Data Reduction on Travel Time Series Databases

Arjen Tebbenhof

Bachelor Thesis Econometrics
Erasmus University Rotterdam

Data Reduction on Travel Time Series Databases

April 2006

Author:
Arjen Tebbenhof <arjen@tebbenhof.nl>
student number: 151779

Technical document specifications:

Typesetting:	LaTeX 2_ε (release date 2001/06/01)
Written in:	TeXnicCenter shell (version 1 beta 6.30, ToolsCenter.org)
Distribution:	TeXWin32 implementation MiKTeX (version 2.4, MiKTeX.org)

Using the dutch manual for LaTeX *by Piet van Oostrum,* http://www.cs.uu.nl/~piet

Preface

Looking back to the period that I had my internship at TNO Inro, and being a small part of a large research, I discovered that 'writing a thesis' is not that structured as I thought it was. Struggling through articles and literature on fields I was not even aware that they existed, my eyes got more and more open and I got a chance to look at the world of practical investigation and researching problems that we actually have in our daily lives.

In my experience, this world of solving practical and futuristic problems couldn't be better shown at the place I was for more than one year. I had the pleasure to work with a variaty of very nice people who helped me feel comfortable at this place.

I owe a lot of gratitude to a few people, and would like to mention some of them here.

First of all, my supervisors at the university: dr.ir. J. van den Berg and dr.ir. U. Kaymak. I seems that their patience has no limits, every time I contacted them (long overdue some deadlines), they continued helping me following the right tracks and looking for ways of making this thesis to an end. Also my supervisors at TNO Inro, ir. C.M.J. Tampère and ir. H.H. Versteegt. Despite their busy schedule, a helping hand was always available, and I had the opportunity to not only delve into solving a problem, but also being part in the research as a whole.

My fellow students, Richard de Vos in particular, made sure that study and fun could actually stay together when writing a thesis. I had a great time and lots of good laughs.

Last, but definetely not least, my family and friends, who were always interested, and never gave up in asking "Did you already finish your thesis?". Mom and dad, Nico and Sandra, my cousin Simone and motivator Rubia.

Most important in this period, and partially (maybe fully) responsible for the fact that this thesis got to a finish eventually, is Inge. Thank you, for helping me, respecting me, and waiting for me. I will never forget what you did and meant for me, showing unconditional patience, and would wish for a way I could do something back in return.

Thank you all.

Contents

List of Figures

Chapter 1

Introduction

1.1 Travel Time Prediction

The supply of up-to-date and accurate traffic information has been going through a large development the last decades. The provided information, like locations and size of congestion, was given through radio stations several times per hour. With the help of IT-applications in the field, the number of media where this information is made available has grown. One can think of teletext and, of course, the Internet.

One may wonder what the underlying idea is behind the demand for this information. Most people can make a good estimation of their travel time given origin and destination of their planned trip. This is used to estimate the time of departure in order to get to the destination at a certain time of arrival. However, always in assumption of "normal" conditions. One can think of the use of highways, and that the average speed approaches the speed limit on (parts of) the route. The use of some common sense with this raw estimation shows that congestion (caused by accidents, extreme weather conditions, road maintenance, events or traffic peak hours) on the selected route will increase the travel time. The more we know "about our route", the better estimate we can make of the travel time. The interpretation of this available information is very important. If normal circumstances have changed, how does the provided information affect your estimation of the travel time? Does it double the travel time, or is the influence only marginal? If there is no congestion now, what is the probability that there will be congestion on my route as I travel it? How will that affect my travel time?

By analysing the current supply of news on traffic information, you might suggest that the information given is not exactly what people want to know. Instead, the best estimation of a travel time over a certain route is what one is interested in in most cases. With the help of the information technology and the Internet, nowadays it is even possible to receive an advice for your journey from door to door, completely personalized and with an estimation of the travel time. You can think of in-car information, provided by GPS, or the (widely known) route planners in the Internet (Expedia, ViaMichelin, MapPoint, Map24, etc.). These online planners are available for both public and private transport.

The interesting challenge now, is to combine the above. On one hand, there are a lot

of ways to get very accurate travel time predictions under "normal" circumstances. On the other hand, (recent and estimated) traffic information could increase a travel time. A combination of these two must be able to give a personal (door-to-door) travel time prediction, based on recent and future traffic information (called traffic state).

A model (framework) that combines these applications, is currently being developed at TNO-Inro (Institute for Traffic and Transport, Logistics and Spatial Development) in Delft. This framework has the ambition to make a good travel time prediction for any given route over a known network, taking into account the recent and estimated traffic state information. At this moment, a time horizon of approximately 300 minutes is covered, but the ambition exists to make this a few days. The predicted travel times for certain links in this network shall be stored in a central database, and this database is updated dynamically with the best possible estimate of current traffic states, based on the most recent information from all available traffic data sources.

First let me discuss some terminology. A travel time profile is a time series of travel times over a certain link in our defined network. We think of a network as a collection of nodes and links. A node is a point where two or more of these links intersect, and a route is a collection of links and nodes, and represents the path in this network from some origin to some destination. See figure 1.1. A travel time profile has certain characteristics just like

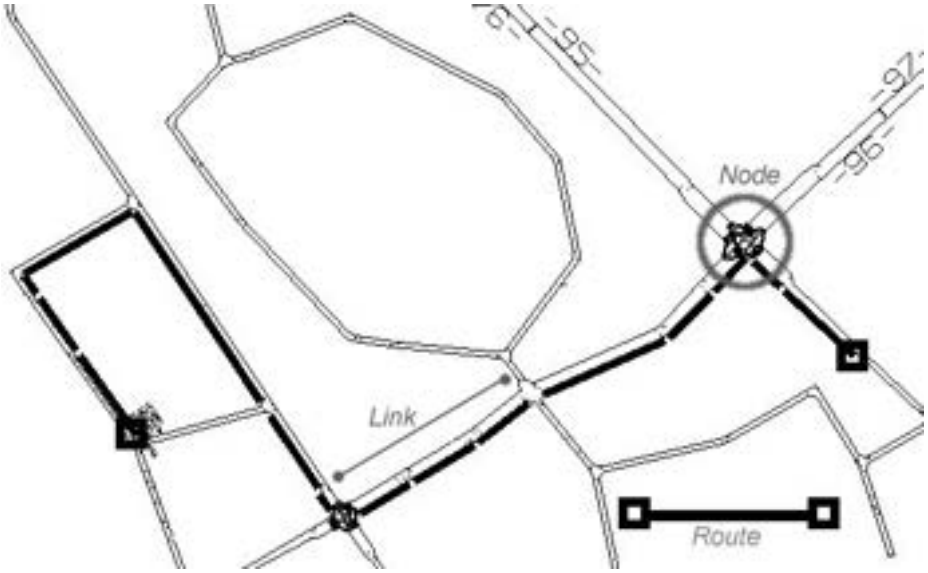

Figure 1.1: A part of a network with a node, link and route

other time series. For example, time series from a stock market have very similar properties to a statistical random walk, as where the sequence of an ECG is often referred to as a periodic signal. Both these examples have their own characteristical behavior. An example of an average travel time profile is shown in figure 1.2. This figure shows a plot of a sample highway in the Netherlands, measured over weekdays of a complete year. Here, you can find a "normal" travel time (the minimum travel time measurement), the median (which gives

a raw feeling of the average profile, insensitive to outliers), and the 15-85 percentile (where 85% of the measurements is still within the plotted bandwith). You can see congestion in the morning and evening peak hours, where travel times are much higher than outside these time intervals.

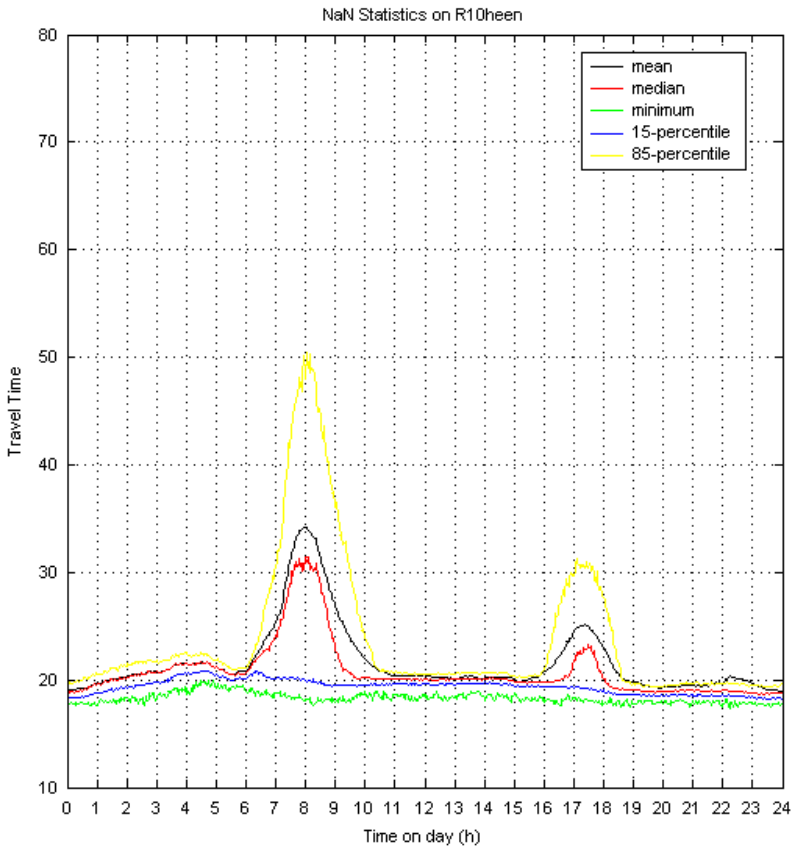

Figure 1.2: Travel time profile of a sample link

From the example in this figure, one can see that under "normal circumstances" (in non-peak hours) a travel over this route will take at least 18 minutes (the minimum), and 15 percent of the travellers need around 20 minutes. In the morning peak hours (around 8 a.m.), there is an average travel time of around 34 minutes, and 85 percent of all the measurements are below 50 minutes of travel time. This is a huge delay, and an increase in travel time of 150 percent! The evening peak hours (around 5 p.m.) have a 23 minute average travel time, where 85 percent is still below 31 minutes. The values are taken from a database with measurements on this route from weekdays in some specific year.

1.2 Prediction Framework

The approach of TNO is to build a framework for door-to-door travel time prediction. The general idea behind this framework is illustrated in figure 1.3.

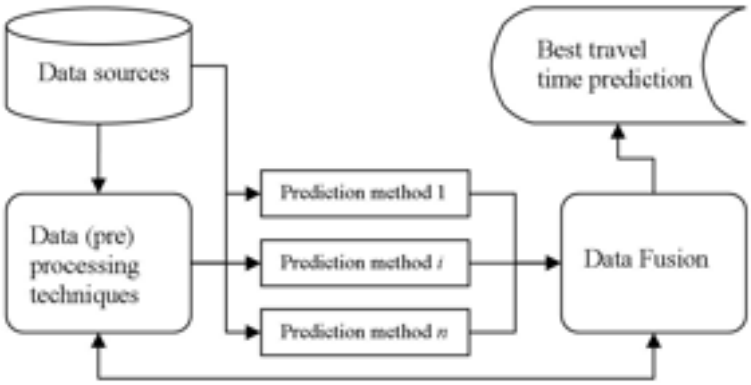

Figure 1.3: General idea behind the framework

The framework uses several modules, each having it's own purposes to make the best and accurate travel time prediction. These modules are connected to each other in such way that all available information will "merge" to the final prediction of a network link. Several data sources must be (and are) available to this system, like historical travel time information, weather conditions, road maintenance, etc. These data sources might need a preprocessing step in order to get them in the right format. After this step, some prediction methods make their individual travel time prediction for some links in our network (or better: all links in our network). These different predictions are "fused" by a final module. This module returns the best travel time prediction for each individual link. A detailed overview of the framework, including an overview of the "prediction methods" can be found in [15].

1.3 Research Objectives

In this design, the fusion method is an important component regarding redundancy of the prediction methods. To reduce the costs (including, but not limited to processing time and purchase of external prediction methods), as few as possible different prediction methods must be taken in account to make a good prediction. If two or more methods give redundant information, then at least one method could be ignored. The question is: "What is a good way to find those redundant information?".

This framework has a predictive nature, and has two characteristics. The first is that (a part of) a travel time profile over some time interval must be compared to already known profiles over the same interval length (and not necessarily the same interval), to find a similar profile. This way, one has an idea of what the travel time profile up until now can do in the near future. The second is that, given some other information than only travel

times on a specific link or route (for example the day and time, weather conditions, or events in the neighborhood of a link), a specific historical travel time profile could have a high predictive value for the given situation. In other words, we want to find external states that could have an explanatory value to a travel time profile.

The amount of available information can increase a lot in time, due to better or more measuring techniques, or due to extension of the traffic network. Is all this information essential and distinctive enough to make a good prediction or fusion of travel times? Therefore, not only redundant prediction methods must be filtered out of the framework, but also redundant travel time information. In fact, we want to reduce the dimension of this information, for the same reason as with the prediction methods: decrease processing time and costs.

The objective for this research is formulated as:

Given a dataset of travel time series, reduce the size of this set in a way that typical prototypes of a travel time profile remain.

1.4 Outline

To gain insight in the framework and its prediction properties and to get a good structural solution for the described problems, I took the following steps. First, I examined the complete framework as a whole, to get a good feeling of the initial problem, the ways to get to a prediction, and the problems that occurred in the separate steps. I then thought of what factors could have an influence on travel times, and whether there was a way to quantify the factors and circumstances that have been found. Some common sense, and also discussions with some of the employees of TNO Inro gave me insight in how a travel time can be influenced. You can find this in chapter 2.

Second, I studied methods that could help me make the available travel time series easier to work with. The objective is to reduce the size of the given database, and I started a search for articles about this subject. For example, how can I reduce the dimension of the available data, and what methods exist to work with function parameters instead of the whole dataset. Some general information about knowledge discovery in large databases is given, from where I start a search for articles and theory that fitted the scope of this problem. After all, we were dealing with travel times, and not just some random time series. Travel times have characteristics of their own, so theories that could take advantage of those characteristics should be used. In this study, a polynomial fit and Fourier Series are applied to a sample dataset to see if there exists some functional relationship between travel time and some timestamp. Furthermore I study the clustering of travel time profiles. If we find a cluster of “similar” travel time profiles, then a set of clusters gives a representation of our complete dataset. This way, the amount of information is reduced to a set of prototype profiles, instead of a larger set of predictions or measurements. Thus, the dimension of the data has decreased. These steps are explained in chapter 3.

In chapter 4 the described methods are tested with a sample dataset. First, the structure of the dataset is explained and the type of available data is shown. This will be followed by a statistical analysis, to get a better feel with the type of data, and can give us insight if any used techniques are giving desired results. Finally, the results of the different reduction techniques are shown.

An overall conclusion can be found in the final chapter.

Chapter 2

PredicTime Framework

In this chapter, I give an introduction to the framework of PredicTime (see section 1.2), followed by a research to external factors that could influence travel times.

2.1 PredicTime

The next section is quoted from [15]

> Accurate travel time prediction is gaining more and more importance. This has resulted in the development of many travel time prediction methods in recent years. However, in order to be able to provide travelers with individual travel time information, a number of improvements are still needed. First of all, there is often only limited information for a limited part of the route. Next, in cases of incidents, where the need for information is becoming more urgent, the provided information to travelers is often quite limited. Thirdly, the information to the travelers has to become more personal, more complete and more predictive. Reliable travel time information can compensate in a number of ways for these shortcomings. For that aim, an open framework for travel time prediction, PredicTime, is being developed. PredicTime will use the predictions of several, separate forecasting methods to predict future travel times from door to door. The system should be able to forecast travel times for virtually any route for each possible time of departure. It should take into account the actual traffic situation, the consequences of incidents, events, road works, etc. Finally, it should be open to new data sources and future prediction methods. Based on a review of travel time prediction methods, it can be concluded that indeed no single method is expected to be the best method under all circumstances. Instead, each method seems to have its own strengths and weaknesses. If the full prediction horizon is to be covered, a combination of methods seems to be the best option. This idea has been incorporated in the functional architecture of PredicTime in figure 2.1.

Looking at this figure, you see a lot of components. Some explanation can be found in the following example.

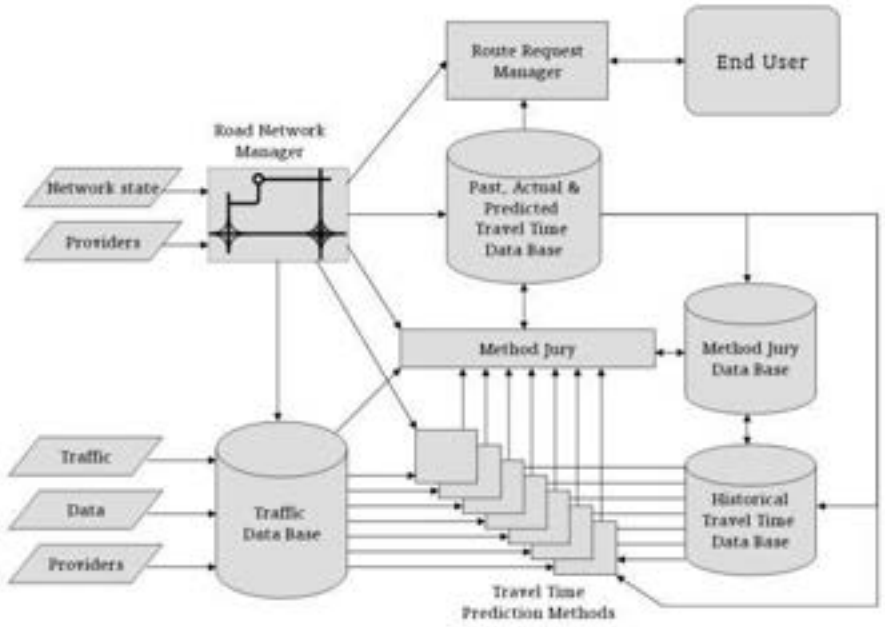

Figure 2.1: PredicTime's open framework for travel time prediction

Imagine that an end user is trying to get a travel time forecast for a given route and given time in the future. He gives his route to the Route Request Manager (RRM) and expects his travel time back. The RRM maps the route on the Road Network Manager to receive the relevant network links. For these individual links, the RRM checks the predicted travel time in the Past, Actual and Predicted Travel Time Database (PAPTT). Once resolved, he returns the output back to the end user. The PAPTT Database thus holds all predictions of travel times over the network links for some time horizon and uses a moving horizon.

This database is updated by the Method Jury, which gives the final prediction for all network links. The Method Jury does not do the actual forecasting, but only merges (fuses) the forecasts of the Travel Time Prediction Methods (TTPM's), as explained in the first paragraph of this subsection. The TTPM's make use of the following information for a forecast: The Traffic Database (contains recent and actual traffic data measurements, potentially including road maintenance information, weather information, etc.) and the Historical Travel Time Database (contains the best ex-post estimated travel times). The merge (or fusion) that the Method Jury does, has the characteristic that it must be *at least* as good as the best prediction of the TTPM's. A Method Jury will therefore be developed to "judge" and combine the predictions made by individual travel time prediction methods using data fusion techniques in order to obtain a better forecast. This data fusion technique builds up it's own database with reference data, the Method Jury Database (MJDB). This reference data consists of time series of recent predictions (made by the Method Jury) and prediction errors of each TTPM (when an ex-post prediction is known).

This information should be available for all network links. The MJDB thus receives input from the Historical Travel Time Database, the PAPTT Database and the Method Jury itself. At this moment, this is still a conceptual description of the database, and no strict conditions or demands are defined for the structure or contents of this database as yet.

Another research for this framework, separate from this thesis, has to be focused mainly on developing specifications for the Method Jury and for travel time prediction methods. These components, combined with the Method Jury Data Base, can be seen as the heart of PredicTime's framework. To develop these specifications, travel time data has been analysed and tests have been carried out with some existing prediction and data fusion methods. In the next section, the available data and information to the MJDB will be presented, and the way the database stores this information.

2.1.1 Available information and storage

The information that is available to the MJDB from the framework per *network link* is summarized as follows:

- Past, Actual & Predicted Database
  - Time series of recent and future (ex-post) travel times
- Method Jury
  - Predictions by Travel Time Prediction Methods
  - Prediction by Method Jury
  - Traffic Data (external factors)
- Historical Travel Time information
  - Time series of past (ex-post) travel times

The next task is to specify the characteristics of the above information, keeping in mind that we want to have access to travel times per known individual network link.

Something else to keep in mind is the fusion technique that the Method Jury uses. If the Method Jury uses a Kalman Filter, the computed weights are method-specific parameters that need to be stored. If the fusion is based on an (adapted) Dempster Shafer technique, then the Method Jury wants to store the different classes that have been applied to the predictions. Maybe other fusion techniques will be implemented in the future. It is then desirable that the Method Jury gets his own storage place for these specific parameters. I name this component the 'MJ Method Parameters'.

Also, our network must be defined before we try to fill the database. For test cases, we can use complete routes as single links, with no dependency between them. But as the project progresses, some links will have a certain effect on (neighbor) links as travel times change. The dependency has to be implemented as well, which I will leave for future work.

2.1.2 Functional Structure

The Method Jury Database is assigned the structure in figure 2.2. You will find the relations between the components, and where there is a data storage or a data analysis.

