Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Project

Multiagent route planning in heterogeneous transport networks

Tomáš Grubhoffer

Supervisor: Mgr. Jan Hrnčíř

Study Programme: Cybernetics and Robotics, Bachelor

Field of Study: Robotics

May 23, 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš Grubhoffer

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Route Planning in Heterogeneous Transport Networks

Guidelines:

1. Survey existing methods for route planning in heterogeneous transport networks.
2. Formalise the route planning in heterogeneous transport networks problem.
3. Propose an algorithm for the formalised problem.
4. Implement the proposed route planning algorithm capable of working with real-world road network, timetables of public transport services and travel demand.
5. Evaluate the resulting route planning algorithm on a set of test scenarios based on real-world data.

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

L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Tomáš Grubhoffer

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Plánování cest v heterogenních dopravních sítích

Pokyny pro vypracování:

1. Prozkoumejte stávající metody pro plánování cest v heterogenních dopravních sítích.
2. Formalizujte problém plánování cest v heterogenních dopravních sítích.
3. Navrhněte algoritmus pro formalizovaný problém.
4. Implementujte navrhnutý algoritmus pro plánování cest, který bude schopen pracovat se skutečnou silniční sítí, jízdními řády hromadné dopravy a cestovní poptávkou.
5. Vyhodnoťte výsledky plánovacího algoritmu na skupině scénářů založených na skutečných datech.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Mgr. Jan Hrnčíř

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

Aknowledgements

Here, I would like to thank my supervisor for his knowledge, advices and patience during the whole project. I would also like to thank my family and my girlfriend for their moral support during the study.

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".

Prague, 23 May 2013 .....

Abstract

This work describes and designs the solution of an algorithm for the Multiagent route planning problem in heterogeneous transport networks. The main task being solved is to find a complete route plan through a heterogeneous transport network for each traveller according to his or her journey request and then optimize all the plans found into one complete plan, where the lowest final price is used as an optimality criterion. The algorithm solves the first-mile problem defined as the path between the requested origin and the PT network. In the first mile part of the route, the passengers can share a taxi.

The algorithm uses the A* algorithm for searching paths through a timedependent graph. The first mile problem is solved by clustering of the passengers and by the VRP (Vehicle routing problem) solver. The algorithm was tested in a real-world scenario with different parametres. The complete price of the transport was decreased using a taxi sharing system.

Abstrakt

Tato práce popisuje a navrhuje algoritmus pro multiagentní plánování cest v heterogenních dopravních sítích. Hlavním úkolem je nalézt kompletní plán cesty skrze heterogenní dopravní síť pro každého cestujícího v závislosti na jejich požadavku a optimalizovat všechny nalezené plány do jednoho úplného plánu, kde je jako optimalizační kritérium použita nejnižší konečná cena. Algoritmus řeší problém první míle, který je definován jako cesta mezi výchozím bodem cestujícího a sítí veřejné dopravy.

Algoritmus používá pro hledání cest skrze graf A* algoritmus. Problém první míle je vyřešen shlukováním cestujících a VRP solveru. Algoritmus byl otestován na skutečných datech s různými parametry. Použitím systému sdílení taxi byla celková cena dopravy snížena.

Contents

Chapter 1

Introduction

This text discusses an algorithm for the multiagent route planning problem in heterogeneous transport networks. A heterogenous network is a connection of two types of networks. The first one we call a scheduled public transport network, where transport is controlled by timetables. We can move through the scheduled PT network for example by bus, train or subway. The second type of network is an individual transport. Individual transport contains car, taxi, bicycle, or walk.

A complete algorithm for planning in heterogenous network has to find demanded routes. Passengers, called agents, can use any type of transport. In this work, we focus especially on the first mile problem and the last mile problem. The first mile problem discusses the first part of agent's trip from the origin to the first PT station. The last mile conversely discusses the last part of trip between the final PT station and the destination.

The task of the algorithm is to find the complete plan of the way through the transport network for each agent. The way is composed from the first mile, the public transport part and the last mile. Another task of the algorithm is to compare found plans. It's because of possibility of sharing individual type of transport (taxi and car) between a group of agents. The algortihm has to compare separate plans and find the most optimal plan for all agents. In our case the optimal plan is the cheapest one.

1.1 Motivation

This paragraph discusses usefulness of the ride-sharing. There are several reasons. The most important is the economical reason. Ride-sharing saves money of passengers. When two agents have similar first mile part of route or the last mile part, they can hire the same taxi and save up to half of the normal cost of the trip, even more when there are more than two agents with the similar trip. Solution used in this project simplifies the sharing only to a taxi sharing system. Taxi can take up to four passengers, which means that maximum reduction of the cost of the trip is a quarter of the original price. Ride-sharing can increase time spent on the agent's trip, but the economical status of the trip is more important for us in this case.

Ride-sharing improves also ecological aspects of the transport. It's similar to the economical reason. When two or more agents share an individual transport, they also reduce air pollution caused by using a car.

Ride-sharing will also decrease traffic density, so the transport would be swifter, there could be less cars in the cities, less traffic jams and all the types of transport would be more effective. For example public transport could be more accurate.

Chapter 2

Literature Review

In this chapter we discuss the literature background of the multiagent planning algorithm. We compare our implementation of major algorithms and methods with algorithms and methods found in the literature.

2.1 Planner combining scheduled and on-demand modes

First, we would to like to mention a method, that solves a similar problem as our multiagent route planning algorithm. This method is described in [5] by Mark E. T. Horn.

In this article author describes a tranport scheduler called LITRES-2 which was tested in Australia. The scheduler is divided into two parts. The first one is the Request broker, where the algorithm of route planning analyses the agent's request. The second one is the Journey planner itself. Journey planner schedules the request using timetables of transport and demand-responsive fleet scheduler.

The author divides transport into three modes which are similar to ours. The first is the timetable mode of transport which includes a Fixed route transport meant as a conventional public transport with timetables. The timetable mode also includes a transport called "SmartShuttle" transport. SmartShuttle is a demanded timetabled transport between PT stations. The second mode of transport is a Free-range mode which includes RovingBus and Taxi transport. The RovingBus transport is similar to the taxi. It's a kind of multiple hire demand-responsive system. The third mode is walking.

Now, the author's steps of algorithm will be described. Firstly, there is an analysis of the Request broker. Secondly, a Journey planner which includes several next steps. In the first step, there is an identification of walkable points. The algorithm uses a spatial interpolation and a clustering technique to define O-pickups and D-setdowns. In our project, we call this points "the first PT station" and "the last PT station". Then, the algorithm defines lower bounds and after identifies initial timetabled pickups of all walkable PT points. It also finds initial RovingBus pickups in the case of using RovingBus transport. After this identification, the algorithm traces legs from the current tier. And then it proceeds to the next one

(complete algorithm is repeated). Now, the algorithm use fleet scheduler and creates the complete plan for all agents.

2.2 The A* algorithm

As mentioned in chapter 1, we have to find complete paths for agents through the transport network. We need an algorithm which will return a minimum cost path in the rated graph. We can use A* algorithm which is described in [4] by P.E. Hart, N. J. Nilsson and B. Raphael. This text describes the problem of finding paths through graphs and the A* algorithm.

Authors are introducing two methods of solving the problem of finding paths. The first one is a mathematical method, which deals with properties of the graph and with algorithms that prescribe an orderly examination of nodes of the graph to establish a minimum cost path. The mathematical method will find the ultimate achievement of solution apartly on calculating difficulty. The second, heuristic method, uses a special knowledge about the domain of the problem being represented by a graph to improve computional efficiency of solutions.

