Less obstructed street traffic

Experience-based routing to alleviate traffic congestion

Project Report
D505E15

Aalborg University
Computer Science

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfsvej 300
9000 Aalborg
http://www.aau.dk

Title:
Less obstructed street traffic

Theme:
Intelligent or Massively Parallel Systems

Project Period:
Fall Semester 2015

Project Group: d505e15

Participant(s):
Andreas Hairing Klostergaard
Benjamin Ahm
Kristian Hauge Jensen
Michael Jensen
Morten Rune Peterson
Morten Korsholm Terndrup

Supervisor(s):
Bin Yang

Copies: 3

Page Numbers: 53

Date of Completion:
December 15, 2015

Abstract:

This report looks at the possibility of using machine intelligence and distributed systems to gather and analyse traffic data. The main focus of this project is machine intelligence where distributed systems is a support part. The system need to split up traffic so the route network is better utilise. A lot of software used in this project is not made by the group, but taken from other free or open sours projects because these elements is not the focus of this project.

Contents

Todo list

Preface

Here is the preface. You should put your signatures at the end of the preface.

Thank the Track-Matching guy.

Aalborg University, December 15, 2015

Andreas Hairing Klostergaard
<aklost11@student.aau.dk>

Benjamin Ahm
<bahm13@student.aau.dk>

Kristian Hauge Jensen
<kje13@student.aau.dk>

Michael Jensen
<mj09@student.aau.dk>

Morten Rune Peterson
<mopete13@student.aau.dk>

Morten Korsholm Terndrup
<mternd13@student.aau.dk>

Introduction

Transportation and traffic problems are an issue in today's societies. It has been estimated that the total number of vehicles in the world will have increased by more than 150% between 2002 and 2030[3], which will in all likelihood aggravate the current transportation problems. Furthermore, many of these problems become accentuated when notable events occur, such as concerts, holidays, and traffic accidents. A well-planned infrastructure can alleviate many of the traffic issues, but is expensive to change when requirements for traffic change. Therefore it is advantageous if traffic problems can be alleviated by better utilisation of existing road networks. Such an alleviation would be possible if drivers would plan their routes based on how the traffic situation usually is in their area. People usually have a good idea of such *traffic patterns* in their local community, and as such it would be useful if those traffic patterns could be defined and used by many drivers, to take into account the local traffic situation. However, it is costly for people to report such patterns for others to use, and therefore it is interesting to explore the possibilities of automatising such a task. This project aims to explore the possibilities of applying the field of machine intelligence and distributed systems to design an intelligent agent that analyses traffic to find patterns, such that traffic problems can be alleviated.

1 Problem analysis

This chapter describes different aspects of the problem of traffic congestion. First the relevance and interested parties are discussed following a description of existing work and their limitations within this problem domain. Thereafter the problem will be more precisely defined in the problem formulation. Then follows the method for solving the problem formulation as well as how the solution will be evaluated in the method and solution criteria sections.

1.1 Relevance

Traffic problems have several consequences and affect several different groups of people and organizations. Table 1.1 summarizes the various issues of traffic congestion and whom they affect. Estimates show that the number of cars in the world is increasing year by year[11]. This increase can potentially lead to an increase in traffic congestions, which in turn increases the need for better infrastructure or more efficient traffic flow.

Many of the roads that are congested are heavily used roads. These roads see more wear than less used roads and therefore requires either special construction or more maintenance, which increases the costs for governments on infrastructure. Maintenance of roads also obstructs the road network, since parts of a road can be unavailable which can result in temporarily slower traffic.

Another issue that is a consequence of traffic congestion is the increased CO2-emissions of vehicles frequently accelerating and decelerating in a congested area[1].

Being stuck in traffic can also affect the well-being of drivers as stress levels increase. A study of the relationship between traffic congestion and driver stress observes that there is a correlation between drivers' stressed behaviour and the traffic situation[16, 2].Stress has several negative impacts

"correlation between drivers' stressed behaviour and the traffic situation." Wat?

Table 1.1: Summary of issues involving interested parties

Interested party	Affected by
Drivers	time wasted in traffic stress from congestion CO2-emissions costs of congestion
Governments	maintenance costs emergency vehicle obstruction CO2-emissions costs of congestion

on the health of the stressed individual, which in worst case can cause or influence several medical conditions. Furthermore, severely stressed drivers are more prone to be driving more recklessly to get out of congestion, endangering themselves and others in traffic[10].

Another critical issue is that emergency vehicles needing to travel through congested areas might be obstructed by traffic congestion. This can in the worst case worsen life-threatening situations which is why avoiding congestion is critical.

The different consequences of traffic congestion leads to economic costs. The combined annual costs of congestions in the U.S., the U.K., France, and Germany is estimated to increase to $293.1 billion by 2030[7]. This translates into an annual, average cost per capita of $1740 and 111 hours wasted stuck in traffic. Therefore, both drivers and governments should be very interested in seeing traffic congestion alleviated or eliminated from the infrastructure.

1.2 Existing work

Many different methods for helping drivers better plan their routes have been made. The simplest way of receiving updates on traffic is through the radio in your car. This method is still used to inform people if there has been some incident or if there is heavy traffic on some road. Though this is only to inform people and they usually do not suggest alternative routes to avoid the heavy traffic. The message is reported in by a person and is typically announced once. There are many different tools for planning a route on different devices and there is a lot of research in this field as well, trying to figure out ways to improve the already existing technologies and techniques. The earliest versions of simple GPS-devices, which could aid in finding a route, only considered the shortest path or the fastest path to your destination.

Source?

The recent years there has been focus on trying to predict other things like live traffic patterns and changing conditions on your route.[4]

In the following section relevant existing technologies and scientific articles will be explored so as to gain insight into the existing work of the problem domain.

Dedicated GPS products

A typical GPS, such as a TomTom device works by using satellites. It locates your position and then locates the point you want to navigate to, then it computes a route on a map, which is usually on a SD-card inside the TomTom device. There is no historical data applied to generate the routes.

It also uses a technology called RDS-TMC (Radio Data System-Traffic Message Channel), which is a service that provides live-time traffic updates to your GPS. RDS-TMC works such that if there is an incident on a road, such as a crash, bad weather or queues it transmits data about this to a central information centre, that further transmit the information to a TMC information service provider. The information service provider encodes the traffic information, and transmits it to a FM radio broadcast where the information is then sent out in RDS signals which the GPS-device receives and decodes to a visual representation. The reporting of incidents or congestion is done manually people getting information from external sources. In Denmark the Highway Agency is responsible for manually sending out this information[14].

Web Mapping Services

Web mapping services (WMS) are an alternative to the first GPS-devices. WMS include popular services such as Google Maps, OpenStreetMap and Apple's Maps. Most of these services can be downloaded to mobile devices such as smartphones or tablets and provide directions for the user. Google maps has a feature where the suggested routes will be coloured green, yellow or red indicating respectively clear, slow-moving or heavily congested traffic. Google Maps also creates the routes from historical data and live data which is sent by sensors and smartphones[6]. The historical data includes information about what day it is and what time of the day it is, to be able to try to predict if there can occur traffic jams. The live data which are sent by sensors are placed by the government or private companies who gather traffic information, whereas the live data from the smartphones are from people who are driving on the roads, reporting automatically how fast they are moving on a particular road. The map is not hardcoded on your device, but depending on your route and your whereabouts, segments of a map is downloaded to your device. Google Maps does pick alternative routes for you, if they exist, they usually suggest two other routes and give an estimated driving time for each route. They give the routes and time estimations from a collection of historical data and current live data from smartphones[9].

collect all methods for obtaining data to section below

Intelligent Transportation Systems

Another type of relevant system is the Intelligent Transportation Systems. ITS are defined as:

″ITS – Intelligent Transport Systems – is a generic term for the integrated application of communications, control and information processing technologies to the transportation system. The resultant benefits save lives, time, money, energy and the environment. The term "ITS" is flexible and capable of being interpreted in a broad or narrow way.″[13]

Per definition, ITS covers a wide area of system types, varying from traffic safety and security systems to traffic congestion relief systems. Consequently, ITS considers many different methods for acquisition, processing and analysis of traffic data and as such, is interesting to further investigate.

Data acquisition

ITS must consider some data if they are to be of any real use for the user. Obtaining data is therefore one of the aspects of such systems. We distinguish between two methods of obtaining traffic data: road-based and vehicle-based data collection.

Road-based methods places the responsibility of collection data on equipment installed on the roads, such that passing vehicles has no involvement in the collection. Road-based methods[13] include technologies such as inductive loops buried beneath the roads to sense passing vehicles, and infrared or ultrasonic sensors mounted on different entities along the roads. Inductive loops has the advantage of working well, regardless of the surrounding environment, however they are costly to implement and maintain in existing road networks, since they need to be buried beneath the roads. The sensors has the advantage of being less costly in implementation and maintenance, but may have problems operating under bad environment conditions such as snowy weather[8, 13].

The vehicle-based methods of collecting traffic data includes the vehicles in the responsibility of data collection. This means that vehicles might be equipped with devices that either cooperate with some service or other equipment installed on the roads. Vehicle based methods include cell-tower triangulation of mobile devices, RFID-based identification of vehicles with sensors on the roads (or other vehicles in a peer-2-peer network) and GPS-enabled devices, communicating with a GPS service provider or satellite. Regardless of using a road-based or vehicle-based method, installing and maintaining equipment on roads for an ITS is expensive[8]. Considering the GPS-enabled devices approach is independent of other physical equipment installed in the area or on the roads, and the widespread popularity of GPS-enabled devices such as smart-phones, this approach seems like the most flexible and cost-efficient way of collecting traffic data.

Data processing

When data is being acquired by various sources, an ITS usually performs some processing. When data is collected from different sources, the system performs data fusion to merge different knowledge about single entities. The data is usually stored in a database system, such as the data can be accessed and analysed when needed. During the processing, the ITS must also perform different kinds of data exchange, to accommodate the different schemas of storage and communication e.g. transforming a communications stream from a vehicle into a database record.

