university of groningen

faculty of mathematics and natural sciences

Visualising large scale temporal geospatial multivariate graphs in a web-based environment

Master Thesis

November 2015

Student: S. Groenewold

Primary supervisor: Prof. dr. A. C. Telea

Secondary supervisor: Prof. dr. ir. M. Aiello

External supervisors: Drs. K. A. Helmholt, Drs. ing. J. van der Geest

ABSTRACT

There are many large scale infrastructures in our modern day society, such as gas transmission system, water supply networks and electrical grids. The behaviour of these kinds of networks can be described using simulations or sensors, using which huge datasets are created. Generally, we can refer to these types of datasets as large temporal geospatial multivariate graphs. The analysis of these datasets is important, since it can lead to new knowledge, for example by allowing researchers to get a better understanding of algorithms they are developing, or system monitors to detect problem areas. Analysing such datasets is however a challenging problem, due to their sheer size, as these datasets contain many attributes, describe large time ranges, and large spatial areas. In particular in a web-based environment where processing power, storage and bandwidth are limited, additional challenges are introduced.

This work proposes an analysis environment where these types of datasets can be explored and analysed by using a set of linked visualisations. Each of the visualisations specialises in an aspect of the data, allowing them to complement each other. To serve the datasets to a web environment an aggregation and storage scheme is constructed, which attempts to give a good balance between enough detail and low bandwidth requirements. This functionality is added to an existing universal analysis framework. The result is a general purpose analysis platform for large temporal geospatial multivariate graphs.

The implemented visualisations allow users to analyse aspects of their datasets which could previously not be viewed using their tools. In particular with respect to the analysis of temporal data significant improvements have been made. The solution combines a map based visualisation with an evolution spectrograph visualisation, which is a type of visualisation that to our knowledge has never before been used to display these types of graphs. Together these visualisation give the user the option to easily relate data back to the real world and view the structure of the network, while also offering a detailed view of the temporal behaviours in the data. The solution follows Shneiderman's mantra by giving a high level overview of the data, and allowing the user to zoom and filter the data, and view details as desired. We show such an analysis tool offers new non-trivial insights in the temporal aspect of these types of datasets.

ACKNOWLEDGEMENTS

First of all I would like to thank TNO for offering the possibility to perform this research and writing this thesis, which has been a great learning experience. In particular I would like to thank Kristian Helmholt and Johan van der Geest for their guidance, useful suggestions and extensive feedback. For his help with the CommonSense framework I would also like to thank Arnoud de Jong. Furthermore I would like to thank the other people at TNO who have helped me out during my time at TNO, including the fellow students who helped make it an enjoyable experience. Additionally I would like to thank Alexandru Telea for introducing me to TNO, and also the great suggestions and valuable feedback. I am also grateful for Marco Aiello his useful remarks. Lastly, I would like to thank Walter van Buijtenen and Manuel Nas from PhaseToPhase for inspiring me with their work.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Many infrastructures can be described as large scale *temporal geospatial multivariate graphs*. Gas transmission systems, water supply networks, electricity grids, road systems and computer networks; these are just a few examples of large scale networks that can be considered as graphs: a set of objects such as homes, intersections or generators, connected by cables, pipes or roads. In other words, nodes and edges of a graph, with locations in the real world. But such a graph only describes the structure of the network, these types of networks are not simple static objects. Pressures vary, voltages change, traffic speeds fluctuate, and each of these attributes can be measured by using sensors, or generated by simulations, over periods of time. A graph with these properties, it having attributes which are temporal in nature, its nodes and edges relating to real world locations, and each element of the graph being described by multiple attributes, is called a temporal geospatial multivariate graph.

These graphs are interesting to analyse, since the types of networks they describe, such as the examples listed above, are immense in size and play an important role in our day to day lives. Because of this it is important to observe the behaviour of these types of networks, so that they can be better understood, and potential problems can be detected early on. Some questions about these datasets can be easily answered, such as when and where in time and space a safety limit is exceeded, or what is the total consumption over some period of time. These questions have straightforward answers which can be computed. Other questions are however harder to answer, such as whether there are temporal periods which show similar patterns, or how a network reacts to a local event, or which parts of a network need to be upgraded. Answers to these type of questions can not be easily computed.

That is why this thesis proposes to create an interactive visualisation platform for these kind of graphs. By visualising the data and interacting with it, users can explore the data. This way they can search for interesting data, and zoom in onto details, while refining their search queries. Exploring these types datasets is however a challenging problem [1, 2]. Due to the large scale of these graphs, sometimes covering entire nations or even the entire world, with large amounts of attributes of which many are temporal in nature, there is simply a huge amount of data to analyse. One can not easily gain insights in these large volumes of data by looking at the raw measurements.

Netherlands Organisation for Applied Scientific Research (TNO), is a non-profit research institute in the Netherlands that performs and applies research for government bodies and public organizations. They often encounters these types of networks in their projects. They develop simulations of these kind of networks, as well as partake in projects involving large sensor networks. An example of a recent project is that of the placement of building sensors which monitor the effects of earth quakes on the property of home owners [3]. Such sensor networks generate huge amounts of data, reporting many measurements throughout the day. These kind of datasets become more common as sensors, as well as mass storage of data becomes cheaper. TNO is interested in finding a generic analysis solution for such networks, which allows them to investigate the data effectively and give insights in patterns in the data, allowing decisions to be made. Genericity is an important aspect here, to allow the solution to be reused with future projects.

In the next sections first the biggest challenges in creating a analysis solution for graphs will be discussed, and the scope of this work will be outlined. Additionally a concrete problem statement will be formulated.

1.1 PROBLEM OVERVIEW

To analyse these datasets, a set of visualisation solutions will be used. Using these visualisations users will be able to explore their datasets, and view the different aspects of the data, such as spatial locations, or temporal behaviours. This solution will be created by extending an existing web visualisation framework currently developed by TNO, called *CommonSense* [4]. The use of this framework was set as a requirement by TNO. The details of this framework will be discussed in Chapter 5. The use of this framework does have an impact on some decisions. With regards to visualisation, this framework is currently focused primarily on displaying spatial information. The spatial view it offers for this purpose is good for understanding the structure of the datasets, and the connection to real world locations. These geospatial temporal multivariate graphs however have more aspects than just the geospatial aspect, as they are also temporal and multivariate. Because of this the visualisation solutions discussed in this work will focus primarily on explaining these other aspects of the data, in a way that cooperates with any existing functionality of the CommonSense framework.

Using a web application such as CommonSense as a visualisation platform introduces some additional challenges. Compared to a native desktop application, less computational power is available, and data storage is limited. Practically all used data will have to reside in memory. Considering the vast size of these datasets, and the limited storage available on the client, all data will never be available locally on the system used for analysis, since there is simply too much of it. Data will have to fetched from a remote server as needed, and only the data that is relevant to the current viewing parameters chosen by the user should be retrieved. To gain an overview of the entire dataset, the dataset will have to be simplified strongly, and only when viewing a small subset of the data can it be retrieved and shown in full detail. Because of this data transformations will be required before the data can be visualised.

The process of transforming raw data into information consists of a number of components. This process can be described as a pipeline, where data is transformed and send between the different parts that make up the pipeline. A high level representation of the pipeline used in this work is shown in Figure 1. The pipeline start with raw data coming into the system. The origin could be anything, such as simulations, or real world sensors. This raw data needs to be collected and stored somewhere so that it can be

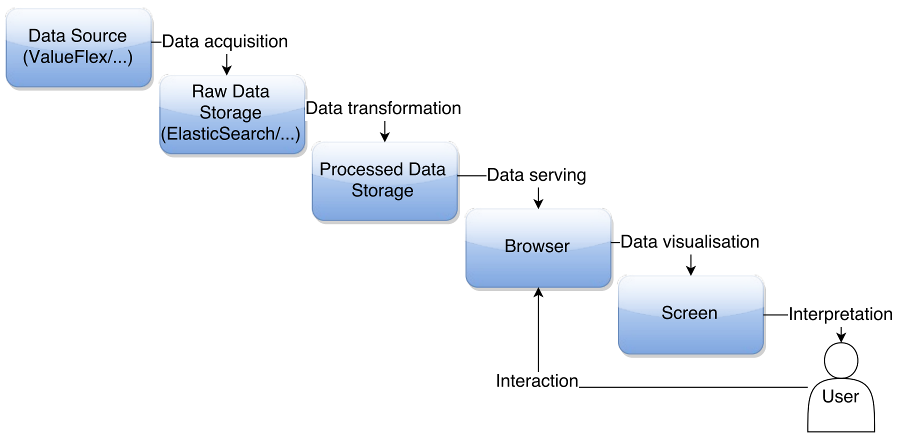


Figure 1: High level representation of visualisation pipeline.

processed. This processing step takes care of unifying different data formats into a single standard. At this stage the data must also be structured and simplified, so that a web client can easily request data subsets and aggregates. This processed data can then be requested by the user its web browser, where the data is visualised so that the user can interpret and analyse the data. Finally the user can interact with the web browser to further explore the data, updating the visualisation accordingly.

Not all stages of this pipeline will be discussed equally intensively in this work, as we primarily focuses on the later stages in this pipeline. A data source is assumed to exists, which can generate appropriate data, and the output of which is stored in some kind of database. In this work, appropriate visualisation techniques will be discussed first. Analysis is the main goal of this work, and the visualisations used strongly impact the ability of users to successfully perform this analysis. The chosen visualisations guide decisions in the other stages of the pipeline. Strongly related to the visualisations is the user interaction. User interaction is important to allow the user to explore the dataset, so that they can look in more detail at interesting subsets of the data. This is particularly important with a dataset this large, since subset selection will be required to be able to gain a detailed view of the data, as it is impossible to visualise all the data at once at higher detail levels. After deciding on visualisation solutions, the problem of transforming the data in a way that it can be effectively consumed by the web application will be discussed. Due to the size of the datasets considered in this thesis, finding a solution to this problem is very important for the success of this research.

1.2 PROBLEM STATEMENT

By analysing aspects of large scale geospatial temporal multivariate graphs which represent important infrastructures, humans can get a deeper understanding of any patterns, trends, oddities or problems in the behaviour of these large scale networks. This analysis requires visualisation, however visualising such large graph datasets is a challenging problem. The solution must be scalable, coping with the detail level of households but also the detail level of a country. The solution must allow the identification of problem areas, both problems that appear at specific points in time, as well as those that only appear when analysing a period of time. Finally, it must be generic and reusable for other similar datasets, and web compatible.

All these previous requirements leads to the following main research question: **"How can a large scale geospatial temporal multivariate graph be visualised, allowing interesting behaviours and pattern to be analysed, in a real time, interactive and generic way in a web based environment?"**

To be able to answer the research question, it is split up into multiple sub questions:

Q1 What visualisation techniques are suitable for visualising large scale geospatial temporal multivariate graphs?

Q2 How can large scale geospatial temporal multivariate graph datasets be processed and stored to allow the data to be retrieved effectively in a web application?

Q3 How can the proposed solutions for data processing and visualisation be implemented in a web application?

Q4 Do the used visualisation techniques allow data oriented questions to be answered effectively in an interactive way?

1.3 STRUCTURE

This thesis is structured as follows. Chapter 2 will attempt to answer Q1, by investigating visualisation requirements and discussing existing solutions. Based on that, a visualisation solution is proposed in Chapter 3. Next, Chapter 4 will discuss the data problem outlined by Q2. It will discuss the advantages and disadvantages of different solutions for aggregating, storing and serving large geospatial temporal multivariate graphs. After this Chapter 5 will discuss the details of the existing web visualisation framework CommonSense mentioned earlier, and how the solutions proposed in Chapter 3 and Chapter 4 can be integrated into it. This will answer Q3. Q4 will be answered next, in Chapter 6. Two different datasets will be analysed in this section, to show how non-trivial insights can be achieved using the tool created in this thesis. Chapter 7 summarises this work and will answer the main research question. This chapter is concluded by outlining some potential improvements and additions that could be investigated in future work.

2 TEMPORAL GEOSPATIAL MULTIVARIATE GRAPH VISUALISATION

In this chapter visualisation techniques appropriate for geospatial temporal multivariate graph data are discussed. To help determine which solutions are most appropriate, first some requirements are discussed in Section 2.1, using an example use case. Afterwards a selection of existing visualisation techniques is described and compared in Section 2.2, listing their advantages and disadvantages.

2.1 REQUIREMENT ANALYSIS

The data of interest in this project is structured as a graph, with its nodes and edges having attributes that vary over time. In this project the structure of this graph is assumed to be constant, where only the attributes changes. Visualising this kind of data is challenging [1, 2], since there is a huge amount of information related to a single object, a point or line, in the dataset. In this work we often refer to these objects as *features*, by which we mean any object that has a spatial location, this can be a point or a line, but also a polygon, including any properties or attributes that belong to that object, temporal or not. This is a typical definition of features in Geographic Information System (GIS) systems [5]. When displaying these features spatially, which is the only possibility in the current visualisation framework, visualising such a large amount of data on a single point is essentially impossible. At least not in a way a human could easily relate it back to the original data. Using a small area around each feature helps, although this can easily cause features to overlap, causing some features to be occluded. The exact location of the feature also has meaning, which can be harder to read when the drawn feature occupies more screen space than it actually relates to.