The A* algorithm finds out the minimum cost path through the rated graph. The rated graph is defined as a logic graph, where all edges are rated by a price function. Authors also mentioned an admissible algorithm term. We call an algorithm admissible if it is guranteed to find an optimal path from the start node to the end node.

The A* algorithm works with a cost function $f(n)$, which can be described as the sum

$$f(n) = g(n) + h(n)$$

where n describes a node, which is A* currently working with, $g(n)$ is the cost of an optimal path from the origin of path to the node n and $h(n)$ is a heuristic function, which in our case returns price of an optimal path from node n to the preffered terminal node of n. The heuristic function needs to be admissible. The A* algorithm starts on the start node. It finds out which nodes are expandable and expands them, calculates the value of $f(n)$, then it compares values of $f(n)$ and moves to the node with the lowest cost of $f(n)$. The A* algorithm always finds the best solution of path through the rated graph.

2.3 The first mile and the last mile problem

The main part of this project deals with the first mile and the last mile problems. Both problems are very similar and could be solved similarly. The algorithm for solving the first or the last mile part of the route has to find the path between an ordered origin or a destination and the PT station.

We found many articles that describe these problems and different methods to solve them. For example in [7] solves the problem with a bicycle rental system. The text describes

complete analysis of bike rental system and its first implementation failure. Another method solves our problem by using segway machine, which is described in [9]. Both of these methods cannot be used in our case, because we have to solve these problems for longer distances.

More interesting and more useful are articles [2, 3]. These texts describe the network-inspired transporting system known as NITS. The main task of NITS is to solve the last-mile problem in low-density urban areas. The goal of NITS is to provide a connection between these low-density urban areas, usually with a very low density of transport services, and areas, where a high-density transport system can be found. It divides the complete road network into the big amount of small service areas, called subnets. All of these subnets are being serviced by a demand-responsive transport (e.g. by a taxi or by a car) designed for a door-to-door service within the subnets. When the customer desires to leave the current subnet he has to enter the PT station called gateway and then to use a train or a bus service for transport to another subnet. If the destination is also a low-density transport area out of the subnet, customer has to use a second demand responsive vehicle.

The most optimal paths of the demand-responsive vehicles lead only through places where customers are waiting. For calculating the paths of vehicles in subnets, the NITS uses a dial-a-ride problem algorithm (known as DARP). DARP finds the optimal path for all customers with their demanded origins and destinations.

This text helps us to find a similar solution of solving the first mile problem.

We divide the graph into locations around PT stations, so we do not need subnets as another input into the algorithm. Each location is defined by it's PT station. For each PT station we create a group which represents it. Customers are clustered into these groups based on the destination (the PT station) of the first mile part of their complete route plan.

Instead of the DARP algorithm we use the VRP algorithm, because the VRP is simpler to solve. The article does not contain an evaluation on higher number of passengers. Our solution of the first mile and of the VRP solver is detaily described in the solution approach part of this text.

2.4 The VRP algorithm

This section describes the background of the vehicle routing problem algorithm. We need this algorithm to find the plan of demand-responsive vehicles which are set into areas around PT stations. The VRP algorithm is described in [1]. The VRP algorithm generalizes the Travelling Salesman Problem (TSP). In the text the VRP is defined as

$$VRP = (G, O, d, m, n)$$

Where $G = (V, E)$ represents the graph, where $V = (0, 1, ..., n)$ represents the vertex set and E represents the set of edges. The set $O = (o_0, o_1, ...)$ represents the origins of customers. d represents the depot ($d \in G$), n is the number of vehicles and m represents their capacities. The origin of all vehicles is set to the depot. All edges of the graph G are defined as

$e \in E = \{(i, j) : i, j \in V, i < j\}$. A travel cost c_{ij} is assigned to each edge.

We have defined the VRP differently for our implementation. Instead of a complete graph we create matrix called costmatrix. The costmatrix represents the map for VRP. The size of the costmatrix is equal to the number of customers plus one for the depot. Each customer is represented as a coordinate of the costmatrix. The depot is also represented as a coordinate which is equal to the size of the costmatrix. The values of the costmatrix are calculated using A* algorithm, which returns weight of the path between two coordinates. The number of vehicles is the same as the number of customers and the capacity of vehicles is set to four, because we use the taxi sharing system.

Each customer has to get into the vertex, which is defined as the depot. The task of the VRP algorithm is to get all the customers into the depot, while using vehicles defined by their capacity, and minimize the cost of this plan.

Our definition of the VRP is simpler than the described in [1]. As the input into our VRP solution we need just to create the costmatrix.

Chapter 3

Problem specification

3.1 Problem instance

Our task is to find a complete route for each agent, using a multiagent taxi sharing system. Each route can be divided into two sections - the first mile and through the PT network. The task of the algorithm is to find complete routes through the road network and the public transport network. Agents can travel together in the first mile.

3.2 Timedependent graph

Now, we specify a problem instance, an input into this algorithm in detail. The first input into this algorithm is a graph G which consists from the road network G_R and the public transport (PT) network G_{PT}.

The road network consists from a set of nodes and a set edges. Each node has its own gps location and ID. Known information about each edge includes relation with the set of nodes, length of the edge and mode of transport through it.

The road network is used for solving the first mile problem, the route from the origin to the nearest PT station. In case of the road network G_R, V_R denote intersections and E_R denote roads between them. We use the road network just for a taxi transport, but in some future project, the road network can be also used for the bike transport or for the car transport. The value of the price function w_R is calculated as a price of the taxi per kilometer and the value of the time function t_R is calculated from the average speed on the taxi. The roadnetwork is defined as

$$G_R = (V_R, E_R, w_R, t_R).$$

The PT network G_{PT} is used for route planning through the public transport network. In the PT network, there are nodes denoting PT stations and edges denoting connections between these stations. The value of the price function w_{PT} is given by the value of a ticket

and the value of the time function t_{PT} is calculated from timetables. All stations have specific timetables, where could be find the nearest departure of PT transport vehicles which agents want to use. In our case, the PT network is composed just from a simple train network.

$$G_{PT} = (V_{PT}, E_{PT}, w_{PT}, t_{PT})$$

Connecting the road network with the PT network we create special *cartraintimedependent* graph, where we transform all edges to *pricedtimed* edges. The *pricedtimed* edge gives information about the time and price spent on it. The PT network nodes are situated into the nodes of roadgraph. Respectively, the PT graph is created just from new edges between nodes in roadgraph. The *cartraintimedependent* graph is defined as

$$G = (V, E, w, t)$$

$V = V_R \cup V_{PT}$; $V = (N_0, ..., N_n)$ is set of all nodes which are defined as $N_i = (id, \varphi, \psi)$ where id is the ID of the node, φ is the longtitude and ψ is the latitude. $E = E_R \cup E_{PT}$; $E = (e_0, ...e_m)$ is the set of all edges. Each edge is defined as $e = \{(i, j) : i, j \in V, i < j\}$. $t : E \rightarrow R_0^+$ is a time function, which returns time t_{ij} spent on the edge between nodes N_i and N_j.

$$t(e) = \begin{cases} t_R(e) \ for \ e \in E_R \\ t_{PT}(e) \ for \ e \in E_{PT} \end{cases}$$

$w : E \rightarrow R_0^+$ is a price function, which returns price p_{ij} spent on the edge between nodes N_i and N_j. This price is calculated from the length of the edge and used mode of transport.

$$w(e) = \begin{cases} w_R(e) \ for \ e \in E_R \\ w_{PT}(e) \ for \ e \in E_{PT} \end{cases}$$

3.3 Travel demand

We also need to discuss one more input into our algorithm, the travel demands. A travel request is described as a set of agents

$$A = \{a_1, ..., a_n\}$$

where $a_1, ..., a_n$ are agents. Every agent $a \in A$ is defined as

$$a = (o, d, \tau)$$

where o is the origin of agent's trip, d is his destination and τ is agent's departure time.