Another processing task is to perform map-matching of raw data. Map-matching is the process of mapping location measurements to a map. This is required because the location measurements often enough is not accurate, and therefore must be matched to a map by processing the data. An example is gps-samples that can be inaccurate depending on the satellite connection.

Data analysis

When data is processed and readily available, the system must perform some kind of analysis to discover useful information about transportation networks. Such information could be about car accidents, roadwork, and traffic congestion. The information can then be utilised to act in the problem domain, to either automatically alleviate a problem or help improve decision making of drivers by giving useful suggestions.

Whenever useful information has been derived, an ITS must distribute the information to the users of the system. Many different communication channels exist, such as message signs on the road, direct change in directions on gps-devices and in-vehicle devices.

1.2.1 Other analysis methods

Abnormal events on a road network, can be defined as negative deviation from normal traffic flow on a given road. The normal traffic flow can be determined by collecting and processing historical data such as the speed of vehicles travelling on the road. Kamran and Haas[8] proposes to partition road networks into road segments, such that information relevant for each segment is linked with the respective segment. This accommodates special cases of roads, such as highways, where some traffic data collected from the road is not necessarily relevant for some other part of the highway. An example would be the E45 highway, where the traffic on E45 in Northern Jutland might not be relevant for the traffic on E45 in Southern Jutland. By maintaining the average vehicle speed on road segments for multiple time-intervals, an agent could gain useful insight into how traffic flows at different times and can consequently deduce a set of traffic patterns indicating congestion.

1.2.2 Current limitations

As described in 1.2, the limitations of road-based methods of inductive loops and sensors cause issues regarding the cost of implementation and maintenance, and environmental issues with accuracy of measurements. Vehicle-based methods, on the other hand, have the usefulness of being less expensive and more flexible, now that most people have a GPS-enabled smart-phone or tablet in their vehicles. However, there are still the concern of the privacy of drivers. Tracking GPS coordinates of vehicles enables surveillance of people's movements, which could potentially be exploited if not handled carefully. Therefore measures must be considered to anonymize the GPS location data such that it cannot be used to backtrack individuals' movements.

1.3 Method

To solve the problem of traffic congestion, we process a large dataset of vehicle movements within a city, to derive useful properties of roads such as average speeds and number of vehicles at certain periods in time. We will use this information, to find *traffic patterns* in the data, such that congestion can inferred from the history of the traffic on a given road (e.g. a main road to the university could possibly have congestion from Monday to Friday every morning from 07.00 to 09.00). We will use these patterns in a route planning process to find a potentially faster route than the shortest path, by avoid choosing a roads going through a congested area. Ultimately, the agent should be able to suggest a driver of a vehicle wanting to travel from point a to b, a faster route and users of the agent should be directed away from congested areas, such that the overall congestion of a city is smaller, thus utilising the road network better.

Data source & collection

Because of the limited time we have for this project, we use a GPS data set from Beijing from 10.000 taxies over a period of a week, as the historical data[12]. If we had more time we would have collected our data on our own. A problem regarding this data is the short time span, since it does not cover holidays, or other special days, which are often at fault for massive traffical problems.We are also going to use a large data set from Beijing to simulate real-time data, this data was gathered over a period of 5 years, and include a testbase of 182 people[5]. Both of the data sets was published by Microsoft Research.

consider moving problems with our method into an evalution section later in the report

1.4 Solution criteria

We propose the following criteria that the agent must fulfil to be considered an acceptable solution to the problem:

1. The agent must provide time-wise equivalent or better routes than those generated by Google Maps.
2. The agent must not choose routes that are disproportionately longer than a shortest path to obtain minuscule time-savings.
3. The agent must avoid adding traffic to congested areas, that is, it must avoid planning routes that either go through existing congestions or create a congestion by overloading the capacity of other roads.
4. Realistic usage of the agent must alleviate congestion, that is, a reasonable fraction of motorists using the agent should mitigate congestion.

To evaluate the agent against criteria 1 and 2, we simulate several individuals wanting to travel through the road network. We set up the simulation with a route that is the shortest path between two nodes, such that the route goes through an area that is known by the agent to be congested in a specific time interval. We then calculate a route with Google Maps and with the agent respectively and compare the properties of these routes. If the route calculated by the agent is equally fast or faster than the route calculated by Google Maps, we consider criterion 1 met. If the distance travelled is not disproportionally longer than that of the Google Maps route, criterion 2 is also met.

To check whether the agent meets criteria 3 and 4 we consider how the number of users of the agent affects the severity of congestions. To do this, we simulate travels of motorists on the Beijing road network and estimate the capacity of congested roads. Then we increase the number of users of the agent incrementally by recalculating their routes. By doing so, we can estimate the correlation between the number of users of the agent and congestion. Consequently, we can approximate how many users of the agent there must be to best alleviate congestion problems. If somethingthen criterion 3 is met. If this amount of users does not exceed the number of users that utilize their smart-phones or tablets for directions,we consider criterion 4 met.

Find out exactly how to evaluate this.

Mention that this is what we meant by "realistic usage" in criterion 4

1.5 Problem formulation

So far we have considered different aspects of the abstract problem of detecting traffic patterns. However, before we construct a solution, the problem should be well-defined such that a solution is clear. The problem is defined by the following problem formulation:

Can an intelligent agent model the road network of Beijing to process and analyse raw GPS data, collected from a network of GPS-enabled vehicles, so as to detect traffic patterns indicating abnormal traffic flow such that a map of the traffic state of a road network can be constructed, that can be used to suggest alternative routes for drivers to avoid congestion and consequently save time and in general obtain better utilization of the road network?

Decomposing the problem formulation, we get the following questions:

- How can a road network and traffic be modelled?
- How can data be collected and communicated between GPS-devices and the agent?
- How can the agent analyse traffic data to detect traffic patterns?
- Can such a system deliver time savings for drivers?
- Can such a system deliver better utilisation of road networks?

With the problem formulation defined we can now construct a solution that answers the above questions, consequently answering the problem formulation itself.

2 Problem Solution

This chapter discusses the proposed solution for the problem formulation. We first describe the architectural design of the developed system and how we model road networks. Furthermore we discuss the communication between client and server components. Then, we describe how to process the GPS data and perform data mining to deduct traffic patterns. At last, we discuss how to use the patterns in the intelligent route planning and issues regarding communicating routes and changes in routes from server to client.

2.1 System architecture

The design in Figure 2.1 shows the overview of the system.

The system has been designed as a combination of a client-server and a repository "blackboard" pattern. The main server module encapsulates most of the processing of the system. The clients represent the GPS-enabled devices that communicates live GPS data through an internet connection to the server, which in turn processes and analyses the data in different subsystems. The server architecture is designed as a repository structure, where the solution repository is the shared medium of the subsystems. This architecture is well-known within artificial intelligence, where the solution is gradually built from different contributing systems, which is why we have chosen this structure of the system. Initially, the solution repository contains nothing but the problem specification, which is the set of route requests from the clients. The job of the subsystems is to transform the specification into a solution. This is done gradually by invoking the different subsystems to perform operations on the problem specification until a partial solution can be sent back to the client(s). The control shell module controls which subsystem that should be invoked, based on the contents of the solution repository.

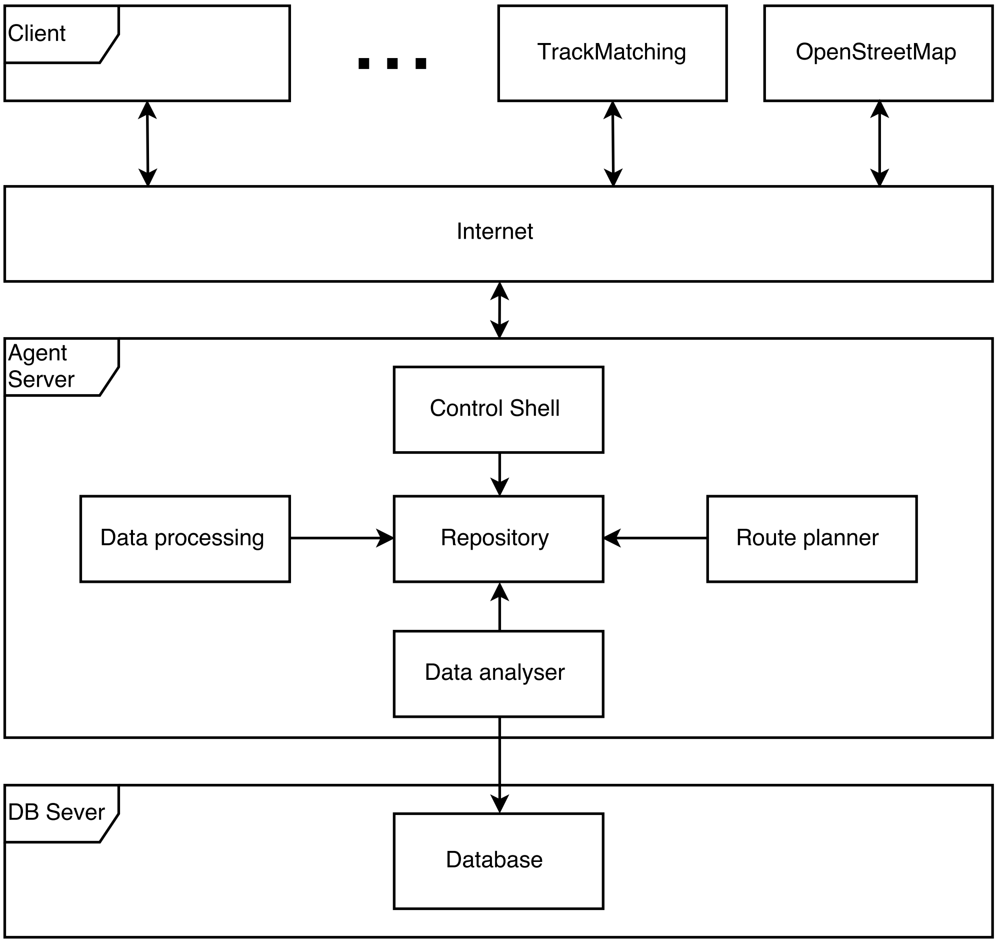


Figure 2.1: A course grained system overview.