To clarify the problem even further, assume a technique exists that could visualise a single value on each pixel of a screen. With colour mapping this is a reasonable assumption. The datasets considered in this work can, worst case, be on the level of countries. In the case of the Netherlands, such a graph would have a number of nodes in the order of millions, or 10^6. Assuming just a single attribute for each node, and ignoring the structure and edges of a graph, 10^6 pixels are required for displaying this data. The most commonly used screen resolution as of writing is 1366x768 [6], which has only approximate 10^6 pixels, which would be roughly enough to visualise all this data simultaneously. This is however just the nodes without their connections, with just a single attribute, at a single time step. For the entire dataset, at this resolution, a single pixel would essentially have to display all the temporal data of all attributes of a single node in the graph, while somehow also showing the structure. Some datasets can have in the order of 10^5 time steps. Even the second most commonly used screen resolution, 1920x1080, has only approximately twice the number of pixels. Clearly, with current technology it is impossible to view all this data simultaneously at its highest level of detail. This thesis attempts to circumvent this problem by allowing the user to explore the data at a high level, with strong aggregation. Once a time period or sub-graph has been identified that exhibits interesting behaviour, more detailed information will be shown.

To help guide design decisions during this research, a specific data source was considered as a use case for the developed solution. In particular for the choice of visualisation techniques this is important, since the usefulness of a visualisation technique depends on the questions that one wishes to answer. Different techniques are used for viewing geospatial infrastructure than those used for detecting correlations between attributes. Other techniques are orientated towards the temporal aspect of data, allowing one to see patterns over time. No single visualisation technique exists that is able to clearly display all these aspects of a geospatial temporal multivariate graph dataset all by itself. Implementing many different visualisation techniques is unfortunately unrealistic however, given the time constraints of this project. Because of this the ValueFlex project will be discussed in this section. ValueFlex is another project of TNO. It provides an example use case for the analysis solution proposed in this work. The ValueFlex project is the primary use case for this solution. Based on the needs of this project, appropriate visualisation techniques are chosen. To provide context for this project, the concept of smart grids will be discussed first. While the chosen visualisation techniques are selected specifically to display the data of the ValueFlex project, Chapter 6 will show the solutions discussed in this work can be applied successfully to two other projects. Even when a new visualisation technique is required for a future project, the data transformation solutions discussed in this work should allow new visualisation to be added relatively easily.

2.1.1 *Smart grid*

Power consumption and private production has increased significantly in recent years [7], which puts more stress on the ageing electrical grid. More power than ever is being put back into the grid by prosumers owning devices such as solar panels and micro-CHPs, and, at the same time, higher peak consumptions occur. This is due to synchronous high peak demands from for example many electrical cars being charged simultaneously at the end of the day. The current electrical grid was not designed with these use cases in mind. With this increase in popularity of devices such as solar panels and electrical cars, the limits of components of electrical grids are being reached, overloading parts of these grids [8]. Since the demand on electrical grids is expected to only increase further [7], solutions for these problems are actively being researched. One solution would be to perform grid reinforcement, which involves replacing part of the electrical grid to satisfy the growing need for power, but there is an alternative. By modifying the grid such that peak loads can be distributed over a longer period of time, and dynamically increasing consumption to mitigate overproduction of energy, the current network can still be sufficient for our needs for the time being. Such an electrical grid that allows for this dynamic management of supply and demand of power is called a smart power grid.

A smart power grid, commonly referred to as a smart grid, is a power grid with advanced sensor and control capabilities, and integrated communications. This additional functionality provides opportunities for the supply and demand of power within a smart grid to be actively managed to maintain balance and avoid overloading the network. This balancing is done by managing so called smart devices within this network. These smart devices have flexibility, which means for example that they can be instructed to use more or less power on demand, or postpone their work to a later point in time. Consider for example an electric car. For most people the car simply needs to be fully charged again in the morning the next day, ready for use. The owner does not care whether the car is charged immediately when he comes home, or in the middle of the night, as long as it is fully charged the next morning. There is flexibility in the exact time frame in which the car is charged. Another example is a large cooled storage. Such a storage has a range in which the temperature must remain. When there is a power surplus this storage could be cooled a bit more than usual. At a later time the cooling system could be turned lower, during which less power will be used, allowing the temperature to increase again. There are many more of these smart devices with flexibility available, such as micro-CHP's, heat pumps and generators, but four types of devices are generally recognised [9].

- Uncontrollable. Devices that have no flexibility, but their production or consumption is measurable. The operation can potentially be predicted. Examples are solar panels, wind turbines, TV and indoor lighting.
- Time shiftable. Operation can be shifted in time, but has to be completed within a given time range. Examples are a washing machine, a dishwasher and an electric car.
- Buffer/storage. Flexibility in either the production or consumption, however the flexibility is bounded by a buffer. Examples are a freezer, a heat pump and a micro-CHP.
- Unconstrained. Devices that can be used completely on-demand, and are not limited by a buffer. Examples are generators.

By having a large system of such smart devices, power shortages and surpluses can be mitigated.

2.1.2 *ValueFlex*

To help get an idea of the actual benefits of using a smart grid and the consequences of using these smart grid solutions, the ValueFlex simulation framework was created. This framework is being developed by TNO. The ValueFlex project has spawned from their PowerMatcher [9] project, a smart grid management solution. ValueFlex simulates the electrical market, the power flow of the electrical network, the demand response system, and the production and consumption of power, making extensive use of models. Each of the components of the simulation generate output data. One of its outputs is the state of the electrical grid during the course of the simulation. This output can be analysed to detect electrical problems in the power grid. ValueFlex is able to simulate large scale electrical grids, over varying time periods ranging from just a few days to multiple years or potentially even a decade. The output from this power flow analysis is essentially structured as a graph, where nodes represent for example substations or generators, and edges represents cables. Each of these components have attributes describing their state. Some instances of these attributes are the voltage, amperage, active or reactive power, resistance and phase angle. Each of these attributes are recomputed for every time step in the simulation. This results in a multivariate temporal graph dataset. Due to the size and complexity of such a dataset, some kind of visualisation, or a set of visualisations is required to analyse it effectively.

An electrical grid can be seen as an geospatial temporal multivariate graph. In ValueFlex, the network is described in terms of generators, buses and branches. The nodes of the graph are the generators and buses of the network, and the branches between them are the edges. These objects have a physical location, and multiple attributes which change over time, making it a geospatial temporal multivariate graph. As such a electrical grid is a good example of the kind of datasets considered in this work.

There are three types of analysis that are desirable to analyse this type of data:

- Value oriented analysis
- Time oriented analysis
- Comparison oriented analysis

VALUE ORIENTED ANALYSIS First of all, it is interesting to analyse the performance of the grid by searching for behaviour purely based on specific attribute values. For example a user might be interested in knowing how their network behaves on a very sunny day, or when all families in a neighbourhood use electric cars. In these cases it is important to see how the grid performs compared to its safe limits. Many of the problems one might encounter in an electrical grid are discussed by Dugan et al. [10]. As discussed previously, one kind of problem can be the network being overloaded by too much simultaneous demand of power. This can for example be indicated by power levels crossing an upper limit on components in the grid. Other problems can also occur, such as a deviating voltage levels. Most European countries use a voltage of 230V, but allow for a deviation of 10% in either direction. If the voltage deviates too far from the standard value, this is a potential power quality problem. It may cause electrical devices to malfunction or even be damaged. Because of this it is important for the voltage to stay within safe margins. The voltage may become lower, also known as a voltage sag [11], as a result of a sudden increase in the load. The opposite can also occur, a voltage swell, where the voltage actually increases past the normal value. This can occur in the reversed scenario, where a large load is suddenly disconnected. Because of these problems, being able to view where and when specific values occur, in particular extremes and values relative to safe limits, is desirable functionality.

TIME ORIENTED ANALYSIS Other problems can be detected by analysing time ranges. Constant fluctuations in the voltage can be problematic. They can cause temperature changes that deteriorate cables, even when the fluctuations themselves remain within safe margins. Eventually this can lead to failures. Commonly this kind of problem can be observed by humans by flickering lights, and its most common causes is arc furnaces [10]. To detect these problems in the output of ValueFlex, one would have to look at the behaviour of components of the grid over time, as opposed to looking at single values as needed to find the problems discussed above.

COMPARISON ORIENTED ANALYSIS Finally another interesting type of analysis is that of comparing the effectiveness of using a demand response solution versus the same network without the demand response solution in place. This is of interest to distribution network operators who need to decide whether to invest in this technology on the grid they manage, or to employ grid reinforcement. To help decide it is useful to be able to see which parts of the network are affected, and how these parts are impacted by such a change. For example, it would be interesting to see if a part of the grid which is showing problems in a specific scenario, copes better in that same scenario with a load balancing solution in place. To do this type of analysis a solution is required which allows multiple simulation runs to be compared.

2.2 EXISTING SOLUTIONS

Many different visualisation solution already exist. In this section some interesting visualisation techniques, which could be applied to the geospatial temporal multivariate graph datasets, are discussed. These visualisations primarily focus on dealing with the temporal aspect of the data, as the spatial and relational aspect is largely dealt with by the existing interactive map in the existing framework. Unfortunately no single ultimate visualisation exists that is able to answer all questions a user might have about the data, each visualisation technique has its strengths, but also its weaknesses. Each of the visualisation shall be compared by their ability to deal with the three following challenges:

DATA SIZE The datasets considered in this work are large in size. Not only does it consists of many features, but each of the these features has multiple attributes, which are also recorded at many different time points. For a visualisation to be useful, it should be able to deal with large multivariate datasets.

data heterogeneity The attributes of these features are also of different types. The features have spatial attributes, such as the geographical position, but also categorical attributes. For example, a pipe in a gas network can be suited for low, medium or high pressure. Others are temporal in nature, or describe a relation, i.e. connection, between features. When aggregating features, it is hard to aggregate all these different types of attributes. Most visualisation techniques avoid this problem by only focusing on one type of attributes.

interactive exploration As discussed before, this work focuses on the questions of users that can not be easily computed into a single number, but that require exploration to answer. This means the visualisations need to allow interaction, so that the user can search for the unknown such as outliers, correlations and verifying hypotheses.

For each of the visualisation techniques mentioned below these challenges are discussed.

2.2.1 *Temporal graph visualisation*

Solutions do exist for the visualisation of temporal graph datasets. Such graph visualisation techniques can be divided into two groups, those that utilize *unfolding*, and those that utilize *animation* [12]. In the former case, the time axis is unfolded along a spatial axis, as is shown in Figure 2, a solution presented in the work of Erten et al.. This particular example attempts to display the time dimension of the data by rendering in 3D. In this visualisation each time step is visualised as a separate graph, and each of these graphs are stacked on top of each other. Corresponding nodes in the different layers are connected by edges. Essentially a 3D graph is created by connecting each of the graphs from each timestep into one larger graph. Unfortunately, this method does not scale well to large graphs or large periods of time.

While it would be possible to display temporal geospatial multivariate graphs datasets with this technique, due to the large data size there would be large amounts of occlusion, making it impossible to gain an overview of the data. As it gives no clear overview exploring the data will be very hard, as a user would have to zoom in to be able to see anything at all. The number of attributes that can be displayed using this technique is also limited. In Figure 2 only the size of the nodes is used to indicate the magnitude of an attribute. Although an additional attribute could be displayed using the colour of the nodes, this still only allows for two different attributes.

Animation, another commonly used visualisation technique, is currently being used in CommonSense. Unlike the technique described above, animation scales incredibly well with large time ranges. However, to be able to perform the time oriented analysis described previously a user would have to memorize temporal patterns in order to compare different parts of the animation. As time ranges are large and detailed in the datasets considered in this thesis, this ask the user to remember a large amount of information. This makes solutions that rely on animation less suitable for the purposes of this work.

2.2.2 *Table lenses*

Another visualisation technique is table lenses [14, 15, 16, 17]. A table lens visualises data as if it were a single large table, where each row consists of a single observation, and each column contains an attribute of these observations. An observation in this case could be a single node in a graph. When viewing the table while zoomed in, the table contains the actual values of these attributes as regular numbers. However, when zoomed out, instead

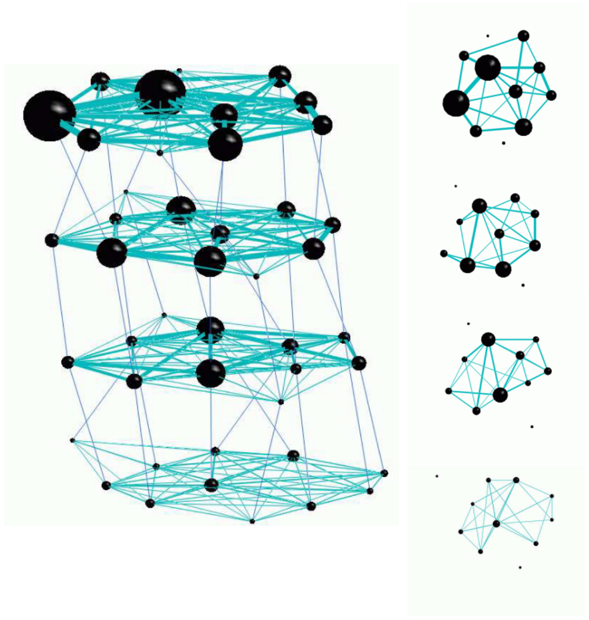


Figure 2: Example of a temporal graph visualisation. A graph of each time step is stacked on top of each other, resulting in a 3D graph. The separate layers are shown on the right. *Source: Erten et al. "Exploring the computing literature using temporal graph visualization"* [13]

of displaying the actual attribute as a number in this table, the attributes are displayed in the form of bar, scaled according to the magnitude of the value. This way this solution offers a high level overview, while also offering the ability to look at details. An example of this technique is shown in Figure 3 from the work of Telea. This technique is useful for finding those observations where attributes take on interesting values. For example, it could be used to find the nodes in the electrical grid where the voltage is too low, by sorting the column corresponding to the voltage attribute. As can be seen in the figure, it is only possible to visualise a limited number of columns, or attributes, since screen space is limited. The user could be given the ability to choose attributes of interest to solve this, since not all attributes are necessarily interesting to visualise at the same time.