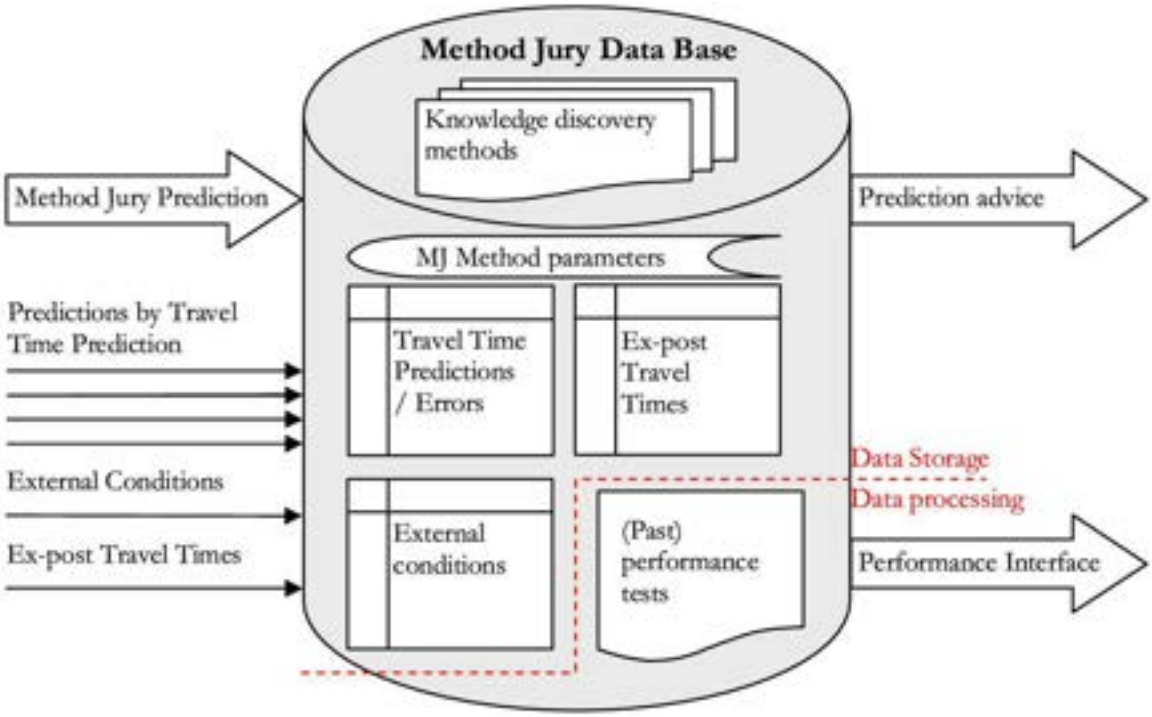

Figure 2.2: Functional structure of the MJDB

Travel Times

If we define a set M of Travel Time Prediction Methods, there are three different travel times to be saved:

1. $\hat{y}_{il,t}$ the travel time predictions calculated by Prediction Method $i \in M$
2. $\hat{y}_{l,t}$ the travel time prediction made by the method jury
3. $y_{l,t}$ the ex-post travel time (i.e. what we should have been predicting)

Each travel time needs to be saved with the indices l and t, connected to vector $S_{l,t}$. This will be explained in the following paragraphs.

Link l

A travel time is only defined for a certain link l on our network. We can always extend this to a combination of links, or define a route as a single link. This all depends on our network

information. The exact way to store the link information is also to be considered. Do we use an index, or do we use a part of a route-name, how do we store vice-versa information, and so on.

Every individual link has also additional information. It can not be said if these properties are static (same value for a fixed time interval, like road type and length) or dynamic (variable due to traffic situation, like maximum speed and number of lanes). Since we don't use this information in this problem case, we accept this information as-is.

Timestamp t

Every travel time occurs at a unique time stamp t. This means that every travel time we know is always connected to a time stamp, let's say "yyyy-mm-dd hh:mm". Of course, we should be able to find all travel times occurred on a Wednesday at 5 p.m., so a good way of storing this is needed.

Factor Vector S

Also, every travel time is connected to a factor vector S, showing external circumstances / conditions that occurred when this travel time was acquired / estimated. This means that we can trace the circumstances that have taken place if we look at a travel time on a certain network link l at time t. We need the indices l and t to connect a "unique" factor vector to a travel time.

2.2 External Factors

A factor has some (qualitative or quantitative) influence on a travel time (prediction) over a network link. One can probably think of a few factors that could have an influence on travel times.

In the first part of this section, I would like to distinguish a few types of factors:

- (Traffic situation) *dependent* and *independent* factors
- Factors with *structural* and *incidental* influence

The factors I will use in this case can be mostly assigned to traffic situation *independent* factors. The opposite, traffic situation *dependent* factors, are much harder to implement because they vary due to (for example) congestion. Some examples of these factors are route deviations and dynamic traffic management (traffic light manipulation). Another problem of these factors is that we do not have complete information about it. Traffic management can be applied on a link or node, without us knowing it.

Structural influences are characterized by the fact that they happen on fixed time stamps or intervals, so we are able to assign an explanatory value to them. The incidental influences however can only be taken in account in predictions with a much higher uncertainty. The only thing valuable we can say about them is that there exists some probability that they happen.

An example of the above factor-types is the following: if a bridge opens every morning at 8:00h and every afternoon at 16:00h then this is a structural factor. But if the bridge opens when a boat arrives, we can only say that there is a positive probability that the travel time increases by a few minutes, depending on the amount of boats crossing a network link in some time interval. This implies that the uncertainty of a prediction will increase, but the prediction itself will only say something about the travel time under normal circumstances. The same is to say about traffic accidents. We *can* say that there is a probability accidents happen on a network link, but we *cannot* predict when they happen. However, we *do* want to know if an accident occurred at a certain time stamp for historical purposes. An accident is an explanatory variable when we look at past travel times. Some external factors increase the probability of an accident, but there are also accidents that are not caused by known circumstances. The incidental influences are not used in predicting, but only for statistical purposes to decrease the noise on time series (see 2.3.1).

The external (independent and structural) factors and conditions will be given to the MJDB either through the Method Jury, or in another (not defined yet) way. It is not clear at this moment if the external conditions are all stored in the Traffic Database, but it is obvious that if Travel Time Prediction Methods need these factors, that they must have access to it. For now, I'll assume that the MJDB gets its information through the Method Jury.

2.2.1 Seasons

We can define a season as a period where there are typical characteristics of traffic situations. These can have economical or recreative causes. One can think of the December-month in a whole, because lots of people then go out to visit family and friends, short holidays, or visit large store locations. For this definition, it is possible to divide the seasons in spring, summer, autumn and winter. A more sophisticated season-definition can also be investigated.

A possible way to quantify this influence is to create an entry *season* in the database with domain $\mathbb{R}^+$. A certain link or travel time may then be associated with a defined season if the entry is non-zero. This could be a Boolean value, or a value between zero and one (as some fuzzy member of more than one season).

Season is the first of three *period-patterns*. It has a small influence on traffic, and is spread over a period of (several) weeks or months and over a large amount of links.

2.2.2 School Holidays

The school holidays can be structured as *boolean* value. Zero indicates no school holiday, one indicates holiday. Per available network link can be saved if there is a holiday in the area of that specific link. In the Netherlands, the holidays are divided in parts north, middle and south, and are separated for primary school and high school. We have to map the links in our network to the known holidays.

Something to think about, is that since lots of Dutch people tend to go south on holidays, a vacation in the north definitely can have an influence on traffic in the south of the

Netherlands. *Boolean* might not be a perfect way to store this if we keep this in mind. Further research should determine what the amount of influence of a school holiday in a different region is.

Holiday is the second of three period-patterns. It has a significant influence on traffic, on a collection of links, and is spread over a maximum of a few weeks.

2.2.3 (Large) events

Last, but not least, are the organized events. Some examples are the "Nijmeegse vierdaagse", the Fun Fair in Tilburg, important soccer - games, music - festivals, Queens's day, and so on. These events (mostly) happen on fixed time stamps, or during an interval of time stamps.

These events have a very high influence on travel times, because the traffic supply increases with sometimes factors of ten times. Then again, it only counts on very specific links in our network, and a range of links in the neighborhood. A possible way to store this data is to define it as a Boolean on the days the events happen. If the event does not occur on a certain time stamp, the value is zero, if it does occur, the value is one. Data analysis on known information about events could be needed to give a better way of storage of this factor.

Large events is the last of three period-patterns. It has a very high influence on traffic, on very specific links, and is spread over just a few days.

2.2.4 Road Maintenance

From AVV (Rijkswaterstaat, Adviesdienst Verkeer en Vervoer) we have a lot of information available on road maintenance. Several tables with data over the year 2002 contain information about work-phases (date), diversions, ramps, openings and closures of roads, road blocks and road signals. Whether we can use this information is very hard to say, because everything available must be rewritten in a format that can be used in combination with travel times. This looks like a very big job, and requires a separate study.

To simulate this influence, we can alter the capacity of (a set of) network links.

2.2.5 Weather

This is a tricky part of our database. At this moment, we do not exactly know how much and in which format our weather conditions will be available to us. But, like holiday, a weather factor will hold the same value for some subset of links in our network. No matter what the condition will be, it will almost surely have the same effect to neighbor links.

Some research at the Royal Netherlands Meteorological Institute ([1]) indicates that in the Netherlands there are 283 stations that measure rain, divided over 15 districts. For six main stations (De Kooy, De Bilt, Eelde, Twenthe, Vlissingen and Maastricht), this institute can provide a lot of information freely. A table with an overview of this information is found in table 2.1.

I collected this information for the year 2001, more recent information could be needed, depending on what is available from other data resources.

- prevailing wind direction in degrees (North, South, West, East, 0 = calm / variable)
- daily mean wind speed (in 0.1 m/s)
- maximum hourly mean wind speed (in 0.1 m/s)
- maximum wind gust (in 0.1 m/s)
- daily mean temperature (in 0.1 degrees Celsius)
- minimum temperature (in 0.1 degrees Celsius)
- maximum temperature (in 0.1 degrees Celsius)
- sunshine duration (in 0.1 hour, -1 for ¡ 0.05 hour)
- percentage of maximum possible sunshine duration
- precipitation duration (in 0.1 hour)
- daily precipitation amount (in 0.1 mm, -1 for ¡ 0,05 mm)
- daily mean surface air pressure in 0,1 hPa
- cloud cover in octants (9 = sky invisible)
- minimum visibility
  - 0 = less than 100m
  - 1 = 100-200m
  - 2 = 200-300m
  - ...
  - 49 = 4900-5000m
  - 50 = 5-6km
  - 56 = 6-7km
  - 57 = 7-8km
  - ...
  - 79 = 29-30km
  - 80 = 30-35km
  - 81 = 35-40km
  - ...
  - 89 = more than 70km

Table 2.1: Information by the Royal Netherlands Meteorological Institute ([1])

The main problem with this data is to assign a link in our network to a certain weather station. As some information stated above can be saved with a fuzzy characteristic (in order to distinguish several classes instead of a continuous domain), a link could also be partially member of more than one district. A topographical overview of the locations can be found in figure 2.3.

A side study has already been done, where all the available information (stated above) has been translated into more common terms. One can think of road capacity reduction factors for fog (visibility decreases), precipitation under light and dark circumstances, and dry weather under light and dark circumstances. A bit common sense says that ice on roads also must have some influence, but I haven't seen any information about this factor yet.

To simulate this factor, it is possible to change the capacity of network links. The influence of precipitation makes most road users drive more carefully, so that they hold more distance from each other and decrease their speed. This is a reduction in road capacity.

2.3 Framework discussion

The MJDB is not a database that just stores travel times and factor vectors. There are several wishes and issues that the framework has to address. In this chapter, I will first

discuss the framework issues that have effect on the framework as a whole, and then indicate the issues of the MJDB itself.

A typical problem that a method jury should solve is the following: It could be possible that TTPM m structurally estimates too low when factor s_1 has a certain value. Therefore the Method Jury must learn from our ex-post data to find this structure and relate the different factors in S to a TTPM to give a certain 'belief' about one. The MJDB only must supply the Method Jury with the right information to find these structural errors.

I would like to give some definitions that I want to use later on in this chapter (see also 2.1.2). Once we have received an ex-post travel time y, we know what our overall error e in the overall prediction $\hat{y}$ was, and what errors e_i our TTPM's gave us. Furthermore, our defined factors can be found in factor vector $\vec{S}$:

$e_i = y - \hat{y}_i$	the prediction error of method $i \in M$
$e = y - \hat{y}$	the prediction error of the Method Jury
$\vec{S} = \begin{bmatrix} s_1 & s_2 & \dots & s_n \end{bmatrix}$	where n is the number of factor variables we store

Performance tests on TTPM's

One can imagine that under some circumstances, more than one Travel Time Prediction Method (TTPM) gives a prediction of a travel time. That is why we have a method jury, to make sure a value that is at least as good as the best prediction actually enters the PAPTT database.

Now suppose we don't know how a TTPM calculated its prediction, it just gives the value and maybe an estimation error. If we do not know the method of the estimation, then it is not trivial that each individual TTPM has a complete distinguishing value from the other methods. In fact, maybe two TTPM's actually use the same data source for their answer, or just have the same estimation method with some additive error term.

Of course we want to know if maybe one or more of these methods are no longer needed. This is typical something that you could extract from the information in the database. We can construct a correlation matrix of the predictions of method i and j, given some circumstances in factor vector S. If there is a correlation, and this correlation is structural, then at least one of those two does not improve our overall travel time prediction.

Factor changes

We also want to validate the different TTPM's. It is obvious that a sudden change in factors from $\vec{S_1}$ to $\vec{S_2}$ also changes the real travel times. Our Method Jury must then be able to check if a TTPM made errors in his previous predictions. It would be very likely that that TTPM makes the same error again.

What we also want to do is to make sure the influence of sudden factor changes does not influence the quality of the Method Jury's prediction. This is necessary if the Method Jury uses a Kalman filter, for example. The time that the Kalman filter needs to adjust it's weights is too long for short term predictions. It then would be more realistic if it replaces his recent history values with ones that apply more to the new factors in vector $\vec{S_1}$.

For the two issues named above, we need to find a two way relation between factor vectors and travel time series. That is, given a factor vector, we want to know if TTPM i

has structural errors in it's predictions. As second issue to solve is to search in which case of a factor vector, a TTPM (or multiple TTPM's) have structural errors. If we see that there are a lot of errors in our error-table, we want to find out if there is some similarity in the external factors.

2.3.1 Other Statistical Tests

As indicated in 2.2, we can do a side analysis to find out the explanatory value of incidental influences on travel times. It is well known that traffic accidents increase the travel times enormously, and we don't want our structural external factors to explain high travel times due to incidental influences. The noise in the regression should decrease and we simultaneously collect information about the probability that incidental influences happen on certain network links. Because this is not a part of this study, this could be future work for the framework of PredicTime.

Figure 2.3: Topographical overview of the weather stations

Chapter 3

Research and Methodology

3.1 Knowledge Discovery

In order to extract useful information from the existing data sources, we need to describe the information that we eventually want to obtain. This means that we need a combination of

- retrieval of existing knowledge (either historical data or other factors that influence travel times),
- a way to store (and manipulate) this knowledge,
- a way to combine our results and answer our questions.

One approach of a solution could be data mining. Data mining can be defined as the efficient discovery of previously unknown, valid, potentially useful and understandable patterns in large databases. We can think of several tools to accomplish this goal. Database tools are needed to handle (collect, clean and combine) the data, statistical tools to analyze the data (logit regressions for example) and the actual data mining tools, to perform the mining process. In the end, we want to provide quantitative outcomes on relations between different attributes, and actionable information.

There is a lot to find in the literature about finding similarities in large databases of time series. I will start with articles that give an overview of similarity search problems with time series. In [8], we find an overview of what tasks within time series data mining have already been researched, and see a segmentation of the problem field to *indexing*, *clustering*, *classification* and *segmentation*. The terms used are defined as follows.

- *Indexing* is to find the nearest matching time series in a database, given a query time series Q and some similarity threshold
- *Clustering* is the problem of finding a natural grouping in a database with time series.
- *Classification* is assigning an unlabeled sequence to one of two or more predefined classes. Reformulated, when we define our classes as the found clusters, assign the unlabeled sequence to the supporting cluster.

- *Segmentation* is modeling the time series in a way that only K piecewise segments of all the n data points in the time series are a good approximation of the original time series.

The difference between classification and clustering can be understood by the following: In the case of this research, one is interested in a typical travel time profile for some route. Because we are looking for different groups within the set of historical travel times, this can be assigned to clustering (the third technique). But, once these clusters are defined, it could also be interesting to look for relations between the set of clusters, and maybe some other states, for example: the day of the week, or the weather conditions (see section 2.2 on page 11. This part refers to classification (the second technique). The latter is for future research.

The essential issue for this research is that we have to find a similar pattern of values over some time horizon. Reformulated, if $Y_1 = y_{1,1}, y_{1,2}, \ldots, y_{1,n}$ and $Y_2 = y_{2,1}, y_{2,2}, \ldots, y_{2,n}$ are two sequences of time series, define and compute $\texttt{Similar}(Y_1, Y_2)$.

With this formulation, there are some problems that we have to think of. For starter, we have to keep in mind that the similarity measure has to allow imprecise measurements, like missing data and outliers. Furthermore, we have to look at some indexing method to see which time series are grouped in one cluster.

Suppose we are given a database with time series: a large collection of sequences of real numbers, representing the measurements (or predictions) of travel times at equal-spaced time intervals, with non-decreasing timestamps. The objective is to reduce the dimension of this database and to find clusters of "similar" time series. [7] brings a definition of this problem, repeated here for reference purposes:

> Given a sequence $\vec{q}$, a set of time sequences X, a *distance measure* d, and a *tolerance threshold* ε, find the set R of sequences closer to $\vec{q}$ than ε, ore more precisely:
>
> $$R = \{\vec{x} \in X \mid d(\vec{q}, \vec{x}) \leq \varepsilon\} \tag{3.1}$$

3.2 Surveys

Time series is a very abstract definition of a sequence of floating point values. Stock market data (compared to the statistical random walks) have other properties then time series for an ECG in medics, and these differ in properties of time series for travel times. According to [8], most articles about this subject test their techniques on only one dataset. No matter what the outcome will be, the proposed methods are proven to work on very specific data, and was not tested on other datasets. The authors define the *data bias* as the conscious or unconscious use of a particular testing dataset to confirm a desired finding. This means that suggested algorithms could not always be as fast and accurate as the author says, and may only be applicabable to time series with the same properties of the testing dataset. A solution is to test algorithms on heterogeneous datasets and compare those results with other techniques. In [7], the author gives a survey of approaches of signature based similarity retrieval for time sequences, found in the literature. In the discussion, he mentions one of

the obstacles that need attention: the presence of noise in various forms makes it necessary to support flexible similarity measures.

3.3 Preprocessing and Imputation

If we were living in an ideal world, there would be no such thing as missing values in a dataset. But real-time datasets (in contrast to synthetical datasets) are rarely clean of missing values, due to for example failures in the data measurement system. A set of faulty or missing values could decrease the accuracy of the complete model.

The results and outcome of this study are a basis for a prediction framework. Eventually, accurate and reliable traffic information can only be gathered from good source data. Therefore, faulty and missing input data is of particular interest. I will shortly show some imputation strategies for preprocessing data. Let me recall the structure of a dataset with travel times, as described in 4.3. There, we can see n data points, and each datapoint can be interpreted as a row with t travel times. A missing value in this context is when one of those t values is missing.

A first way to solve this problem, is to delete a complete datapoint whenever one of the t values is missing. The total dataset will decrease of course, and it depends on the structure of the dataset whether this is a smart thing to do. Suppose a given dataset has n datapoints, and these datapoints are also an equal division of measurements over 5 days of the week. Suppose furthermore that every monday morning the measurement unit needs maintenance work for some time interval. In worst case scenario, one might delete all measurements done on mondays, only because there are missing values for a small interval. In this scenario, deleting the complete datapoint is not a good option.

A second way to deal with missing values is to replace them with the string "NaN", which is used in several statistical packages, for example in MatLAB. The big advantage of this string is that the original structure of the dataset remains intact and all results are based on all real-time measurements. However, chosing this option leaves "gaps" in some intervals of a datapoint. Descriptive statistics can work with these gaps, but algorithms that search for some functional structure within a dataset cannot work with these gaps. At this scenario, the first given option of deleting a complete datapoint gives better results.

The last strategy that I will describe is to replace the missing values with an imputated value. This way, the original size of the dataset stays intact and there are no gaps in de measurements. Of course, the first question that comes in mind is with wich value one should replace a missing value with. There exist a lot of ways, I will name a few:

- linear interpolation
- nearest neighbor interpolation
- cubic spline interpolation
- neural networks

The basic idea of linear interpolation is that new values are computed linearly between the last known value before the gap, and to the first known value after the gap. More

specifically, let τ be a number between 0 and 1 which represents at what point we want to interpolate a value y between time t and $t+1$. Then we can define the linearly interpolated value $\hat{y}(t+\tau)$ as follows:

$$\hat{y}(t+\tau) = (1-\tau) \cdot y(t) + \tau \cdot y(t+1) \tag{3.2}$$

Intuitively, this might be a good substitution, as long as the gaps are not too big. The bigger the gap gets, the more difficult it gets to get a good approximation with linear interpolation. Additionally, when we speak of a gap, the assumption is made that there are only missing values in the middle of a datapoint. Of course, at the beginning or at the end of a datapoint, missing values could occur just as easily. Linear extrapolation is a bad option in most of the times, because it extrapolates the last known stucture of the measurements, or works towards the first known structure. This means, when you visualise the interpolated values in a graph, negative travel times could occur.

A better idea to deal with this problem is to look at neirest neigbor interpolation. This method sets the value of an interpolated point to the value of the nearest existing data point. This method is better suitable for the edges of a data point, because the direction of successive measurements has no influence on the interpolated value. However, if the known datapoints are successive to one and another, there exists another option for interpolating the missing values. Suppose $y_{n,0}$ to y_{n,t_1} are missing. Then the values of this data point's predecessor, y_{n-1,t_0} to $y_{n-1,T}$ are good points to work with for linear interpolation. This is only true if datapoints are given continuous in time, and does not give correct information if some days are missing.

It totally depends on the structure of the dataset to determine which option is best suitable. There exist a lot more options, this is left as a separate study per dataset.