Finally, the multiagent planning problem can be defined as $M = (G, A)$, where G represents the complete transport network and A represents all travel requests.

3.4 Expected output

Now, we describe an instance of a solution. Expected output is a set of plans P for all agents

$$P = \{p_1, ..., p_n\}$$

This set P is composed from plans p where plan p_i is found plan for agent a_i and can be described in a form of

$$p_i =< o_1 \rightarrow ... \rightarrow o_f \rightarrow ... \rightarrow o_n >$$

where $o_i \rightarrow o_{i+1}$ represents a travel between two nodes in the transport network. In each plan, there are also special nodes – o_f is the first PT station of agent's path. Each path $p =< o_1 \rightarrow ... \rightarrow o_n >, p \in P$ needs to be a valid path from o_1 to o_n in the graph G.

Chapter 4

Solution approach

4.1 Description of the algorithm

In the previous chapter, we described the necessary input and the expected output. Here, we would like to present our solution of the multiagent route planning problem. We divided the algorithm into four phases: the *graph creation phase*, the *initial planning phase*, the *agents clustering phase* and the *joint planning phase*. All phases take output from the previous phase as their input and create a new output for the next phase.

4.1.1 Graph creation phase

In the first step, we need to create a graph G which consists from the road network G_R and the PT network G_{PT}. The steps of building the graph G are described in the problem specification. In this phase the algorithm also creates requests, which are described as a set of agents A. When the graph is built and the requests are checked, the algorithm moves to the next phase.

4.1.2 Initial planning phase

We have to find a complete route from the origin o_i to the destination d_i for each agent $a_i \in A$ separately. Agents route can be divided into two parts - the first mile part and the route across the PT transport. For finding the route through the graph G, we use the A* algorithm, which is described in the background part of this text. For the A* algorithm we need to define the heuristic function $h(n)$. The heuristic function $h(n)$ is in our case defined by the destination of plan and speed (in kmph) of the agent move in the graph. The value of the heuristic function for node $n \in G$ is described as

$$h(n) = distance(n, terminalnode)/averagecrusingspeed$$

where *terminalnode* is the demanded destination, *distance* is a function, which returns the distance between nodes. In our implementation the *averagecruisingspeed* is set on 70 kilometres per hour.

After the A* search, all agents have their own route plan, from which can we find information about its price, length and duration. An example of found initial plan is given in the Implementation chapter 5.2.

4.1.3 Agents clustering phase

When the initial plans for all agents are made, we can cluster agents into groups, which will be used as input for the next phase. At first, we cluster agents by the first PT stations where agents enter the PT transport network. The number of groups created by PT clustering is $n = m + 1$, where m is the count of the PT stations located in the graph. There is one more group for a situation when agents don't travel through PT network. Groups can be described as a set

$$g = (g_0, g_1, ..., g_n).$$

Each group $g_i \in g$ has an ID and also carry information about the node in which the PT station is situated. Each PT station has the same ID as ID of node in which the PT station is situated.

Second clustering is based on departure time of each agent. In this clustering, we create new groups, which are technically subgroups of groups created in the clustering by PT station. We cluster all agents into subgroups by the hour of the day. For each group $g_i \in g$ we create set of subgroups $g'_i = (g^0_i, ..., g^j_i)$ where the top index represents the hour of the day. The set $g' = (g'_0, ..., g'_n)$ represents all created subgroups. For each agent $a_k \in g_i$ with departure time τ_i we find the subgroup $g^j_i \in g'$ represented with an hour j and the time slot $T = (j, j + 1)$, where $\tau_i \in T$. An example of the clustering phase can be found in 5.1.

As the input of this phase, we get set of groups

$$g' = (g^0_0, ..., g^m_0, ..., g^m_n)$$

where n is a number of created subgroups, $i \in m$ represents the time slot.

4.1.4 Joint planning phase

In the next part, we take created subgroups and find a new route plan for agents in them using the VRP solver. We use VRP solver only for the first mile part of the route. VRP solver returns a complete plan for each group $g^j_i \in g'$ using the taxi sharing system. Because of the taxi capacity, the maximum number of agents who could travel together is four.

Each plan for each group $g^j_i \in g'$ is calculated separately. As input into the VRP solver we need costmatrix, which represents the map for the VRP. Each group has its own costmatrix. In the VRP solver we also need to set a number of vehicles which could be used for transporting. The number of the vehicles equals to the number of agents in the group. The VRP solver also needs to define the agents' depot, which is set as the PT station, described in the group information.

The algorithm returns agents with a complete plan of route. From this plan, we can find out the information about the price, the length and the duration of agent's trip. An example of this phase is given in 5.4.

4.1.5 Pseudocode

This paragraph is a short summary of the algorithm. We connected phases which are described aboveand create the table which represents the pseudocode bellow.

Input

- roadgraph G_R connected with PT network graph G_{PT}
- set of agents $A = \{a_1, ..., a_n\}$ where each agent has an origin, destination and departure time

Initial planning phase
For $i = 1, ..., n$ do

- for agent a_i find a single agent plan from origin o_i to destination d_i
- find the first PT station of the plan p_i for future clustering

Agents clustering phase

- cluster agents by the first PT station into groups $g_1, ..., g_N$ where N is the number of PT stations.
- cluster each group separately by departure time of agents into the subgroups $g' = (g_0^0, ..., g_0^m, ..., g_n^m)$

Joint planning phase

- use VRP solver for each subgroup $g_i^j \in g'$ and find new plan for all agents in the subgroups.
- return set of subgroups g'
- each subgroup $g_i^j \in g'$ has its own plan of paths for all agents in the subgroup

Output

- for each agent $a_i \in g_i^j$ return plan p_i which is based on complete plan of the group g_i^j

4.2 Last mile solution

In this project we are working just with the first and the second part of agent's journey through the network. In the first part, we solve the first mile problem, which is the leg between the origin and the first PT station, and in the second one we find the complete route through the public transport network.

In this section, we would like to describe how could be solved the last part of agent's plan, known as the last mile problem, which is the path between the last PT station and the demanded destination. The solution of the last mile part of agent's plan is very similar to the first mile part. In the first step, we find a path between the last PT station and agent's destination for each agent. Secondly we cluster agents into groups in the same way as in the first mile problem - by the last PT station of the PT part of the route and by the time when agent checks out of the PT station. For each agent $a_i \in A$ we set the depot as the agent's a_i destination. Then we start the VRP algorithm, which returns us a complete plan for the last mile part of the complete route.

After that, we can put all three parts, the first mile part, the path through PT network and the last mile part together and create the complete plan for all agents.

Chapter 5

Implementation

This chapter describes our implementation of the multiagent route planning problem. We describe our method of solving all steps from the solution approach. To illustrate each step we create small scenario.

5.1 Design and architecture

As the first task, we must generate input elements into our program. The timedependent graph is built from an OSM (Open Street Maps [10]) file. In the OSM file, which is technically XML file, we can find information about nodes, edges and relations between them. Sample OSM file node structure:

```
<node id="316944691" version="1" timestamp="2008-12-06T13:42:22Z"
uid="80081" user="JamesCorner" changeset="214438"
lat="53.5974177" lon="-1.201174"/>
```

Each node in the OSM structure is defined by its node id (in example 316944691) and its gps location ($lat = "53.5974177", lon = " - 1.201174"$). Edges are denoted as a sequence:

```
<way id="8195401" version="3" timestamp="2008-01-25T18:25:54Z"
uid="5121" user="user_5121" changeset="692326">
  <nd ref="343663"/>
  <nd ref="343662"/>
  <tag k="highway" v="motorway_link"/>
  <tag k="layer" v="1"/>
  <tag k="oneway" v="yes"/>
  <tag k="bridge" v="yes"/>
</way>
```

The OSM file contains very detailed information. It also describe the type of road - in the example we can find out that the sequence of edges describe a highway bridge. This information is not important for us in this implementation. We need just the information about relation between the nodes and the edges.