This technique, when preserving a one-to-one mapping, is however limited by the number of rows available in the table, which is determined by the number of vertical pixels available on the users screen. If more table rows than there are pixel rows must be shown, multiple rows have to be aggregated into a single row of pixels. This aggregation could be something simple such as the average or maximum value, but this might suppress interesting outliers. As such a more advanced technique could be used instead that highlights relevant details. For example, Holten et al. [18] describe an approach that weighs rows based on how frequently they appear in their local neighbourhood. By using such aggregation techniques to render multiple rows on a single pixel row, table lenses can potentially scale to a very large number of rows. Once an interesting set of rows has been identified, the user can zoom in to view the raw data. This way this visualisation is able to cope with the large data sizes presented in this work, and allow for effective exploration. It however achieves this at the cost of only focusing on some type of attributes, ignoring for example relations present in the graph.

Temporal multivariate graphs can be mapped onto this table lens visualisation in a number of ways. Each row could represent a single node or edge, where each column contains an attribute of the feature, aggregated over the entire time range. This potentially allows interesting features to be detected, as extreme values in attributes can be detected by sorting on columns. The downside of this approach is that all temporal information is lost, removing any details such as spikes from the data. Since the goal is to analyse the

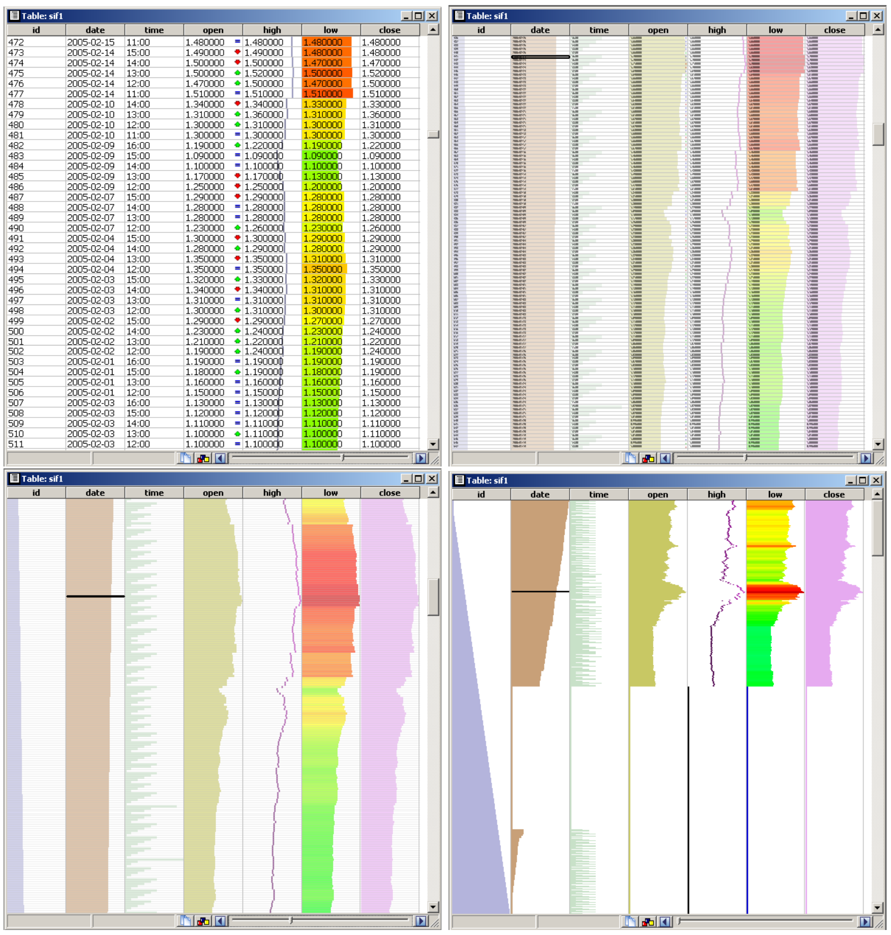


Table: sif1

id	date	time	open	high	low	close
472	2005-02-15	11:00	1.480000	= 1.480000	1.480000	1.480000
473	2005-02-14	15:00	1.490000	▼ 1.490000	1.480000	1.480000
474	2005-02-14	14:00	1.500000	▼ 1.500000	1.470000	1.470000
475	2005-02-14	13:00	1.500000	▲ 1.520000	1.500000	1.520000
476	2005-02-14	12:00	1.470000	▲ 1.500000	1.470000	1.500000
477	2005-02-14	11:00	1.510000	= 1.510000	1.510000	1.510000
478	2005-02-10	14:00	1.340000	▼ 1.340000	1.330000	1.330000
479	2005-02-10	13:00	1.310000	▲ 1.360000	1.310000	1.360000
480	2005-02-10	12:00	1.300000	▲ 1.310000	1.300000	1.310000
481	2005-02-10	11:00	1.300000	= 1.300000	1.300000	1.300000
482	2005-02-09	16:00	1.190000	▲ 1.220000	1.190000	1.220000
483	2005-02-09	15:00	1.090000	= 1.090000	1.090000	1.090000
484	2005-02-09	14:00	1.100000	= 1.100000	1.100000	1.100000
485	2005-02-09	13:00	1.170000	▼ 1.170000	1.130000	1.130000
486	2005-02-09	12:00	1.250000	▼ 1.250000	1.200000	1.200000
487	2005-02-07	15:00	1.290000	▼ 1.290000	1.280000	1.280000
488	2005-02-07	14:00	1.280000	= 1.280000	1.280000	1.280000
489	2005-02-07	13:00	1.280000	= 1.280000	1.280000	1.280000
490	2005-02-07	12:00	1.230000	▲ 1.260000	1.230000	1.260000
491	2005-02-04	15:00	1.300000	▼ 1.300000	1.290000	1.290000
492	2005-02-04	14:00	1.280000	▲ 1.290000	1.280000	1.290000
493	2005-02-04	13:00	1.350000	▼ 1.350000	1.310000	1.310000
494	2005-02-04	12:00	1.350000	= 1.350000	1.350000	1.350000
495	2005-02-03	15:00	1.320000	▲ 1.330000	1.320000	1.330000
496	2005-02-03	14:00	1.340000	▼ 1.340000	1.310000	1.310000
497	2005-02-03	13:00	1.310000	= 1.310000	1.310000	1.310000
498	2005-02-03	12:00	1.300000	▲ 1.310000	1.300000	1.310000
499	2005-02-02	15:00	1.290000	▼ 1.290000	1.270000	1.270000
500	2005-02-02	14:00	1.230000	▲ 1.240000	1.230000	1.240000
501	2005-02-02	13:00	1.210000	▲ 1.220000	1.210000	1.220000
502	2005-02-02	12:00	1.190000	▲ 1.240000	1.190000	1.240000
503	2005-02-01	16:00	1.190000	= 1.190000	1.190000	1.190000
504	2005-02-01	15:00	1.180000	▲ 1.190000	1.180000	1.190000
505	2005-02-01	13:00	1.160000	= 1.160000	1.160000	1.160000
506	2005-02-01	12:00	1.150000	= 1.150000	1.150000	1.150000
507	2005-02-03	16:00	1.130000	= 1.130000	1.130000	1.130000
508	2005-02-03	15:00	1.120000	= 1.120000	1.120000	1.120000
509	2005-02-03	14:00	1.110000	= 1.110000	1.110000	1.110000
510	2005-02-03	13:00	1.100000	▲ 1.110000	1.100000	1.110000
511	2005-02-03	12:00	1.100000	= 1.100000	1.100000	1.100000

Figure 3: Example of a table view enhanced using the table lens technique, zooming from the level of text (top left), to an overview of the whole table (bottom right). *Source: Telea "Combining Extended Table Lens and Treemap Techniques for Visualizing Tabular Data"* [17]

temporal data, as this is currently hard to do with the interactive map offered by the existing framework, effectively throwing away the temporal data is not a good solution.

A different way to map this data would be to have a column which contains, for each row, the timestamp at which the data of the row was recorded. A similar approach is used in Figure 3, although in this case the date and the time are separated into two columns, and each combination of a date and a time is unique to one row. Timestamps would not be unique in a temporal multivariate graph, since the entire graph is defined at each time stamp. When each feature of a graph is mapped to a separate row, multiple rows with the same timestamp would exist. Assuming stable sorting, by having a column with timestamps, a user could sort the rows on the features they originate from, grouping all rows that belong to a single feature, and do a second sort on time. This would then show how each feature behaves over time for each of the attributes. As discussed before, a graph could have 10^6 features, with 10^5 timesteps. This will mean there will be in the order of 10^{11} rows. To gain an overview of this data, the user would have to zoom out very far. However, at this point there would be many more rows than there a pixel rows available. In fact there would be so many rows, that after aggregating them, all rows belonging to a single feature would be mapped to the same row in the table lens. The result is that the same image is acquired using this mapping as was achieved using the previous mapping. The only difference is that when zooming in, the user is able to view the temporal behaviour of a limited number of features.

Another mapping would be to map a timestamp to a row, as opposed to a feature. Each row would then present an aggregate of the entire graph at that point in time. This essentially gives a set of vertical graphs of the aggregated attributes of the entire graph over time, with each column containing one graph. The same could be accomplished with the previous mapping, by sorting on time and zooming out till all rows representing data of the same timestamp is aggregated into a single line. This visualisation allow the user to see where in time interesting behaviour occurs, at the level of the entire graph. For example, if a large portion of the electrical grid suffers from a large demand of power, this will show up in the visualisation. Small details in the attributes in the graph are however lost, and a localised problem in an infrastructure might be lost in this visualisation.

2.2.3 *Parallel coordinates*

A different approach, which focuses on correlations, is parallel coordinates [19, 20, 21]. In this approach, a vertical line is drawn for each attribute to be analysed. Next, for each observation a point is plotted on each of these vertical lines, creating one-dimensional plots on each line. Finally, per observation, all points are connected by a polyline. The result is one such line for each observation in the dataset. An example of this, as created by Robert Kosara, is shown in Figure 4. The main strength of this approach is the ability to find correlations between attributes. Where two neighbouring attributes are strongly correlated, many parallel lines will appear, as shown between the 3rd and 4th vertical line in Figure 4. When they are inversely correlated, the lines will form a cross, as happens between the first two and the last two vertical lines in the example figure. Uncorrelated attributes will result in the lines appearing to orientate randomly, filling up the area between the two vertical lines.

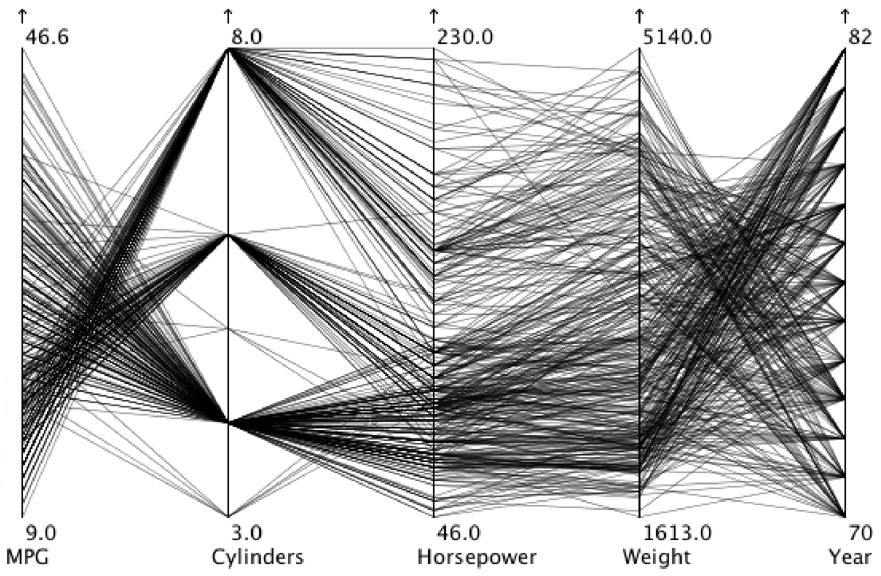


Figure 4: Example of a parallel coordinates plot, showing the correlations between different attributes of cars. *Source: Robert Kosara,* `https://eagereyes.org/techniques/parallel-coordinates`

Parallel coordinates however do not easily incorporate the time aspect present in the data without aggregating over a dimension. A similar approach as suggested for the temporal problem with table lenses can be used, where instead of visualising individual components of the electrical grid, a single timestep is aggregated over the entire graph to become a single observation. Then one line in parallel coordinates can be a single timestep. Additionally these lines could be coloured to indicate their age, allowing older and newer observations to be distinguished, although this may become a bit chaotic when viewing a very large time range with many timesteps.

This technique scales well with many observations. With large numbers of observations, while the lines will overlap, the patterns that indicate correlations will remain visible when strong enough. A problem however is that these correlations are only visible between neighbouring attributes. Because of this the usefulness of this approach is strongly tied to the order of the columns. This problem can be alleviated by allowing the user to re-order the columns, but it is not reasonable to expect the user to try out all possible permutations just to find these correlations. Some prior knowledge might make this task somewhat more reasonable for the user.