2.2 Client-server communication

As mentioned in the former section, we will be using a client-server structure. The reason behind this choice, is the need to be able to send information back and forth. Without a client server solution we would not be able to collect data, nor be able to communicate between different devices. Smartphones used by drivers will be acting as clients, where they need to be able to send information such as location and speed to the server in some given time interval. The server will handle this information and send it further for analyzing, the server will also send updates regarding the route back to the clients, to make sure, if events occur on a route that a client is on, it will be possible to reroute the client.

The client server interface is built upon the Java ServerSocket and the Java Socket, which handles opening a port, accepting a connection, and connecting to such a port, through a TCP connection. We chose TCP because it is a stream oriented protocol, which is ideal for transferring a lot of sequential data from one point to another, which increases the throughput. The other option would have been UDP, which is a message oriented protocol. UDP is used for sending messages from one point to many points, which minimizes the time it takes for a message to be sent and received. UDP does not remember the sequence of what you are sending, but TCP does, which of course in our need is what we prefer.

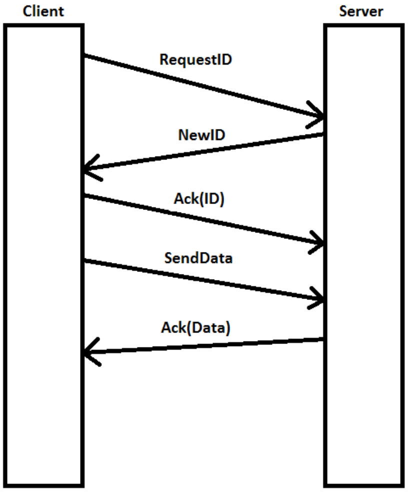


Figure 2.2: Client-server illustration

Figure 2.2 illustrates how our client-server structure works. The client sends a RequestID command to the server. The server responds with an ID, which is an integer that increases every time a RequestID command is received. The reason the server provides the ID is to prevent clashes, which might occur if the clients generated the IDs. The client sends back an acknowledgement to the server when it has received the ID. When the client has received the ID, it then acknowledges the ID received. The client sends the data to the server that is needed to create a new route, or alter a route, or gives information about the route it is following. The server then sends an acknowledgement that it has received the data.

The messages that are sent back and forth are converted into bytes. In Figure 2.3 it is illustrated how the bytes are packed. 32 bits are reserved for our ClientID, which is empty when the first message is sent to request an ID. Next we have our RequestID command which takes 16 bits and for the other commands there is 8 bits. We have reserved one bit as a Last message ID, which is used to be able to tell if a big message, that is split into 2 or more files has ended.

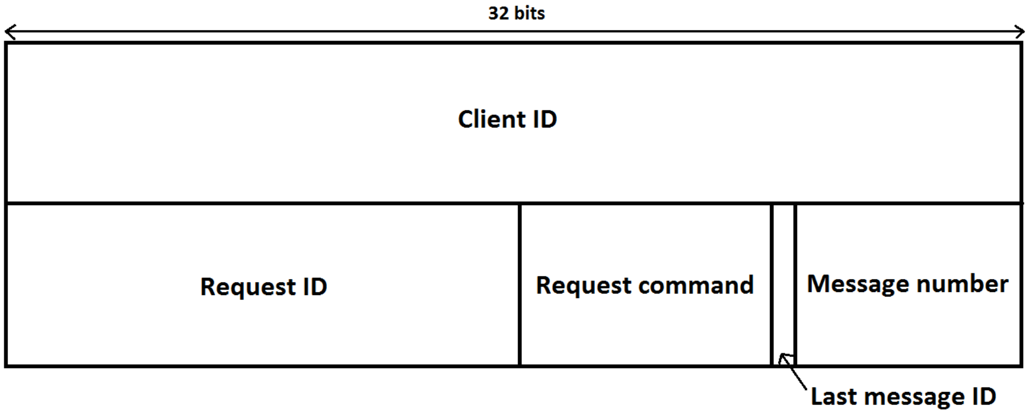


Figure 2.3: Illustration of header composition

det der vi snakkede om med striberne

The client will ask for a GPS update every 5 seconds, this is done to be able to maintain a precise location of the client. This information, plus the speed and time is sent to the server. When a message is sent either from the client or the server, they are both set to wait for 5 seconds to receive the acknowledgement message, if no such message is received the message is discarded. We could have continued to try and resend the message several times, but a lost GPS coordinate from a user is not a critical problem for our system and will not have any major effect on our calculations.

As just explained the client harvests data from the user, this data can be used as historical data and is something that we would have used, but because of our limited time for this project we, as explained earlier, are using a data set from Beijing.

On the client we have implemented OpenStreetMap and a getRoute method where the client asks for a route between two points, this is done at the moment by sending 2 node ids, node ids are explained in 2.3.1, to the server where then the fastest path is calculated between the 2 points. The server then sends GPS locations of the route which then is shown in the OpenStreetMap on the client.

2.3 Data preprocessing

Here we discuss how to process the Beijing dataset. The overall process is illustrated in 2.4.

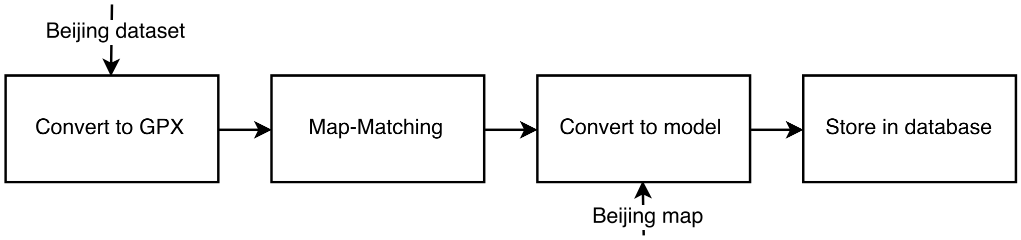


Figure 2.4: From raw gps data to database

The gps dataset is converted into the GPX format which is subsequently map-matched to the road network of Beijing. The map-matched data is then converted into the model objects of the system, and are then stored in the respective tables of the database. Likewise, the map of the Beijing road network is converted to model objects and stored in the database. By doing so, we have a knowledge base of the road network of Beijing and data about vehicle movements in a unified format.

maybe we should perform some data cleaning

2.3.1 Map generation

To model the road network of Beijing we need to have the specific information about all the roads in the road network. This information is available several places, but most of these services are not free. We have chosen to work with OpenStreetMap which is an open source map service, where users collaborate to keep the city maps updated. There are some problems in using this kind of service, such as the possibility of wrong data, limited data, etc. But because this service was the only free service that we could find, we see no other option than having in mind that these problems can occur. We have also found that OpenStreetMap cannot export large areas of a map, such as the whole city of Beijing. Therefore we have chosen to use one of its mirror sites, Geofabrik, which holds extractof the whole world, segmented into countries and these are updated daily. This makes it possible for us to extract the whole map for China.

Er det godt nok bare at skrive etc? + kilde?

Forstår ikke helt det?

Format & Pre-processing

The maps from Geofabrik comes in a compressed format called pbf. To extract the relevant data from this format we are using a tool called Osmosis, to query the file using several filters to be able to extract only the information relevant.

We are using two types of filters with Osmosis, the first is to limit the area that we want to extract, since the map we got is for all of China. The second filter is used to extract only roads from the restricted area. With this filter we have the possibility to specify which roads we want to have in the output. Because all paths are marked in OpenStreetMap as a way, we want to filter paths in order to remove side-walks, bicycle ways, and other ways that are not possible to use with a car.

The extracted data from the Osmosis query is then exported to a file which is structured in a way that resembles the XML format. There are two types left in the export, which are nodes and ways. Nodes represent road intersections and are also used to indicate points along a road if the road is curved. Both nodes and ways can include key-value pairs that contain information, i.e. if a way is one-way, if a intersection have a traffic light, ect. Ways are almost similar structured, but they also contain a nd tag which contains a value the id of a node related to the way.

igen, er etc godt nok at skrive?

Forstår ikke den sidste sætning her

```
<node id, version, timestamp, uid, user, changeset, longitude, latitude >
        <tag key, value />
</node>
```

Listing 2.1: Node representation

```
<Way id, version, timestamp, uid, user, changeset >
        <nd ref />
        <tag key, value />
</way>
```

Listing 2.2: Way representation

We have illustrated the representation of nodes, ways, edges and segments in Figure 2.5. The red nodes represent intersections, the green nodes are points along a way, edges are the black lines between every node, the way is the entire black line, and segments are represented by the blue lines between intersection nodes.

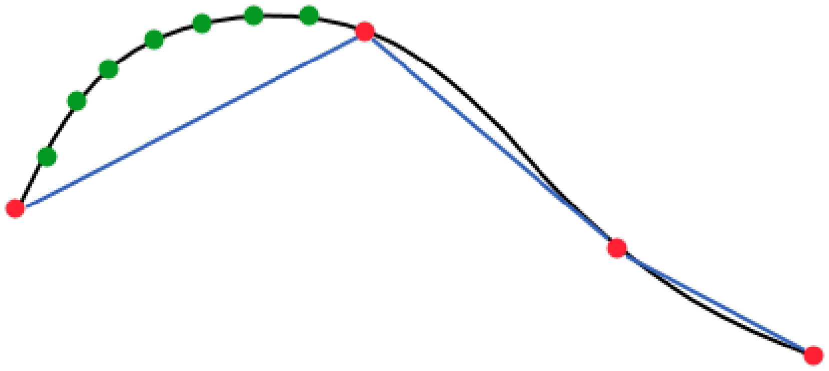


Figure 2.5: A way with nodes

Generation

An XML parser is used to extract the information from the Osmosis export that is needed to model the road network. When a node or way tag is read a equivalent data structure is created and stored.

```
Node{
   long id;
   double longitude;
   double latitude;
   Way[] connectedWays;
}
```

Listing 2.3: Datastructure for a node

```
Way{
   long id;
   Node[] connectedNodes;
   int type;
}
```

Listing 2.4: Datastructure for a way

We want to model the road network as a graph, but the relation between ways and nodes are not in this format when exported from Osmosis, so we have to work on the data structures generated from the parser. Besides the original way and node data structures, we have chosen to create two other classes such that we are able to construct a graph, these are called edges and segments.