Based on this information we create *timedependent* graph in which we have the same information as in the OSM file, but from the *timedependent* graph we can also get information about time spent on edges. Now, we need to create the PT graph. There are two possible ways to solve this task. The first solution is to built an independent PT graph connected with roadgraph by walking edges. The second possible solution is to create list of edges, which describe the PT graph and put it into the roadgraph. The condition for the second solution is that the nodes, which are in relation with these edges, must lie on the roadgraph.

We use the second method. We create a new type of edge called *GraphStopEdgeInterval*. This type of edge carries information about its initial node, terminal node, used mode of transport and the interval of the PT. The interval is defined as a time between two departures of the PT vehicle. If we know all PT stations and their location, we can build the PT graph and connect it with the road graph.

In the next step we take our *timedependent* graph and create a new graph, which carries information about used mode of transport on all edges, called *cartraintimedependent* graph.

In our implementation we connected the roadgraph for the taxi service with the PT network, which is represented as a train line.

For better illustration, we present a screenshot of the created *cartraintimedependent* graph. We have created special function (based on [11]) which paints the graph into the KML (Keyhole Markup Language) file. The KML file can be opened using the Google Earth program [6].

Figure 5.1: Screenshot of the part of transport graph taken in Google Earth

The green lines represent the road network. The blue lines represent the PT network.

Secondly, we need to create the travel demand. The travel demand is described as a set of journey requests. Each journey request is described by the origin, the destination and the requested departure time. For creating a journey request, we can use the generated *timedependent* graph. We pick randomly two nodes from the graph, set them as agent's origin and destination. Then we generate a random time and set it as agent's departure time. We also generate an agent's ID and set their group number ID as 0. And then we set agent's plan as null. Agent's plan describes the path between origin and destination. Then we put the agent into the travel demand list.

As an example scenario, we created small list with ten requests. The origins of all requests are randomly set into the areas around two PT stations (requests with IDs 0-4 are set into the location around the Doncaster PT station, requests with IDs 5-9 are set around the Kirk-Sandall PT station). The destinations of all requests are set into another PT station. The departure time of each requests is randomly set into a two hour timewindow. We present an text output from this phase bellow

```
Agents id: 0 group: 0 Origin ID: 790628892 departure time: 12:12
Agents id: 1 group: 0 Origin ID: 790994959 departure time: 13:38
Agents id: 2 group: 0 Origin ID: 272858886 departure time: 12:09
Agents id: 3 group: 0 Origin ID: 715245877 departure time: 13:16
Agents id: 4 group: 0 Origin ID: 715245759 departure time: 12:00
Agents id: 5 group: 0 Origin ID: 369600406 departure time: 13:51
Agents id: 6 group: 0 Origin ID: 617024889 departure time: 13:46
Agents id: 7 group: 0 Origin ID: 286586357 departure time: 13:03
Agents id: 8 group: 0 Origin ID: 715321456 departure time: 12:01
Agents id: 9 group: 0 Origin ID: 368735056 departure time: 12:07
```

Now we find an initial plan for all agents. For this we use the A* algorithm. Inputs for the A* algorithm are the graph, two nodes describing the demand called startvertex and endvertex and the heuristic function.

When inputs are set, A* algorithm calculates a complete route for the agent. We pick an agent from the list, find path through the graph for him and set this path as an agent's plan.

As an example of this plan, we present you a KML file with a figure of an agent's plan. The first mile part of the plan is colored in red, the PT part is colored in green.

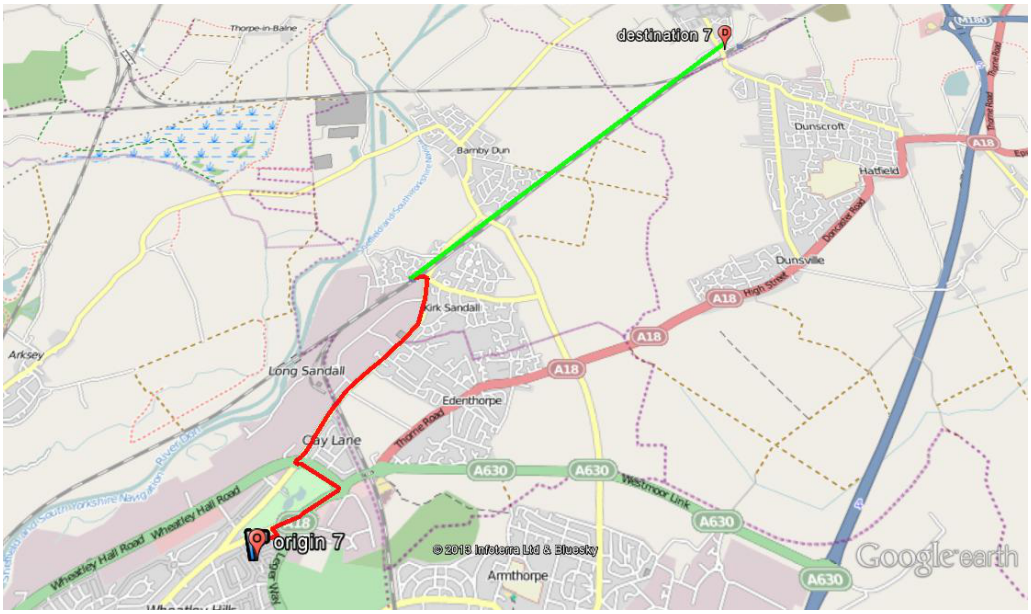

Figure 5.2: Plan of the route for agent with ID 7

We have a list of all agents, where each agent has his own plan through the transport network. From this plan, we find out the information about agent's travel. For example, we find the PT station, where agent enters the PT network. We also find the price or time necesarry to spent using this plan.

This information is used for the clustering part of our algorithm. First, we take all PT stations and create a list of groups, which carries information about them. Each group has its own ID and describes the node of the PT station. We also create a special group with ID 0, which is used for agents who do not use the PT network, respectively their path does not go through the PT network. Now, we take the list of agents, compare the information about the first PT station of their plan with the stations in the list of groups and when the PT station equals, we set the agent's groupID to the ID of group, which describes the same PT station.

As an example of the PT clustering we present our small scenario again. After the PT clustering, three groups were created:

```
Group id: 0          PT station ID: no station
Group id: 364648619 PT station ID: 364648619
Group id: 410367517 PT station ID: 410367517
```

We got agents seperated into the groups, which are situated into small locations around the PT stations. Now we need to cluster them into groups by time. Each group is situated

into an hour timewindow. We pick a group from group list created in the step before. From this group, we create subgroups, distributed by hours of the day. Then we pick our list of agents, find agents from the same group, put them into the new subgroups in dependence of their departure time and their new groupIDs. Now we have smaller groups, which are situated into the locations around PT stations and into the requested hour.