Whether this technique is useful strongly depends on the data being analysed, and the questions a user wishes to answer. The visualisation focuses mostly on clarifying the relations between attributes, and individual observations are lost as a result of all the overlapping lines. In the case of ValueFlex, this may not be the most suitable visualisation, since the questions outlined in Section 2.1.2 are more focused towards the values of attributes, and their temporal behaviour, instead of correlations. This may however be a useful visualisation solution for other projects.

2.2.4 *Multidimensional projections*

Another way to visualise the data is to utilize multidimensional projections. The idea of this approach is to consider the observations to be points in an N-dimensional space. The similarity of observations can then be described by the distance between these points in the N-dimensional space. Since there are currently no techniques to display or view a high dimensional dataset, these high dimensional points are projected onto a low number of dimensions, such as two or three. This low dimensional data can then be visualised by using, for example, a simple scatter plot [22], as shown in Figure 5 from the work of Paulovich et al. The challenge is to project these points in such a way that the distances in the high dimensional space are preserved as much as possible in the low dimensional space. Martins et al. [24] give a survey of several recent dimensionality reduction algorithms.

This technique is primarily useful for finding relations between observations. Clusters indicate similarity, elongated structures indicate correlations, and outliers indicate deviating observations. This may be useful for finding nodes, subgraphs or time steps where interesting behaviour occurs. To be able to use this technique however, a mapping needs to be found to map the the graphs considered in this work onto this N-dimensional space. There are numerous ways to do this. One mapping would be to consider the attributes of a point in the N-dimensional space to be all the attributes of all the components of an electrical grid at one time point. This could be done for all components, or just one type, for example the nodes, and then having such a multidimensional projection per component type. This way time steps can be compared to each other. A different mapping would be to instead consider a high dimensional point to represent all the values of all the attributes of a single component over the whole time range. This mapping would visualise the similarity of different components with respect to their behaviour over time.

This technique is very scalable, since points can be rendered very small if necessary. It can also easily deal with many attributes, even those of different types. The results are however fairly abstract. While it shows that some observations for example are similar, it does not show why they are similar. Some recent research [25] has attempted to explain the clusterings in a multidimensional projection by, by colouring these clusters by the attribute most similar within the cluster, and labelling them accordingly. However, the results presented in their work describe datasets with only a limited number of attributes, each having only 12 metrics. While the colouring by attribute

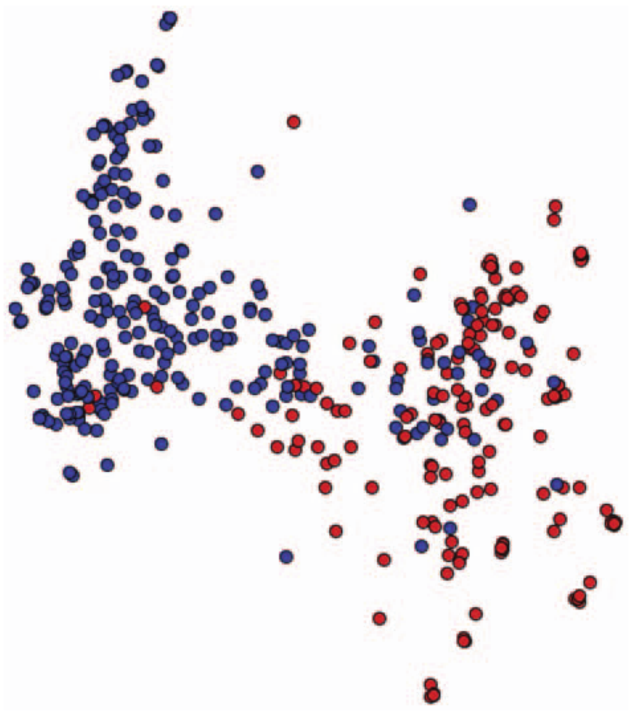


Figure 5: Example of a scatter plot of points projected from a high dimensional space. In this example, two clusters appear in the projected data. *Source: Paulovich et al. "Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping"* [23] © 2008 IEEE

can be done as performed in their work, the labels would be meaningless in a scenario with one of the mappings discussed above, where an attribute refers to the value of an attribute of a feature at one point in time. Thus the resulting clusters are still not explained.

2.2.5 *Calendar views*

A quite different visualisation technique is that of calendar views [26]. The primary strength of this visualisation technique is to view patterns over time. Figure 6 shows an example of a calendar view from the work of Van Wijk and Van Selow. It describes the number of present employees over a time period of a year. To construct this visualisation the time range is split into days, and the days are clustered in terms of their similarity. Each cluster has its own colour assigned to it, and the days belonging to a cluster are coloured accordingly in the calendar part of the view. On the right the average value over the duration of a day is shown for each of the clusters.

The strength of this approach lies in the fact that it becomes possible to easily spot patterns on varying time scales. In the graph section, a clear pattern emerges within a day, where people arrive around 9 AM and leave around 4 PM. But from the calendar part, it also clearly shows that, as expected, attendance is very low during the weekends, and on Fridays significantly less employees are present. Seasons also clearly appear, as there for example are less people present during the summer.

This technique can deal very well with a large time scale, but it does require it to be meaningful to divide the time range into equal chunks such as months, weeks and days. For datasets such as those generated by ValueFlex, this is most likely the case, since for example the production of solar panels varies in a fairly consistent pattern each day. Industries will also use less energy during the weekends. Problematic however is that the graph visualisation used is not particularly well suited for the data available in this project. In the dataset shown in Figure 6 a single attribute, the number of

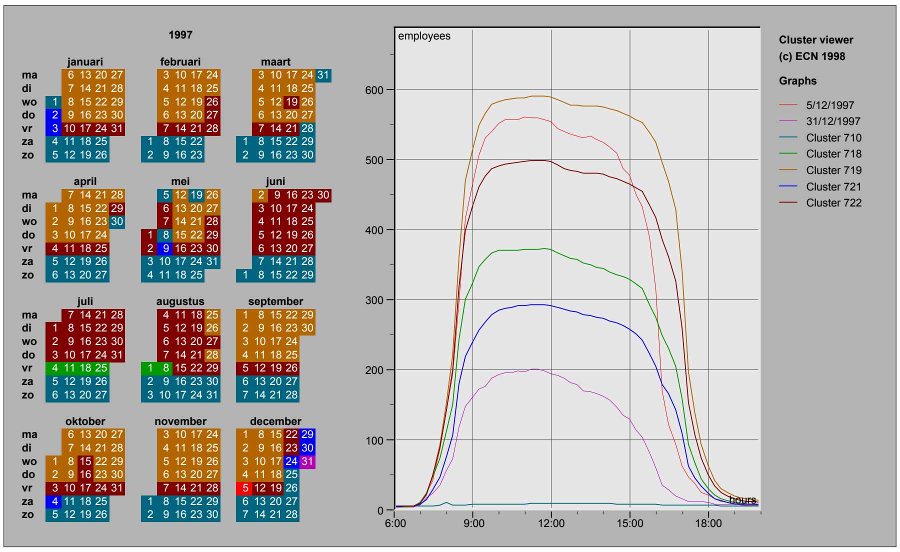


Figure 6: Example of a calendar view, showing the number of present employees over the course of a day. *Source: Van Wijk and Van Selow "Cluster and calendar based visualization of time series data"* [26] © 1999 IEEE

present employees, is displayed at each timepoint. In the case of a temporal graph, the state of an entire graph is recorded for each time step. Because of this, some strong aggregation would again be required to be able to use this visualisation in its current form.

2.2.6 *Evolution spectrographs*

An evolution spectrograph [27], an example of which is shown in Figure 7 from the work of Wu et al., essentially visualises a matrix by colour mapping its values. In the given example it is used to visualise the history of a code repository. Each row represents a file, with each column showing the value of an attribute of that file at a point in time. Such a row is similar to a sparkline, a small line chart without axes or coordinates. By putting many of these rows together a small multiple is created. In Figure 7 the shown attribute is the change in the number of incoming dependencies of a file. This clearly shows when the files are created, and how they evolve over time. A similar technique has been used to visualise genome expression data [28]. A temporal graph could also be mapped well onto this visualisation, where each row displays the temporal behaviour of a single attribute of a feature, such as a node. This gives a highly detailed overview of the temporal behaviour of the entire graph. Unfortunately, as is the case with all previous visualisations, the size of the dataset is too large to visualise all the data without aggregation. Multiple rows will have to be rendered to a single pixel, and the same holds for the columns. Zooming and panning controls could be added to allow the user to retrieve the unaggregated data in an area of interest, showing a limited number of rows or columns using a one-to-one mapping with pixels.

A benefit of this visualisation is that it has a very high information density, potentially using every pixel of its drawing area to display information. For example, the table lens visualisation uses the width of a column to display a single value. Such a bar will use multiple pixels to display a single value. In the case of an evolution spectrograph, when the number of rows and columns match or exceed the number of pixel rows and columns, a single value will be mapped to each pixel, maximizing the use of screen space. The drawback of this is that, as a user, the value indicated by a colour is generally harder to read and compare than one represented by the length of a bar.

This techniques also suffers from details being lost to aggregation. If a single row has a single point in time where an extreme value occurs, this spike can be lost in the overview if simple averaging is used. Ideally the user is allowed to choose the aggregation method, so that, for example, extreme values can be highlighted by taking the maximum or minimum instead of the average. Another downside of the evolution spectrograph is that it is only able to show a single attribute at a time.

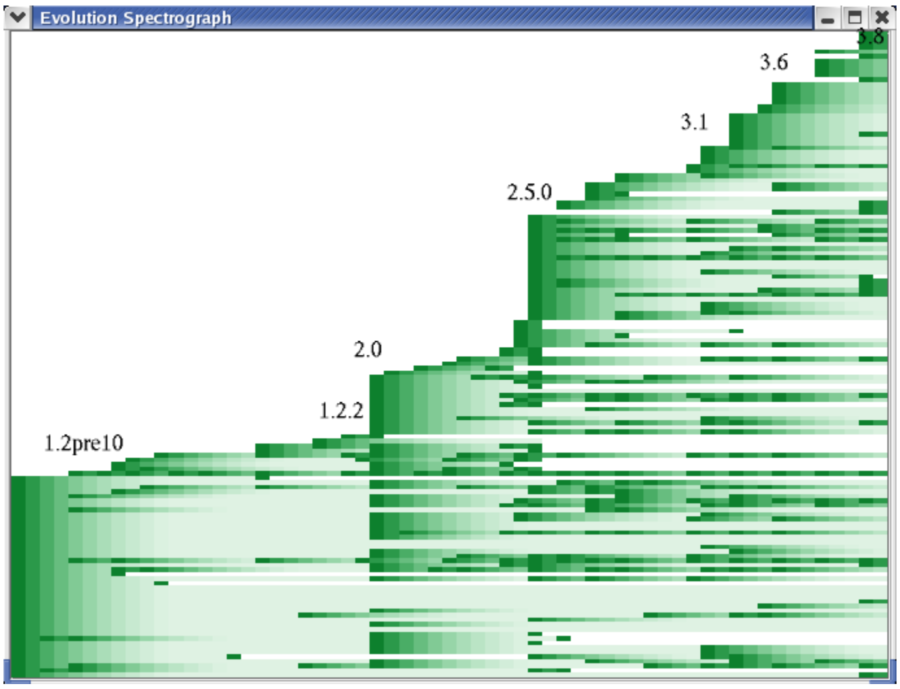


Figure 7: Example of an evolution spectrograph. *Source: Wu et al. "Evolution spectrographs: Visualizing punctuated change in software evolution"* [27] © 2004 IEEE

2.2.7 *Linked views*

While linked views are not a visualisation by themselves, they are a good way to strengthen the exploration capabilities of visualisations. The idea is to link multiple visualisations, or views, of the same dataset together. An early example of this approach can be found in the work of Becker et al. [29], but linked views are a very common technique used in visualisation applications [30, 31, 32, 33]. In the work of Becker et al., a matrix of scatterplots is used, where each scatterplot has a different combination of attributes. The tool then allows the user to select points in any of the scatter plots, which results in those points to be highlighted in all of the scatter plots. This is the main concept of linked views, where an interaction such as selection or filtering affects all the linked views.

Linked views are great for exploring a dataset when used with a diverse set of visualisations. For example, one of the time orientated visualisations such as the evolution spectrograph, could be used to find a point in time that exhibits interesting behaviour in some nodes in the graph. By selecting this time point on the spectrograph, this could in turn cause a spatially oriented visualisation to show this same timepoint, allowing the user to see how the interesting behaviour is spread over the spatial structure of the graph. Another possible link would be to have the ability to select a node, or perhaps a subgraph, which is then visualised or highlighted in a temporal orientated visualisation. This allows the user to see how this node or subgraph behaves over time. Especially in the case of a temporal multivariate graph, where it is impossible to visualise all its aspects simultaneously without any overlap, these kind of interactions are important to allow a user to explore the data effectively.

2.3 DISCUSSION

No literature appears to exist regarding the visualisation of large scale temporal geospatial multivariate graphs, thus in this chapter we discussed how some commonly used visualisations could be applied to our data. The spatial visualisation already present in the visualisation framework is a powerful tool for analysing the structure of a graph, and relating data to physical locations. It is however very limited in its ability to explore temporal data. The analysis requirements posed by the ValueFlex project are primarily orientated towards exploring and analysing time ranges. Thus we discussed a number of existing visualisations that could be used to help perform the types of analysis desired for the ValueFlex use case. Different visualisations are however best suited for different tasks. Table lenses, calendar views and evolution spectrographs are oriented towards showing large time ranges, while parallel coordinates and multidimensional projections allows correlations to be detected. None of these visualisations can however be applied directly to our data. To use them we came up with a number of possible ways to map the data onto the visualisations, the choice of which has a strong impact on what the resulting visualisation looks like. In the next chapters we will show how the visualisation techniques discussed in this chapter can be used, and extended, to create a solution that supports the analysis features desired in the ValueFlex use case.