Edges are the links between every two nodes and a segment is the link between to intersection nodes. When generating these classes from ways and nodes, we have to check whether a way has a tag attached to it which describes if the way is one-way or not, because if it is we have to create edges and segments in both directions.

```
Edge{
   long id;
   double length;
   long origin;
   long destination;
   long segment;
}
```

Listing 2.5: Datastructure for an edge

```
Segment{
   long id;
   double length;
   long way;
   long origin;
   long destination;
}
```

Listing 2.6: Datastructure for a segment

The relation between edges and segments is that each edge of a segment contains a reference to the segment that it is a part of, the same is true from the relation between ways and segments.

2.3.2 Data filtering

insert coordinates

We perform a simple spatial and temporal filtering the GPS data, by filtering GPS samples from outside the physical city limits. The limits are given by a bounding box defined by the coordinates . Cleaning the data before further processing is key, since not all of the data is accurate or even useful for reasoning about the traffic. We Even though of the data that is filtered is reliable enough, we delimit our focus on congestion in the city center, so we chose to look aside this data. Since most of the data is clustered in the city center anyway, we do not lose significant parts of the dataset by filtering these samples.

Furthermore, some GPS samples are of very bad quality. Some routes are (according to the GPS samples) at one moment driving through Be-

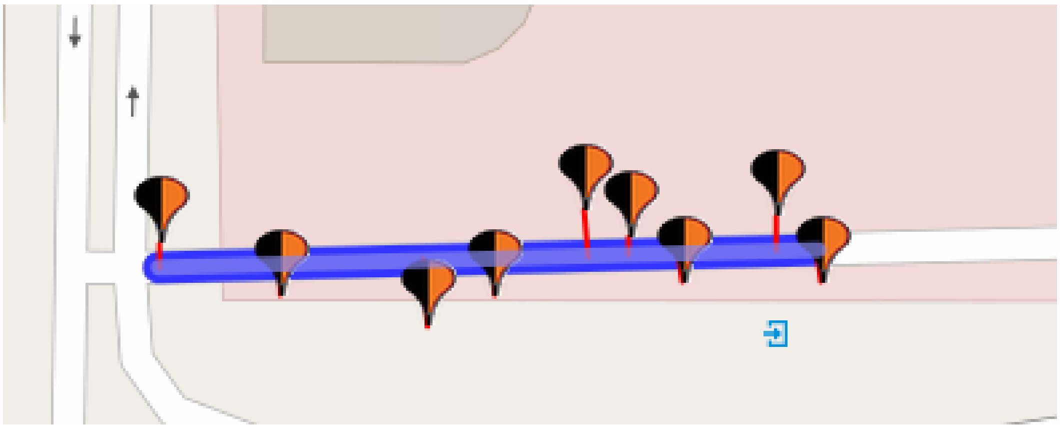


Figure 2.6: GPS samples and the actual road driven on.

jing, and the next moment suddenly driving in Africa or Europe. This phenomenon could be caused by weak connection to the GPS satelites e.g. when driving through are tunnel. The inacuraries these corrupt samples introduce, are taken care of as well by filtering samples outside of the city limits. Fortunately, we do not lose all of the samples associated with the rest of the route involved - just the corrupted sample.

GPS samples that is shown to have completely unrealistic properties is also cleaned from the dataset. An example would be the speed driven between two samples relative to the change in distance between the sampling timestamps. If the speed required to move the physical distance in that time is unrealistic (say, well above what a car can actually drive), we filter these "bad" observations. Note that we actually postpone the filtering of these samples, as it is easier to take into account after map-matching of the GPS samples has been completed.

2.3.3 Map matching

Generally, GPS coordinates measured are often imprecise due to external factors, and do therefore not reflect the precise location of, in this case, a vehicle. Consequently, the samples of the Beijing dataset must be treated as approximations of GPS locations. This causes some problems in terms of how we should decide which road the vehicle was traveling on and what its speed was.

The problem is illustrated in figure 2.6. The orange-and-black colored GPS samples are imprecise in the sense that they are not mapped directly onto the blue road. The straightforward solution would just be to map the samples to the road closest to the collection of samples. However if the road network is dense, we can have multiple candidates for roads, since the distance from the samples to the two roads might be equal.

Consequently, we must perform the process of *map-matching* the dataset of GPS samples to the most likely roads that vehicles was traveling on. Several algorithms and implementations exist that perform this task, including TrackMatching, GraphHopper and N-Gemma.Limited informal evaluations of these implementations showed that TrackMatching provided a more precise map-matching than GraphHopper, both of which work with GPX-formatted GPS samples. Unfortunately, the documentation for N-Gemma was incomplete, and as such, we decided to not investigate further the use of N-Gemma, since TrackMatching was an adequate solution to the map-matching problem, despite it being a webservice.

Consider adding a source for each tool.

Alternatively, we considered implementing a map-matching algorithm ourselves, however given that the focus of the project is on constructing an intelligent routing agent; not an efficient map-matching algorithm, we decided that our time would be better spent elsewhere, and opted for utilising an existing implementation for map-matching.

TrackMatching input and output

We chose to use the GPX format for input and GeoJSON as the output of the map-matcher.The output is represented as a tree structure in Figure 2.7. The *links* element of each *route* is an array of link objects.

why?

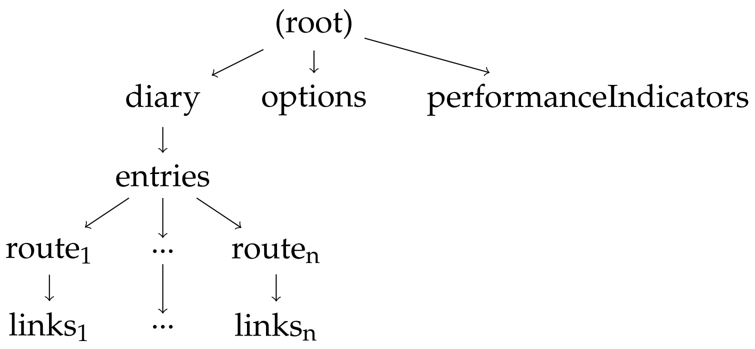


Figure 2.7: Structure of GeoJSON output.

A simple example of a link object can be seen in Listing 2.7. The **err** (error) attribute describes how accurately the waypoints were matched to a way; the lower the error, the more likely that the correct way was picked.

The **src** (source) and **dst** (destination) attributes correspond to nodes in the map on the same way, which is identified through the **id** attribute (i.e., nodes #src and #dst are on way #id). The source node in the next link is the destination node in the previous link, except for the first link. The coordinates in the **geometry** attribute contains the longitudes and latitudes for each node that is passed through in the link. The **wpts** (waypoints) attribute contains the coordinates, id (within the link) and timestamp for each waypoint matched to the link.

mangler noget her: hvorfor er det interessant? HVad bliver de forskellige dele brugt til?

```
{
  ’’err’’:398.43,
  ’’src’’:2639384604,
  ’’dst’’:2639384636,
  ’’id’’:258604821,
  ’’geometry’’:’’{\’’type\’’: \’’LineString\’’, \’’coordinates\’’: [[1.2997645E7,
       4836996],[1.2997724E7, 4837050],[1.2997795E7,
       4837084.5],[1.2997899E7, 4837123]]}’’,
  ’’wpts’’:[{
    ’’x’’:1.2997645E7,
    ’’y’’:4836996,
    ’’id’’:’’0’’,
    ’’timestamp’’:’’2008-02-02T15:10:20Z’’
  },
  {
    ’’x’’:1.2998036E7,
    ’’y’’:4837174,
    ’’id’’:’’1’’,
    ’’timestamp’’:’’2008-02-02T15:10:25Z’’
  }]
}
```

Listing 2.7: Content of a simple link object.

2.4 Knowledge representation

In this section we describe what we are going to consider knowledge in the system and how we are going to represent that knowledge. More specifically, we consider how we can model knowledge about the road network and traffic in Beijing. We consider the knowledge representation for several reasons. First of all, the raw GPS samples and map data is not really useful to reason about, as there are a lot of unnecessary details included. Thus, we want to make sure that the model adequately portrays the real roads and traffic of Beijing only with the required level of detail. Secondly, we want to be able to reason about this knowledge to later be able to find patterns in the traffic.

2.4.1 Road network representation

To represent the road network, we must model what the road network consists of. A road network generally consists of roads and intersections of roads. Obviously there are many different types of roads, such as bridges, boulevards and one-way roads, which all have different properties related to how you can travel on them. Instead of differentiating between types of roads, we can see them as a single type of road, having attributes with different values. As such, to represent the road network we use a weighted, directed graph.

Let G be a weighted directed graph $G(V, E)$, where V is the set of nodes corresponding to intersections, E is the set edges corresponding to road-segments between nodes. Each edge represents the direction of a road from the origin intersection, n_o to the target intersection n_t. That is:

$$e \in E \text{ is a tuple } e = (u, v) \text{ where } u, v \in N$$

Furthermore, two-way roads are represented as two edges,

$$e_1, e_2 \text{ where } e_1 = (u, v) \text{ and } e_2 = (v, u)$$

Each edge, e, also has an associated weight, $weight(e)$ where

$$weight : E \times T \times W \rightarrow \mathbb{R}_+ \tag{2.1}$$

is the weight function as defined more precisely in Section 2.6.3.