3.4 Dimension Reduction

We find a summary of dimension reduction techniques in [8]. Named methods are *Discrete Fourier Transforms* (DFT, [14], [5], [2]), *Wavelets* (DWT, Haar Wavelets, [11], [13]), *Principal Component Analysis* (PCA, [5]), *Self-Organising Maps* (SOM, [4]) and *Singular Value Decomposition* (SVD, [13]). These processes can be summarized as signal processing techniques ([10]).

Different methods that index sequences, as researched by [7], do not work well when the data has a high dimension. A solution to this problem is dimensionality reduction, and extract a signature of the original dataset. Then indexing on the signature space yields the desired result.

In order to make sure we approximate the original dataset, a bounding condition is introduced: the distances between two signatures must not be greater then the distances between the original data. This bounding condition is a relaxation on the original problem. An example is using a *Discrete Fourier Transform* (DFT) on the dataset, and use the first three to five biggest amplitude coefficients as a signature. Then, Euclidean distance in the signature space is smaller than the real Euclidean distance in the original set. However, this works only on complete sequences, and not on subsequences.

Finally, another smoothening method is merging several sequences into one sequence and identifying the prototype for it ([9]). To make sure that we compare time series, independent of their mean value and bandwith (so scaling and translation problems are reduced), a normalization of the data can be done ([5]).

In the next chapter, I will show some results of appllying a Discrete Fourier Transform method as a technique of smoothening data. This is based on the fact that travel times over a day typically already shows a similar structure. If the results are good, one might think of using the DFT parameters for forther clustering methods, in stead of the already known techniques.

3.4.1 Polynomial Fit

A polynomial fit is straightforward least squares fitting with an nth degree polynomial function of type

$y = a_0 + a_1 x + \ldots + a_k x^k$

and the residual is given by

$R^2 = \sum_{i=1}^{n} [y_i - (a_0 + a_1 x_i + \cdots + a_k x_i^k)]^2$

One could guess that a travel time profile over a day could be estimated with some degree polynom. The higher the degree, the better the fit will be. Of course, a high degree will give us more parameters and therefore this method will only be interesting if the degree of the polynom is "low", so we could compare its fit with other used techniques.

The theory behind a polynomial fit can be read from [18].

3.4.2 Fourier Series

We can also try to give a representation of the travel times by a sum of approximating functions. If we think of a travel time profile over a single day as a signal, then a Fourier series might be a good solution. The idea behind Fourier series is that most signals can be represented as a sum of sine and cosine waves with some periodicity as a property. If this is not the case (indeed, with this research it is not, since we are dealing with a travel time profile over a length of 24 hours, with no periodicity within this interval), onde can assume that a day is a complete period.

The theory can be read in Appendix D.1 and in [17].

3.5 Clustering

Clustering is defined as: *Find groups of similar objects, given a set of objects.* Of course, this is a very general definition. The word *groups* represent the number of similar travel time profiles. The set of objects should be considered as the complete dataset. In this case, an approach of the word *similarity* can be described as below.

A clustering or classification can only be done if we predefine when two time series are similar. As mentioned earlier, a lot of papers are available concerning similarity measurement in time series databases. Similarity in time series is defined as follows: if $Y_1 = y_{1,1}, y_{1,2}, \ldots, y_{1,n}$ and $Y_2 = y_{2,1}, y_{2,2}, \ldots, y_{2,n}$ are two sequences of time series, define

and compute `Similar`(Y_1, Y_2). In [8], the authors implemented several of those techniques and reported the result as an error rate of the various similarity measures. This is tested on two (publicly available) synthetic datasets. The best outcome is the *Euclidean distance* (see also [5], [13]), where the authors also state that this is only done on 2 datasets. There may exist databases where other measures of similarity actually *do* outperform the Euclidean distance. However, [7] suggests that *time warping* is a more robust distance measure with more possibilities, but without any loss of performance.

There exists a large number of clustering techniques: k-Means Clustering (See 3.5.1), k-Mediods algorithms, Hierarchical algorithms (used in [16] and [5]), Density based algorithms, density approximation, Cluster, then Classify (See 3.5.2) and Greedy ([3]).

Algorithms for *Fuzzy clustering* (i.e. a sequence does not belong to only one cluster, but can belong to more clusters with some degree of membership) are found in [10]. There exists a good reason tot suspect that this might apply to our case, due to the various external factors that have their influence on the travel times, and therefore also on the database.

3.5.1 k-Means clustering

Most datamining techniques do not work for time series due to their unique structure. The high dimensionality, very high feature correlation, and the (typically) large amount of noise that characterize time series data present are hard to solve. The initial clustering is performed with a very "quick and dirty" clustering of the data. The results obtained from this clustering are used to initialize a clustering at a slightly finer level of approximation. This process is repeated until the clustering results stabilize or until the "approximation" is the raw data.

An appliction can be found in [11] and [3].

3.5.2 CTC (Cluster, then Classify)

Some classification might fail, becuase there may be two or more distinct shapes that are typical of the class or group. To avoid this problem, an algorithm needs to be able to detect the fact that there are multiple prototypes for a given group, and classify an unlabeled instance to that class if it is sufficiently similar to any prototype. The CTC is an algorithm which does exactly this. For further details, see [9].

3.5.3 Fuzzy C-means Clustering

With fuzzy clustering, a data point may belong to multiple clusters with different degrees of membership. This means that different travel time profiles, that may look the same, have overlapping characteristics. A degree of membership is some "Belongingness" of a point to a cluster. The sum of the degrees of memberships of a single datapoint adds up to 1. The advantage of fuzzy clustering is its interpretability, which is based on the fact that the membership functions partition the data space appropriately.

Fuzzy clustering can adapt to noisy data. In stead of clustering crisp and accept an large error, a datapoint belongs to multple clusters at some weight and the centroids of the

cluster indicate at what value most datapoints are. Also, classes that are not well separated are handled good with the fuzzy clustering algortihm.

The above can be best explained with the following.

Given:

- $Y = \{y_1, y_2, y_3, \dots y_n\}$, a set of input vectors
- $V = \{v_1, v_2, v_3, \dots v_c\}$, a set of cluster centers
- $c =$ number of clusters and $n =$ number of objects
- U is the fuzzy partition matrix whose entries $\mu_{i,j} \in [0, 1]$ represent the degree of membership of the jth vector point x_j in the ith fuzzy cluster such that

$$\sum_{j=1}^{c} \mu_{ij} = 1 \ , 1 \leq j \leq n \tag{3.3}$$

$$0 \leq \sum_{j=1}^{n} \mu_{ij} \leq n \ , 1 \leq i \leq c \tag{3.4}$$

If the above is given, then Fuzzy Clustering minimizes the objective function:

$$J_S(X, U, V) = \sum_{j=1}^{n} \sum_{i=1}^{c} (\mu_{ij})^m d^2(x_j, v_i) \tag{3.5}$$

where $m =$ fuzzification factor and $d^2(x_j, v_i) =$ distance between x_j and v_i based on the defined distance norm.

The algorithm is as follows:

1. Given X, number of clusters c, distance norm d, fuzzification factor m and termination condition $e > 0$.
2. Initialize the degree of membership matrix U according the constrains on μ_{ij}.
3. Calculate the c cluster centers based on U

$$v_i = \frac{\sum_{j=1}^{n} \mu_{ij}^m x_j}{\sum_{j=1}^{n} \mu_{ij}^m} \tag{3.6}$$

4. Compute the the distance of each input vector to all the clusters centers based on the defined distance norm.

The above is used in the next chapter, where the available data is processed.

Chapter 4

Experiments and Results

4.1 Used Datasets

In report [12], the authors are discussing a way to compute the travel time over a certain trajectory of a highway at some timestamp. The authors have put measurement units along highways in the Netherlands, with 0.5 to 2 kilometers distance between them. Every minute, an average velocity of all passing vehicles is stored. From these velocities, a fictive car is put at the beginning of the traject and moves along the traject with the average speeds that are stored earlier. When this fictive car has reached the end of the traject, the travel time over that traject is then also known. By letting a fictive car leave at the beginning of the traject every two minutes, one can easily get a travel time profile for all the possible hours in one day. By repeating this for all the weekdays in one year, the desired result has been reached. In the complete dataset, a travel time at 8:00h AM is a travel time of trip that started from departure location at 8:00h AM. Thus the destination is reached at time (starting time + travel time).

The available data has a lot of potential information, besides just the travel times themselves. Available is data about every weekday of the year 2001, with included information like length of the traject, date (implicating day of the week), school-holidays and type of weather. The dataset contains 2-minute time-intervals on the columns and days on the rows, according to the data structure shown in 4.3.

Considering each single row, one can think of this matrix as a travel timer during day d. The dates of the travel time measurements are stored in a separate file in a column of date-values, like 2001/03/14. This data is available for 5 highways, in both directions on the specified traject. In table 4.1, you can find some information of the trajects.

4.2 Data Preprocessing

As described in section 3.3, I first extracted the actual travel times for the given trajects. Additionally, I extracted some external information on the fly. The datasets had a lot of "missing" values. That is, travel times lower than or equal to zero are encountered in the dataset. I did not find an explanation for these values in the report, but they certainly can

Name	Road	Route	Length (km)
R04heen	A2 / A58	Eindhoven - Tilburg	27.6
R04terug		Tilburg - Eindhoven	28.4
R09heen	A2	Beesd - Utrecht	28.4
R09terug		Utrecht - Beesd	28.4
R10heen	A27	Gorinchem - Utrecht	34.9
R10terug		Utrecht - Gorinchem	34.9
R19heen	A12	Gouda - Den Haag	31.5
R19terug		Den Haag - Gouda	31.5
R32heen	A7 / A8 / A10 / A2	Purmerend - Amsterdam	32.6
R32terug		Amsterdam - Purmerend	31.5

Table 4.1: Information of the trajects in the dataset

not be travel times. One might think of technical problems with the measurement devices. In subsection 4.2.1 I will explain what the initial preprocessing step is (filtering), and in subsection 4.2.2 I will explain what I did with with the missing values.

4.2.1 Filtering

A complete overview of the data in the datasets can be found in [12]. In the first preprocessing step, I did the following things:

- extraction of travel times, days in rows, time stamps in columns
- extraction of the dates (i.e. yyyy/mm/dd)
- extraction of day of the week (i.e. monday, tuesday, wednesday, thursday, friday)
- replacement of travel times equal to or lower than zero with "NaN"

I then saved several different ASCII files, according to the scheme in table 4.2. The travel

Type	Filename	Dimension	Contents
Dates	'nan' + *name* + 'dates'	$n_1 \times 1$	yyyy/mm/dd
Days	'nan' + *name* + 'days'	$n_1 \times 1$	$\{1, 2, 3, 4, 5\}$
Data	'nan' + *name*	$n_1 \times 720$	travel times
Data	'nan' + *name* + 'replacedvalues'	$n_1 \times 720$	original data
Data	'nan' + *name* + $\{mon, tue, wed, thu, fri\}$	$n_2 \times 720$	travel times per day

Table 4.2: File Contents

times in these files are now used for the rest of the experiments. All data is in a good readable format now.

The following overall results could be made from this descriptive analysis:

- The minimum value of the average travel times in each graph can be seen as an indication of a travel time under circumstances without delay.

- The afternoon peak hour is the most intense on Fridays and also starts much earlier on Fridays than on other days.
- The morning peak hours are the most intense on Mondays and Tuesdays, less on Wednesdays and Thursdays and on Friday it has the least influence on travel time.
- One can 'mirror' the plot of *heen* over the plot of *terug* and get almost the same values of travel times

Conclusions R04 (Eindhoven - Tilburg v.v.):

- Friday afternoon peak hour starts extreme early

Conclusions R09 (Utrecht - Beesd v.v.):

- After 22:00h also a peak in travel times on R09terug (road maintenance?)

Conclusions R19 (Den Haag - Gouda v.v.):

- Afternoon peak has to extreme points instead of one (one at 16:00h and one at 18:30h)

Conclusions R32 (Amsterdam - Purmerend v.v.):

- No real afternoon peak, only high travel times from noon until 20:00h towards Purmerend.

4.2.2 Missing values

The NaN-entries in the dataset are fine when just looking at statistics, but when we want to do good clustering, these values are not understood. Therefore, we have to make a choice what to do with them. There are several ways to imputate NaN entries, I will describe the ways I tested.

- Complete deletion of NaN
- Replacement of NaN

In the first case, I will completely delete a day where we encounter NaN-entries. This means that if one measurement is NaN, then the cmplete day will be removed from the dataset. The advantage of this way is that in no manner, the structure of a travel time profile is changed by artificial values. But the disadvantage is that the complete dataset is reduced dramatically if there exist lots of NaN entries over different days.

In the latter case, all values are replaced by an imputated value. A simple linear interpolation was not an option, because in lots of cases the beginning and end of a complete record in the dataset were also missing. Therefore, I wrote an algorithm with the following steps. The exact implementation of this algorithm can be found at page A in appendix A.

1. If values at the beginning of a record are missing, I first extrapolated these values
2. If values at the end of a record are missing, I then extrapolated these values
3. Finally, I interpolated the values in the rest of the record, with only the original data as input

You can see the used algorithms in appendix A on page A-4. This way, I am sure that the new record (travel time profile of 1 day) is only different from the original one at the datapoints where we had missing values, i.e. NaN's.

4.3 Statistics

In section 3.3, I explained how one can obtain a proper dataset. But what information is already present in a dataset? Of course, gaining insight in the structure of a dataset could already help us understand and qualify outcomes of a clustering algorithm. Or even better, it could help us understand outcomes of any algorithm that claims to extract usefull information on this dataset.

Since we are dealing with the problem of data reduction in travel time series, let us keep in mind that the least we need is a dataset with travel time profiles, as desribed below.

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,t} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,t} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n,1} & y_{n,2} & \cdots & y_{n,t} \end{bmatrix} \quad \text{with} \quad \begin{array}{ll} y_{d,t} \in [0, \infty) & \text{travel time} \\ d \in \{0, 1, \ldots, n\} & \text{day} \\ t \in \{0, 1, \ldots, t\} & \text{time interval index} \end{array}$$

One way to gain insight in a dataset, is to look at basic descriptive statistics, like the mean and median of a time interval, over all available days in one year. This gives a feeling of how an average travel time profile looks like for a given link on a complete day (24 hours). However, mean and median values do not say anything about the variation of the values. Therefore, additional statistics like the 15^{th} and 85^{th} pecentile should give an idea of this variation. The 15^{th} percentile is a good indication of the travel time on a "bad" day. In 85% of the cases, one will be faster than this value. The 85^{th} percentile is a good indication for the travel time on a "good" day. In only 15% of the time, one has a travel time longer than this value. In figure 4.1, you can see a sample graph of the statistics of some dataset.

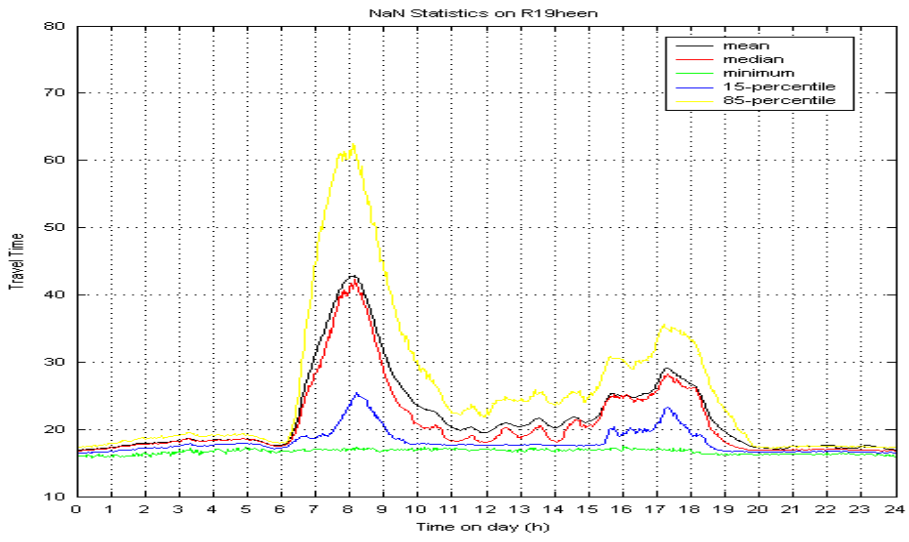

Figure 4.1: Sample statistics

At normal circumstances, every dataset contains some minimum travel time of that

specific traject, which could depend on speed limits or safety insights of the driver. For example, it is practically impossible to drive over a traject with unlimited speed. These speed limits can either be structural (indicated with speed limit signs), or the technical speed limit of the vehicle. Therefore, there must exist some minimum value for each (part of) a traject, which is an interesting value.

Using the filtered dataset, we see that the statistics do not differ a lot from the original statistics. This is enough reason to assume that a data reduction technique will give the same result with either a dataset with imputated values, or with missing values replaced bij NaN-entries. In this research, the dataset with imputated values was used, because the algorithms do not handle NaN-entries.

4.3.1 Mean and medians

As described above, there are two ways to give an indication of an average travel time profile on a certain traject. In [12], the authors discuss if they should use the median or the mean to give an approximation of the measured speed. The same discussion holds for the statistics over a complete year. The mean is not robust to outliers, because an outlier above the "normal" travel time can never be corrected by an outlier in the oppostite side: there exists a minimum travel time, which is a lower bound. A better choice would be the median, because in most cases, it is not far from the mean and is robust against outliers. But what we see with the median is that no single outlier is taken into account, also the structural ones. If one thinks that an outlier is an accident that should not be taken into account on qualifying a traject, than the median is the better option. However, if an outlier is also a structural indication of performance, one should definetely take them into account and either judge them separately or include them in the overall statistics. In case of an accident, this is no problem, but in case of short and heavy congestion due to traffic supply, this is very important. I agree with the authors that the mean is then a better value to give an indication of an average travel time profile if one wants to use it for prediction purposes.

In appendix B, you find several figures of each dataset used. First, you see a figure with the named statistics on a specific traject, where missing values are deleted. Below, two figures are displayed with the same structure, but now the missing values are either replaced bij NaN's or imputated as described above. This is done for all the available datasets.

Furthermore, in appendix C, you find a similar set of figures, but now only the mean per type of weekday. This means that I computed the mean per day-of-the-week, and then displayed these values in a new graph. This is also done for either deleted values, replaced by NaN entries, and imputated values.

After applying the described imputation methods, it can be concluded that only the trajects R32_heen and R32_terug have different statistical results. I chose not to include these trajects in the rest of the research.

4.4 Travel Time Profile as a Function

In this section, I try to find a way to describe the travel time profiles as a function. The theory is explained in the previous chapter, and the standard functions within the package MatLab are used: *polyfit* finds the coefficients of a polynomial $p(x)$ of degree n that fits the data, $p(x(i))$ to $y(i)$, in a least squares sense. The result p is a row vector of length $n+1$ containing the polynomial coefficients in descending powers.

4.4.1 Polynomial function

As seen from the figures in Appendix B, one can guess an average day could be described by some degree polynomial function with a maximum at the peak-hours in the morning and one at the peak hours in the evening. I tried an eight-degree polynomial fit on the 10 figures of sample mean (with a normalized domain, in stead of $[0, 2, \ldots, 720]$). The value eight is chosen after a comparison of the results over the available datasets with larger or smaller coefficients. It seems that the best fit is accomplished with seven local extremes in the polynomial, which implies a degree of eight. In Table 4.3, you can find the coefficients for the polynomial function $P(x)$, where x is the (normalized) highest value of a time interval.

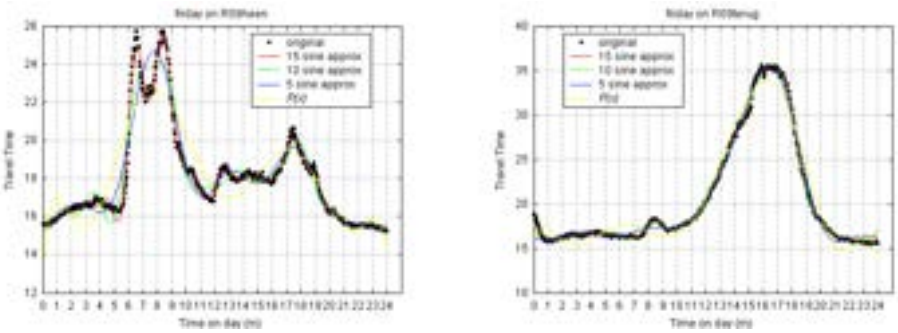

Figure 4.2: DFT approximation on means for Utrecht - Beesd v.v.

In figure 4.2 you can see two graphs for the polynomial function $P(x)$. The right graph seems to have a good fit, however, in the left graph, you can see that $P(x)$ is not even near the original data around for example 8:00h a.m. With most of the results for the polynomial fit this is the case, and the polynomial fit can not be seen as a good solution for data reduction.

The rest of the results are plotted in Appendix D (indicated in the legend by $P(x)$).

4.4.2 Fourier Series

We define the interval we look at as $[0, T]$, where T is the end of the day (i.e. 24 hours, or 1440 minutes). So, with Fourier Series, we are interested in expressing the travel times as a sum of weighted sinusoids (each with a different frequency). Our goal is to give a good representation of a travel time profile, with as small as possible parameters.

	p_8	p_7	p_6	p_5	p_4
R04heen	-0.17484	-0.036457	1.2861	0.79375	-3.1657
R04terug	-1.4854	0.10485	9.2003	0.10086	-17.941
R09heen	-6.1869	2.9695	38.365	-18.487	-75.221
R09terug	-4.4513	-0.4237	28.468	5.2019	-58.041
R10heen	-3.2656	1.5447	20.066	-8.885	-39.336
R10terug	-3.4187	0.23118	22.113	1.1918	-46.927
R19heen	-6.5522	3.1676	40.387	-17.504	-78.476
R19terug	-4.2281	0.5277	26.911	-0.78787	-55.036
R32heen	-6.2161	2.4732	37.849	-15.913	-72.431
R32terug	-0.59591	-0.47034	3.5606	3.5329	-6.0441
	p_3	p_2	p_1	p_0	std$(v(x))$
R04heen	-2.535	2.2408	1.6667	16.508	0.40116
R04terug	-1.6605	9.8515	1.5978	17.149	1.1203
R09heen	36.756	45.14	-23.077	17.831	2.9095
R09terug	-16.149	37.645	15.045	16.819	2.3437
R10heen	16.169	24.199	-9.4379	20.248	1.9365
R10terug	-9.0023	33.274	11.15	19.737	2.9813
R19heen	29.364	45.84	-14.335	20.513	3.3453
R19terug	-4.2142	36.204	6.7749	17.178	2.2987
R32heen	32.578	41.915	-20.889	21.169	3.8631
R32terug	-8.2597	1.1334	6.1663	21.385	1.4507

$$P(x) = p_8x^8 + p_7x^7 + p_6x^6 + p_5x^5 + p_4x^4 + p_3x^3 + p_2x^2 + p_1x + p_0$$
$$v(x) = P(x) - \text{mean}(x)$$

Table 4.3: Coefficients for estimated polynomial function on travel times

In Appendix D, the graphs of a Discrete Fourier Series are shown for the given links in the network. Summations of 5, 10 and 15 sinusoids are plotted in the graphs (indicated in the legend by xx_sine_approx). The data that is used to produce these graphs is the dataset that is filtered by day-of-the-week, as described in section 4.3.