3 VISUALISATION DESIGN

As an analysis framework for large scale temporal geospatial multivariate graphs, this work proposes to use a set of linked views. Due to the many aspects present in the data, entire datasets can not be displayed effectively using a single visualisation. However, by connecting visualisation that each focus on a different aspect of the data, each can be used to help explore and refine the search domain in another view. For example, when trying to find parts of an electrical grid where overloading occurs, one can first find a time slice of the dataset where problems occur using a temporally orientated visualisation. After this, this time slice could be viewed in a spatial orientated view, where the problematic components can be identified spatially.

In this chapter a design for an analysis platform for the large datasets discussed previously is outlined. First a set of visualisations which we believe are appropriate for the visualisation of these datasets is discussed in Section 3.1. They focus on conveying spatial information, structure, attribute values and temporal behaviour. Next the interactions that should be available to a user are discussed in Section 3.2.

3.1 VISUALISATIONS

We propose to use two linked visualisations to analyse large scale temporal geospatial multivariate graphs. The first visualisation is an interactive map, which focuses on the spatial information and structure present in such graphs. The second visualisation is more oriented towards showing the temporal aspect of these datasets. Together we think these visualisations allow large scale temporal geospatial multivariate graphs to be viewed and analysed effectively. Both of these techniques will be discussed in the following sections.

3.1.1 *Interactive map*

We think a rendering of the graph on a map representing the real world as a part of the analysis tool is important. Such a map would show the real world positions of elements in the graph, as well as how these elements are interconnected. Fortunately, as mentioned previously, such a map visualisation is already present in the CommonSense framework, an example of which is shown in Figure 8. It shows a map of the real world, on top of which a road network is rendered in black. A map rendering offers a very intuitive way for users to view the structure and spatial positioning of an infrastructure. Understanding these relations may be crucial in understanding behaviours present in a dataset. For example, strong correlations between different parts of a graph could be explained by these parts being spatially close to each other. Or conversely, when two distant objects are behaving in a correlated fashion the explanation may be simply that the two elements are connected directly. Thus being able to understand how parts of a graph are spatially located and interconnected is important for analysing these datasets.

Such a drawing of the infrastructure on a map opens possibilities for showing additional information, such as attributes, using a number of simple techniques. For example, a common occurrence in these types of networks is that not all parts of it are of equal type. In the case of a power grid, some parts may be rated for high voltage, while others only deal with low voltage. They may also serve different uses, as some nodes may represent generators while others display the location of a transformer. Being able to easily distinguish these different types of elements would allow a user to

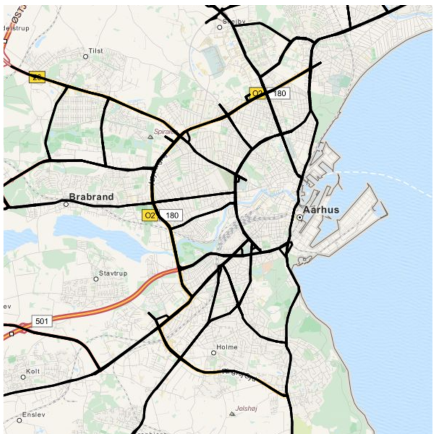


Figure 8: Example of a graph structure drawn on an interactive map, allowing it to be related back to real world locations.

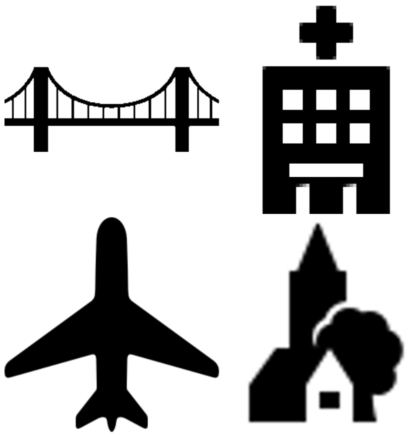


Figure 9: Examples of icons present in the CommonSense framework, used to indicate the locations of bridges, hospitals, aeroplanes and towns.

better understand the infrastructure they are viewing. What type of object a feature is, is simply a categorical attribute. In the case of nodes, a simple way to visualise such an attribute would be to display an icon on the location of it. For electrical grids standard symbols exist for distinguishing different elements [34, 35, 36], the use of which creates a convenient link to previous knowledge of users. Some icons are already being used in CommonSense, such as those shown in Figure 9, which are used to indicate the locations of bridges, hospitals, aeroplanes and towns.

More techniques can be used to show additional attributes on such a map rendering. CommonSense currently offers three techniques to show attributes. First of all the icon discussed above can be colour mapped to indicate the value of an attribute at that location. Secondly, the boundary around the icon can be coloured. Finally, also the thickness of the outline can be varied, allowing for another attribute to be visualised. While these last two techniques can also be applied to display attributes on top of the lines, since there is no icon to colour on lines, it is only possible to visualise two attributes on a line. It should however be noted that not all three indicators are equal in how well they can convey information, and they can conflict with each other. While a coloured area can be read fairly well with a legend if a good colour map is used, this is much harder when looking at the thickness of the outline. If difference are large enough it is possible to compare two different locations, but it is impossible to get an exact reading of the absolute value at some location, since a legend will be hard to create for this indicator. Care must also be taken when colour mapping information onto the icon and outline of a feature. The same colour map could be used, but this results in the same colour having multiple meanings. Care must also be taken when using multiple colour maps however, such as two colour maps on both the nodes and the edges. It is easy to create an overlap in the colour maps used. A user can easily get confused in this scenario, and as such using all the indicators at the same time is not advised.

Unfortunately these techniques all only show a single value, making them unsuitable for showing how temporal attributes behave over time. At best these techniques can show a single slice of time, which is described by a single, possibly aggregated value. However, by offering the user a way to choose which time slice to visualise, using these attribute mapping techniques is still useful for understanding the state of an infrastructure at a given point in time. For finding a point in time that exerts interesting behaviour, or for analysing temporal behaviours, the second visualisation described next should be used.

The datasets considered in this work are temporal in nature, while the map visualisation can only show three values at the same location, using which only a single time slice of the dataset can be shown. In theory the temporal aspect of the data could be explored using just this visualisation, by constantly moving back and forth through time, which results in an animation. However, without information about how the data behaves over time, the user can not do anything but simply pick time slices at random, looking for a time slice at which interesting behaviour appears. Patterns over time will be close to impossible to view, as that would require a user to remember previously viewed slices. Considering the quantity of data considered in this thesis, that would be very challenging. To help guide the user to a point in time in which they are interested, and view temporal patterns in the data, the evolution spectrograph is proposed.

The strength of the evolution spectrograph lies in its ability to give a detailed temporal overview of the data. Since all elements of the dataset are shown simultaneously, connections in their temporal behaviour can easily be detected. Imagine an area in an electric network where all nodes always simultaneously demand more power. This could be an industrial area where multiple factories depend on each other. Such behaviour will show up with multiple rows showing simultaneous spikes, allowing this kind of correlation to be easily identified.

To map the data of a graph a simplification of the data is required. For the spectrograph to be easy to understand, data rendered on it must be consistent. This means that each row should represent similar data, i.e. if one row displays the voltage, the next row should not show the resistance of a cable. Because of this the spectrograph can only show a single type of features, and only one attribute which must be present on all of the features shown. The same attributes are often not present on both the edges and nodes of a graph, thus only either of the two can be shown. While this may limit the analysis potential of this visualisation somewhat, generally different types of features are not comparable, thus nothing is lost. If desired, multiple spectrographs could be used simultaneously, each showing a different type of feature.

The visualisation as described by Wu et al. is adapted slightly for the purposes of this project. One change made to the spectrograph as discussed by Wu et al. in this project is to render using linear interpolation, instead of the nearest neighbour interpolation technique used in Figure 7. The values of the attributes, in reality, will be changing constantly over time, and not suddenly update on a set interval. Rendering the data as if these sudden changes actually occur may confuse the user, thinking that is the actual behaviour of the data. While the results of linear and nearest neighbour interpolation will be very similar with the incredibly high data to pixel ratio in the ValueFlex scenarios described previously, the significance of the need of this interpolation will become more clear in Section 3.1.3. An example of the result achieved by applying these changes is shown in Figure 10.

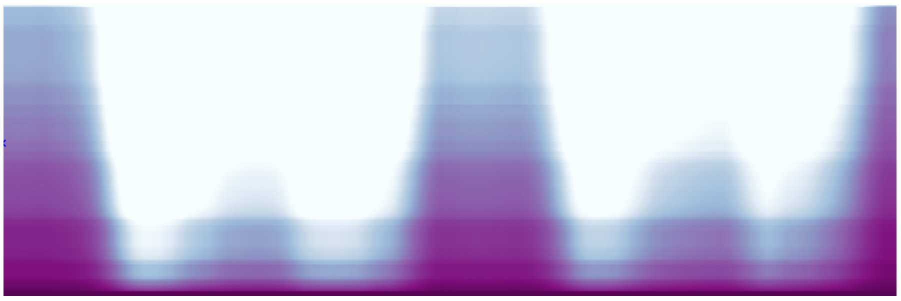


Figure 10: Example of an evolution spectrograph using linear interpolation.

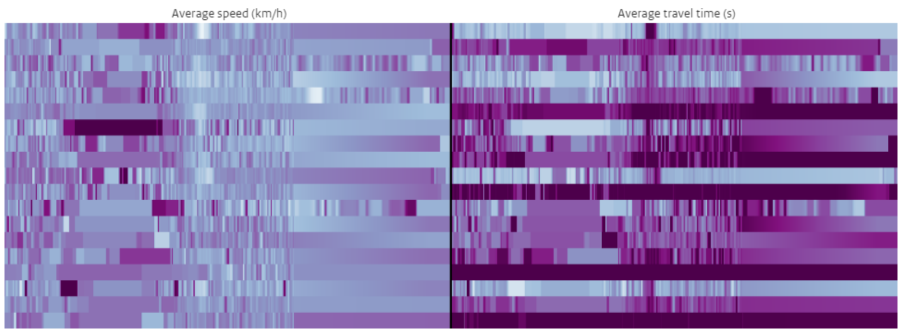
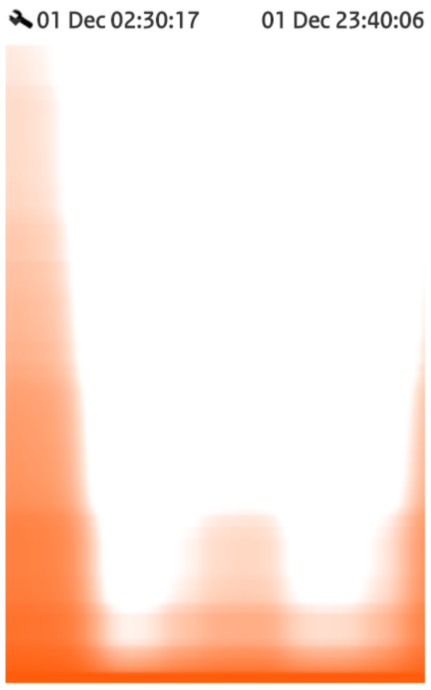


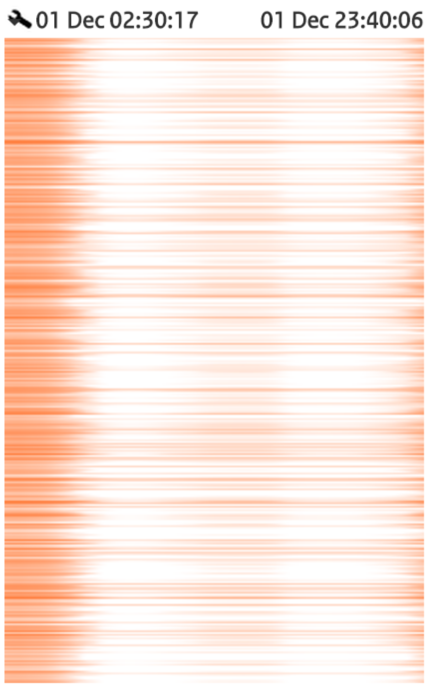
Figure 11: By drawing multiple spectrographs side by side we can visualise multiple attributes, and look for correlations.

The evolution spectrograph as presented by Wu et al. has another downside. As mentioned before it only allows for a single attribute to be visualised. The visualisation is also described as a static solution, without allowing any user interaction to further investigate sub sections of the data. The former problem can be circumvented by displaying multiple evolution spectrographs, one for each attribute of interest, next to each other, with each showing the same time range with the same features, but a different attribute. An example of this is shown in Figure 11. Each row in both spectrographs represents the same feature. The amount of times this can be repeated is limited by the amount of screen space, so it does not scale to a large number of attributes, but it does allow for a few attributes to be compared. This comparison, when the spectrographs are placed side by side, will also allow users to detect correlations between the attributes. The data exploration experience can be improved by adding basic user interaction, such as zooming and panning. This way the user can easily look at different time ranges, a more fine grained time scale, or a subset of rows.

Another challenge with this visualisation is what ordering to use for the rows. In the work of Wu et al., the rows are sorted by the creation date of



(a) Evolution spectrograph with rows sorted by average value.



(b) Evolution spectrograph with rows in arbitrary order.