Usually, the weight of an edge is used to choose between edges in search algorithms. Similarly, we construct our weight function for this purpose, as described in **??**

Skal ref til weight function

Representing the road network as a graph is sufficient to capture the important attributes of roads and intersections, since the these attributes can be taken account for by adjusting the weight function to represent differences between different types of roads.

2.4.2 Traffic representation

We must also consider how to represent knowledge about previous traffic in the Road Network. Since traffic changes over time, we consider this a separate model than the model for the road network, as the traffic describes dynamic properties of the road network, whereas model of the road network represents the static properties.

From the Beijing dataset, we can find many useful properties about the traffic. More specifically, we wish to capture the notion of a person observing the state of traffic when driving on a road at a specific time. An example would be, that a driver could observe that the traffic flows at a lower speed than the speed limit on the road to work in the morning. By collecting many of such *observations* of drivers at different times, we build a large set of observations that can be used to reason about how the traffic situation is at certain time periods. To capture these properties, we define a set of *observations* on road segments.

Let an *observation*, o, be:

$$o = (t, d, w, s, e)$$

where

$$\begin{aligned} t &\in T \text{ is the time of day} \\ d &\in D \text{ is a date} \\ w &\in W \text{ is the day of the week} \\ s &\in \mathbb{R} \text{ is the speed and} \\ e &\in E \text{ is road segment where the observation was made.} \end{aligned}$$

From the map-matched dataset, we obtain the set of all observations on the segments in the road network, by constructing a new observation every time a vehicle is driving on the segment. Fortunately, the time of day, segment identifier, date and week of the day can directly be found in the data. The remaining problem is then to estimate the observed speed. Observations are generated on a link-to-link basis.

We have encountered a number of problems when generating observations:

1. Speeds cannot be accurately calculated when there are fewer than two waypoints on a link.

2. Sometimes, a few consecutive waypoints are identical. This is only a problem when all the waypoints are identical, as that is effectively equivalent to having only a single waypoint.

3. The map used by TrackMatching may differ from the one used locally. In one case, a way in the TrackMatching map was represented as two separate ways in the map we used. In another, a node in the map used by TrackMatching did not exist in ours.

4. Some of the drivers in the Beijing dataset drove in the wrong direction on a one-way road. This is either due to an error in the map (i.e. wrong classification of the road) or reckless driving.

5. Due to the inaccuracies in GPS technology, someone staying in the same place will appear to move back and forth on a way. The same will happen in a few cases when the map-matcher includes waypoints that should have been pruned.

These problems are solved by skipping links where they occur. This increases the average quality of observations in exchange for a slight decrease in the quantity.

2.4.3 Estimating observation speed

Nyligt omskrevet. Læs og ret.

mangler hvorfor vi har valgt at gøre det sådan her (done?)

The speed of an observation is estimated based on the data in a single link. The average speed between each pair of waypoints in the link is calculated by

$$\frac{od(wp_1, wp_2)}{t_2 - t_1}$$

where $od(a, b)$ is the orthodromic distance (i.e., the distance along the surface of a sphere) between two points; wp_1 and wp_2 are waypoints, and t_1 and t_2 are their respective time in hours. We use the median of the speeds between each pair of waypoints as the speed of the observation, since it is more resistant to outliers than the mean.

We generate observations for each link that is not skipped. Given a link that contains a set of road segment E, a sequence of coordinates $Coords = (c_1, ..., c_i, ..., c_n)$, and a sequence of GPS waypoints $WP = (wp_1, ..., wp_i, ..., wp_n)$ projected onto a road, where $c_i = (long_i, lat_i)$ and $wp_i = (d_i, w_i, t_i, x_i, y_i)$, we construct a new observation o_e for each $e \in E$ such that:

$$o_e = (t_1, d_1, w_1, S, e)$$

where

$$S = median(\{s_1, ..., s_i, ..., s_{n-1}\})$$

The speeds in S are calculated using:

$$s_i = \frac{dist(wp_i, wp_{i+1})}{t_{i+1} - t_i} \qquad \text{for } 0 < i < n$$

where

$$dist(wp_1, wp_2) = \begin{cases} od(wp_1, wp_2) & \text{if } |cbw| = 0 \\ od(wp_1, c_1) + od(c_1, c_2) + ... & \text{if } |cbw| > 0 \\ +od(c_{m-1}, c_m) + od(c_m, wp_2) & \end{cases}$$

and cbw is the sequence of all coordinates $(c_1, ..., c_m)$ in Coords between wp_1 and wp_2 on the way.

2.5 Learning traffic patterns

In order to reason about the traffic in Beijing, and provide some measure of how the previous traffic possibly affects our belief in how the future traffic situation will be in the road network, we analyse the preprocessed data to look for patterns. In this section, we consider methods for extracting such patterns and propose how to apply them on the Bejing dataset.

2.5.1 Methods

From our knowledge representation of the road network and traffic, we want to predict how traffic affects the travel time of roads in the road network. In order to do this, we need a measure of the traffic on roads. There are several ways one could construct such a measure, incuding

- *capacity*: How many vehicles can travel on a road, safely, while not negatively affecting the average speed driven on the road?
- *classification*: Define a class feature of a road, that defines the state of the road. An example could be $Traffic \in \{free, light, medium, heavy\}$. A given road then has a classification based on the observations of speed within some time period.
- *speed*: we measure the traffic on a given road, by how fast you can drive on the road. In addition, one could measure the ratio between speed and speed limit.

The capacity of a road would be possible to estimate from the speed limit, length of a road, safety distances and number of cars. However, drivers do not always drive in a uniform manner, which makes it difficult to accurately estimate capacities.

Performing some discrete classification could help to distinguish between different traffic states. Intuitively, a person might be interested in knowing if a road is clear, mildy congested or heavyily congested. Such information could be captured by classifying roads by a roatio between speed limit and actual speed. This raises the questions of which classes should be used and how many. If there are not enough classes, it might be difficult to choose which road to take if both are congested more or less equally.

To more easily be able to evaluate roads where traffic is similar, we must use a measure that has a finer granularity. By looking directly at the actual speed driven on roads, we can determine more precisely the costs of taken a particular road. The actual speed, together with the length of a road is also the most important factors in in evaluating the travel time for a road. Therefore, we take a regression approach for predicting traffic.

2.5.2 Regression model

To predict the traffic speed on a given road segment, at a given time we can consider a speed prediction a function:

$$speed : E \times T \times W \rightarrow \mathbb{R} \tag{2.2}$$

that maps a road segment, time of day and day of week to the a real valued speed prediction. The remaining question is then, how to actually perform the prediction of traffic on a segment given a specific time.

Predicting the numeric value of some variable based on previous observations is typically associated with regression. Regression fits input and output pairs to a function, such that future values for input pairs can be predicted. Fitting a regression function on the observations for a particular road segment, provides a way to predict the traffic on the segment. However, since traffic varies over time, one could argue that a polynomial regression function would fit the observations better than a linear function as illustrated in figure **??**. Even though the higher order regression functions generally fits the training data better, such a fit might cause *overfitting* observations, where the accuracy of future predictions decreases and the a linear fit might fit the future observations better.

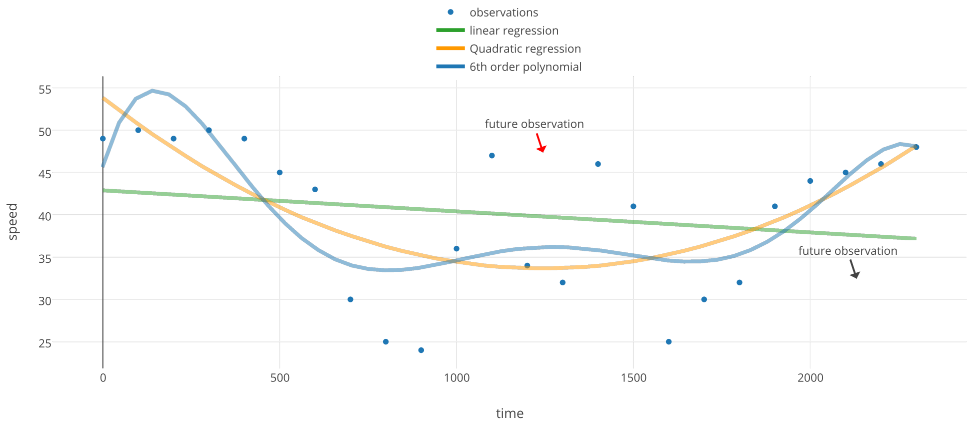


Figure 2.8: Different regression fits on the same dataset.

However, a linear regression over the observations for a whole day is not as useful for predicting the fluctuations in traffic during the day, because linear functions by definition cannot model large fluctuations. For this reason, we instead utilise several linear regression models over the course of a day. More specifically, we split a day into partitions $p_1, ..., p_i, ..., p_n$ where $p_i = (t_1, ..., t_j, ..., t_k)$ and t_1, t_2 are time-stamps. Thus, a partition is defined as a period of time between t_1 and t_2 that contains every possible timestamp t_j from t_1 to t_k. For every p_i, we perform a linear regression fit, f_i, over the observations within that period of time where $f_i : E \rightarrow \mathbb{R}$. In practice, the overall regression function, *speed*, for a road segment becomes a piecewise linear function:

$$speed(e, t, w) = \begin{cases} f_1(e) & \text{if } t \in p_1 \\ f_2(e) & \text{if } t \in p_2 \\ & \vdots \\ f_n(e) & \text{if } t \in p_n \end{cases} \tag{2.3}$$

where $f_1, ..., f_n$ are linear regression functions for each partition over the observations for segment e. The idea is illustrated in Figure 2.9.