In the small scenario we created these groups:

```
Group id: 1 PT station ID: 364648619 Description: group time (hour of the day): 12
Group id: 2 PT station ID: 364648619 Description: group time (hour of the day): 13
Group id: 3 PT station ID: 410367517 Description: group time (hour of the day): 13
Group id: 4 PT station ID: 410367517 Description: group time (hour of the day): 12
Group id: 0 PT station ID: no station
```

The group with id 0 stays without time description, because it serves for agents who travel out of the PT network. This group also does not enter the VRP part of the algorithm. The agents were clustered into these groups following:

```
Agents id: 0 group: 1 departure time: 12:12
Agents id: 2 group: 1 departure time: 12:09
Agents id: 1 group: 2 departure time: 13:38
Agents id: 3 group: 3 departure time: 13:16
Agents id: 5 group: 3 departure time: 13:51
Agents id: 6 group: 3 departure time: 13:46
Agents id: 7 group: 3 departure time: 13:03
Agents id: 4 group: 4 departure time: 12:00
Agents id: 8 group: 4 departure time: 12:01
Agents id: 9 group: 4 departure time: 12:07
```

Possible solution of the next step is to cluster agents into groups by the length between their origins, which returns us groups with agents, who can travel together in taxi to the first PT station. Based on the initial testing, this solution is not so effective, so instead we use the Vehicle routing problem solver (VRP).

The VRP solver creates a complete plan for agents in subgroups. For the VRP, we need to create a costmatrix, which describes all possible paths between agents origins in the subgroups themselves and between the PT station. The coordinates i, j of costmatrix describe the price between the origin of the agent i and the origin of the agent j. The size of the costmatrix is the number of agents in the subgroup plus one for the PT station. For filling the costmatrix, we use the A* algorithm again, because it returns us the smallest price. In the diagonale of the costmatrix we set the price to 0. Then we fill the costmatrix using the A* algorithm. When the costmatrix is filled, we can start the VRP algorithm phase.

As an example of the costmatrix we present the values of costmatrix, which represents the map for the VRP solver of the group 4. This group consists of three agents (with IDs 4,8,9). The size of the costmatrix for this group is 4.

$$\begin{pmatrix} 0.0 & 204.39363 & 160.25441 & 304.04202 \\ 204.39363 & 0.0 & 73.96908 & 123.65029 \\ 160.25441 & 73.96908 & 0.0 & 198.36757 \\ 304.04202 & 123.65029 & 198.36757 & 0.0 \end{pmatrix}$$

The value 204.39363 in the coordinates $[2, 1]$ and $[1, 2]$ is the weight of the route between two agent with ID 4 and 8, which are set into these coordinates. It is important to say that the coordinate number in the costmatrix may not be the same as the agent's ID.

We use created costmatrix as the input into the VRP algorithm. In the VRP, we need to set several elements - map, customers, depot and vehicles. The costmatrix is set as a map for the VRP. As customers we use our agents. As the depot for the VRP we set the PT station of the subgroup. Then we need to set vehicles, their number, capacity and their location. In our case, we use taxi as a vehicle, so the maximum capacity is set to four. The starting location of all vehicles is set to the depot. After all elements being created, we can start the VRP algorithm and calculate new plan for all subgroups.

Thanks to the VRP solver we have a plan of the first mile part of the route and the path through PT network for all agents, including information about which agents can travel together, the price and the time spent on this plan. Connecting both these path we create a complete plan for all agents.

For illustrating the complete output from the algorithm we present images of the planned path before and after the VRP solver for group 4.

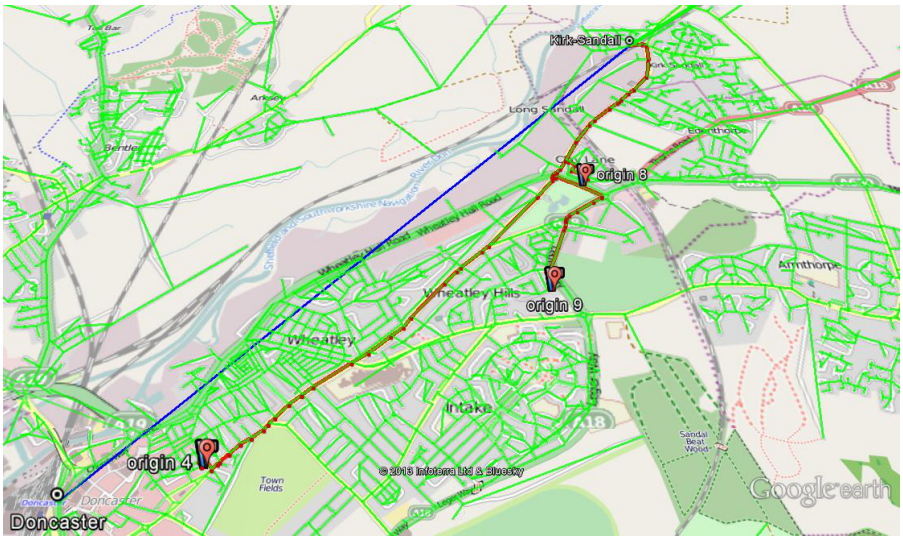

Figure 5.3: Complete route plan for group 4 before the VRP

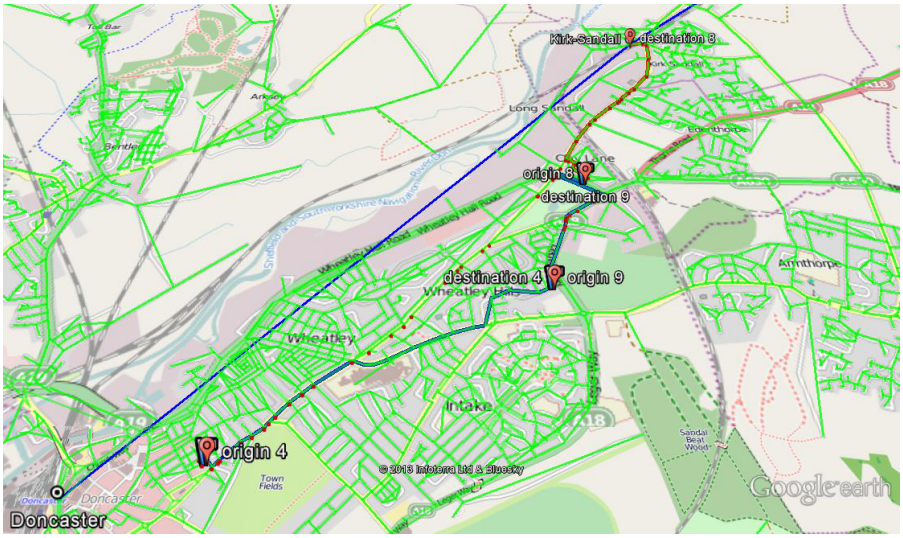

Figure 5.4: Complete route plan for group 4 after the VRP

In 5.4 we can see the complet plan for the group 4. The path starts at the agent's 4 origin. Then the path leads to the agent's 9 origin and then to agent's 8 origin. Then the vehicle go to the depot described as destination 8 in the picture.

5.2 Implementation details

This part discusses details of used algorithms in our implementation of the multiagent route planning algorithm. In our implementation, we have used two algorithms - the A* algorithm and the Vehicle routing problem solver.

5.2.1 The A* algorithm

As we said, for searching paths through the transport network we use the A* algorithm. A short description of the A* algorithm can be found in the background part of this text. We repeat that the A* algorithm evaluates each node, it has visited, by evaluation function $f(n)$. The evaluation function is defined as the sum of $g(n)$, the cost function, and the $h(n)$, the heuristic function. It is not necessary to discuss the cost function, because it returns the lowest cost of the path to the node n. The price of all edges is calculated from the length of each edge and the used mode of transport. We simply multiply the length of the edge in kilometres by the price of used transport per kilometer.