Figure 12: Interpreting the evolution spectrograph is harder without any ordering, in particular how big a part of the network exerts similar behaviour.

the files they represent. With the data used in this thesis, this is not effective. It is common for all elements to start recording values at the same time, in particularly in simulations. Choosing an appropriate ordering is however important, as a scenario where consecutive rows have a very different pattern over time in their attribute values is problematic. This results in a very noisy and hard to analyse pattern, as shown in Figure 12. In the sorted version a clear noticeable peak in the middle of the time range only appears for the bottom quarter of rows. In the unsorted example it is very hard to estimate how many rows show this peak. It is also quite hard to compare the individual rows, and as a result it is hard to be able to tell which rows are similar. Such a noisy pattern can be avoided by sorting the rows by some metric that describes the unique features of a row. This will put similar values closely together.

A simple solution to this problem is to sort the rows on their average values. This is what was used to generate the images shown in Figure 12, and is used in the rest of this work. While this will not perfectly place the most similar rows together, it will be a decent approximation, reducing the existence of these noisy patterns. This ordering is also easy to understand by an end user, and relatively cheap to compute. Optionally the user could be given the ability to choose to use the maximum, minimum, median, or any other function that can easily reduce a row to a single value.

An alternative solution would be to perform hierarchical clustering. This is also what is done in the work of Eisen et al. with genome data. This would more accurately allow similar rows to be placed together, and would allow for a configurable distance function to be used. It would however be expensive to compute the distance matrix, and would also require some explanation of the resulting clustering, for example by using dendrogram. Without such an explanation it would be unclear why a clustering is constructed as it is. Ordering by a metric such as the average is a rough approximation of this clustering, which is good enough for the purposes of this project.

An even different possibility would be to group rows based on their spatial or relational proximity. This technique would work under the assumption that proximity of elements implies behavioural similarity. In the case of a gas network this may be an accurate assumption, but this assumption may not hold in all cases. However, if spatially close objects show similar behaviour, sorting by average should again give a rough approximation of the same ordering.

How the evolution spectrograph is realised will be discussed in Section 5.2.2.

3.1.3 *Aggregation*

The visualisation design follows a client-server model, with data management performed by a server, and interactive data rendering performed by a client. This is discussed in full detail in the next chapter. An issue related to this design decision, which has been skipped over while discussing the previous two visualisation, is that of the aggregation of data with respect to the visualisations. As mentioned before, ValueFlex simulations can cover, in the most extreme case, the size of a country over a time period of multiple years. Even ignoring the problem of storing such quantities of data in a browser, and rendering this data in a interactive fashion, clearly there are less pixels on conventional screens than there is data, so it will be impossible to have a one to one mapping between data and pixels. On the interactive map features would be heavily occluded by each other. In the case of the evolution spectrograph, there are too many features, and the time ranges are too detailed to fit all the data directly on to it. Rendering a larger image and allowing the user to pan around could be used as a solution, although images would become so large it would be impossible to view an overview of the data. Displaying all features simultaneously on the map would result in a very cluttered result with large amounts of overlap.

One approach would be to tackle these problems on the client side. The evolution spectrograph could aggregate all data that is rendered to a single pixel. Care must be taken to preserve interesting details, such as maxima or minima, but solutions exist for this, as will be discussed in the next chapter. The occlusion problem on the map could be combated by clustering overlapping points, representing clusters with a single representative marker. It has however been decided to not deal with these problems on the client side. The reason is simple, this would require all the data to be retrieved first, before it could be simplified. As concluded before, this is not a feasible solution. Because of this the problem of aggregating and simplifying the data will have to dealt with before the data reaches the web client. Solutions that can be used to do this will be discussed in the next chapter. So, while the visualised data will still be an aggregate of the original data, the aggregation will not be performed by the web application. To approximate the original data on the client side, interpolation can be used as discussed in the case of the evolution spectrograph.

3.2 INTERACTIONS

A very important aspect of these visualisations, or views, is the interaction between them. The presence of interactions is what makes them linked views. The main idea of linked views is that they show the same dataset, or subset thereof, but in a different way. Since each view is only strong at displaying some aspects of a dataset, having multiple views allows them to complement each other. While the map visualisation is great for showing the relation between objects and their real world counterparts, and the structure of an infrastructure, it is limited in its capabilities of showing the temporal aspect of a dataset. This aspect of the dataset can however be visualised very well using a evolution spectrograph.

A great way to explore datasets is using Shneiderman's Mantra [37]. The idea is to first provide an overview, which gives a rough idea of where in the dataset interesting behaviour may occur. A user can then zoom and filter the presented data to a subset about which they wish to learn more about. These subsets are then visualised with more detail. If the dataset is very large, as is the case in this work, this process can be repeated multiple times. This approach works well in combination with linked views. For example, a user first select a spatial region of interest, such as a city. Clearly the map is most suitable to make this selection, as it shows the relation between the data and the physical world. Next the user may wish to select a specific time slice to analyse. The spectrograph is more suited for this task, as it shows an overview of the data over time. This way the dataset is filtered down to a subset that can answer a question the user has.

In this work we distinguish three aspects of the data to which filtering and zooming actions can be applied:

- the spatial domain
- the temporal domain
- attributes

Filtering on any of these aspects will affect the datasets visualised in each of the views, thus resulting in these views changing. How a user can perform these filtering and zooming actions will be discussed in Chapter 5.

3.3 DISCUSSION

In this chapter we discussed what selection of visualisations could be used to perform analysis of large scale temporal geospatial multivariate graphs, and some of the challenges that have to be overcome with these visualisations. How everything discussed in this chapter has been implemented will be discussed in Chapter 5.

4 SERVING LARGE DATASETS

In this chapter we discuss the problem of making large scale temporal geospatial multivariate graphs available in a web environment. First we take a closer look at the proportions of the data in Section 4.1. We then discuss the advantages and disadvantages of using a server side rendering solution versus a client side rendering solution in Section 4.2. Finally we discuss solutions for aggregating and storing these large graphs in Sections 4.3 and 4.4.

4.1 PROBLEM DESCRIPTION

The datasets considered in this thesis are very large. Networks such as those discussed before; gas transmission systems, water supply networks, electricity grids, road systems and computer networks; span entire countries. When considering a country such as the Netherlands, these networks have in the order of millions of nodes, with individual nodes for each household. For each of these nodes multiple attributes exist, such as voltage, amperage, and power. Each of these attributes varies over time.

As a concrete example of the size of these datasets, again consider ValueFlex. The largest simulations assumed to be performed in this work, as described by its developers, would be a simulation at the level of a country, over a year. As discussed before, in the case of the Netherlands, this would mean a dataset has worst case in the order of millions of nodes. These simulations run, in the most extreme case, over the period of a year, at a five minute interval. This means a simulation has roughly a hundred thousand time steps. These datasets are also multivariate, meaning they have multiple attributes. Not all attributes are necessarily temporal, such as the geographic location, but in the case of ValueFlex, there are at least the voltage magnitude, the voltage angle, the active and reactive power, and the amperage. While this is not a particularly high number of temporal attributes, the quantities described above are enough for a dataset to grow into the order of terabytes in size. Assuming 32 bit floating point numbers:

$$10^6 * 10^5 * 5 * 4 = 2 * 10^{12} \text{ bytes}$$

Which is approximately 1.8 terabytes of raw data.

Unfortunately, the CommonSense framework does not currently have any support for the large scale datasets considered in this work. All data is loaded up front, after which all data is displayed at once. This works well for the small datasets currently being visualised using this framework, but for large datasets this approach simply wont work. Not only would sending over these terabytes of data take a very long time to transfer to a client, the client would be unable to store it all in memory. The data must be prepared and structured so that a client can efficiently request only the data it needs for the current viewing parameters, with an appropriate level of detail.

4.2 SERVER SIDE VERSUS CLIENT SIDE RENDERING

A commonly used technique when visualising data is to shift the responsibility of rendering to a server which has access to all the high detail information. Clients can then request images from this server depending on which subset of the data they are viewing. This is an often used technique in GIS systems for the map image tiles showing the earth, in applications such as Google Maps [38], Bing Maps [39], Nokia Here [40] and other competing map viewing applications. The data used to generate these image tiles is relatively static. Unless, for example, the local infrastructure has changed, or a new house has been built, the data does not change. While this may happen quite often at a global scale, locally updates will be very uncommon. In this case server side rendering is very efficient. Data can be rendered once, and the result can be cached, since each future request of the same map tile should return the same image. All rendering could even be done up front, which is exactly what is commonly done in the case of map tiles. Only once the map data changes, do images change. Even then only the affected images would have to be re-rendered.

So ideally server side rendering is used for completely static datasets, displayed in a static way. The technique has however also been applied successfully with more dynamic visualisations. One such an example uses this technique to display three billion tweets [41]. These tweets are categorised by the operating system of the mobile phone used to send the tweet. This results in four groups, Android, iPhone, Blackberry and other. For each of these tweets the location it was sent from is recorded. For each tweet in the dataset, a point is rendered, coloured by the operating system it originates from. The user is allowed to dynamically choose which of the origin operating systems to show, being able to toggle each individual group on and off.

The user interaction offered in this case is however still very limited. In this three billion tweets example, all the user can do is toggle on or off the four categories. Effectively, there are 15 possible combinations, resulting in 15 different images for each raster tile. While this of course means 15 times as many images will have to be rendered than when not offering this interaction, this is still a manageable amount to pre-render. Even if not everything is rendered beforehand, results can be cached as it is likely different users will view the same tiles.

The datasets considered in this thesis are a lot more detailed. In the case of the three billion tweets example, a single point just belongs to a category, meaning it has only two attributes, a location attribute and a categorical attribute. The datasets considered in this work however have significantly more attributes, which have different values for each different point in time on which it is defined. Also, with the visualisations proposed in the previous chapter, users have a lot more fine grained control over the visualisation. Users can choose which attributes to show on the map, which colour mapping to use, and most importantly, choose a time slice. Allowing the user to choose a time slice alone results in thousands of possible results for a single image. This leads to a huge number of possible configurations for the map visualisation alone, and thus a huge number of possible renderings. It is impossible to store pre-rendered images of all these configurations, thus all images will have to be rendered on demand. Doing so would require a very powerful server to be able to serve multiple users simultaneously, and does not scale well at all. Because of this, server side rendering is not a particularly well suited solution for this project.

The alternative is to instead send the data itself, and delegate the rendering to the client. Clearly sending all the data at once, however, overloads the client. As such a solution is required that cleverly only sends data relevant to the current viewing parameters. An appropriate level of detail is also important, as the highest level of detail is unnecessary when viewing the entire world. If the same level of detail is used regardless of the zoom level, the same problem of sending all the data still occurs at the high overview zoom level. But just sending a small subset or aggregate of the spatial features is not enough. If the fully detailed time data is still attached to individual features, a small spatial subset of the data will still be quite large. Aggregation must be performed both in the spatial and the temporal domain. Also, the level of detail desired in the temporal domain is not necessarily related to the level of detail desired in the spatial domain. A user may wish to view the network at a high level, but only over a time span of a few minutes. Conversely, a user may wish to analyse only a small subset of a network, but over a large time period. Because of this it has been decided to separate the spatial data from the temporal data, and use a specialised solution for each. This way spatial data and temporal data can be easily requested at separate levels of detail.

In the following sections the spatial aspect is discussed first, after which the temporal aspect is covered. For both aspect, aggregation solutions are discussed, which allow the data to be requested at an detail level appropriate for viewing parameters chosen by a user. Data storage and serving solutions are also covered. It is assumed the spatial data has some reference which can be used by the web client to retrieve the appropriate temporal data.

4.3 SPATIAL DATA

The spatial data in the datasets considered in this project consists of the locations of the nodes and edges of a graph. This data must be aggregated in some way, so that different versions of the same data can be created, with different levels of detail. The goal of this is to always have a roughly constant number of elements on the screen of the user, no matter what part of the world, at what zoom level a user may look. It is important to realise these nodes and edges are connected, since they are part of a graph. This has large consequences for the aggregation process. With a simple point dataset, overlapping points can simply merged or discarded. Aggregation can be performed very locally. In this case points are however connected. Deleting a node has consequences for the structure of the graph. How a graph can be aggregated will be discussed in the next section. After this solutions for storing and serving the aggregated data are discussed. These sections will only cover the spatial structure of the datasets, not the data required to display the map on top of which the structure is to be rendered. As discussed previously pre-rendered images are commonly used for this. The serving of such tiles is standardised [42], and many free services exist that serve these image tiles. Some of these services will be utilized in this project as discussed in Chapter 5.

4.3.1 *Simplification*

No standard algorithm appear to exist for simplifying geospatial graphs based on spatial proximity. Two problems need to be tackled to perform simplification. Firstly, some algorithm is needed that decided which nodes and edges need to be combined to acquire a graph of an appropriate detail level. This could be simply some distance threshold between the nodes, but other solutions exist. Once these sets have been identified, some new point that represents the original points needs to be constructed. This new point needs a location, attributes, and edges, which all need to be determined based on the original nodes which are combined to create the new node.

Selection

The simplification of spatial graphs does not appear to be an extensively researched area. Solutions for the simplification of general graphs do exist [43], however, the graphs considered in those works are not spatial in nature. The location of nodes in spatial graphs have important meaning, while in general graphs nodes do not have set positions. These techniques are generally concerned with grouping up strongly connected parts of graphs spatially by moving nodes. This is however not necessarily the behaviour desired for the data in this project. Strongly connected does not necessarily imply spatial proximity. By clustering points that lie far apart critical spatial information could be lost. These general solutions are therefore not a good fit for this problem.