In figure 4.2 you can see an example of the Fourier Series result. The three plotted Fourier Series give a fair representation of the travel time profile. Of coure, the best result can be achieved with increasing the number of sinusoids (or increasing the degree of the polynom), but with 10 sinusoids a fair result is plotted.

The approximation of a travel time profile with an 8th degree polynomial function does not give accurate results. The choice of a higher degree could be interesting, but the high increase in parameters will make the approach less usefull, because the results for the sum of 10 sine or cosine functions give better results as can be seen from the plotted figures.

4.5 Clustering

Considering the discussion in section 3.5.3, here you wil find the graphical results of the fuzzy c-means clustering method. The code can be viewed in Appendix A. I applied the

algorithm on the R04_terug, R09_terug, R10_heen and R19_heen datasets with 3, 4, 5 and 6 clusters. The NaN entries can not be handled by the algorithm, and are therefore not used. The datasets where incomplete columns are deleted have almost the same statistical results as the imputeded dataset, and therefore I only applied the algorithm to datasets with imputated values.

In the TNO research, there exists a strong feeling that the weekdays are the most important factor on finding different clusters in the available data. Therefore, the comparison is made between the centroids of the found clusters and the mean weekday values.

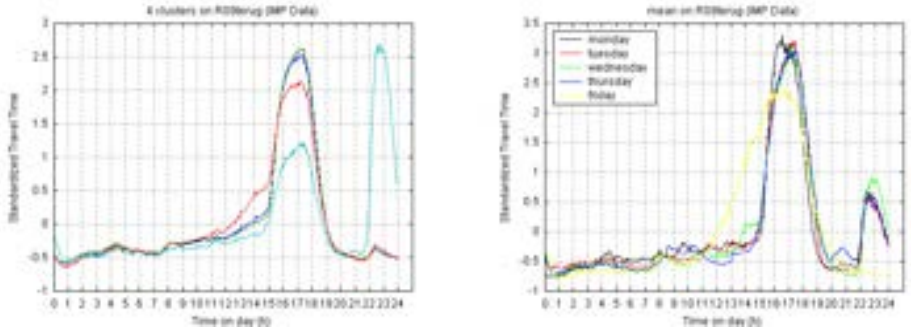

Figure 4.3: 4 cluster centroids and mean values per day-of-the-week on R09terug

In the example in figure 4.3, you can see a plot of the 4 cluster centroids in the left graph, and the plot of the mean values, per day-of-the-week. We can here see the clustering of values that weren't seen in the mean values. Around 22:30h, you see a very high value of one specific centroid. In the dataset there were enough outliers to assign them to a specific cluster, but not enough to have a large enough influence on the mean.

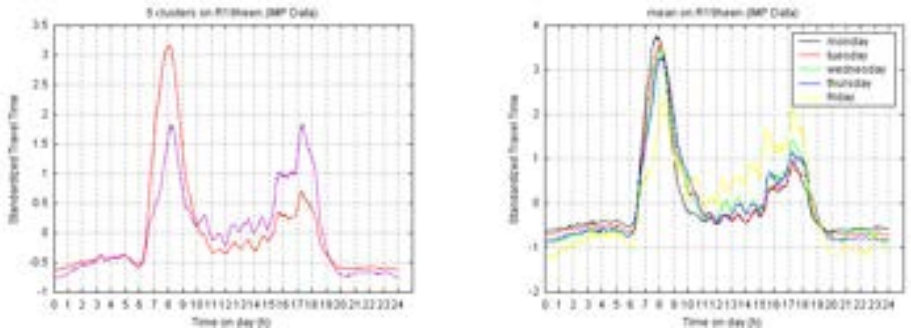

Figure 4.4: 4 cluster centroids and mean values per day-of-the-week on R19heen

Furthermore, in figure 4.4 we see that the search for 5 (and also 6) clusters, that 4 cluster centroids seem to be the same. It seems that only 2 clusters have been found, but there are 5 centroids. This means, with this particular dataset, a method for dynamically finding the optimal amount of clusters could solve this problem.

The clusters do not seem to reflect the assumption that the day-of-the-week is of large influence on the cluster centroids.

In the figures E.1, E.2, E.3 and E.4 in appendix E, you can see the clustering algorithm applied to the normalized imputated travel times dataset. The first thing we can see in the figures is the more clusters we try to find, the better values the centroids will have. All normalized travel times in the cluster algorithm are within the interval of the normalized mean travel times. This is no strange behaviour because all data is fitted within the cluster centroids, so the perfect clustering is only found when the number of clusters are the same as the number of datapoints in the set.

4.6 Interpretation

The results from the 4.4 and 4.5 can be interpreted in the following way.

Both the Polynomial fit als the Fourier Series are both regression methods. This means that on a specific link or route, the function parameters are used to calculate the travel time in stead of using the historical travel time series. The Fourier series analysis gives fair results with 10 of 15 sinusoids in comparison with the polynomial fit. If any development can be made to this algorithm, it might be possible to either dynamically find the optimal amount of sinusoids in such way that computation time is the lowest possible (the data is reduced as far as possible). Or to work with some moving horizon in the coefficients of the sinusoids. Begin with an initial, and try to adjust them as time progresses.

If one has found a good clustering technique for a given dataset, the prediction (with for example the framework mentioned in chapter 2) process could be applied. As a consequence that there now exists an initial clustering of the original dataset, one is able to check whether the available prediction methods in the framework really give distinctive predictions. If different predictions continuously end up in the same cluster, then at least one of those prediction methods is not needed. Also, when we search for similar time series given one on a subinterval, this is nothing more than finding the right cluster for this time series. Just like we did with first finding the clusters, we now have to assign a profile to the right cluster.

Chapter 5

Conclusions

From the external factors, discussed in this study, it can be concluded that the framework should try to include any influence that these factors should have. Events and road maintenance are factors that in most cases are well known in advance, but season influences and weather conditions are known with higher incertainty. These are also based on predictions.

The data reduction techniques in this study should give a fair dataset to be used with the framework developed at TNO Inro. The available data for this study was restricted to a dataset for five highways in the Netherlands. Every other dataset could be used, and a good analysis of the available information gives insight in the structure and data type.

The data filtering techniques give insight in the dataset. What are the peak hours, where are missing data, and what data should be used when giving a representation. All statistics and plotted graphs are not that important for the reduction techniques itself, but do however show where to expect interesting results with reduction of datasets, and furthermore, using these reduced datasets for prediction purposes in the framework.

Inconsistencies in the used datasets, for example outliers or missing values, are dealt with by using inter- and extrapolation. The dataset where records with missing values are completely deleted give a wrong representation of the data, and therefore the dataset with imputated values should be used for the named reduction techniques.

From the used data reduction techniques, the polynomial fit has poor results in comparison to the Fouries Series. So if one is to use some regression method, this signal interpretation on a travel time profile should give better results, as in this study, a fair representation of a travel time profile is shown in the results.

The Fuzzy C-means cluster technique, as applied in our experiments, show different interpretation of results for the various datasets. Whereas in some datasets, the hypothesis that the day-of-the-week should be of influence on the travel time is accepted, on other datasets it is rejected. However, overall conclusion can be made that a fuzzy clustering technique for data reduction method has promising results. A more thorough analysis of the centroids, and in this case a validation technique for the clustering, is be needed to make conclusions if all used techniques are applicabable for use in the prediction framework.

5.1 Future Research

The reduction techniques based on regression (polynomial and fourier series) are only compared with a dataset based on mean values. Therefore, to give a good representation, a comparison between the to methods itself should give better insight in what degree to use, of how many sinusoids to use. A good idea could be to find some validation method, to either compare within a reduction technique, or between different reduction techniques.

The fuzzy C-Means clustering algorithm works by assigning a datapoint to a certain cluster, and not crisp, but fuzzy. I have tried to fit a fixed number of clusters, but there is enough to find in the literature about dynamically assigning the amount of clusters. Another research followed by this study could be to find explanatory states for each cluster, for example, "this cluster has travel time profiles, where in 85% of the cases, it was a monday outside of school holidays".

As long as computers work faster, it should be better and better possible to use large datasets in the same amount of computational time. That would conclude that is study is useless. However, the number of ways to collect data (and also the amount of available data) for travel times increases radiply. One can think about tracking systems through GPS satellites: It is already possible to register the exact travel time for a very precise distance or route (like a black box in an airplane). And, this amount of data will also increase in time. Therefore data reduction techniques will probably always be needed, to make sure

References

[1] Daily values of temperature, sunshine, cloud cover and visibility, air pressure, precipitation and wind, 2000 - 2003. Freely available from http://www.knmi.nl/voorl/kd/. 13, 14

[2] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag. 22

[3] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Knowledge Discovery and Data Mining*, pages 16–22, 1998. 24

[4] S. De Backer, A. Naud, and P. Scheunders. Non-linear dimensionality reduction techniques for unsupervised feature extraction. *Pattern Recognition Letter*, 19:711–720, 1998. 22

[5] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Data mining the stock market: Cluster discovery. In *Sixth ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, pages 487–496, 2000. 22, 23, 24

[6] R. L. Herman. An introduction to fourier and complex analysis with applications to the spectral analysis of signals. pages 169–171, 2006. A-27

[7] M. L. Hetland. A survey of recent methods for efficient retrieval of similar time sequences, 2001. 20, 22, 24

[8] E. Keogh and S. Kasetty. The need for time series data mining benchmarks: A survey and empirical demonstration. In *ACM SIGKDD*, 2002. 19, 20, 22, 24

[9] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 239–241, New York City, NY, 1998. ACM Press. 23, 24

[10] M. Last, Y. Klein, and A. Kandel. Knowledge discovery in time series databases, 2001. 22, 24

[11] E. K. Michail Vlachos, Jessica Lin and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series, 2003. 22, 24

[12] A. Oostveen and J. V. Toorenburg. Verdeling trajectsnelheden. Technical report, Transpute BV, Gouda, november 2002. 27, 28, 31

[13] K. pong Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999. 22, 24

[14] D. Rafiei. Fourier-transform based techniques in efficient retrieval of similar time sequences, 1998. 22

[15] H. Versteegt and C. Tampère. PredicTime: State of the art and functional architecture. Technical Report 03-7N-024-73196, TNO Inro: Institute for Traffic and Transport, Logistics and Spatial Development, February 2003. 4, 7

[16] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories, 2002. 24

[17] E. Weisstein. Fourier series. mathworld.wolfram.com. 23

[18] E. Weisstein. Least squares fitting – polynomial. mathworld.wolfram.com. 23

Appendix A

Source Codes

at_avv_preprocess.m

```
close all;
clear all;
clc;

% original files with travel times, days in columns and time-intervals on rows
filenames = {'04heen','04terug','09heen','09terug','10heen',...
        '10terug','19heen','19terug','32heen','32terug'};
% number of measured data on 1 day (24 hours)
dimension = 720;
% prefix that should be added to the filenames for saving new data
prefix_nan = 'nan_';                    % days with missing values are replaced by NaN's

for i=1:size(filenames,2)
  % Find out dmax (the number of columns) in the data files. Only applicabable to AVV dataset!
  fprintf(1,'Opening %s\n',filenames{i});
  file = strcat('result',filenames{i},'.tsv');
  [pathstr,name,ext,versn] = fileparts(file);
  fid = fopen(strcat(name,ext),'r');    % open file i
  tline = fgetl(fid);                   % read first line
  name = strrep(name,'result','R');     % replace "result" by "R" in filenames
  matches = findstr(tline, name);       % find indices of occurences in first line
  dmax = length(matches);               % and assign this as the number of columns

  % Load the dates where we have travel times of (1 x dmax matrix)
  fprintf(1,'Reading dates');
  dates_nan = dlmread(file,'\t',[7 1 7 dmax]);

  % Load the travel times itself, 2 min. intervals (dimension x dmax matrix)
  fprintf(1,', data');
  data_nan = dlmread(file,'\t',[700 1 (700+(dimension-1)) dmax]);
  % lowest traveltime per day (1 x dmax matrix)
  datamin = min(data_nan);

  % Load the days where we have travel times of (1 x dmax matrix)
  fprintf(1,', days.\n');
  for l=0:7
    tline = fgetl(fid);                 % go to the line with the days
  end
  days = {};
  [versn,tline] = strtok(tline);
  for l=1:dmax
```

```
  [days{l},tline] = strtok(tline);
end
mon = strmatch('ma',days);
tue = strmatch('di',days);
wed = strmatch('wo',days);
thu = strmatch('do',days);
fri = strmatch('vr',days);
clear days;

days_nan = ones(1,dmax);
for l=1:size(mon,1)
  days_nan(mon(l)) = 1;
end
for l=1:size(tue,1)
  days_nan(tue(l)) = 2;
end
for l=1:size(wed,1)
  days_nan(wed(l)) = 3;
end
for l=1:size(thu,1)
  days_nan(thu(l)) = 4;
end
for l=1:size(fri,1)
  days_nan(fri(l)) = 5;
end

fclose(fid);
clear versn pathstr tline matches fid;

nan_value = ones(dimension,dmax);
fprintf(1,'Finding NaN''s...\n');
l = 0;
for m=1:dmax                          % For each day
  for n=1:dimension                   % For each time-interval
    if data_nan(n,m) <= 0             % If travel time <= 0
      nan_value(n,m) = data_nan(n,m); % Backup original data
      data_nan(n,m) = NaN;            % Replace time-value with NaN-value
    end
  end
end

% file with the dates, 1 column
fid_dates_nan = fopen(strcat(prefix_nan,name,'_dates','.txt'),'w');
% file with the days, 1 column
fid_days_nan = fopen(strcat(prefix_nan,name,'_days','.txt'),'w');

% first, write the dates and days to the files (array of NaN items)
fprintf(1,'Writing dates and days for NaN array...\n');
for n=1:size(dates_nan,2);
  jaar = floor(dates_nan(n)/10000);
  maand = floor((dates_nan(n) - (jaar*10000))/100);
  dag = floor(dates_nan(n) - (maand*100) - (jaar*10000));
  if maand < 10
    maand = strcat('0',int2str(maand));
  else
    maand = int2str(maand);
  end
  if dag < 10
    dag = strcat('0',int2str(dag));
  else
    dag = int2str(dag);
  end
```

```
  dates_nan_new = strcat(int2str(jaar), '/', maand, '/', dag);
  fprintf(fid_dates_nan,'%s\n',dates_nan_new);     % day per row
  fprintf(fid_days_nan,'%1.0f\n',days_nan(n));     % day per row
  clear jaar maand dag dates_nan_new;
end
fclose(fid_dates_nan);
fclose(fid_days_nan);

% transpose the matrices for external prediction methods (the days are listed on rows)
data_nan = data_nan';
nan_value = nan_value';

% file with the travel-times on those days, time-intervals in columns and days on rows
fid_data_nan = fopen(strcat(prefix_nan,name,'.txt'),'w');
fid_data_nan_replaced = fopen(strcat(prefix_nan,name,'_replacedvalues.txt'),'w');

fprintf(1,'Writing data for NaN array PER WEEKDAY...\n');

fid_data_mon = fopen(strcat(prefix_nan,name,'_mon.txt'),'w');
fid_data_tue = fopen(strcat(prefix_nan,name,'_tue.txt'),'w');
fid_data_wed = fopen(strcat(prefix_nan,name,'_wed.txt'),'w');
fid_data_thu = fopen(strcat(prefix_nan,name,'_thu.txt'),'w');
fid_data_fri = fopen(strcat(prefix_nan,name,'_fri.txt'),'w');

fprintf(1,'\t%s\n',strcat(prefix_nan,name,'_mon.txt'));
for k=1:size(mon,1)
  fprintf(fid_data_mon,'%2.2f',data_nan(mon(k,1),1));
  for j=2:720
      fprintf(fid_data_mon,'\t%2.2f',data_nan(mon(k,1),j));
  end
  fprintf(fid_data_mon,'\n');                  % linebreak for a new day
end
fprintf(1,'\t%s\n',strcat(prefix_nan,name,'_tue.txt'));
for k=1:size(tue,1)
  fprintf(fid_data_tue,'%2.2f',data_nan(tue(k,1),1));
  for j=2:720
      fprintf(fid_data_tue,'\t%2.2f',data_nan(tue(k,1),j));
  end
  fprintf(fid_data_tue,'\n');                  % linebreak for a new day
end
fprintf(1,'\t%s\n',strcat(prefix_nan,name,'_wed.txt'));
for k=1:size(wed,1)
  fprintf(fid_data_wed,'%2.2f',data_nan(wed(k,1),1));
  for j=2:720
      fprintf(fid_data_wed,'\t%2.2f',data_nan(wed(k,1),j));
  end
  fprintf(fid_data_wed,'\n');                  % linebreak for a new day
end
fprintf(1,'\t%s\n',strcat(prefix_nan,name,'_thu.txt'));
for k=1:size(thu,1)
  fprintf(fid_data_thu,'%2.2f',data_nan(thu(k,1),1));
  for j=2:720
      fprintf(fid_data_thu,'\t%2.2f',data_nan(thu(k,1),j));
  end
  fprintf(fid_data_thu,'\n');                  % linebreak for a new day
end
fprintf(1,'\t%s\n',strcat(prefix_nan,name,'_fri.txt'));
for k=1:size(fri,1)
  fprintf(fid_data_fri,'%2.2f',data_nan(fri(k,1),1));
  for j=2:720
      fprintf(fid_data_fri,'\t%2.2f',data_nan(fri(k,1),j));
  end
```

```
    fprintf(fid_data_fri,'\n');                 % linebreak for a new day
  end

  fclose(fid_data_mon);
  fclose(fid_data_tue);
  fclose(fid_data_wed);
  fclose(fid_data_thu);
  fclose(fid_data_fri);

  clear mon tue wed thu fri;
  clear fid_data_mon fid_data_tue fid_data_wed fid_data_thu fid_data_fri;

  % write data to new output-file
  fprintf(1,'Writing data for NaN array WHOLE DATASET...\n');
  for m=1:size(data_nan,1)
    fprintf(fid_data_nan,'%2.2f',data_nan(m,1));
    fprintf(fid_data_nan_replaced,'%2.2f',nan_value(m,1));
    for n=2:size(data_nan,2)
      fprintf(fid_data_nan,'\t%2.2f',data_nan(m,n));
      fprintf(fid_data_nan_replaced,'\t%2.2f',nan_value(m,n));
    end
    fprintf(fid_data_nan,'\n');               % linebreak for a new day
    fprintf(fid_data_nan_replaced,'\n');      % linebreak for a new day
  end
  fclose(fid_data_nan);
  fclose(fid_data_nan_replaced);

  fprintf(1,'Closing file %s.\n\n',strcat(prefix_nan,name,'.txt'));
  clear days_nan dates_nan data_nan nan_value dmax datamin;
end
```