2.5.3 Partitioning scheme

In order to use the piecewise linear regression approach, we must devise a partitioning scheme. One such scheme could be the one in Figure where

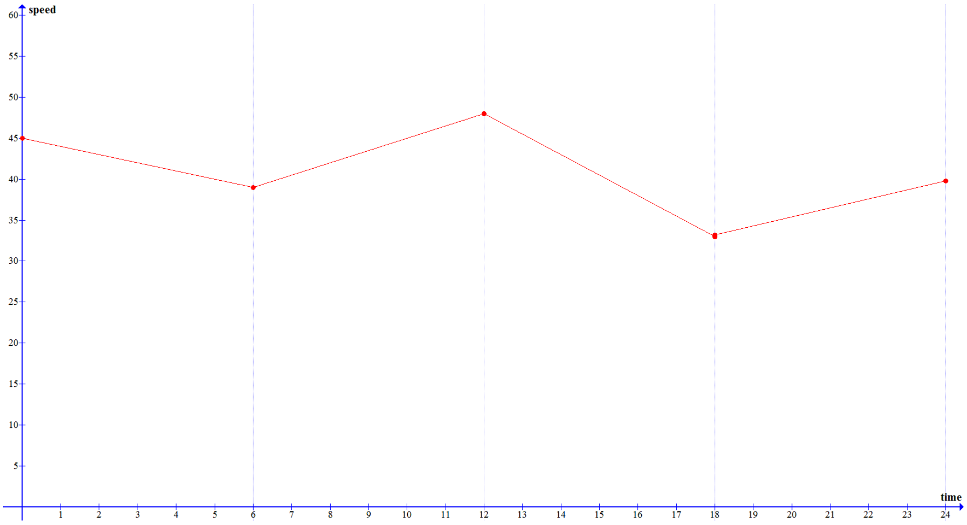


Figure 2.9: Example of piecewise linear regression - day of 24 hours are partitioned into 4 parts. Every partition has a linear function describing traffic in this time period.

every day a split into 4 equal partitions: night, morning, afternoon, evening. The following problems are raised by the choice of partitioning scheme:

- Where should we split the day, such that important changes in traffic are captured?
- How many observations is required to perform regression?
- How do we guarantee enough observations in every partition?

While it makes sense to have equally sized partitions, it does not guarantee that there will be enough observations to perform a accurate regression for every partition (an example would be, that there are usually less vehicles driving at night, thus less observations in this partition). Furthermore, since traffic patterns most likely are different for different road segments, a static partitioning is not guaranteed to capture important patterns in traffic, such as morning traffic. Therefore, we must incorporate a finer granularity in the partitioning scheme, possibly different for every segment, to capture the differences traffic patterns and to ensure that there will be enough observations to actually perform regression.

This leads us to a dynamic partitioning scheme, where the partitioning is based on the density of observations and how well the function fits the data. A measure to use for such a partitioning, could be the Sum of Squared Errors, which is a measure of how close a predicted value is to the real value. Partitioning is then done, based on minimizing the Sum of Squared Errors, as this yields more accurate regression models.

2.5.4 Model trees

To implement the piecewise regression model described in Section 2.5.2, we must either construct an algorithm or apply an existing one to the data. One could develop a partitioning scheme and then perform a linear regression on each partition. However, such an approach could potentially yield regression models that does not take into account the growth of predecessor and successor partitions' models. Furthermore, it is not clear how an effective partitioning scheme should be devised, to facilitate dynamic partitioning.

Fortunately, existing work related to the piecewise linear regression model approach exists, in so called *model trees* and *regression trees*. As illustrated in Figure 2.10, a *model tree* is a decision tree that has regression models that can either be functions or constants at the leaves. Hence, based on the value of the feature being decided on the model tree has a regression model for any possible world over the set of features. A regression tree is a specialization of a model tree, where the regression models at the leaves are constants.

Several algorithms exist for inducing a piecewise linear regression including *CART*, *M5*, *M5′* - an improved version of M5 - and *MARS* . Wang and Witten[17] argues, that M5′ performs somewhat better than M5 and produces model trees that are more comprehensible due to the size of the trees. Furthermore, open-source implementations of M5′ are available for use, which provide a useful basis for inducing model trees for the traffic data.

need refs for these algorithms

For implementing M5′ model trees and data filters we use the machine learning tools and algorithm implementations in Weka[15]. Weka contains an implementation of the M5′ model tree induction algorithm (called M5P in Weka). Weka can be invoked directly from Java code, and therefore integrates well with the rest of the system.

M5′ model tree induction

M5′ constructs a model tree, like the one illustrated in Figure 2.10, from a set of training examples, *T*.

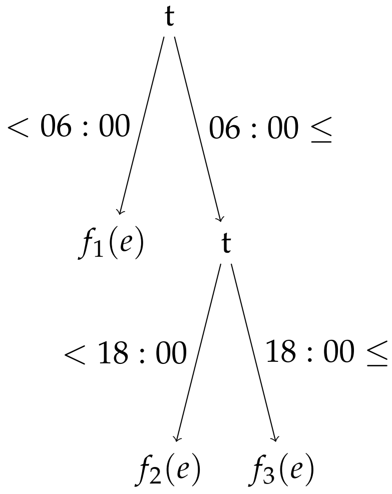


Figure 2.10: Example of a simple model tree with three partitions and three linear regression models. The *t* feature corresponds to the input time feature in equation 2.3

It works by recursively constructing a decision tree by splitting T into subsets T_1, T_2 where the split point is determined by examining all possible splits over all non-class features and the domains of those features. The split that yields the highest reduction in standard deviation is chosen at each step. The resulting tree is then pruned for nodes and leaves that yield a worse standard deviation reduction than parent nodes. This pruning process makes sure the final model is not overfitting the training examples and can better be applied to unseen data. Finally, the resulting regression models at the leaves of the tree are mutually smoothed as to minimizes sharp transitions from model to model.

Using this approach, M5′ facilitates a dynamic partitioning scheme, based on the measure of standard deviation and smooths transitions between partitions.

2.5.5 Generating speed functions

Using the M5′ implementation in Weka, we generate model trees for road segments on a day-by-day basis as described in Section 2.5. However several issues are raised about the data used for inducing the model trees:

- Noisy data can skewer the regression models to be inaccurate.
- Incorrectly mapmatched observations can skewer regression models.

- Observed speeds above the speed limit introduces the possibility that the system can suggest driving faster than allowed.
- A segment might not have enough observations to construct a (useful) regression model.

Filtering the data

To minimize the effect of these issues, we perform filtering on the observations based on the *Median Sample Rate* and mapmatching error. Furthermore observations speeds higher than two times the speed limit are filtered, because such high speeds are more likely to be noise data. Observations with speeds between 100% to 200% the speed limit are lowered to the max speed, since such observations might be realistic, but we still do not want them to affect the regression model.

Additional thresholds for filters can be seen in Table 2.1 (the green row is the one we use).

Speed	Length	MSR	MME	#Wpts	Observations
-	-	-	-	-	4584306
>1 km/h <speed limit * 2	>5 m	<= 20	<100	>= 5	269527
>1 km/h <speed limit * 2	>5 m	<= 20	<100	-	867128
>1 km/h <speed limit * 2	-	<= 20	<100	-	878426
<speed limit * 2	-	<= 20	<100	-	898774
<= speed limit	-	<= 20	<100	-	861766
<= speed limit	-	<300	<100	-	2861949
>1 km/h <speed limit * 2	>5m	<= 60	<100	-	1715070
>1 km/h <speed limit * 2	>5m	<= 60	<100	>= 3	913779

Table 2.1: Observations remaining after using various filters. MSR = median sample rate of the link, #Wpts = number of waypoints in the link.

MSR is the median sample rate calculated using only waypoints that are at least 1.4 metres apart to avoid having , *#Wpts* is the total number of waypoints on the link, *Length* is the distance driven between the first and

make percent of how many observations (in table) are filtered. Benjamin the last waypoint in the link, and *Observations* is the number of observations remaining after applying the filters. The tresholds are based on informal study of how high requirements of MSR and MME (mapmatching error) was possible, without discarded too much data. Additionally, we perform normalisation of the timestamps of observations, due to Weka3's internal representation of time being problematic when doing regression.

Distribution of data

With the filtered observations we are creating speed functions for every day of the week for all the segments. But since the remaining dataset of observations covers just a little percentage of the segments on all days, there will be some segements that wont have enough observations to create a solid foundation to run the M5P algorithm. Because of this shortage of observations we have decided to use two ways of adding observations to a given segment. The first is to look at the segments neighbours, that have the same roadtype on the same day, the reason that we are looking to neighbouring segmens are that they often reflect the traffical situation of the given segment. If neighbouring segments do not have additional information to add to a segment, then we are looking at the observations for the segment on other days. Here we are differentiating between weekdays, and weekends, which we have classified as weekdays beeing monday, tuesday, wednesday and thursday, weekends are therefore the rest. The reason for this distinction is that we argue that some traffical patterns can be devided into these groups, such as weekday traffic when going to and from work.