Another area where a similar problem is encountered is when resampling polygonal meshes[44]. A polygonal mesh could be seen as a graph, as both are made of points, vertices and nodes, which are connected by lines, edges. The resampling algorithms used for the resampling of polygonal meshes are concerned with acquiring a new polygonal mesh which closely approximates the original mesh, while using less vertices. As a result the rendering of such a surface is more efficient, as well as requiring less space to store the surface. The goals of these algorithms is however again different from what is desired in this case. These algorithms try to describe a densely sampled surface as accurately as possible using fewer vertices, while preserving the original shape. Generally no restrictions are applied however to where the resampled vertices appear on the surface, in theory allowing vertices to appear anywhere on the surface. This means new vertices can potentially appear far from the locations of the original vertices, by which the spatial meaning of points is lost. In the case of a polygonal mesh this is fine, since the information is described by the shape of the mesh as a whole. Individual points carry little meaning. The spatial location of points is however important in a real world infrastructure, thus this information should not be discarded.

A different way to look at this problem is to consider the dataset as a simple set of points, instead of a graph. This allows the problem to be simplified to a clustering problem. After clusters have been identified, the cluster can be merged into a single point. This approach may however be problematic in a strongly connected graph. As such a graph has many edges, less clusters have to be used to still achieve a consistent number of features in the output. This may be undesirable, as in extreme cases this can mean a strongly connected graph has a lot fewer nodes than a graph with few edges. In a highly connected graph it may thus be desirable to either instead of or in addition to clustering one may wish to reduce the number of edges directly, for example by edge bundling [45]. As such this solution only works as expected when most edges are short, and only connect nodes within a local cluster, as they can then be aggregated together. Many long edges should be avoided. This appears to be a valid assumption in this case, because in real infrastructures, having a single node connected to many other distant nodes is inefficient. This would require many long cables or pipes to achieve. Generally only a limited number of sources, i.e. power plants, are connected through the network to many destinations, i.e. households. In these infrastructures the long distance transportation occurs on a high throughput connections, for example with high pressure or high voltage in the case of gas and electricity. Near destinations, connections branch off this main line, going into medium pressure or voltage networks. One could compare these infrastructures with a tree type structure, with one, or few roots, but many leaves. Such a network satisfies our requirement of most edges being small and local, as locally a dense network of low pressure or voltage components are used, while for long distance transport single large capacity components are used.

So it is possible to solve this problem using clustering. Of importance to our solution is a consistent density in the result. If a huge dataset can be simplified down to a thousand points, but almost all points lie closely together and overlap, the data may be small enough for visualisation, but is still hard to interpret. At the same time, the spatial size a cluster describes should be as small as possible, so that the clustered point can represent the locations of the original data as accurately as possible. In the case of a city filling the users screen, all points within the city may lie closely enough together to warrant to cluster them, but of course it is not very interesting to see a single point. So the spatial size of a cluster must be limited somehow. Also, any clustering solutions that upfront require a desired number of clusters to be specified are not desirable, since it is hard to know in advance how many clusters should be used.

A solution that satisfies all these requirements is hierarchical clustering. Traditionally this is done by constructing a distance matrix which contains the distance between any pair of points in the dataset, and grouping up the points that lie closest to each other. This is repeated recursively until a desired number of points remains. Computing the distance matrix alone has a complexity of $\mathcal{O}(n^2)$ however, which is quite computationally expensive. With datasets that have in the order of millions of nodes, this becomes problematic. A slightly different approach is to use a quad tree. Using this technique a space is continuously subdivided into smaller squares, until each square only contains a given number of points. The result of this process on a set of points is shown in Figure 13, where a limit of one point per cell is used. After constructing such a tree, a simplified version of the dataset can be constructed by traversing the entire tree up to a given depth. Which depth is used decided how detailed the result will be. The cells encountered along this traversal can either be empty or contain a single point, or contain more child cells with more points. When empty or containing only a single point nothing needs to be done, the single point can be directly included in the simplified graph. The cells that contain multiple points need to be aggregated into a new single point. This approach has been applied successfully to point data in a geospatial setting on huge datasets, with great performance [46]. A solution to efficiently generate such a quad tree is the Approximate Nearest Neighbour, or ANN, library [47]. This library can construct data structures of millions of points to efficiently perform nearest neighbour queries, one of which is the quad tree.

This approach is a great fit for this use case. Since the quad tree has at most 4^d cells at a depth d, there is a strict upper limit to the possible number of points in the end result, which limits the level of detail. The clusters described by cells also clearly describe a limited spatial area, as a cluster can never contain an outlier. Different levels of detail can also be achieved easily by reconstructing a simplified graph using the cells at different depth of the tree. Now that clusters have been selected, they need to be aggregated.

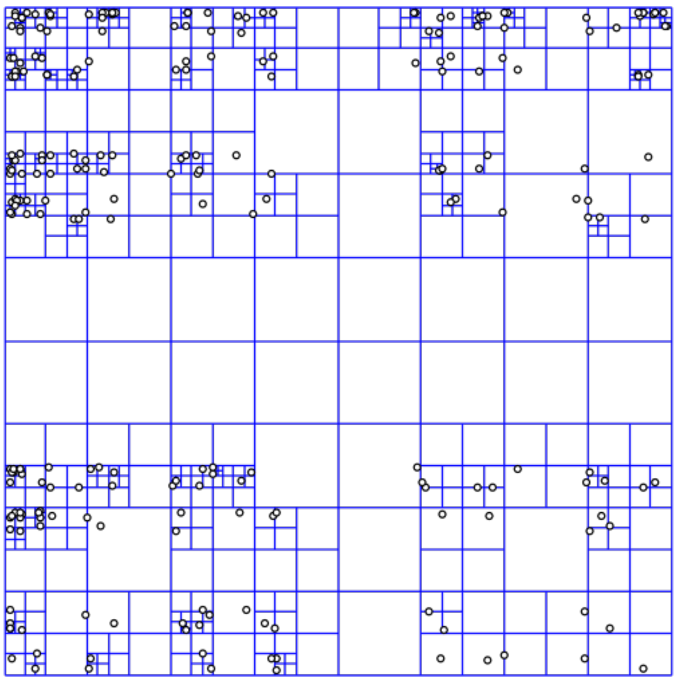


Figure 13: Example of a quad tree. Each cell is subdivided into four smaller cell until each cell contains only a single point. *Source: Wikipedia user David Eppstein,* `https://en.wikipedia.org/wiki/File:Point_quadtree.svg`*, public domain license.*

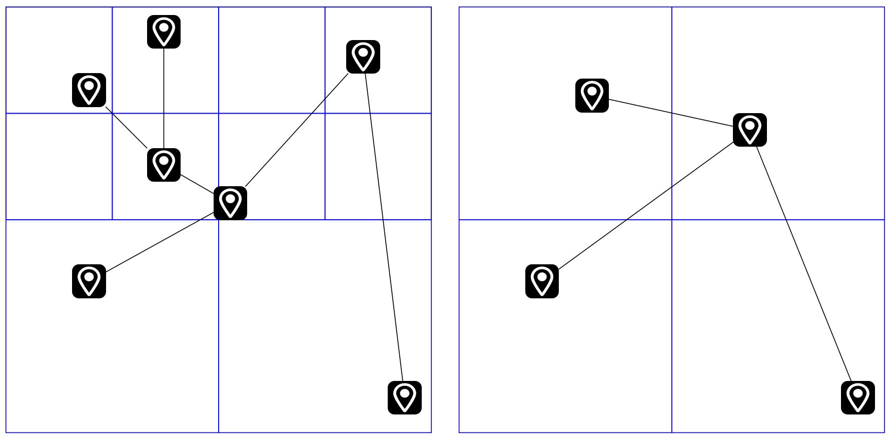

(a) Before aggregation (b) After aggregation

Figure 14: Example of the effect of the proposed graph aggregation algorithm on an example graph, aggregated at depth 1.

Aggregation

Since each node has a location, relations with other nodes in the graph, and temporal attributes, aggregating groups of nodes is not trivial. Each of these problems will have to be dealt with to be able to successfully combine a group of nodes into a single node. The problem of location can be solved quite easily by taking the mean location of the group of nodes to combine. Since the raw data combined into a point can only originate from a small spatial area due to the way the group is selected, as long as the new point lies within the cell of the quad tree, where the new point ends up exactly is not very critical. While more sophisticated methods could be used for location selection, such as weighing the nodes for example by throughput capacity, the difference in position will be so minor a user will hardly be able to tell the difference. Because of this the mean location should be good enough for our purposes.

The problem of nodes being connected by edges can be solved by connecting the newly constructed node, to any nodes outside of the aggregation group which are connected to any of the nodes within the group. This is commonly referred to as vertex contraction in the context of graphs. Two nodes, which are not necessarily connected by an edge, are combined into a single new node. If they are connected the edge between them is simply removed. The new node is adjacent, i.e. connected, to all nodes to which the original two nodes were adjacent. The aggregation of a group of nodes into a single node then is the same as repeatedly contracting a node in the group with any other node in the group, until a single node remains. An example of the result of using these rules for aggregation is shown in Figure 14. A very simple graph, with a quad tree subdivision is shown in Figure 14a. The smallest cells lie at depth 2. The aggregation of this graph at depth 1, after computing the mean position of the nodes within each quad tree cell, and the set of adjacent nodes of each cell, is shown in Figure 14b. Due to the existence of attributes it becomes slightly more complex however.

The most difficult problem is that of combining the attributes. Both vertices and edges can have attributes, both of which need to be combined. No single best solution exists here, since the best aggregation technique strongly depends on the data, and what questions a user wishes to answer using the data. There are two kinds of attributes, temporal attributes, for which different values are available at different points in time, and static attributes, which are the same throughout the entire time range. For numerical attributes such as pressure it makes the most sense to use something such as an average, which gives an indication of the overall pressure in the aggregated group. However, for attributes such as consumption or throughput, adding the individual values seems more appropriate, so the consumption of the aggregated group as a whole can be viewed. Adding these values together potentially introduces a new problem, as the values of different nodes no longer lie in the same ranges. For example, if consumption rates at nodes normally lie between 0.9 and 1, then comparing a single node to a cluster made out of two or more nodes is meaningless, since the values of the cluster will always be much larger. Whether to use the average, mean, median, maximum, minimum, variance, standard deviation, sum, or any other function will have to be decided by an expert of the data. Similar problems occur with other types of attributes, such as text and categorical attributes. Text could be simply concatenated, or perhaps each individual textual attribute will be present as a separate attribute in the new node. These problems could be solved in many ways, but their aggregation can not be solved in a generic way, and will have to be specified on a case by case basis. In any case, it is important it is clear to the user what kind of aggregation technique has been performed, as this explains what information is lost and what is attenuated by the aggregation. This can help the user distinguish actual interesting patterns from any artefacts that result from aggregation.

In the case of temporal attributes, attributes can also be defined at different points in time for different features, which makes combining these features even harder. Solving this in a general way is again very hard, but usually the temporal attributes will be numerical in nature. A way to solve this problem of different sampling rates for numerical attributes is to consider all values of an attribute as a signal, which can be resampled. This resampling could be done by reconstructing a continuous signal by interpolation, and sampling the reconstructed signal at desired points in time. This way values for the attribute for both features can be sampled on common time points. Preferably this set of common time points at least contains the union of the timepoints at which the individual attributes were sampled for the original features. This way the original values should not be lost, as the reconstructed continuous signal should give the same value when sampled again. What function to use to combine these resampled signals is left to an expert. How these temporal attributes, including the combined ones, are simplified is discussed in Section 4.4.

4.3.2 *Storage*

Now that an algorithm for simplifying the spatial aspect of datasets has been described, a solution needs to be found to store and serve the resulting data. Multiple solutions exist. The first storage solution, spatial databases, serve data within a bounding box defined by the clients viewing area. The second solution uses a set of vector tiles, which use a similar concept as the map image tiles encountered in the server side rendering techniques. The client can again request these depending on the viewing area chosen by the user. Another interesting option is graph databases. Each of these solutions will be discussed in the following sections.

Spatial databases

The first spatial data storage solution would be to utilize a spatial database. A spatial database is a specialised database designed specifically to deal with spatial information, allowing it to be queried spatially. With such a database it is for example possible to retrieve all features within a given area. Using such a storage solution would allow the client to simply send a bounding box of the current view, after which the spatial server can compute which features lie within this box and send only those specific features. There are many existing solutions. Special purpose solutions exist, such as SpaceBase [48] and GeoMesa [49], which have been specifically created for storing spatial data. However, many of the general purpose database solutions have either build in support (MongoDB [50], Neo4j [51]), or modules available to add spatial support (PostgreSQL [52], CouchDB [53], SQLite [54]).

After choosing a database one has to decide when to perform the simplification of the data. Not all these spatial databases offer aggregation features, and even those that do seem to be limited in their customizability. Admittedly, the aggregation of spatial is a hard problem that can not easily be solved in a generic way, as discussed before. Most of these databases are also not specifically build to support graph structures, and store features as separate points lines and polygons. They do not have a concept of a graph. Because of this these databases are not capable of simplifying the data of this project effectively in a generic way.

One option to solve this problem would be to perform these simplifications on the fly on the output of the database. This however leads to a large amount of extra computations on every query sent to the database, which can be particularly problematic with a view containing large amounts of high detail features. This problem is enhanced by the fact that each query is essentially unique. Query results can not easily be cached between different users, and even consecutive queries of the same user, while largely overlapping, will generally require the result to be computed from scratch. There is no simple way for the server to know which data the user already has, nor can the client easily request only the new features that came into view. To constantly reaggregate all data from the highest level of detail to the lowest level of detail would be very inefficient, and poorly scalable. The only way to avoid this seems to be preprocessing the data, and storing multiple version of it at different levels of detail. Then somehow the server will have to decide which version to use, for example by looking at the size of the bounding box of the query.