at_statistics_imputation.m

```
close all;
clear all;
clc;

% files with the travel times, days in columns and time-intervals on rows
filenames = {'04heen','04terug','09heen','09terug','10heen',...
             '10terug','19heen','19terug','32heen','32terug'};
prefix_del = 'del_R';    % days with missing values completely deleted
prefix_nan = 'nan_R';    % timestamps with missing values are replaced by NaN's
prefix_imp = 'imp_R';    % timestamps with missing values are replaced by Imputated value
dimension  = 720;
method = 'nearest';

for i=1:size(filenames,2)
  % Load the travel times itself, 2 minutes intervals
  fprintf(1,'Opening R%s\n',filenames{i});
  [pathstr,name,ext,versn] = fileparts(strcat(filenames{i},'.txt'));
  clear versn pathstr;

  data_nan = dlmread(strcat(prefix_nan,name,ext),'\t');
  mon_nan = dlmread(strcat(prefix_nan,name,'_mon',ext),'\t');
  tue_nan = dlmread(strcat(prefix_nan,name,'_tue',ext),'\t');
  wed_nan = dlmread(strcat(prefix_nan,name,'_wed',ext),'\t');
  thu_nan = dlmread(strcat(prefix_nan,name,'_thu',ext),'\t');
  fri_nan = dlmread(strcat(prefix_nan,name,'_fri',ext),'\t');

  nans     = isnan(data_nan);
```

```
nans_mon = isnan(mon_nan);
nans_tue = isnan(tue_nan);
nans_wed = isnan(wed_nan);
nans_thu = isnan(thu_nan);
nans_fri = isnan(fri_nan);

data_del = data_nan;
mon_del = mon_nan;
tue_del = tue_nan;
wed_del = wed_nan;
thu_del = thu_nan;
fri_del = fri_nan;

data_imp = data_nan;
mon_imp = mon_nan;
tue_imp = tue_nan;
wed_imp = wed_nan;
thu_imp = thu_nan;
fri_imp = fri_nan;

fprintf(1,'Deleting NaN values and store in new matrix');
fprintf(1,'.');
l_all = 0;
for n=1:size(data_nan,1)      % For each time-interval
  if nansum(nans(n,:)) > 0        % If travel time == NaN
    data_del((n - l_all),:) = [];        % Delete the date from dates_del
    l_all = l_all + 1;                    % increase # of removed days
  end
end
fprintf(1,'.');
l_mon = 0;
for n=1:size(mon_nan,1)      % For each time-interval
  if nansum(nans_mon(n,:)) > 0        % If travel time == NaN
    mon_del((n - l_mon),:) = [];        % Delete the date from dates_del
    l_mon = l_mon + 1;                    % increase # of removed days
  end
end
fprintf(1,'.');
l_tue = 0;
for n=1:size(tue_nan,1)      % For each time-interval
  if nansum(nans_tue(n,:)) > 0        % If travel time == NaN
    tue_del((n - l_tue),:) = [];        % Delete the date from dates_del
    l_tue = l_tue + 1;                    % increase # of removed days
  end
end
fprintf(1,'.');
l_wed = 0;
for n=1:size(wed_nan,1)      % For each time-interval
  if nansum(nans_wed(n,:)) > 0        % If travel time == NaN
    wed_del((n - l_wed),:) = [];        % Delete the date from dates_del
    l_wed = l_wed + 1;                    % increase # of removed days
  end
end
fprintf(1,'.');
l_thu = 0;
for n=1:size(thu_nan,1)      % For each time-interval
  if nansum(nans_thu(n,:)) > 0        % If travel time == NaN
    thu_del((n - l_thu),:) = [];        % Delete the date from dates_del
    l_thu = l_thu + 1;                    % increase # of removed days
  end
end
fprintf(1,'.\n');
```

```
l_fri = 0;
for n=1:size(fri_nan,1)      % For each time-interval
  if nansum(nans_fri(n,:)) > 0        % If travel time == NaN
    fri_del((n - l_fri),:) = [];        % Delete the date from dates_del
    l_fri = l_fri + 1;                    % increase # of removed days
  end
end

fprintf(1,'Saving new matrix (deleted values).');
write2file(strcat(prefix_del,name,ext),data_del);
fprintf(1,'.');
write2file(strcat(prefix_del,name,'_mon',ext),mon_del);
fprintf(1,'.');
write2file(strcat(prefix_del,name,'_tue',ext),tue_del);
fprintf(1,'.');
write2file(strcat(prefix_del,name,'_wed',ext),wed_del);
fprintf(1,'.');
write2file(strcat(prefix_del,name,'_thu',ext),thu_del);
fprintf(1,'.');
write2file(strcat(prefix_del,name,'_fri',ext),fri_del);
fprintf(1,'.\n');

fprintf(1,'Imputating NaN values and store in new matrix');
fprintf(1,'.');
for n=1:size(data_nan,1)      % For each time-interval
  if nansum(nans(n,:)) > 0        % If travel time == NaN
    data_imp(n,:) = imputeer(data_nan(n,:),method);    % replace the NaNs with interpolated value
  end
end
fprintf(1,'.');
for n=1:size(mon_nan,1)      % For each time-interval
  if nansum(nans_mon(n,:)) > 0        % If travel time == NaN
    mon_imp(n,:) = imputeer(mon_nan(n,:),method);      % replace the NaNs with interpolated value
  end
end
fprintf(1,'.');
for n=1:size(tue_nan,1)      % For each time-interval
  if nansum(nans_tue(n,:)) > 0        % If travel time == NaN
    tue_imp(n,:) = imputeer(tue_nan(n,:),method);      % replace the NaNs with interpolated value
  end
end
fprintf(1,'.');
for n=1:size(wed_nan,1)      % For each time-interval
  if nansum(nans_wed(n,:)) > 0        % If travel time == NaN
    wed_imp(n,:) = imputeer(wed_nan(n,:),method);      % replace the NaNs with interpolated value
  end
end
fprintf(1,'.');
for n=1:size(thu_nan,1)      % For each time-interval
  if nansum(nans_thu(n,:)) > 0        % If travel time == NaN
    thu_imp(n,:) = imputeer(thu_nan(n,:),method);      % replace the NaNs with interpolated value
  end
end
fprintf(1,'.\n');
for n=1:size(fri_nan,1)      % For each time-interval
  if nansum(nans_fri(n,:)) > 0        % If travel time == NaN
    fri_imp(n,:) = imputeer(fri_nan(n,:),method);      % replace the NaNs with interpolated value
  end
end

fprintf(1,'Saving new matrix (imputated values).');
write2file(strcat(prefix_imp,name,ext),data_imp);
```

```
fprintf(1,'.');
write2file(strcat(prefix_imp,name,'_mon',ext),mon_imp);
fprintf(1,'.');
write2file(strcat(prefix_imp,name,'_tue',ext),tue_imp);
fprintf(1,'.');
write2file(strcat(prefix_imp,name,'_wed',ext),wed_imp);
fprintf(1,'.');
write2file(strcat(prefix_imp,name,'_thu',ext),thu_imp);
fprintf(1,'.');
write2file(strcat(prefix_imp,name,'_fri',ext),fri_imp);
fprintf(1,'.\n');

fid_nan_stats     = fopen(strcat('stats_nan_R',name,'.txt'),'w');
fid_nan_stats_mon = fopen(strcat('stats_nan_R',name,'_mon.txt'),'w');
fid_nan_stats_tue = fopen(strcat('stats_nan_R',name,'_tue.txt'),'w');
fid_nan_stats_wed = fopen(strcat('stats_nan_R',name,'_wed.txt'),'w');
fid_nan_stats_thu = fopen(strcat('stats_nan_R',name,'_thu.txt'),'w');
fid_nan_stats_fri = fopen(strcat('stats_nan_R',name,'_fri.txt'),'w');
fid_del_stats     = fopen(strcat('stats_del_R',name,'.txt'),'w');
fid_del_stats_mon = fopen(strcat('stats_del_R',name,'_mon.txt'),'w');
fid_del_stats_tue = fopen(strcat('stats_del_R',name,'_tue.txt'),'w');
fid_del_stats_wed = fopen(strcat('stats_del_R',name,'_wed.txt'),'w');
fid_del_stats_thu = fopen(strcat('stats_del_R',name,'_thu.txt'),'w');
fid_del_stats_fri = fopen(strcat('stats_del_R',name,'_fri.txt'),'w');
fid_imp_stats     = fopen(strcat('stats_imp_R',name,'.txt'),'w');
fid_imp_stats_mon = fopen(strcat('stats_imp_R',name,'_mon.txt'),'w');
fid_imp_stats_tue = fopen(strcat('stats_imp_R',name,'_tue.txt'),'w');
fid_imp_stats_wed = fopen(strcat('stats_imp_R',name,'_wed.txt'),'w');
fid_imp_stats_thu = fopen(strcat('stats_imp_R',name,'_thu.txt'),'w');
fid_imp_stats_fri = fopen(strcat('stats_imp_R',name,'_fri.txt'),'w');
fprintf(fid_nan_stats,    'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_nan_stats_mon,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_nan_stats_tue,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_nan_stats_wed,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_nan_stats_thu,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_nan_stats_fri,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats,    'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats_mon,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats_tue,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats_wed,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats_thu,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_del_stats_fri,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats,    'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats_mon,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats_tue,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats_wed,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats_thu,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');
fprintf(fid_imp_stats_fri,'date\tmean\tmedian\tmin\t15perc\t85perc\tstddev\n');

fprintf(1,'Computing statistics...\n');
for j=1:dimension
  nan_stats(j,:) = [nanmean(data_nan(:,j)) nanmedian(data_nan(:,j)) nanmin(data_nan(:,j)) ...
                prctile(data_nan(:,j),[15 85]) nanstd(data_nan(:,j))];
  nan_stats_mon(j,:) = [nanmean(mon_nan(:,j)) nanmedian(mon_nan(:,j)) nanmin(mon_nan(:,j)) ...
                prctile(mon_nan(:,j),[15 85]) nanstd(mon_nan(:,j))];
  nan_stats_tue(j,:) = [nanmean(tue_nan(:,j)) nanmedian(tue_nan(:,j)) nanmin(tue_nan(:,j)) ...
                prctile(tue_nan(:,j),[15 85]) nanstd(tue_nan(:,j))];
  nan_stats_wed(j,:) = [nanmean(wed_nan(:,j)) nanmedian(wed_nan(:,j)) nanmin(wed_nan(:,j)) ...
                prctile(wed_nan(:,j),[15 85]) nanstd(wed_nan(:,j))];
  nan_stats_thu(j,:) = [nanmean(thu_nan(:,j)) nanmedian(thu_nan(:,j)) nanmin(thu_nan(:,j)) ...
                prctile(thu_nan(:,j),[15 85]) nanstd(thu_nan(:,j))];
  nan_stats_fri(j,:) = [nanmean(fri_nan(:,j)) nanmedian(fri_nan(:,j)) nanmin(fri_nan(:,j)) ...
```

```
              prctile(fri_nan(:,j),[15 85]) nanstd(fri_nan(:,j))];
del_stats(j,:) = [mean(data_del(:,j)) median(data_del(:,j)) min(data_del(:,j)) ...
              prctile(data_del(:,j),[15 85]) std(data_del(:,j))];
del_stats_mon(j,:) = [mean(mon_del(:,j)) median(mon_del(:,j)) min(mon_del(:,j)) ...
              prctile(mon_del(:,j),[15 85]) std(mon_del(:,j))];
del_stats_tue(j,:) = [mean(tue_del(:,j)) median(tue_del(:,j)) min(tue_del(:,j)) ...
              prctile(tue_del(:,j),[15 85]) std(tue_del(:,j))];
del_stats_wed(j,:) = [mean(wed_del(:,j)) median(wed_del(:,j)) min(wed_del(:,j)) ...
              prctile(wed_del(:,j),[15 85]) std(wed_del(:,j))];
del_stats_thu(j,:) = [mean(thu_del(:,j)) median(thu_del(:,j)) min(thu_del(:,j)) ...
              prctile(thu_del(:,j),[15 85]) std(thu_del(:,j))];
del_stats_fri(j,:) = [mean(fri_del(:,j)) median(fri_del(:,j)) min(fri_del(:,j)) ...
              prctile(fri_del(:,j),[15 85]) std(fri_del(:,j))];
imp_stats(j,:) = [mean(data_imp(:,j)) median(data_imp(:,j)) min(data_imp(:,j)) ...
              prctile(data_imp(:,j),[15 85]) std(data_imp(:,j))];
imp_stats_mon(j,:) = [mean(mon_imp(:,j)) median(mon_imp(:,j)) min(mon_imp(:,j)) ...
              prctile(mon_imp(:,j),[15 85]) std(mon_imp(:,j))];
imp_stats_tue(j,:) = [mean(tue_imp(:,j)) median(tue_imp(:,j)) min(tue_imp(:,j)) ...
              prctile(tue_imp(:,j),[15 85]) std(tue_imp(:,j))];
imp_stats_wed(j,:) = [mean(wed_imp(:,j)) median(wed_imp(:,j)) min(wed_imp(:,j)) ...
              prctile(wed_imp(:,j),[15 85]) std(wed_imp(:,j))];
imp_stats_thu(j,:) = [mean(thu_imp(:,j)) median(thu_imp(:,j)) min(thu_imp(:,j)) ...
              prctile(thu_imp(:,j),[15 85]) std(thu_imp(:,j))];
imp_stats_fri(j,:) = [mean(fri_imp(:,j)) median(fri_imp(:,j)) min(fri_imp(:,j)) ...
              prctile(fri_imp(:,j),[15 85]) std(fri_imp(:,j))];
time = strcat(int2str((2*j)-2),'-',int2str(2*j));
fprintf(fid_nan_stats,    '%s\t',time);
fprintf(fid_nan_stats_mon,'%s\t',time);
fprintf(fid_nan_stats_tue,'%s\t',time);
fprintf(fid_nan_stats_wed,'%s\t',time);
fprintf(fid_nan_stats_thu,'%s\t',time);
fprintf(fid_nan_stats_fri,'%s\t',time);
fprintf(fid_del_stats,    '%s\t',time);
fprintf(fid_del_stats_mon,'%s\t',time);
fprintf(fid_del_stats_tue,'%s\t',time);
fprintf(fid_del_stats_wed,'%s\t',time);
fprintf(fid_del_stats_thu,'%s\t',time);
fprintf(fid_del_stats_fri,'%s\t',time);
fprintf(fid_imp_stats,    '%s\t',time);
fprintf(fid_imp_stats_mon,'%s\t',time);
fprintf(fid_imp_stats_tue,'%s\t',time);
fprintf(fid_imp_stats_wed,'%s\t',time);
fprintf(fid_imp_stats_thu,'%s\t',time);
fprintf(fid_imp_stats_fri,'%s\t',time);
for k=1:6
  fprintf(fid_nan_stats,'%8.8f\t',nan_stats(j,k));
  fprintf(fid_nan_stats_mon,'%8.8f\t',nan_stats_mon(j,k));
  fprintf(fid_nan_stats_tue,'%8.8f\t',nan_stats_tue(j,k));
  fprintf(fid_nan_stats_wed,'%8.8f\t',nan_stats_wed(j,k));
  fprintf(fid_nan_stats_thu,'%8.8f\t',nan_stats_thu(j,k));
  fprintf(fid_nan_stats_fri,'%8.8f\t',nan_stats_fri(j,k));
  fprintf(fid_del_stats,'%8.8f\t',del_stats(j,k));
  fprintf(fid_del_stats_mon,'%8.8f\t',del_stats_mon(j,k));
  fprintf(fid_del_stats_tue,'%8.8f\t',del_stats_tue(j,k));
  fprintf(fid_del_stats_wed,'%8.8f\t',del_stats_wed(j,k));
  fprintf(fid_del_stats_thu,'%8.8f\t',del_stats_thu(j,k));
  fprintf(fid_del_stats_fri,'%8.8f\t',del_stats_fri(j,k));
  fprintf(fid_imp_stats,'%8.8f\t',imp_stats(j,k));
  fprintf(fid_imp_stats_mon,'%8.8f\t',imp_stats_mon(j,k));
  fprintf(fid_imp_stats_tue,'%8.8f\t',imp_stats_tue(j,k));
  fprintf(fid_imp_stats_wed,'%8.8f\t',imp_stats_wed(j,k));
  fprintf(fid_imp_stats_thu,'%8.8f\t',imp_stats_thu(j,k));
```

```
    fprintf(fid_imp_stats_fri,'%8.8f\t',imp_stats_fri(j,k));
  end
  fprintf(fid_nan_stats,'\n');
  fprintf(fid_nan_stats_mon,'\n');
  fprintf(fid_nan_stats_tue,'\n');
  fprintf(fid_nan_stats_wed,'\n');
  fprintf(fid_nan_stats_thu,'\n');
  fprintf(fid_nan_stats_fri,'\n');
  fprintf(fid_del_stats,'\n');
  fprintf(fid_del_stats_mon,'\n');
  fprintf(fid_del_stats_tue,'\n');
  fprintf(fid_del_stats_wed,'\n');
  fprintf(fid_del_stats_thu,'\n');
  fprintf(fid_del_stats_fri,'\n');
  fprintf(fid_imp_stats,'\n');
  fprintf(fid_imp_stats_mon,'\n');
  fprintf(fid_imp_stats_tue,'\n');
  fprintf(fid_imp_stats_wed,'\n');
  fprintf(fid_imp_stats_thu,'\n');
  fprintf(fid_imp_stats_fri,'\n');
end
fprintf(1,'Closing files...\n');
fclose(fid_nan_stats);
fclose(fid_nan_stats_mon);
fclose(fid_nan_stats_tue);
fclose(fid_nan_stats_wed);
fclose(fid_nan_stats_thu);
fclose(fid_nan_stats_fri);
fclose(fid_del_stats);
fclose(fid_del_stats_mon);
fclose(fid_del_stats_tue);
fclose(fid_del_stats_wed);
fclose(fid_del_stats_thu);
fclose(fid_del_stats_fri);
fclose(fid_imp_stats);
fclose(fid_imp_stats_mon);
fclose(fid_imp_stats_tue);
fclose(fid_imp_stats_wed);
fclose(fid_imp_stats_thu);
fclose(fid_imp_stats_fri);

clear fid_nan_stats fid_nan_stats_mon fid_nan_stats_tue fid_nan_stats_wed fid_nan_stats_thu ...
      fid_nan_stats_fri fid_del_stats fid_del_stats_mon fid_del_stats_tue fid_del_stats_wed ...
      fid_del_stats_thu fid_del_stats_fri fid_imp_stats fid_imp_stats_mon fid_imp_stats_tue ...
      fid_imp_stats_wed fid_imp_stats_thu fid_imp_stats_fri;

fprintf(1,'Generating graphs for %s\n\n',strcat(prefix_nan,name));

figure
plot(0:2:1438,nan_stats_mon(:,1),'-k',0:2:1438,nan_stats_tue(:,1),'-r',...
  0:2:1438,nan_stats_wed(:,1),'-g',0:2:1438,nan_stats_thu(:,1),'-b',...
  0:2:1438,nan_stats_fri(:,1),'-y'),...
  grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
  title(strcat('Mean of days with NaN data on R',name)),axis([0 1440 10 80]);
LEG = legend('mon','tue','wed','thu','fri',0);
set(gca,'XTick',0:60:1440);
set(gca,'XTickLabel',0:1:24);
set(gca,'YTick',10:10:80);
filename = char(strcat('daymean_nan_R',name));
exportfig(gcf,strcat(filename,'.eps'),'width',10,...
  'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');
```

```
figure
plot(0:2:1438,del_stats_mon(:,1),'-k',0:2:1438,del_stats_tue(:,1),'-r',...
  0:2:1438,del_stats_wed(:,1),'-g',0:2:1438,del_stats_thu(:,1),'-b',...
  0:2:1438,del_stats_fri(:,1),'-y'),...
  grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
  title(strcat('Mean of days with Del data on R',name)),axis([0 1440 10 80]);
LEG = legend('mon','tue','wed','thu','fri',0);
set(gca,'XTick',0:60:1440);
set(gca,'XTickLabel',0:1:24);
set(gca,'YTick',10:10:80);
filename = char(strcat('daymean_del_R',name));
exportfig(gcf,strcat(filename,'.eps'),'width',10,...
  'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');

figure
plot(0:2:1438,imp_stats_mon(:,1),'-k',0:2:1438,imp_stats_tue(:,1),'-r',...
  0:2:1438,imp_stats_wed(:,1),'-g',0:2:1438,imp_stats_thu(:,1),'-b',...
  0:2:1438,imp_stats_fri(:,1),'-y'),...
  grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
  title(strcat('Mean of days with Imp data on R',name)),axis([0 1440 10 80]);
LEG = legend('mon','tue','wed','thu','fri',0);
set(gca,'XTick',0:60:1440);
set(gca,'XTickLabel',0:1:24);
set(gca,'YTick',10:10:80);
filename = char(strcat('daymean_imp_R',name));
exportfig(gcf,strcat(filename,'.eps'),'width',10,...
  'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');

clear mon_nan tue_nan wed_nan thu_nan fri_nan data_nan;
clear mon_del tue_del wed_del thu_del fri_del data_del;
clear mon_imp tue_imp wed_imp thu_imp fri_imp data_imp;
close all;

figure
plot(0:2:1438,nan_stats(:,1),'-k',0:2:1438,nan_stats(:,2),'-r',...
  0:2:1438,nan_stats(:,3),'-g',0:2:1438,nan_stats(:,4),'-b',0:2:1438,nan_stats(:,5),'-y'),...
  grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
  title(strcat('NaN Statistics on R',name)),axis([0 1440 10 80]);
LEG = legend('mean','median','minimum','15-percentile','85-percentile',0);
set(gca,'XTick',0:60:1440);
set(gca,'XTickLabel',0:1:24);
set(gca,'YTick',10:10:80);
filename = char(strcat('stats_nan_R',name));
exportfig(gcf,strcat(filename,'.eps'),'width',10,...
  'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');

figure
plot(0:2:1438,del_stats(:,1),'-k',0:2:1438,del_stats(:,2),'-r',...
  0:2:1438,del_stats(:,3),'-g',0:2:1438,del_stats(:,4),'-b',0:2:1438,del_stats(:,5),'-y'),...
  grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
  title(strcat('Del Statistics on R',name)),axis([0 1440 10 80]);
LEG = legend('mean','median','minimum','15-percentile','85-percentile',0);
set(gca,'XTick',0:60:1440);
set(gca,'XTickLabel',0:1:24);
set(gca,'YTick',10:10:80);
filename = char(strcat('stats_del_R',name));
exportfig(gcf,strcat(filename,'.eps'),'width',10,...
  'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');

figure
plot(0:2:1438,imp_stats(:,1),'-k',0:2:1438,imp_stats(:,2),'-r',...
  0:2:1438,imp_stats(:,3),'-g',0:2:1438,imp_stats(:,4),'-b',0:2:1438,imp_stats(:,5),'-y'),...
```

```
    grid,ylabel('Travel Time'),xlabel('Time on day (h)'),...
    title(strcat('Imp Statistics on R',name)),axis([0 1440 10 80]);
  LEG = legend('mean','median','minimum','15-percentile','85-percentile',0);
  set(gca,'XTick',0:60:1440);
  set(gca,'XTickLabel',0:1:24);
  set(gca,'YTick',10:10:80);
  filename = char(strcat('stats_imp_R',name));
  exportfig(gcf,strcat(filename,'.eps'),'width',10,...
    'height',10,'fontmode','fixed','fontsize',5,'color','cmyk');

  close all;
end
```