Besides looking for more observations we are also figuring out which of these observations to add to ensure that most of the day is covered by observations. The way that we check if there are enough observations, are by splitting the observations into intervals of 6 hours, as seen in Figure 2.11, for each of these intevals we check that there are at least k observations. We have not yet found the best k value, but this is one area that are possible to look at in the future work. We are also looking at the 25, 50 and 75 percentile if these values are laying in between a treshold of 6, 12 and 18. If this is not the case additional observations are added to move these points towards the values. The reasoning for using this way to ensure that the data is uniformly distributed, have shown not to be the best idea since we will accept a segment to be proccessed by M5P if it just have k observations en each interval, and the 25, 50 and 75 percentile is to some point satisfied. This does not hinder that observations can just clutter at some points, and

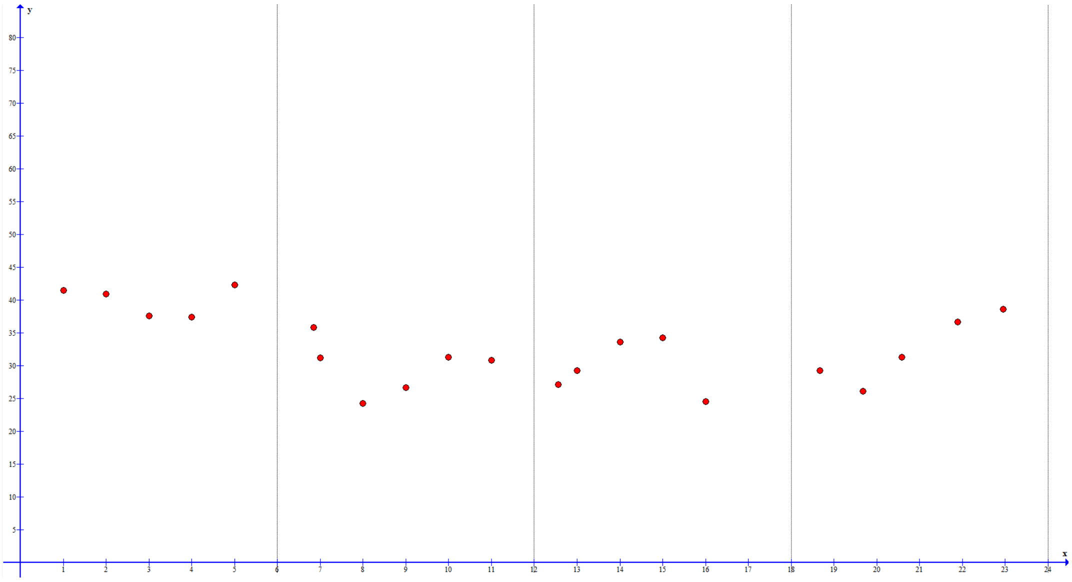


Figure 2.11: Example of intervals with 5 observations in each interval

therefore they are not uniformly distributed.

2.6 Route planning

In this section the process of finding a route will be describe. The aim of this section is to give a clear overview of what elements the route planning consist of. This process can be divided into sub process that are each responsible for an area of work.

2.6.1 Path finding / A*

An A* is used to finding a path through the graph. There are other versions of a A* search that are faster then a normal A*, but these do not work with the weight function that are being used. The time complexity of this search is $O(b^n)$ where b is the branching factor and n is the number of nodes in the graph.

The function is given a start and a goal node, the A* utilise the physical destined between these point as a heuristic to make sure the path it takes goes in the right direction. Then it finds the time and ask the database for the cost of the segments it can travel from the node it's at. The time will charge as the A* is adding segments to the path, this is don to make sure the the cost of travailing a segment in the graph match up with when the driver will be driving on the road.

2.6.2 A*

In the coming section the math and algorithms used will be look at. Why they where chosen over other algorithms of there kinde. Bi-directional graph search is in it's simplest form just a two normal Dijkstra's where one starts from the start node and the other starts from the goal node. The reason for using this algorithm over other graph search is that the the time for finding a path is $O(b^{(n/2)})$ where a normal Dijkstra's takes $O(b^n)$, where b is the branching factor and n is the number of nodes in the direct graph. Bi-directional also have the benefit of working well when running in paralleled.

By using bi-directional search there is one problem that needs to be address, this is will the search meet op before the whole graph is search. There are more then one way of solving this problem, in this project A* have been chosen for solving this problem. The A* makes sure that the search from the start node wakes a path towards the end node and the other way around for the search that starts from end node.

By implementing the bi-directional search this way, the worst case running time becomes $O(b^n)$. The reason for this is that each search is going towards the other search's start node and not it's frontier. The search will stop if one of the two searches hits a node that the other search have visited.

2.6.3 Weight function

As mentioned in Section **??**, we are interested in calculating how fast one can travel along the road segment (i.e. the travel time), given a road a road segment, time and day of the week. With the regression model for predicting speed on a road segment from time and day of the week, we can compute a prediction for the travel time. Let

$$weight : E \times T \times W \to \mathbb{R}_+ \tag{2.4}$$

be the *weight function* that maps an edge e, timestamp t, and day of the week w, to a positive, real-valued travel time.

The weight function works, by calculating the *speed* function described in Equation 2.3 and dividing the predicted speed by the length of the given road segment. That is, given $e \in E$, $t \in T$, $w \in W$:

$$weight(e, t, w) = \frac{speed(e, t, w)}{len(e)} \tag{2.5}$$

where:

$$len : E \to \mathbb{R}_+ \tag{2.6}$$

is a length function, that maps edges to their length.

3 Test and Evaluation

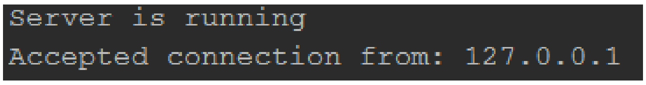
3.1 Test of client-server communication

To be certain that our client server communication works as it should, we will do both "real-life" practical tests and black-box testing, e.g. making sure the client and server behaves as it should on different occasions.

3.1.1 Connection between client and server

To make sure that the client(s) can connect to our server we have tested it in the following ways.

- From local host, we started the server from the IntelliJ IDE and afterwards we started the client from Android Studio and connected to the server. In IntelliJ we could see that the server accepted the connection. Which is shown in Figure 3.1.



```
Server is running
Accepted connection from: 127.0.0.1
```

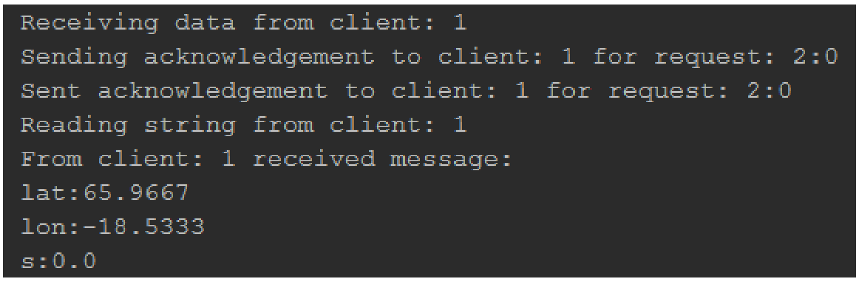
Figure 3.1: Server running and accepting connection

- From seperate clients, we started the server as with the local host testing, but to test the targetted device, e.g smartphones, we downloaded the application to 7 smartphones and connected them to the server. It showed in IntelliJ's prompt window that the connections from each smartphone were successful.

3.1.2 Message received matches the message sent

To be certain the longitude, latitude and speed that is sent is received correctly on the server. We did the following test:

- We gave the emulator in Android Studio a specific longitude and latitude and then used the application to send the data to the server. In IntelliJ's prompt window we got the following information displayed in Figure 3.2. Which confirms that the message sent from the client is received correctly on the server.



```
Receiving data from client: 1
Sending acknowledgement to client: 1 for request: 2:0
Sent acknowledgement to client: 1 for request: 2:0
Reading string from client: 1
From client: 1 received message:
lat:65.9667
lon:-18.5333
s:0.0
```

Figure 3.2: Data received on the server from the client

Use mortens phone to get a proper speed test instead of the 0.0 from the emulator

-

3.1.3 Is the three-way handshake condition met

To be sure that if some message is lost or connection between the client and server is not stable, which we also described in section 2.2, we are sending acknowledgements between client server. As Figure 3.2 illustrates when the server receives data from the client the server responds with an acknowledgement message to the client, that it has received the data, the client then again responds with it's own acknowledgement message to tell the server, that it knows that the data it sent has been received.

3.1.4 If server is down

The user of the application has to be notified if they cant receive or send their data to the server. This is handled by giving a pop-up message to the user of the application that there is no connection to the server.

3.1.5 GPS location and speed

The location of the client and the speed it is moving at has been tested in a car with one person driving the car and another using a smartphone, where the smartphone displays the current speed and the location after a click on a button. The speed was compared to the car's speedometer which matched and the GPS location that was shown was later checked on a map, where the locations given by the smartphone also were correct.

Add more tests to client-server! Dont know if all of these "tests" are needed

3.2 Evaluation process

In order to evaluate the properties of the routing and regression models compared to the problem formulation, we carry out a simulation of routing. More specifically, we want to evaluate:

- How close does the regression models reflect reality?
- Does the routes found based on the weight function reflect reality?
- Will routes found based on the weight function be faster than original route?

To investigate the above questions, we take the following approach: First, we consider how to

1. First, we select a route $R_{uninformed} = (n_1, ...n_m)$ from the GPS data that originates in n_1 and ends in n_m, through a congested area. We then, directly from the data, determine how long it took to drive this particular route.
2. Secondly, we compute a new weight, $weight(s, t, w)$ for every segment $s = (n_i, n_{i+1}) for 1 \leq i \leq m$ with the weight function described in Section 2.6.3. Let this route with the adjusted costs be $R_{adjusted}$.
3. Here, we compare $R_{uninformed}$ and $R_{adjusted}$ to see if there are any significant changes in the total cost of this route. »»»> 9ee4f321aba7509999979acd83b2f25af4ff03
4. Now, we traverse the graph to find a new route, $R_{informed}$ originating in n_1 and ends in n_m such that the route is the shortest path from n_1 to n_m by using the cost of each segment determined by our cost function.
5. We compare $R_{uninformed}$ and $R_{adjusted}$ to see if there are any significant changes in the total cost of this route, and what the difference in the predicted and actual time is.
6. Finally, we compare $R_{informed}$ with $R_{adjusted}$ and see if $R_{informed}$ has chosen a route, that saves time in going from n_1 to n_m.

Route	$T_{uninformed}$	$T_{informed}$	T_{adj}	**Difference**
r_1	1234	1234	123	3
r_2	1234	1234	13123	22
r_3	1234	12333	1212	3
r_4	1234	123	22	44
r_5	1234	123	13	44

Table 3.1: Results of the simulation

We follow the above method for to see any common patterns. The findings are summarized in Table 3.1 Lorem ipsum v Lorem ipsum

x number of route

4 Future work

In this section we will cover possibly area of (spændene) for this project. This is don to highlight topics discover during the project where there wasn't time to implement.