In our project, we are working with two modes of transport. The first one is the public transport, which is in our case represented by a train line. The second we can describe as a on-demand transport (taxi service). The prices of taxi services are much higher than prices of the train. This comes out from the real world data, where it is usual that the prices of the taxi services are more expensive.

We also need to define the heuristic function $h(n)$ for our A* implementation. The heuristic function estimates the time spent between the node n and the demanded terminal node. Duration of this time is calculated from the distance between the present node n and the target node divided by the cruising speed on the graph. We set the cruising speed on both PT graph and road graph on seventy kilometres per hour.

5.2.2 Vehicle routing problem solver

The VRP solver is described in the Literature review part. Because of the difficulty of programming an individual VRP algorithm, we have searched the internet for a library with the VRP solver. For our purposes we discovered the Metavrp (the Metaheuristic vehicle routing problem java library). The biggest reason, why we picked Metavrp instead of another solutions, is that it already is a java library and it is easier to integrate into our program, which is also written in Java language. The Metavrp is a simple opensource library, which can be downloaded from [8]. It solves the VRP and the travelling salesman problems.

We need to modify Metavrp for our implementation. We created our own costmatrix. The costmatrix is in our implementation built from all agent's' origins and paths between them and between the first PT station. The original Metavrp created the same number of customers as is the costmatrix size, otherwise the algorithm threw an exception. We modified the Metavrp algorithm and removed this exception. We also set the number of recalculations of the VRP algorithm on fifty thousand (in the original implementation, it was set on ten thousand). We removed the text outprints too, which makes the Metavrp algorithm much faster. As the input from the VRP we create path for vehicles for each group. The path is a sequence of numbers, where each number describes the position of the customer or the depot. This sequence must be transfered into the path through graph. While building the costmatrix, the algorithm also remembers routes between the coordinates of costmatrix, so each pair of coordinates has its own route plan. From this and the output sequence from Metavrp we can build a complete plan for each group.

5.2.2.1 Short description of the VRP solver

In this section we present a table with a short description of an implementation of the VRP solver.

1. Initialization
   - initialize customers $a_0, ..., a_n$
   - create and set agent's' depot a_{n+1}
   - create cost matrix $C[n+1, n+1]$
   - calculate value of the cost matrix for each coordinate using A* algorithm
   - create n vehicles $v_0, ..., v_n$ with their capacities and set their starting position into the depot
2. Calculate
   - minimize the cost of plan of getting all agents into the depot
   - return the sequence of coordinates of the best solution
3. Output
   - create plan for all customers from the sequence returned by the calculation phase

5.3 Programming the multiagent route planning algorithm

Here, we would like to say several words about the program implementation. For programming the algorithm we used Java language. We present an action diagram of the *Run* class in 5.5, which is the class where the algorithm run completely.

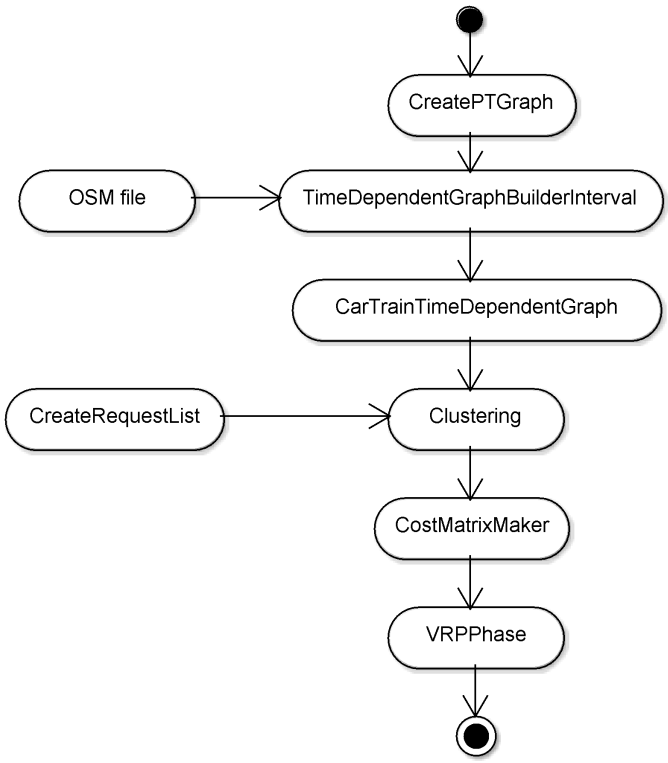

Figure 5.5: Diagram of the Run class

The class *TimeDependentGraphBuilderInterval* builds a *timedependent* graph from an OSM file and from the output from *CreatePTGraph* class where the PT network is created. Then, the *timedependentgraph* is send to the *CarTrainDependentGraph* class where it is transformed into the *CarTrainDependentGraph*.

The requests are created in the *CreateRequestList* class. This class create random agents with their demands and transform them into the requests.

The *CreateRequestList* class send a list of request into the *AStarShortestPath* function, which creates an initial plan for each request by searching the *CarTrainDependentGraph*. After then, the path is used for clustering agents into the groups. The clustering phase can be found in the *Clustering* class. The *Clustering* class returns list of requests separated into groups.

This list serves as an input into the *CostMatrixMaker* class, where the costmatrix is built for each group. Then, the costmatrix is send to the *VRPPhase* class, where the elements for the VRP solver are created. It also run the VRP and create a complete plan for each group.

The implementation is based on [11]. It uses some modificated classes from this software.

The description given here is an overview. The detailed description of the algorithm can be found in Javadoc on attached CD.

Chapter 6

Evaluation

6.1 Scenario

In this section we specify a concrete testing scenario for the multiagent planning algorithm. We focus on the area around the city of Doncaster in Yorkshire in the United Kingdom. Using the Osmosis program we extract detail road graph around this train line from the United Kingdom map into OSM map file. The created OSM file takes information about all nodes, for example their real gps location, and about the relations between them. The manual for cutting osm maps and information about osm map files and Osmosis program can be found at website [10]. The openstreetmap is project which creates and distributes world geographic data for free.

As the PT network we pick one line of train, taken from real world data, situated in this location. The line of train includes eight stations - Doncaster, Kirk-Sandall, Hatfield-Stainford, Thorne, Crowle, Althorpe, Scuntthorpe and Barnetby. For easier implementation we create this PT network connecting nodes which can already be found in the roadgraph. We just have to set the used mode of transport on these connections to train transport. Another thing, which is needed to specify is the train arrival interval. For better and easier calculations we set the arrival interval of each train on the half of an hour.

Now we set the number of agents in travel demand. For testing this scenario, we want to have an average of 4-16 agents in each group. The demands are distributed into the ten hour timewindow (agents travel between 8am - 6pm). With eight PT stations, the algorithm created about 88 groups. The size of the travel demands should be between 400 and 1200.

The demanded path of an agent is created the following way. We pick two random nodes from the graph in minimum distance of 10 kilometres. Because we are working just with the first mile, we set the first node as the agent's origin. As the destination, we set the closest PT station to the second node. This guarantee that the plan consists from the first mile part and the PT part. The agent's departure time is also created randomly from the 8am - 6pm timewindow.

6.2 Metrics for evaluation

We measured cost improvements of the subgroups situated in hour timewindows depending on agents density. Secondly, we measured the complete runtime of the multiagent route planning algorithm also in dependence of the agents density. We picked three different agent densities - 400, 800 and 1200. For each density we made several runs and measured cost improvements. We also measured the runtime in all densities. As a result of this test we present several graphs bellow.

6.3 Results

The results of testing the scenario are presented in graphs bellow. The results can be divided into three sections - the results of the runtime test of the algorithm, the cost improvement results and the results of prolongation of agent's path.