at_fcm.m

```
close all;
clear all;
clc;

% files with the travel times, days in columns and time-intervals on rows
filenames = {'04heen.txt','04terug.txt','09heen.txt','09terug.txt','10heen.txt',...
             '10terug.txt','19heen.txt','19terug.txt','32heen.txt','32terug.txt'};
% prefix that should be added to the filenames where the cleaned data can be found

prefixstr = {'nan_R','del_R','imp_R'};
prefix_statstr = {'stats_nan_R','stats_del_R','stats_imp_R'};
ftstr = {' (NAN Data)',' (DEL Data)',' (IMP Data)'};
prefix = 1;

daystr = {'_mon','_tue','_wed','_thu','_fri'};
N=720;                              % # intervals
dx=2;                               % Interval step (in minutes)
n_c = 10;                           % number of clusters
w = 2;                              % exponent for partition matrix
n_i = 50;                           % maximum iterations
e = 0.001;                          % minimum improvement
info = 0;                           % show info

repstatstr = {'mean','median'};
repstat = 1;

for prefix=2:3
  for i=1:size(filenames,2)           % and for each file
    xdata=0:dx:(N-1)*dx;                % set time-domain
    % Load the travel times itself, 2 minutes intervals (dmax x N matrix)
    [pathstr,name,ext,versn] = fileparts(filenames{i});
    clear versn pathstr;
    fprintf(1,'Opening %s\n',strcat(prefixstr{prefix},name,ext));
    ydata = dlmread(strcat(prefixstr{prefix},name,ext),'\t');
    [n_g,n_t]=size(ydata);
    % ydates = dlmread(strcat(prefixstr{prefix},name,'_days',ext),'\t');

    % subtract the mean
    mean_values=mean(ydata')';
    mean_matrix=mean_values*ones(1,n_t);
    ydata=ydata-mean_matrix;

    % divide by the standard diviation
    standard_dev=std(ydata')';
    std_matrix=standard_dev*ones(1,n_t);
```

```
    ydata=ydata./std_matrix;

    % Add two time points with zero and scale down the original time points
%    ydata=[zeros(n_g,2) ydata];
%    xdata=xdata-xdata(1)+1;
%    xdata=[-1 0 xdata];

    for d=1:5
      fprintf(1,'Opening %s\n',strcat(prefix_statstr{prefix},name,daystr{d},ext));
      stat_values(d,:) = dlmread(strcat(prefix_statstr{prefix},name,daystr{d},ext),'\t',[1 1 N 1])';
      mean_values=mean(stat_values(d,:)')';
      mean_matrix=mean_values*ones(1,n_t);
      stat_values(d,:)=stat_values(d,:)-mean_matrix;
      standard_dev=std(stat_values(d,:)')';
      std_matrix=standard_dev*ones(1,n_t);
      stat_values(d,:)=stat_values(d,:)./std_matrix;
    end
%    stat_values=[zeros(d,2) stat_values];

    % options(1): exponent for the partition matrix U (default: 2.0)
    % options(2): maximum number of iterations (default: 100)
    % options(3): minimum amount of improvement (default: 1e-5)
    % options(4): info display during iteration (default: 1)

    fprintf(1,'Computing clusters.\n');
    for clusts=2:n_c
      fprintf(1,'number of clusters: %s\n',int2str(clusts));
      clear stats mat_U mat_obj_fcn;
      [stats,mat_U,mat_obj_fcn] = fcm(ydata, clusts, [w n_i e info]);
%       plot(stats(:,1), stats(:,2),'o');
%       maxU = max(mat_U);
%       index1 = find(mat_U(1,:) == maxU);
%       index2 = find(mat_U(2, :) == maxU);
%       line(X(index1,1), X(index1, 2), 'linestyle', 'none', 'marker', '*', 'color', 'g');
%       line(X(index2,1), X(index2, 2), 'linestyle', 'none', 'marker', '*', 'color', 'r');

      filename = char(strcat('clust',int2str(clusts),prefixstr{prefix},name));
      figure(i),...
        subplot(1,2,1),
        plot(xdata,stats),...
        grid,ylabel('Standardized Travel Time'),xlabel('Time on day (h)'),...
        title(strcat(int2str(clusts),' clusters on R',name,ftstr{prefix}));
        set(gca,'XTick',0:60:1440);
        set(gca,'XTickLabel',0:1:24);

        % mean,median,min,15perc,85perc,stddev
        subplot(1,2,2),plot(xdata,stat_values(1,:),'-k',xdata,stat_values(2,:),'-r',...
        xdata,stat_values(3,:),'-g',xdata,stat_values(4,:),'-b',xdata,stat_values(5,:),'-y'),...
        grid,ylabel('Standardized Travel Time'),xlabel('Time on day (h)'),...
        title(strcat(repstatstr{repstat},' on R',name,ftstr{prefix}));
        LEG = legend('monday','tuesday','wednesday','thursday','friday',0);
        set(gca,'XTick',0:60:1440);
        set(gca,'XTickLabel',0:1:24);

      exportfig(gcf,strcat(filename,'.eps'),'width',20,...
        'height',6,'fontmode','fixed','fontsize',6,'color','cmyk');
      close all;
    end
    clear stat_values ydata xdata;
    fprintf(1,'Done with %s.\n\n',strcat(prefixstr{prefix},name,ext));
  end
end
```

Appendix B

Dataset Statistics

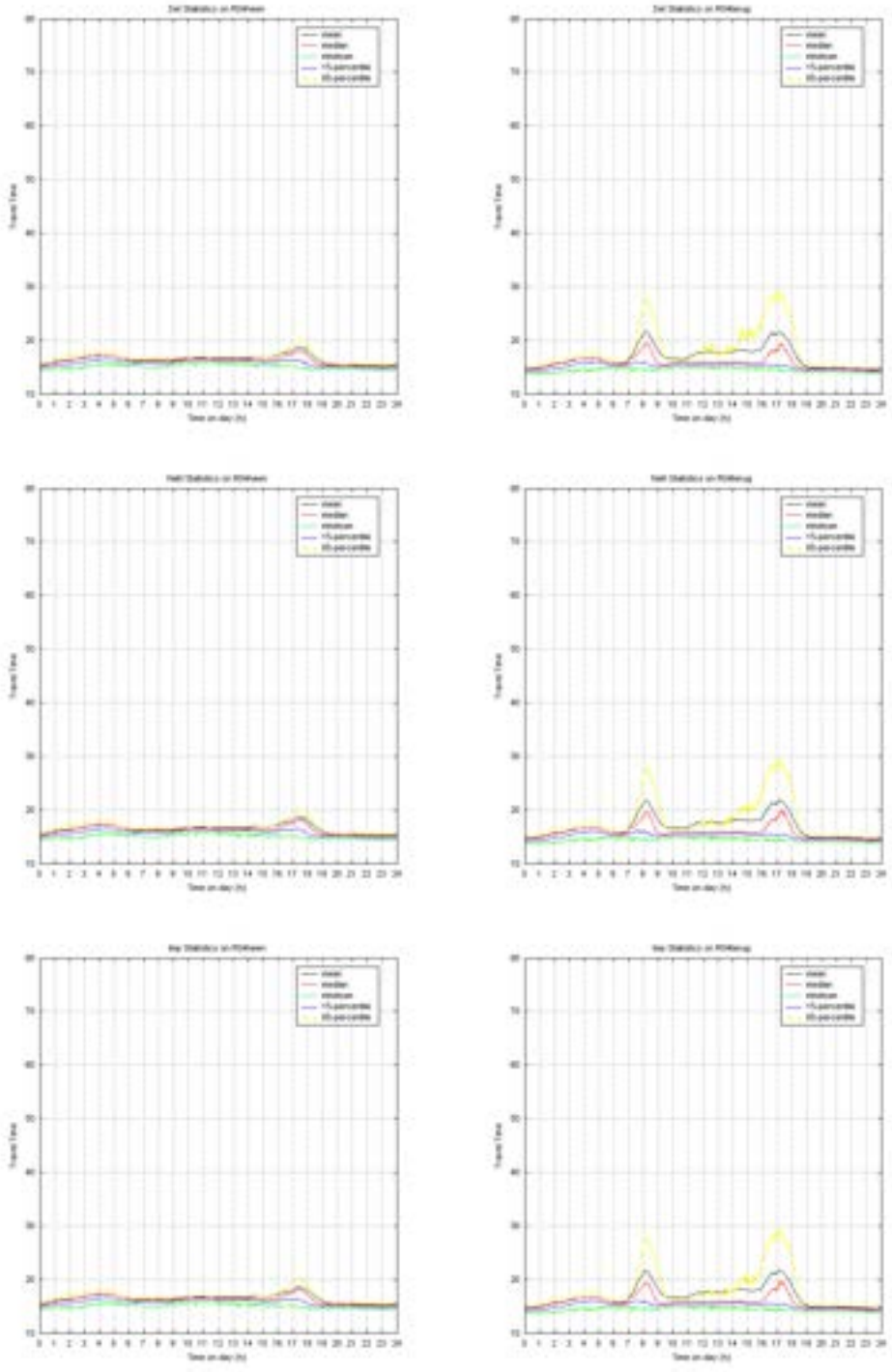

Figure B.1: Tilburg - Eindhoven v.v.

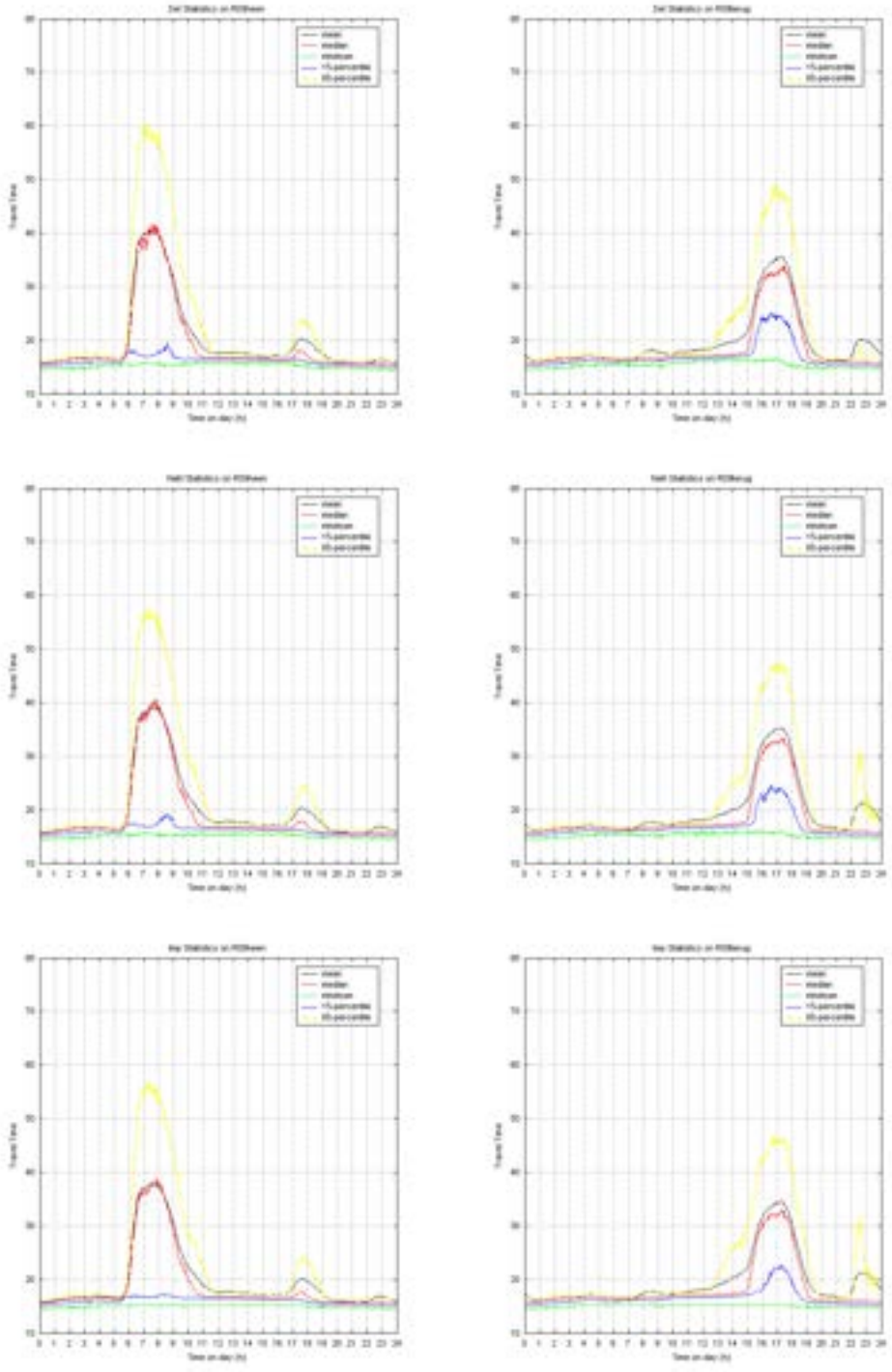

Figure B.2: Utrecht - Beesd v.v.

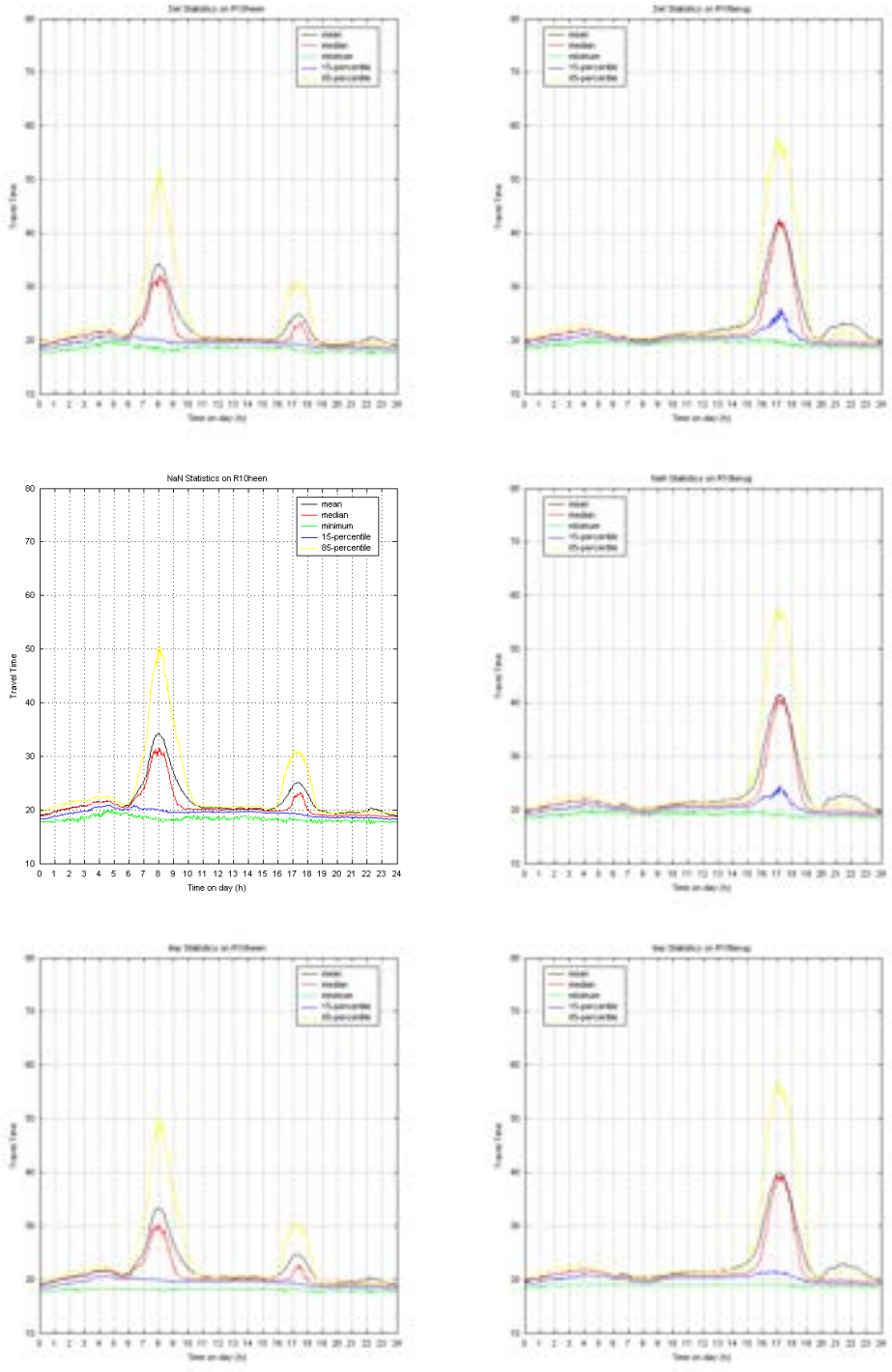

Figure B.3: Utrecht - Gorinchem v.v.

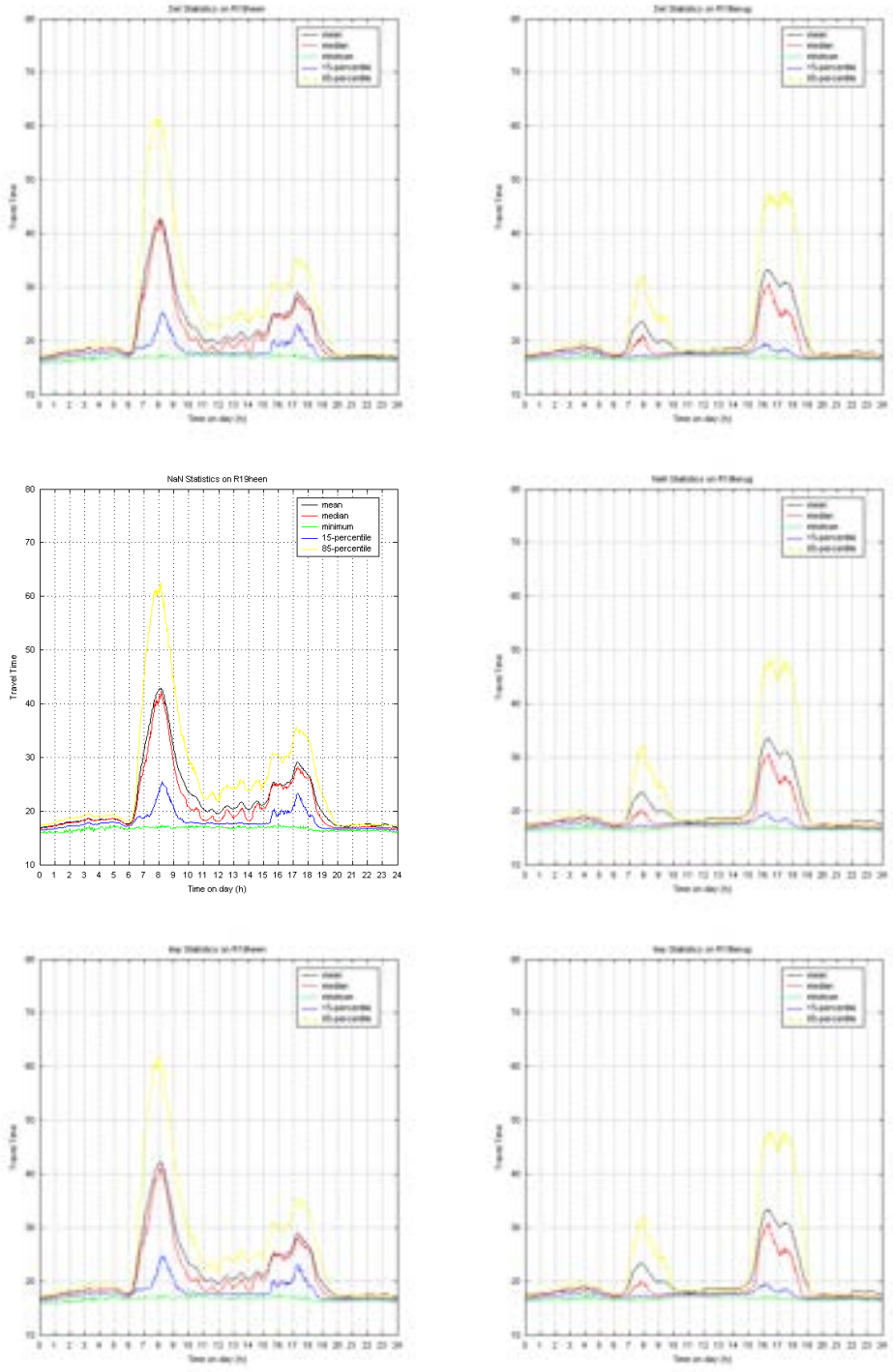

Figure B.4: Den Haag - Gouda v.v.