4.1 Contraction hierarchies

The map is represented as a graph as this makes it a lot easier to work with path finding problems. The problem with representing a map this way is that the graph gits so big and there is a lot of unnecessary points for finding a route from A to B. To minimise the amount of data the graph consist of, shortcut have been introduce. These shortcuts are called segments, and they go from intersection to intersection, so all intermediate points on the original map is then ignored. This makes the graph a lot smaller and faster to search through.

4.2 Map-matching

Map-matching as describe in section 2.3.3 looks at the way this project have implemented this process, but this have highlighted some problems. Given a set of way-points the mapping function will give a route if one or more of these way-points are wrong the function will still make a route with them, never looking if this is possible.

This means that the map data need to look at again to fix these errors. A solution would be to build the map-matchere ourselves "it was out of this project scope". It would allow us the merge the mapping and error detection into one process. Another possible solution could be to use another map-matching program, but more time would be need to make an informed decision on which of the service from section 2.3.3 to or if non of them are usable.

4.3 Data gathering

For this project to work properly, the data needed is a big part of it. Some of the problems with the data used in this project are described in section 2.3.2. For the system to be able to make good decision a lot of data is needed and should be well distributed over all routes. It is hard to tell when there is enough data for each route.

To collect this data a smart and easy way needs to be in play. In section 1.2 some of the possible way of collecting data is analyse. The data should consist of location, time and speed, location and time is in the data we got now but we need to calculate the speed form the data we got now. Since there can be some error with the location data then the speed calculation will yield a wrong result, and the cost function is depended on the correctness of the speed. The client-server solution we proposed is set up to collect the data needed, if we had more time for this project we could have collected data on our own and had a lot better data to work with, since the data we can collect through the client would be the GPS coordinates, time and speed, where speed has been a huge issue for us because it was not given in the dataset we worked with.

Instead of checking the distribution of data as explained in 2.5.5, we could also have been working on this data set by running multi pass over it. This could have been done such that in the first pass only processed segments that contained enough observations, and which were to some degree uniformly distributed over the whole day. Then for the second pass process the segments which now also had neighbours of the same roadtype, that now had speed functions, and then use these as a part of determining the speed functions of these segments. Then at the last pass, if any segments were left then we could have used the median speed of all segments of the same roadtype to determine those speed functions.

ref til aau artikel

4.4 Bi-directional A*

The path finding algorithm describe in section 2.6.1 is a sub optimal for finding a path through a graph. In this section another version of an A* will be looked at, the bi-directional A* algorithm.

Bi-directional graph search is in it's simplest form just a two normal Dijkstra's where one starts from the start node and the other starts from the goal node. The reason for using this algorithm over other graph search is that the the time for finding a path is $O(b^{(n/2)})$ where a normal Dijkstra's takes $O(b^n)$, where b is the branching factor and n is the number of nodes in the direct graph. Bi-directional also have the benefit of working well when running in paralleled.

The problem with using this algorithm with our cost function is that it is time depended and we have no way of making a good estimation of the when the driver will be at the goal node. This means that only the search from the start node will be correct with the time for the cost function.

This means that to implement a more efficient algorithm for finding a path through the graph, a way to estimate the time for when the goal node will be reach is needed.

- problems with mapmatching and alternative solutions
- smoothing of piecewise functions.
- Contraction hierarhies in dynamic road networks
- Utility/decay på kilder til observationer under regression generation
- better cleaning of GPX files (some problems with a LOT of inferred roads with no waypoints)
- include more features in the prediction. Min/max speed, number of vehicles on road etc....

4.5 Client-Server

There are several things that could be added to the client and server as future work, things that we would have done if we had more time. As for the client a specific route option, where you would be able to plot in your position, where you want to go and get a time estimate of your travelling time. A voice like there exists on other GPS, which tells you what direction to turn and if you have made a wrong turn.

Another thing would be live updates, as in if the traffic conditions have changed which would make the server send a new route to the client. When given a route the server could also give 2 alternative routes with the time differences and distance, like Google Maps does. When the client asks for a route the server could respond with 3 routes, which ordinary GPS does, the fastest, the shortest and the cheapest regarding mileage.

At the moment the client only works on Android version 5 and above, it would of course be advantageously to have the client working on lower Android versions, IOS and Microsoft smartphones.

5 Conclusion

This project presented an approach for modeling road network and traffic as weighted directed graphs with weight functions as time-based piecewise linear regression models over traffic observations of speed derived from GPS samples.

The piecewise linear regression models was found to have an average predictive accuracy of NUMBER% of actual observed travel time on routes. This follows from several issues still present in the regression models such as limited data and improvable filtering, and could futher be improved by gathering a more useful dataset where speed is directly measured.

Furthermore the system was found to, in certain cases, be able to find alternative routes avoiding traffic congestion thus potentially provide time-savings for users of the system. By providing such a feature the system can help motorists avoid heavy traffic and be more evenly distributed on the road network.

There are many interesting areas for future work including gathering better GPS data, more accurate map-matching , optimizations in terms of utilizing advanced graph methods such as contraction hierarchies to speed up query time. Furthermore, we find it important to perform field-testing in the future, to get a much better perspective of the accuracy of the system as a whole.

To be able to collect the data needed to predict traffic patterns which our agent needs, we have constructed a client-server architecture, which is described in 2.2. The client collects data, speed, time and GPS locations, and sends that to our server where we can store the information.

Bibliography

[1] Matthew Barth and Kanok Boriboonsomsin. *Traffic Congestion and Greenhouse Gases*. http://www.accessmagazine.org/articles/fall-2009/traffic-congestion-greenhouse-gases/. Accessed 12-10-2015. 2009.

[2] Jeanette Stokols Daniel Stokols Raymond W. Novaco and Joan Campell. *Traffic congestion, Type A Behaviour and Stress*. http://psycnet.apa.org/journals/apl/63/4/467.pdf. Accessed 12-10-2015. 1978.

[3] Joyce Dargay, Dermot Gately, and Martin Sommer. "Vehicle ownership and income growth, worldwide: 1960-2030". In: *The Energy Journal* (2007), pp. 143–170.

[4] Google Maps APIs Elena Kelareva Product Manager. *Predicting the Future with Google Maps APIs*. http://googlegeodevelopers.blogspot.dk/2015/11/predicting-future-with-google-maps-apis.html. 2015.

[5] *GeoLife GPS Trajectories*. http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/. Accessed 06-10-2015. 2012.

[6] Google. *Google Maps*. http://www.google.com/maps/about/. 2015.

[7] INRIX. *INRIX Cost of Congestion Report*. http://inrix.com/economic-environment-cost-congestion/. Accessed 12-10-2015. 2013.

[8] Shoaib Kamran and Olivier Haas. "A Multilevel Traffic Incidents Detection Approach: Identifying Traffic Patterns and Vehicle Behaviours using real-time GPS data". In: (2007).

[9] NCTA. *HOW GOOGLE TRACKS TRAFFIC*. https://www.ncta.com/platform/broadband-internet/how-google-tracks-traffic/. 2013.

[10] David Shinar. *Aggressive driving: the contribution of the drivers and the situation*. http://www.sciencedirect.com/science/article/pii/S1369847899000029. Accessed 12-10-2015. 1998.

[11] John Sousanis. *World Vehicle Population Tops 1 Billion Units*. http://wardsauto.com/ar/world_vehicle_population_110815. Accessed 06-10-2015. 2011.

[12] *T-Drive trajectory data sample*. http://research.microsoft.com/apps/pubs/?id=152883. Accessed 06-10-2015. 2011.

[13] Technical Comittee of The World Road Organization PIARC. *ITS Handbook*. The World Road Organization - PIARC, 0.

[14] Vejdirektoratet. *TMC - Status og udbredelse*. http://asp.vejtid.dk/Artikler/2009/04%5C5481.pdf. 2009.

[15] Machine Learning Group at the University of Waikato. *Weka 3: Data Mining Software in Java*. Accessed 13-12-2015. URL: http://www.cs.waikato.ac.nz/ml/weka/.

[16] DWIGHT A. HENNESSY & DAVID L. WIESENTHAL. *The relationship between traffic congestion, driver stress and direct versus indirect coping behaviours*. http://www.tandfonline.com/doi/abs/10.1080/001401397188198. Accessed 12-10-2015. 1997.

[17] Ian H. Witten Yong Wang. "Inducing Model Trees for Continuous Classes". In: (1996).

A Routefinding pseudocode

Algorithm 1 A* ready for bidirectional search

```
function A*($Graph$, $Start$, $End$)
    create vertex set $Q$                // Set of unvisited nodes

    for all vertex $v$ in $Graph$ do     // Initialization
        $dist[v] \leftarrow$ INFINITY         // Unknown distance from $Start$
                                         // to $v$

        $heur[v] \leftarrow$ H($v$, $End$)       // Heuristic value from $v$ to $End$

        $prev[v] \leftarrow$ UNDEFINED        // Previous node in the optimal
                                         // path from $Start$

        add $v$ to $Q$                     // All nodes are initially unvisited

    $dist[Start] \leftarrow 0$

    while $Q$ is not empty and OtherDirectionNotDone do
        $u \leftarrow$ vertex in $Q$ with min $dist[u] + heur[u]$
        remove $u$ from $Q$
        VisitFromThisDirection($u$)

        if $u = End$ or IsVisitedFromOtherDirection($u$) then
            for this direction: $done \leftarrow$ true
            return ReconstructPath($prev$)

        for all neighbours $v$ of $u$ do  // where $v$ is still in $Q$
            $alt \leftarrow dist[u]$
            if $alt < dist[v]$ then
                $dist[v] \leftarrow alt$
                $prev[v] \leftarrow u$
    return empty
```

Algorithm 2 Bidirectional search

```
function BidirectionalSearch($Graph$, $Start$, $End$)
    $path1 \leftarrow$ run A*($Graph$, $Start$, $End$) concurrent
    $path2 \leftarrow$ run A*($Graph$, $End$, $Start$) concurrent

    if $path1 =$ empty then
        return $path2$
    else
        return $path1$
```