6.3.1 Runtime of the algorithm

At first, we want to present the result of the runtime test of our implementation of the Multiagent route planning algorithm. In the first graph you can find the dependings of the run time on the number of agents in demand. We tested run time on three demands sizes - 400, 800 and 1200. We measured the time in miliseconds, which is needed for creating the random agents list, find their initial paths, clustering phase and the VRP planning phase. The runtime of the creation of the graph and the KML files drawing is not included in this runtime test.

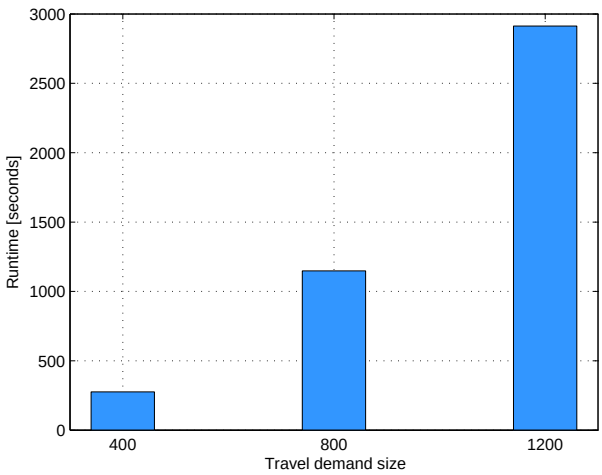

Figure 6.1: The runtime test of the multiagent route algorithm in dependence with the size of the demand

The runtime of the algorithm highly increases with growing density of agents. This high increase is mainly caused by the VRP solver. With growing density of agents, the count of

agents in groups increase and the sizes of costmatrixes increase with it. That makes calculation of VRP more complicated.

In the next runtime test, we show the runtime of each part of the algorithm separately. We ran this test again for three sizes of demand. As the result of this test, we present graphs bellow.

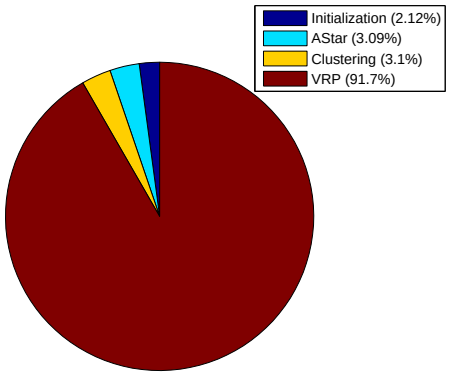

Figure 6.2: The runtime test of the multiagent route algorithm with phases separately (size = 400, complete runtime = 5.76 minutes)

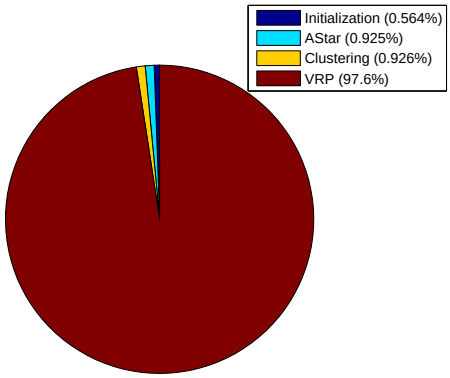

Figure 6.3: The runtime test of the multiagent route algorithm with phases separately (size = 800, complete runtime = 22.66 minutes)

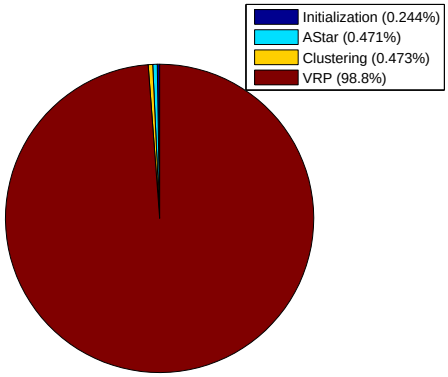

Figure 6.4: The runtime test of the multiagent route algorithm with phases separately (size = 1200, complete runtime = 51.14 minutes)

In these graphs we can see the duration of each phase of the multiagent route planning algorithm. We separate algorithm into four phases - the initialization, where graphs and demands are created, the AStar phase, where an initial plan for each agent is created, the clustering phase, where agents are ordered into groups, and the VRP phase, where the final plan is calculated. The fastest phase of the algorithm is the clustering phase, which is also the simplest one. The longest is the VRP phase. It is caused by calculating the values of costmatrixes by A* algorithm and then by final optimalization of the plan. With growing size of the demand the runtime of the VRP highly rise.

6.3.2 Cost improvement of agents paths

In this section we would like to discuss results of the average cost improvement of agents' paths in the test of the algorithm. The average cost improvement is calculated from all average cost improvements of all groups. The average cost improvement of a group is calculated subsequently:

$$AverageCostImprovement = (CostBeforeVRP - CostAfterVRP)/CostBeforeVRP$$

This equation returns us the percentage of the average improvement. As the result of cost improvement we present the graph bellow where is showed the average cost improvement in dependence with the size of the demand.

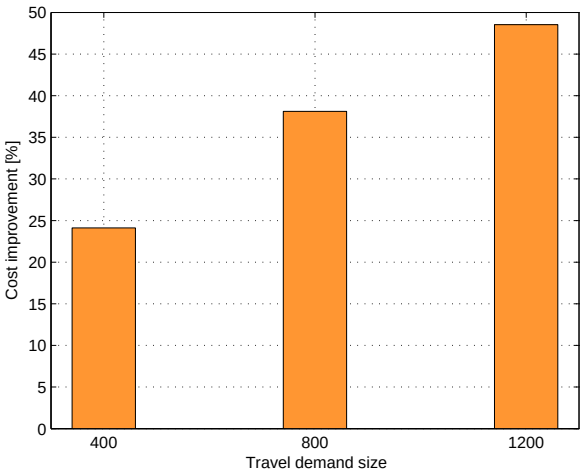

Figure 6.5: The runtime test of the multiagent route algorithm in dependence with the size of the demand

The cost improvement gets higher with growing density of agents. When the size of the list of the demand grows, the density of the map for the VRP grows too. Better solutions can be found and the cost improvement gets higher.

6.3.3 Average prolongation of agents paths

Now we would like to present results of the average prolongation of agents path. The average prolongation of the complete plan is calculated from average prolongations of all group plans. We calculated both distance prolongation (in kilometres) and time prolongation (in minutes) in relation with the size of the demand. We present both results in graphs bellow.

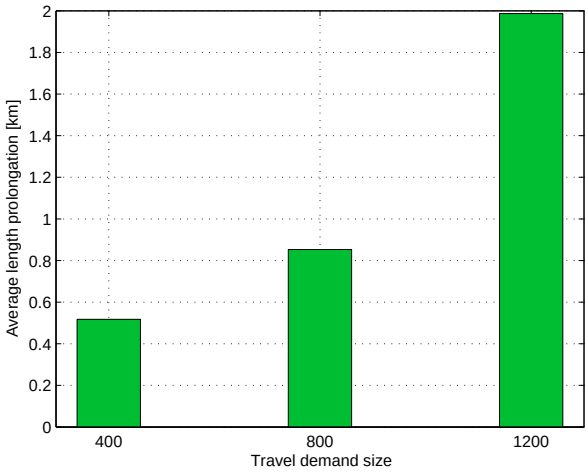

Figure 6.6: The average length prolongation of the plans in relation with the size of the demand

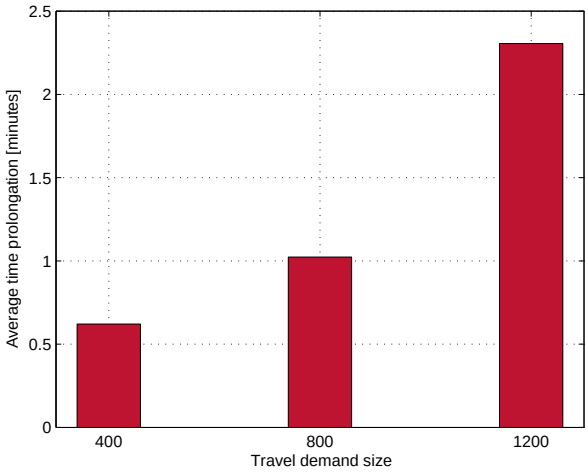

Figure 6.7: The average time prolongation of the plans in relation with the size of the demand

The results are quite interesting. These graphs show us that the with growing size of the demand list (density of agents), the prolongation increase. Possibly it might be caused just by randomization of creating the requests.