Appendix C

Dataset Means per day

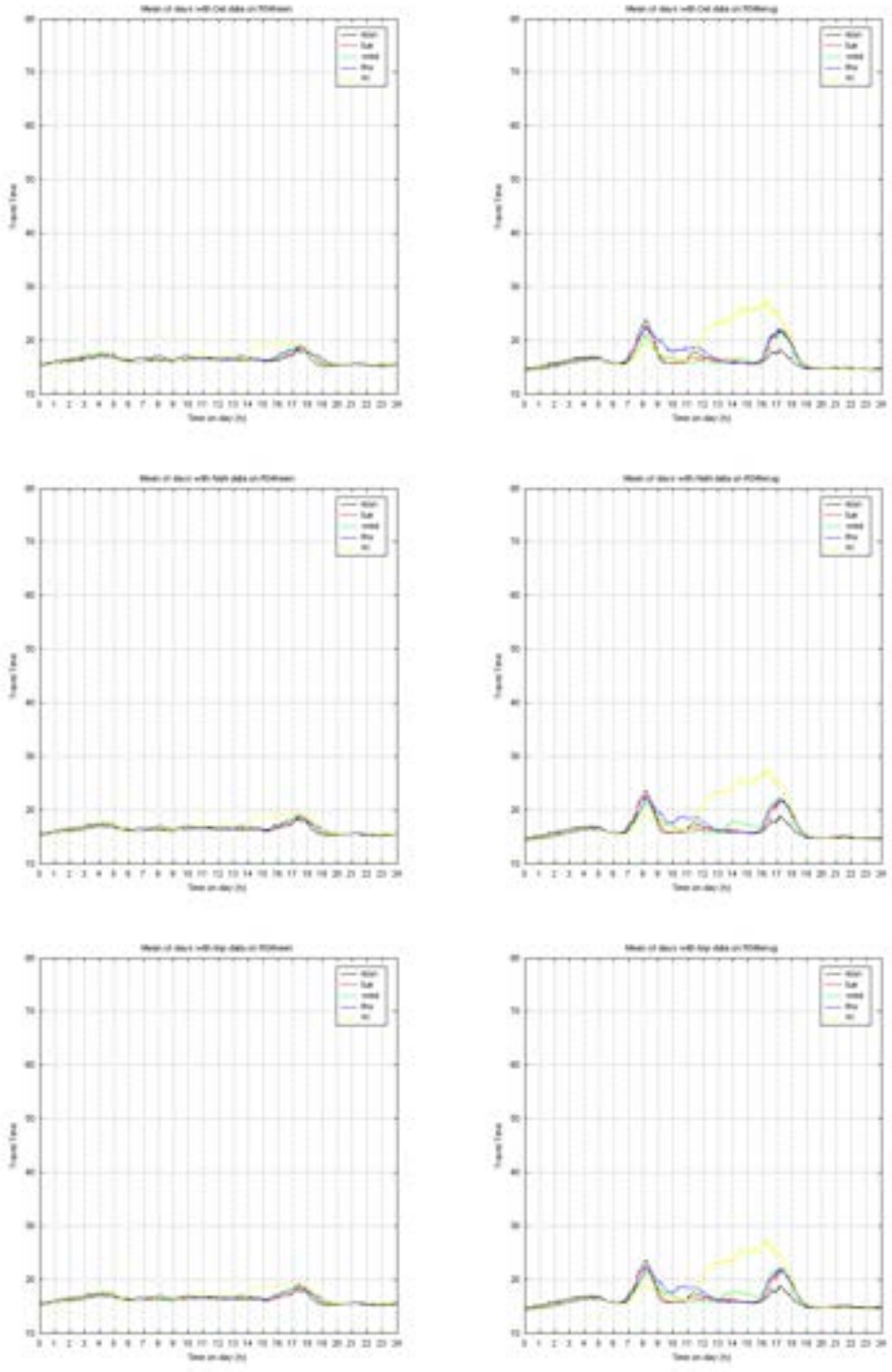

Figure C.1: Tilburg - Eindhoven v.v.

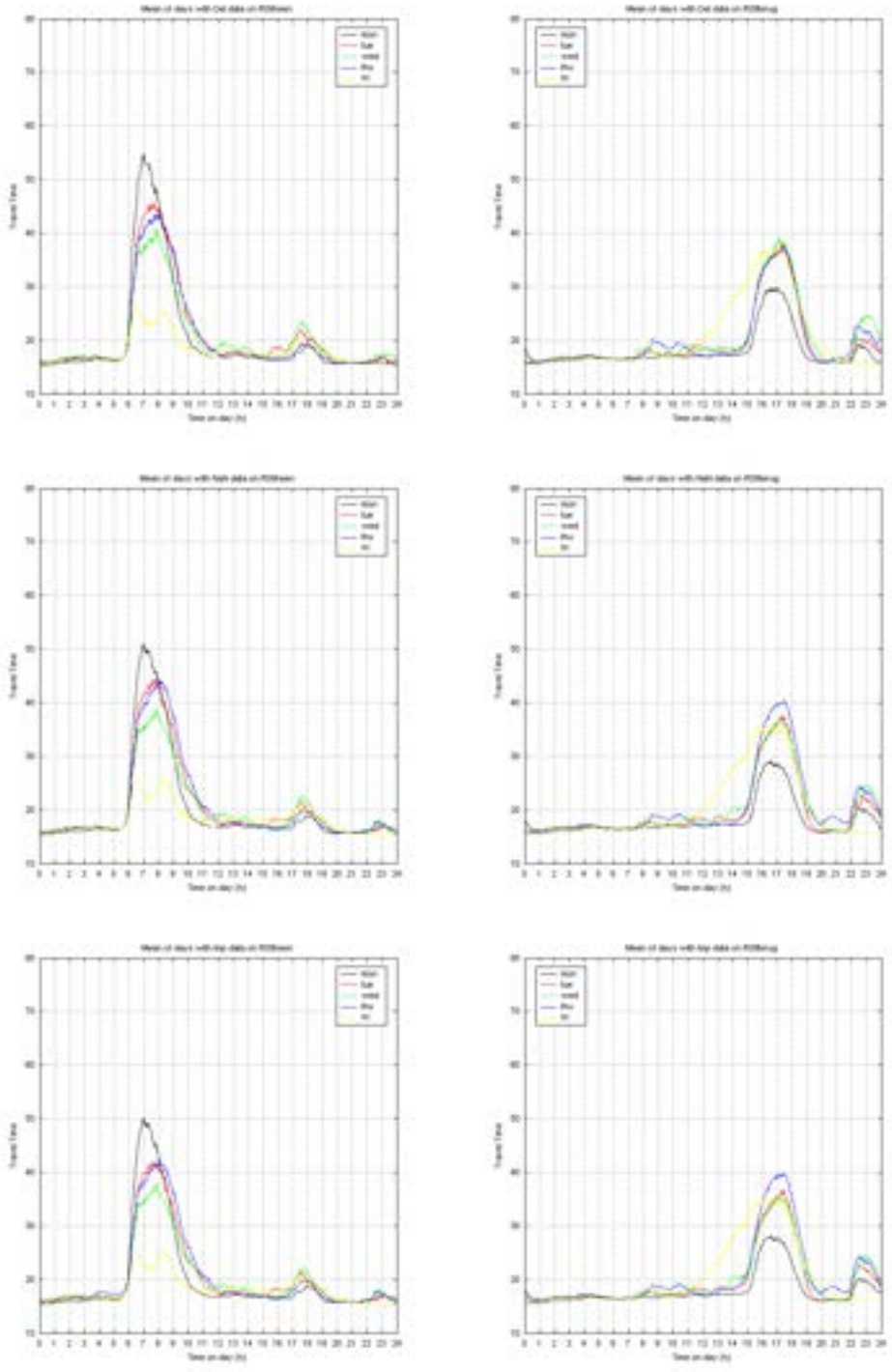

Figure C.2: Utrecht - Beesd v.v.

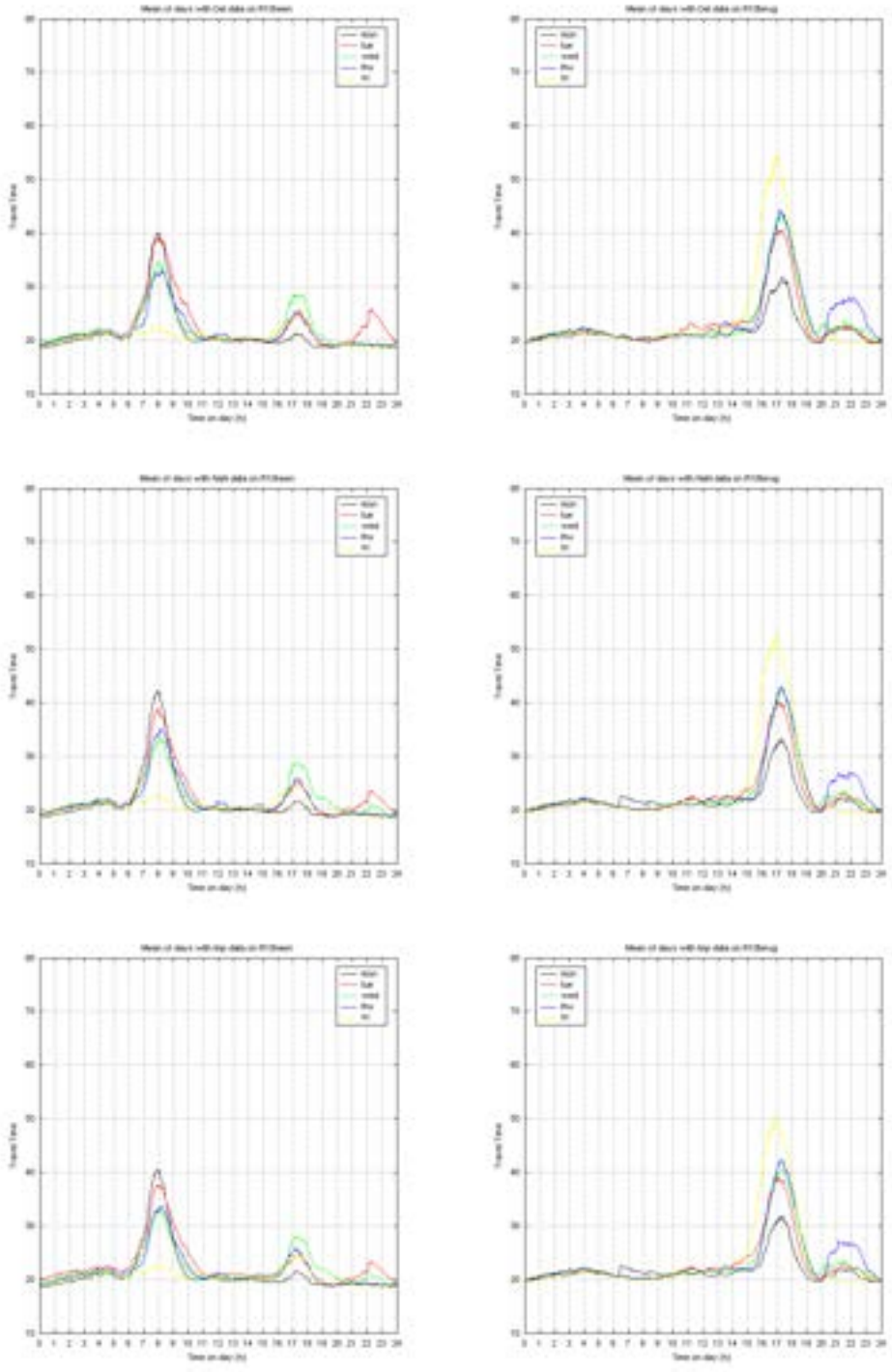

Figure C.3: Utrecht - Gorinchem v.v.

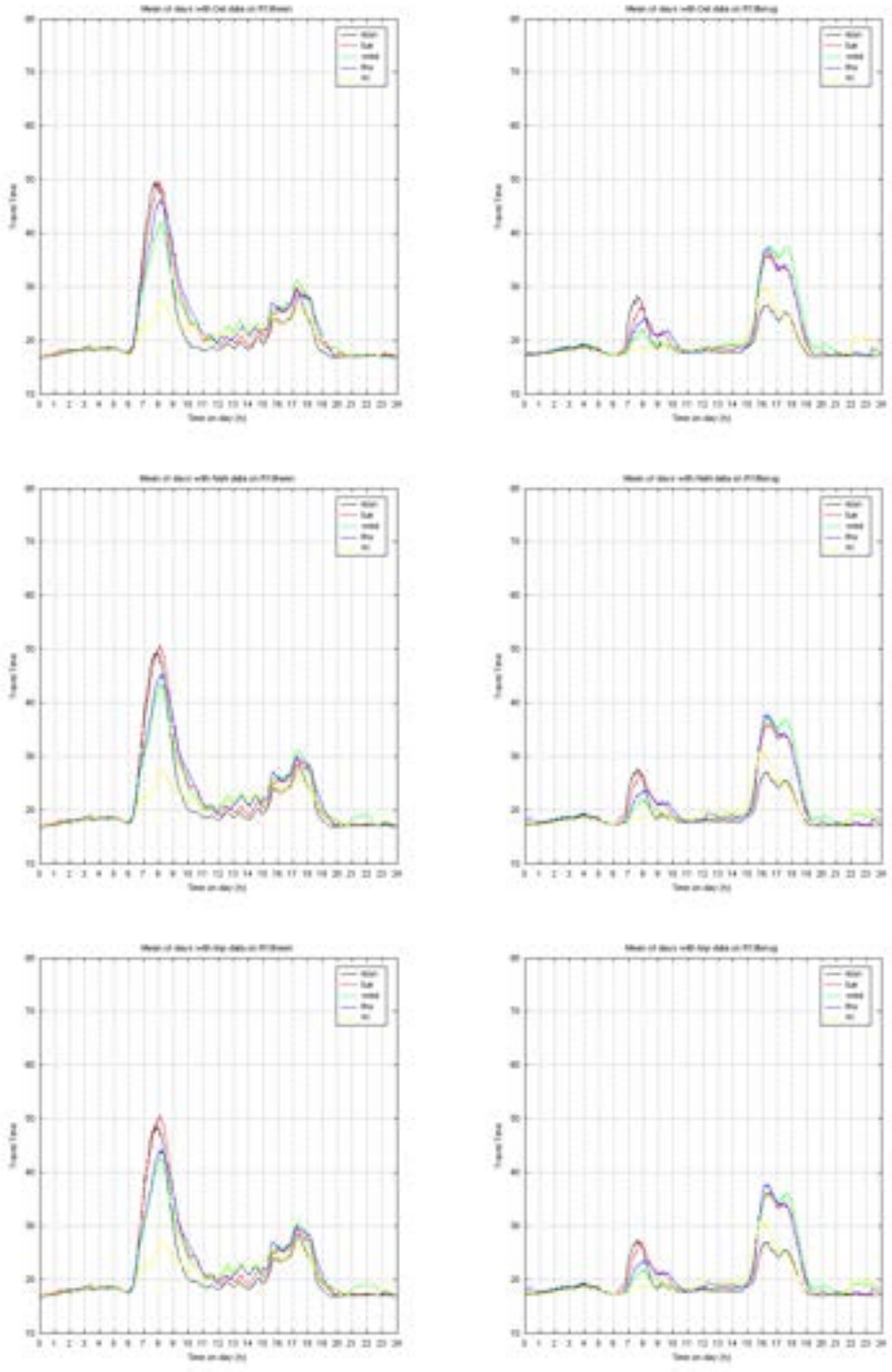

Figure C.4: Den Haag - Gouda v.v.

Appendix D

Profiles as a function

D.1 Continuous Fourier Series

The following is taken from [6].

A *continuous* Fourier series is defined as:

$$f(t) = \frac{A_0}{2} + \sum_{p=1}^{\infty} [A_p \cos(\omega_p t) + B_p \sin(\omega_p t)] \tag{D.1}$$

where the angular frequency is defined as

$$\omega_p = 2\pi f_p = \frac{2\pi}{T} p \tag{D.2}$$

Assumed in the above definition is that $f(t)$ is periodic with period $T : f(t+T) = f(t)$. We now need to extract the Fourier Coefficients (A_p, B_p). This can be done using the property of *orthogonality* of the *trigonometric basis*.

The coefficients $A_p, p \in \{0, 1, 2, \ldots\}$ and $B_p, p \in \{1, 2, \ldots\}$ are the amplitudes, and defined as follows:

$$A_p = \frac{2}{T} \int_0^T f(t) \cos(\omega_p t) dt \tag{D.3}$$

$$B_p = \frac{2}{T} \int_0^T f(t) \sin(\omega_p t) dt \tag{D.4}$$

However, this can only be used when using continuous time data. Since our travel time measurements are only taken at timestamps of every two minutes (any higher frequencies are thus eliminated) and so we have a discrete case here. Both in the context of of frequencies, but also in the context of travel times.

We define the *discrete* Fourier series in the following way:

$$f(t_n) = \frac{A_0}{2} + \sum_{p=1}^{M} A_p \cos(\omega_p t) + B_p \sin(\omega_p t) \tag{D.5}$$

where the time, time step and *trigonometric arguments* are given by

$$t_n = n\Delta t,\ \Delta t = \frac{T}{N},\ \text{and}\ \omega_p t = \frac{2\pi pn}{T} \tag{D.6}$$

Now, all we need to do is determine M and the unknown coefficients. Again, we use the orthogonality principle, but now replacing a continuous integral by a discrete sum. Afterwards, we determine the unknowns in terms of the given sample travel times, which in fact are samples of our function $f(t)$.

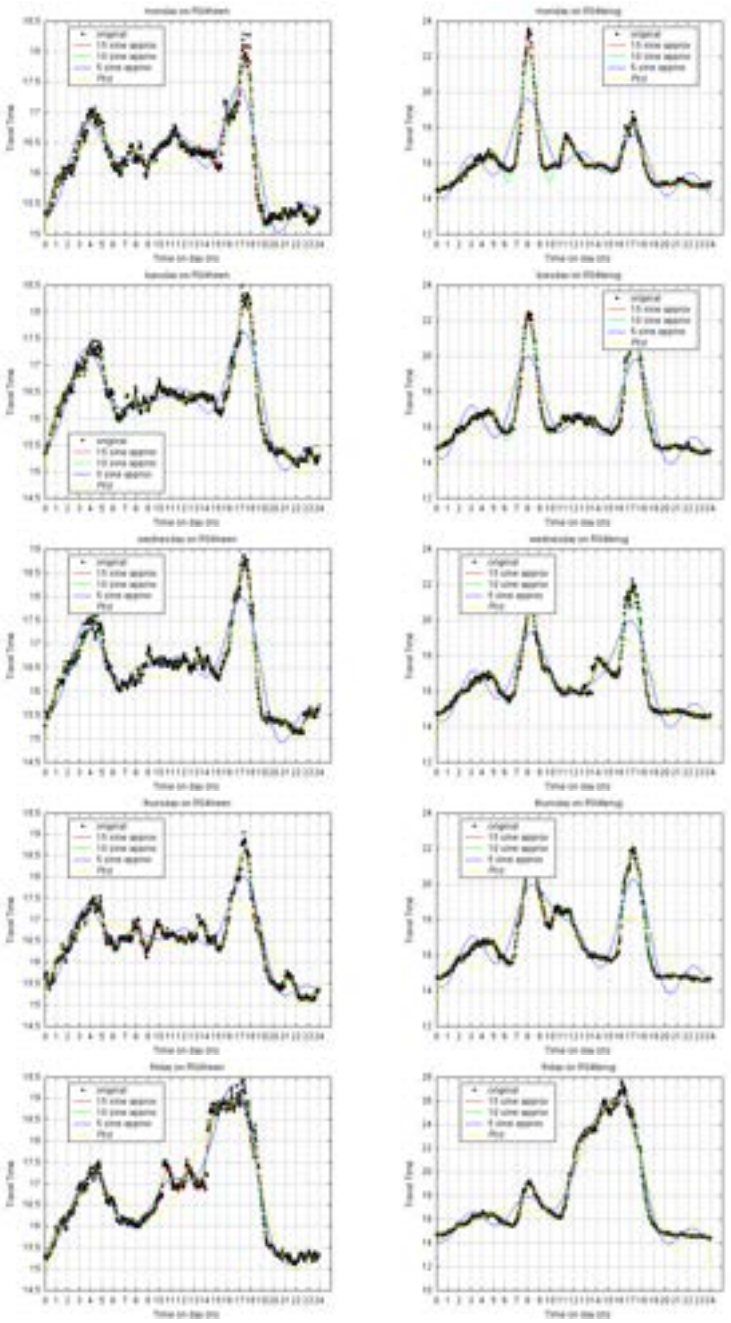

Figure D.1: DFT approximation on means for Eindhoven - Tilburg v.v.

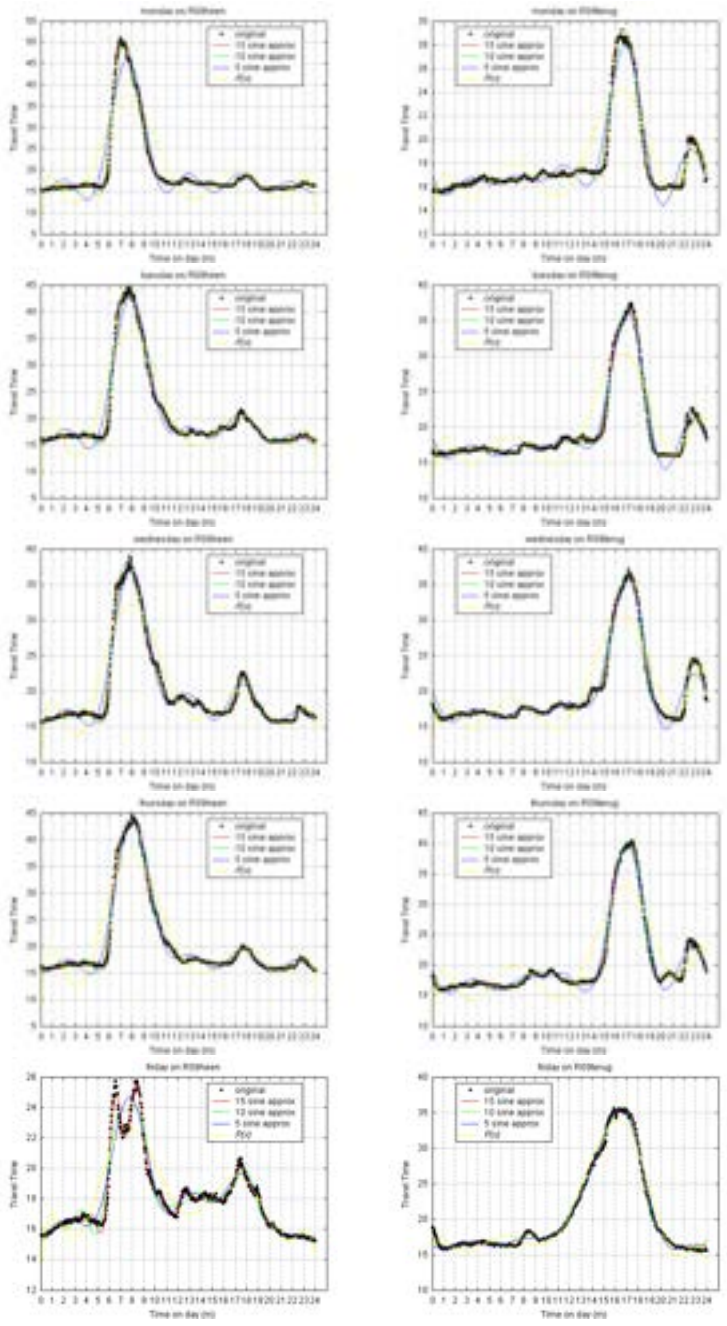

Figure D.2: DFT approximation on means for Utrecht - Beesd v.v.

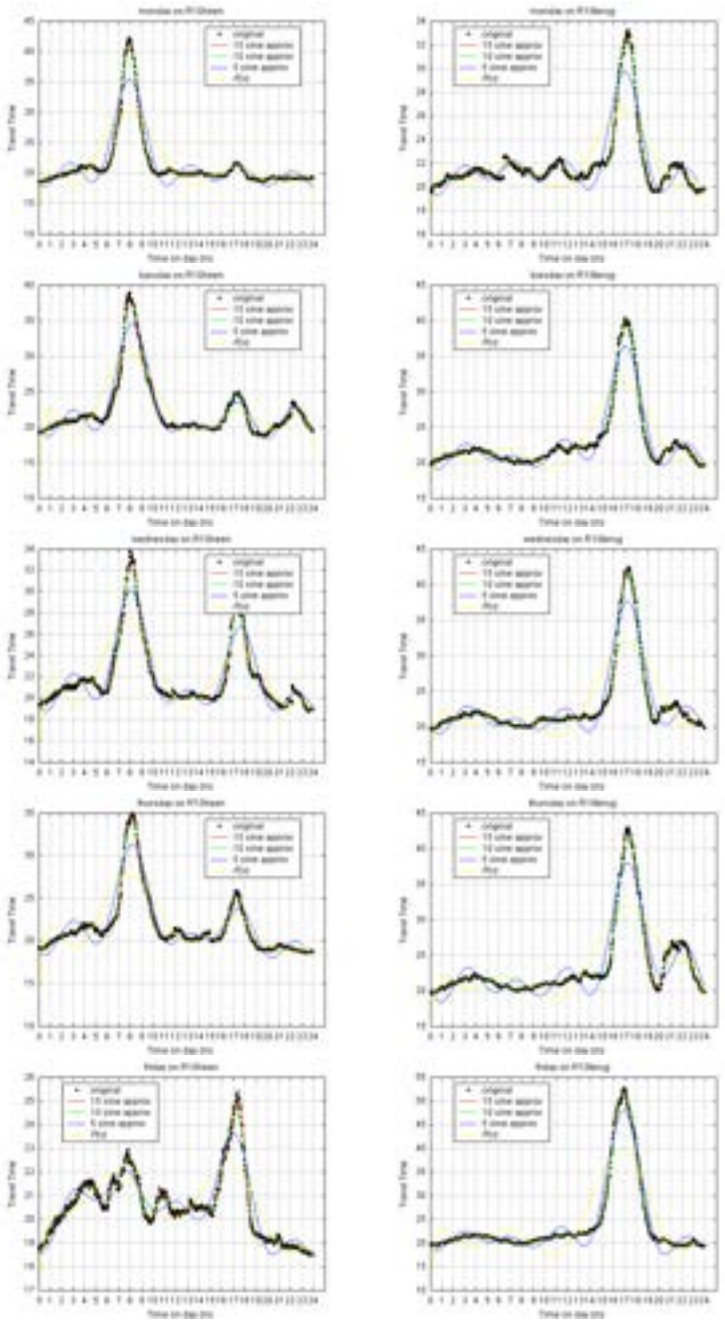

Figure D.3: DFT approximation on means for Utrecht - Gorinchem v.v.

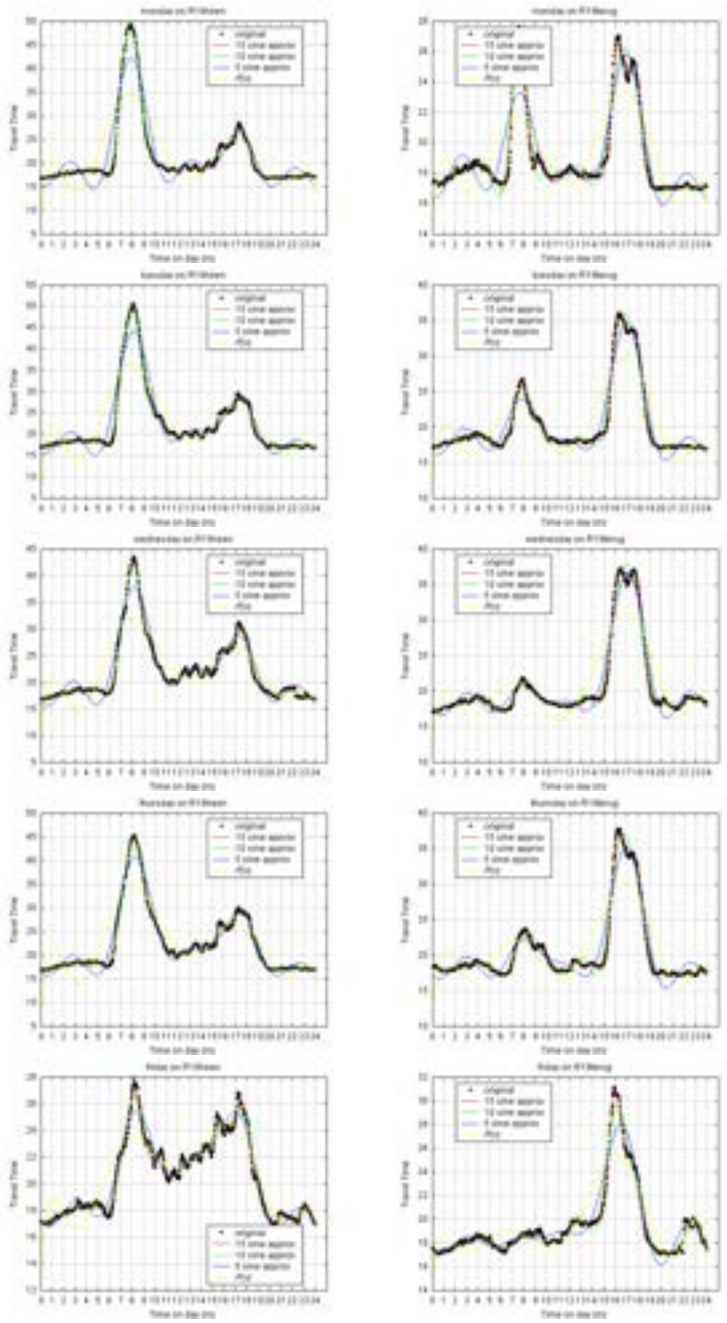

Figure D.4: DFT approximation on means for Den Haag - Gouda v.v.

Appendix E

Clusters illustration, imputated values

Every figure exists of 8 graphs. In the rows you fill find the clustering algorithm applied with respectively 3, 4, 5 and 6 clusters. In the left columns the centroids of the clisters are shown, on the right the mean value per weekday is plotted.

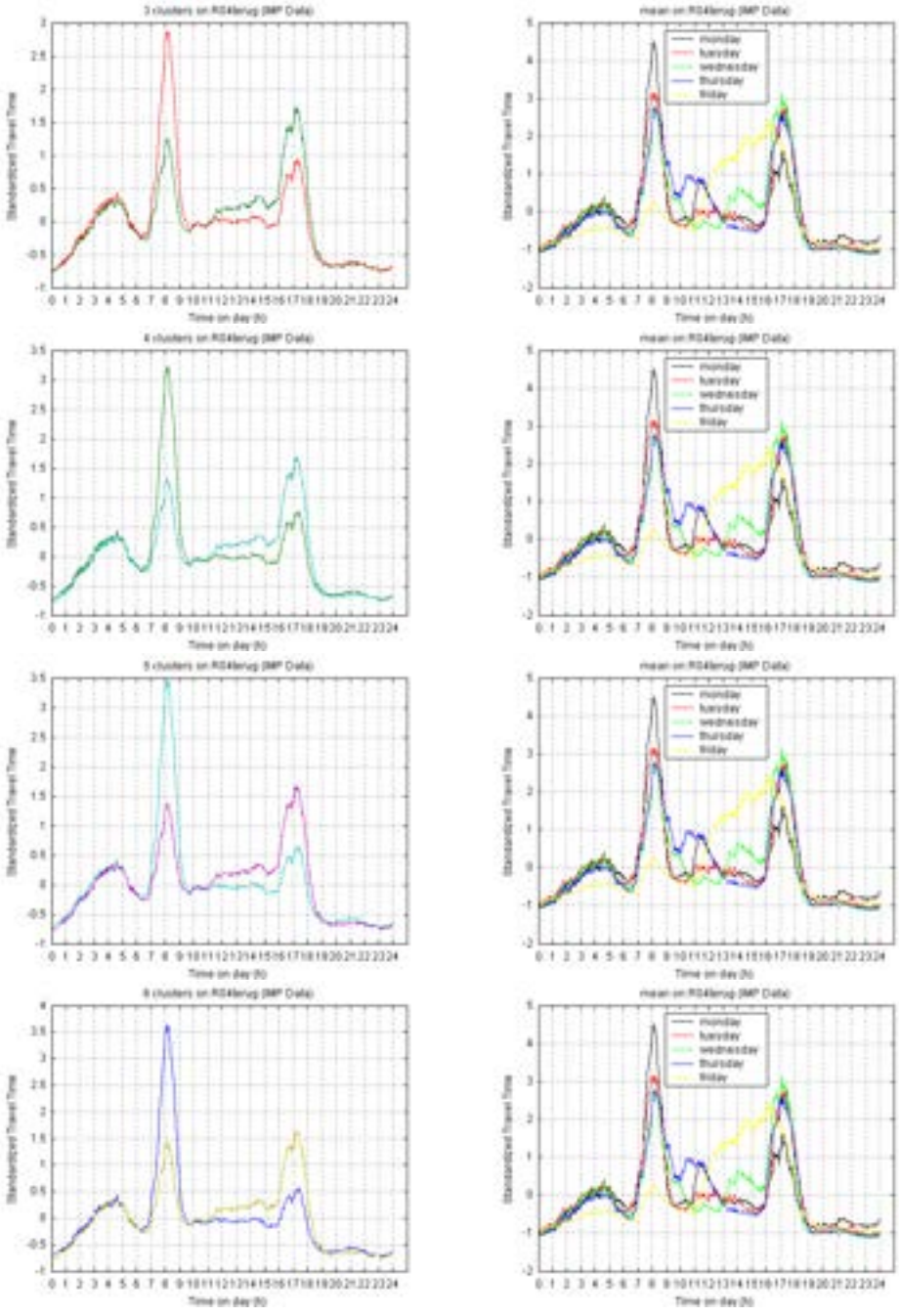

Figure E.1: Tilburg - Eindhoven

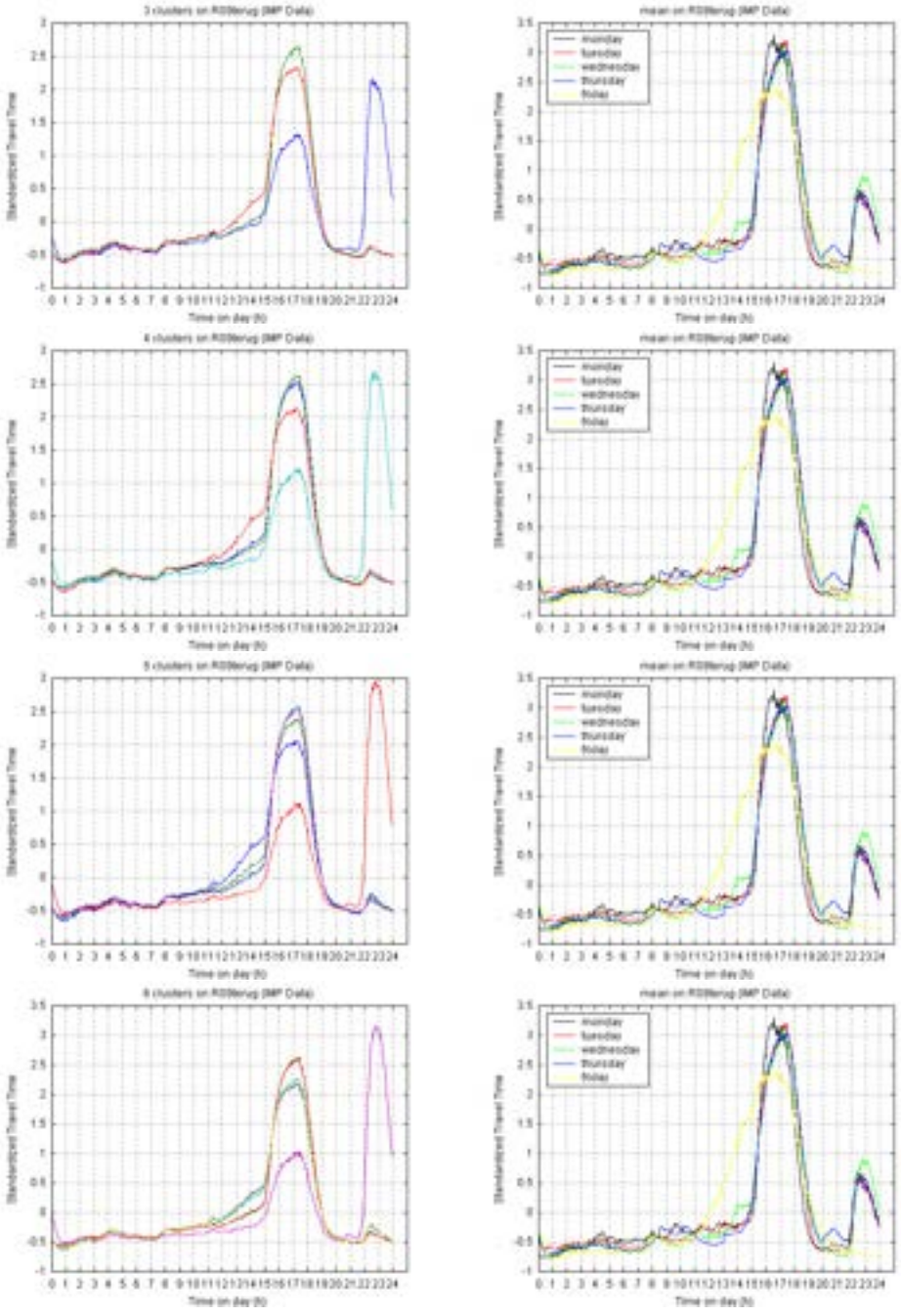

Figure E.2: Beesd - Utrecht

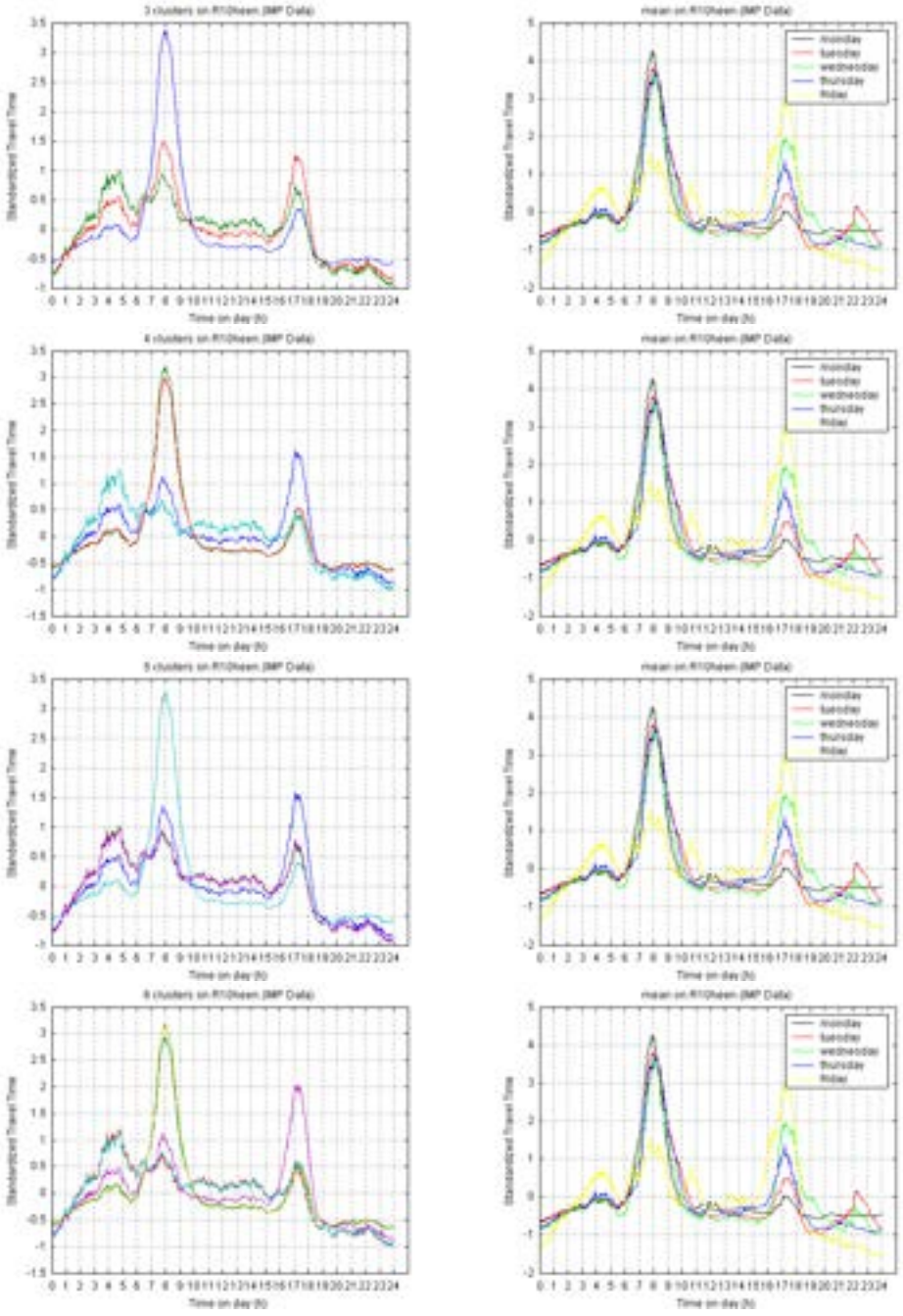

Figure E.3: Utrecht - Gorinchem

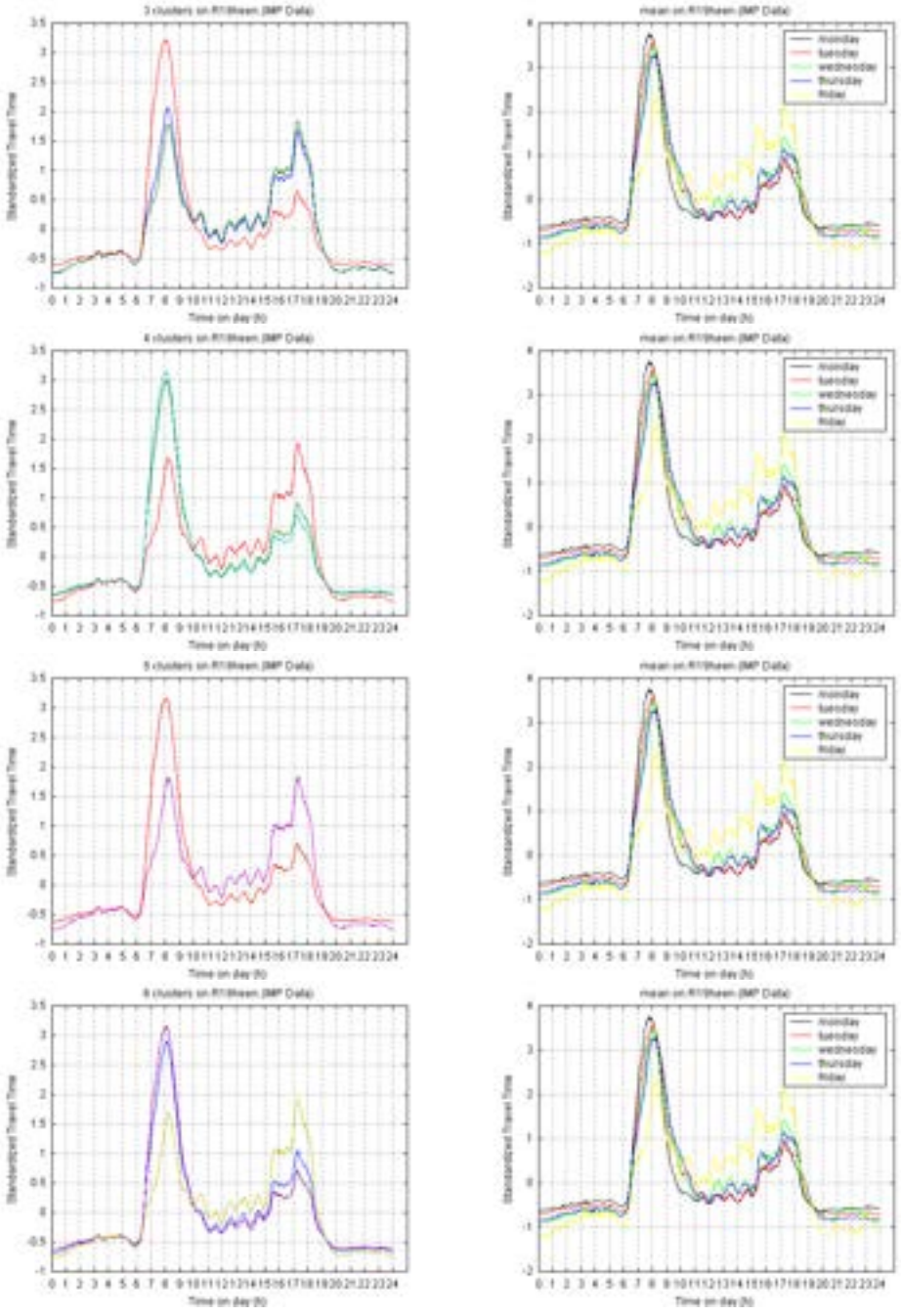

Figure E.4: Den Haag - Gouda