We also present the results of the prolongation of group plans in the dependence with the cost improvement. The prolongation increase with growing cost improvement.

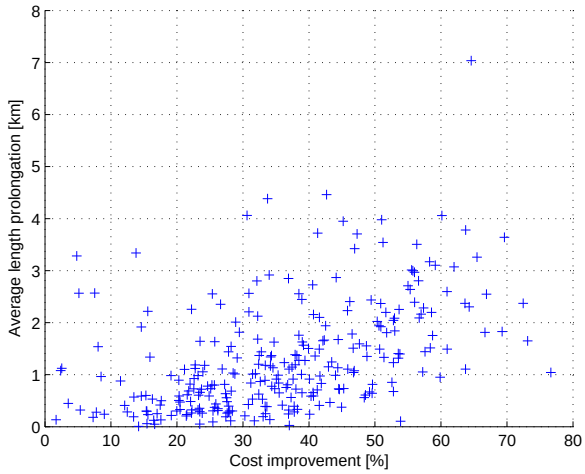

Figure 6.8: The average length prolongation of the plans in relation with average cost improvement

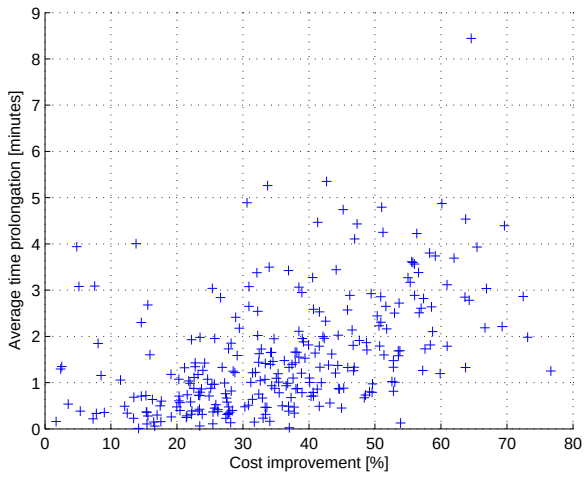

Figure 6.9: The average time prolongation of the plans in relation with average cost improvement

6.4 Problems

In the testing phase of this project some problems occured. For all results, that are shown there, we made ten runs and make an average of them. The results were always different. That was because of randomization of created demands. Another problem has been found in the VRP solver - 1.2% paths were calculated wrongly. The prices of these paths were higher after calculating VRP. We removed these plans from the evaluation. These problems might be caused by modification of the MetaVRP library.

Chapter 7

Conclusion

7.1 Discussion and future work

In this work, we found possible solution of the multi agent route planning algorithm. We have separated the algorithm into four steps: the *graph creation phase*, the *initial planning phase*, the *agents clustering phase* and the *joint planning phase*. We described all these phases of the algorithm and ran them in several tests.

The initialization part of the algorithm meaning the *graph creation phase* ran without any problems. The graph and the demand were created very fastly. In the *initial planning phase* where inital plans of path through graph for each agent is found, we have used the A* algorithm. The implementation of the A* algorithm returns the lowest cost path in an acceptable time. The agents in the *agents clustering phase* were clustered into groups in a short time. In the the *joint planning phase* where joint plan is created, we have used the vehicle routing problem solver. The Metavrp library occurs as quite slow, but solving the VRP for a group of agents can be parallelized. The calculation of the VRP plan takes a lot of time. It might be caused by our modification of the algorithm.

The solution of the multi agent route planning problem in heterogeneous transport network, which is described by this text, solves the first mile problem using a taxi sharing system. The first mile problem is defined as a route between requested origin and the PT network.

In the future project we will focus on the VRP solver, which needs to be improved by optimizing the calculation and decreasing the run time of the VRP. We will also create new solution where will be the last mile problem solved. The last mile problem is defined as the path between the PT network and the requested destination.

Bibliography

[1] Jean-Francois Cordeau and Gilbert Laporte. Modeling and Optimization of Vehicle Routing and Arc Routing Problems. *: Handbook on Modelling for Discrete Optimization*, 88:151–191, 2006. doi: 10.1007/0-387-32942-0_6. URL <http://dx.doi.org/10.1007/0-387-32942-0_6>.

[2] Derek Edwards, Aarjav Trivedi, Arun K. Elangovan, and Steve Dickerson. The network-inspired transportation system: A hierarchical approach to integrated transit. 1:1507–1512, October 2011. doi: 10.1109/itsc.2011.6083066. URL <http://dx.doi.org/10.1109/itsc.2011.6083066>.

[3] Derek Edwards, Arun K. Elangovan, and Kari Watkins. Reaching low-density urban areas with the network-inspired transportation system. 1:826–831, September 2012. doi: 10.1109/itsc.2012.6338650. URL <http://dx.doi.org/10.1109/itsc.2012.6338650>.

[4] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 0536-1567. doi: 10.1109/tssc.1968.300136. URL <http://dx.doi.org/10.1109/tssc.1968.300136>.

[5] Mark E. T. Horn. Procedures for planning multi-leg journeys with fixed-route and demand-responsive passenger transport services. *Transportation Research Part C: Emerging Technologies*, 12(1):33–55, 2004. doi: 10.1016/j.trc.2002.08.001. URL <http://www.sciencedirect.com/science/article/pii/S0968090X03000767>.

[6] Google Inc. Google earth, 2013. URL <http://www.google.cz/earth/>. [Software].

[7] Zhili Liu, Xudong Jia, and Wen Cheng. Solving the Last Mile Problem: Ensure the Success of Public Bicycle System in Beijing. *Procedia - Social and Behavioral Sciences*, 43:73–78, January 2012. ISSN 18770428. doi: 10.1016/j.sbspro.2012.04.079. URL <http://dx.doi.org/10.1016/j.sbspro.2012.04.079>.

[8] Metavrp. Metaheuristic vehicle routing problem java library - google project hosting, 2013. URL <http://code.google.com/p/metavrp/>. [Java library].

[9] Susan A. Shaheen and Rachel S. Finson. Bridging the Last Mile: A Study of the Behavioral, Institutional, and Economic Potential of the Segway Human Transporter. URL <http://publications.its.ucdavis.edu/publication_detail.php?id=293>.

[10] OpenStreetMap Wiki. Osmosis - openstreetmap wiki, 2013. URL <http://wiki.openstreetmap.org/w/index.php?title=Osmosis&oldid=888510>.

[11] Radek Holý Zbyněk Moler, Jan Hrnčíř. planner-data-importer: A tool for creating a road planning graph based on the given osm file developed in the agent technology center, 2013. [Software].

CD content

Included CD contains text of this thesis in PDF format, the source code of our implementation and the Javadoc of the code.

Directory	Description
text.pdf	the text of this thesis
code	contains the source code
doc	contains the Javadoc