Thus the simplification must be performed by a custom solution before storage in such a database. The challenge then becomes to find an appropriate mapping of the data onto the database that allows the data to be queried effectively by a client. The simplest way seems to store multiple copies of the data at different levels of details in the database, while allowing the client to choose the most appropriate detail level for the current view. Each query result will however still have to be build from scratch, no caching is possible.

Tiling

A different solution is tiling. We have mentioned before this technique is used commonly for images in map applications. These tiles are constructed using a structure which is very similar to a quad tree. They are constructed by building a pyramid of equal size images, where all the images together in one layer of the pyramid make up the original image. At the top of the pyramid the image is described by a single image, and all layers below have four times the number of tiles as the level above it. So considering a tile has for example 256x256 pixels, at the top most level the image is describe by 256x256 pixels, at the level below it by 512x512 pixels, below that by 1024x1024 pixels, etc. As illustrated in Figure 15 from the work of García et al., lower levels have more detail than the higher levels. After constructing such a pyramid a client can selectively request tiles from this pyramid from a part of the source image of interest, at an appropriate level of detail. For an overview the detail available in the top most tile may be sufficient, but when viewing a very small part of the image a much lower level of the pyramid may be appropriate.

The main advantages of tiling is that with it, unlike the spatial databases, caching of queries is easily possible. Results can be (pre-)computed once and used many times. This same approach could be use on the feature data, by splitting up the graph in small tiles, with different amounts of detail for varying zoom levels. However, instead of renderings of the data, these tiles would contain the actual features, point and lines, as raw data. These tiles can then be used by the client to render the data as the user desires. By using

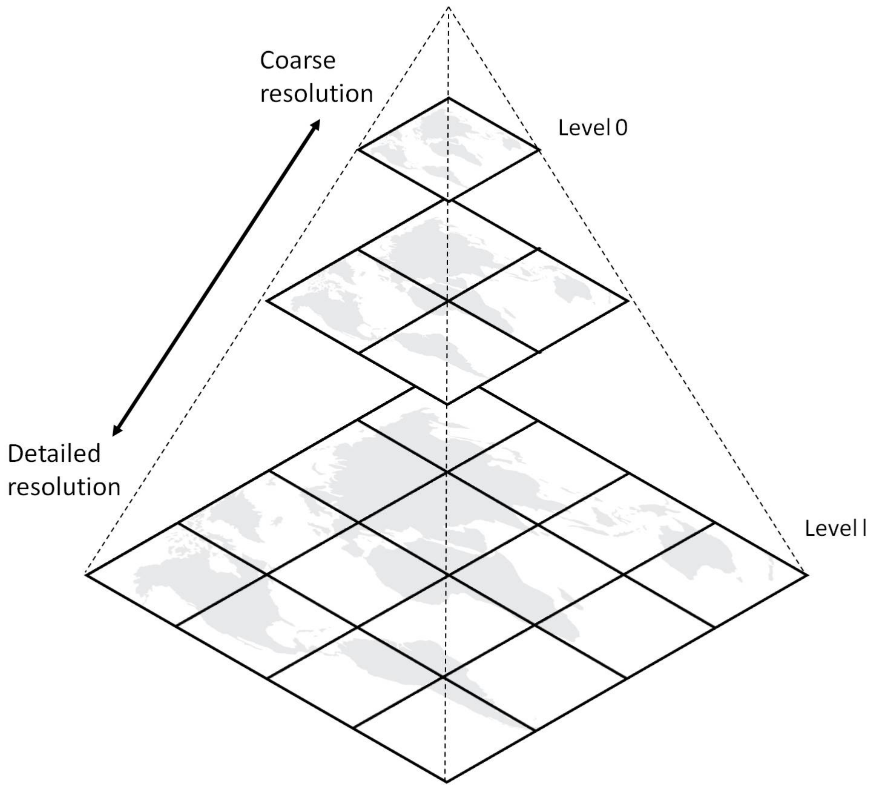


Figure 15: Illustration of a tile pyramid. Each layer of the pyramid describes the data with more tiles, and thus more detail than the layer above it. *Source: García et al. "Web Map Tile Services for Spatial Data Infrastructures: Management and Optimization"* [55]

tiles the client is able to only retrieve the data required for the current view, in terms of location and zoom level.

This tiling is exactly what has already been done for simplifying the spatial data. The construction of such a pyramid of tiles uses exactly the same logic as is used for the construction for the quad tree in the simplification process. If instead of extending the zero depth tile of the quad tree exactly over the spatial dimensions of the data, the tile is extended over the entire world, a tile in the quad tree matches up exactly with a tile in the tile pyramid. Thus after performing the simplification, constructing the tiles requires relatively little effort.

One problem with this tiling approach however is the fact that lines and polygons can cross the boundaries of tiles. In these cases parts of a single feature reside in multiple tiles. This scenario is very likely to occur very often in the case of the large infrastructures considered in this thesis, considering these infrastructures cover entire countries. One solution to this would be to clip the feature, cutting it into two or more separate pieces, each of which is stored in the associated tiled. This way, when a feature covers multiple tiles, each tile only contains the part of the feature that covers it. A side effect of this is however that multiple separate pieces of the same feature are presented to the client, even though each piece belongs to a single feature. Some extra work would be required to still consider it a single object with just one set of attributes. As each feature has an identifier required to retrieve the associated temporal data, the individual pieces can be identified and stitched together. This is important for example when selecting features, the selection of a single part must result in the selection of all visible parts. Alternatively, instead of cutting up the feature, the whole feature could be stored in every tile it overlaps. By doing so the complete feature is available whenever one of the tiles it lies within is visible. Again extra work would be required however, to avoid multiple copies of the feature to be placed on top of each other. This solution might also turn out to be problematic when many tiles are overlapped by a single feature, resulting in large amounts of duplicate data.

Many different formats exist to store these tiles in, for which many different implementations have been created. A very popular format for geospatial data is GeoJSON [56], which is also the format CommonSense currently uses, albeit not in a tiled form. GeoJSON specifies a standardised structure using which features can be stored in the JSON [57] format. The benefit of this format is that it can be natively parsed by JavaScript and is human readable. Using any JSON parser, which exist for almost every programming language, the generated tiles can be easily converted into files with this format. These files can then be easily served using a simple web server providing a RESTful API, which is a common way to serve these types of tiles. Alternatively, a closely related format could be used, called TopoJSON [58]. As the name suggest it also describes a JSON structure, although one that is generally more compact than that of GeoJSON, thus reducing the size of the resulting data.

While the benefits of using GeoJSON or TopoJSON are that it is human readable and easy to use in JavaScript, it is fairly verbose and contains significant amounts of redundant information. An alternative would be to use a binary format, such as Mapbox vector tiles [59]. By using protocol buffers [60] the vector data are encoded efficiently, significantly reducing the data in size. Many tools exist for the generation and parsing of these kind of tiles [61, 62, 63, 64, 65, 66]. Using this solution would allow for more data to be displayed with the same use of bandwidth, making it an interesting solution.

Graph databases

Another storage solution worth mentioning is graph databases. As the name implies these databases are optimized for storing graph shaped data, allowing for queries regarding the structure of the graph. Many popular implementations exist [67, 68, 69], including the earlier mentioned Neo4j which has native support for spatial data. The main strength of these kind of databases lies in the traversal of the edges it stores, so that queries asking for the neighbours of a node can be performed efficiently. This can be very useful when a user wishes to select sub networks of the entire network.

However, such a database will most likely not be a good fit for this project. While graph queries may be interesting in some scenarios, graph databases generally offer no options for spatial queries, which are required for selecting portion of the graph appropriate for the current view. Simplification will also be hard to do, although this could be solved by storing multiple preprocessed copies of the data. If the graph queries are truly important, it is probably more effective to implement these queries on the client side, considering the size of the graphs sent to the client will be limited in size regardless, making such queries possible in real time even in a resource limited web environment. Since the biggest advantage of using a graph database can also be achieved in another way, it will be more appropriate to choose a database specialised in spatial data.

Now that it is clear how to simplify and store the spatial aspect of the data, the temporal aspect will be covered.

4.4 TEMPORAL DATA

Each feature can potentially have temporal attributes, and possibly have multiple of them. As discussed before, storing this temporal data separate from the features themselves allows spatial and temporal data to be viewed at independent levels of detail. To do this, each feature has been given a unique identifier, including those in the simplified graphs. These identifiers can be used to look up the temporal data belonging to that feature.

Similar problems of aggregation, storage and serving as encountered with spatial data are encountered in the context of temporal data. Fortunately, time series data is a more commonly encountered data type than geospatial graph data, in the context of web applications. Many solutions exist for the aggregation and storage of these kind of datasets. Some of these solutions will be discussed next.

4.4.1 *Simplification*

As with the spatial data, a solution is needed for constructing new versions of the data with different levels of detail. When viewing temporal data at the level of weeks, the values at individual seconds are too detailed for visualisation or analysis. At this point an aggregate with a sampling closer to the number of pixels available for rendering should be used. This is essentially the same problem as encountered with the spatial data. The lower detail versions of the data could be computed on demand. However, when viewing the entire time range at a low zoom level, the entire time range must be aggregated each time, for every feature in view, for every attribute it has. This would be too computationally expensive for it to be interactive or scalable. Thus to facilitate these different temporal levels of detail efficiently, the data should be preprocessed, as was done with the spatial data.

Just as we proposed with the spatial data, for the temporal data we can also pre-compute and store a number of different copies of the data with varying levels of detail. Each lower detail version could have half the number of samples as present in the next most detailed version, similar as was done with the spatial data, although more or less detailed levels could be used as desired. The client can then request data of an appropriate level of detail. Computing these lower detail version is again a non-trivial problem however. As a lower detail version of the data has less samples than a higher detail version, some method is required to compute a single value from multiple samples. A simple way to perform the needed aggregation would be to just take the average. This may be sufficient for some time series, for example those that do not contain small variations, or those where small variations are not important. Averaging would remove many small details from a dataset however. If these small details are important, they must be preserved so that they appear in higher level overviews.

Instead of taking a simple average, taking the minimum, maximum or most deviating value may be more appropriate. Holten et al. [18] describes a method to weigh samples by how different they are from their neighbouring values. What approach is best to use depends strongly on the features present in the underlying signal, and the goals of the visualisation. A noisy signal needs a different approach than a signal with many spikes or sharp transitions. Thus, unfortunately, we again can not provide a general purpose solution that works well in all scenarios. The most appropriate aggregation technique must be decided on a case by case basis.

4.4.2 *Storage*

The temporal data in this project is essentially sensor data, i.e. time series data that belongs to one of many entities. Many solutions already exist that allow sensor data to be stored and queried efficiently. Some of these solutions also incorporate aggregation functionality. ElasticSearch [70], the solution currently being used for storage of ValueFlex simulation data, offers a wide range of aggregation functions and queries. With regard to functionality, it is a great fit for this project. It seems less capable however of performing these aggregations on larger dataset in a timely manner, which is unsurprising considering the quantity of data. Thus pre-computation of the aggregated data is still required. This however means none of the functionality that makes ElasticSearch special will be used, thus a database more specifically designed for serving temporal data may be appropriate. As the name implies, it is better suited as a search engine.

When performing the aggregation by a custom solution, in theory any existing storage solution can be used. Many databases have been used effectively for sensor data, such as PostgreSQL [52], MongoDB [50], HBase [71], CouchDB [53] and Cassandra [72]. The work of Van der Veen et al. [73] compares the performance of some of these storage solutions, showing Cassandra provides great read performance if queried appropriately. Each of these databases could store the different detail copies discusses above, allowing the client to query the appropriate version dynamically.

Another promising solution would be to use InfluxDB [74]. Similar to Cassandra, it is specifically designed to deal with sensor data. Additionally, it also offers functionality to aggregate data automatically by using so called continuous queries. These queries, which have as primary function precomputing expensive queries, are run periodically when new data is added. The results of these queries is then stored, allowing the result to be queries efficiently. This can be used very well for computing the down sampled version of the temporal data, and would allow live data to be easily supported.

4.5 DISCUSSION

In its current form the existing framework CommonSense is designed for small amount of data, rendering it incapable of displaying large datasets. While there are many existing software solutions that are designed to help deal with this problem, none appear to exist that are specifically designed to perform the aggregation of spatial graph data. To simplify the problem, spatial and temporal data is split up, so both can be easily requested by a client at different levels of detail. To be able to serve the data efficiently, strong aggregation is required. While general recommendations can be given regarding aggregation, what aggregation best suits a project strongly depends on the data and questions a user may wish to ask.

5 REALISATION

In this chapter the existing visualisation framework CommonSense is first discussed in Section 5.1. We list what functionality is already present, and what still need to be implemented to achieve the functionality discussed in the previous chapters. Afterwards the implementation of the new functionality is discussed in Section 5.2.

5.1 COMMONSENSE

The existing visualisation framework, CommonSense [4], is a framework developed by TNO. This framework was used in this project as it is a commonly used solution at TNO, and integration into this platform allows the new functionality to be easily reused for future projects. Its design philosophy is to offer a generic platform for the visualisation of any kind of spatial data, by providing a highly configurable web environment, with a set of generic visualisation solutions, which can be customized to specifically fit the needs of each individual project. An example of what CommonSense can do is shown in Figure 16.

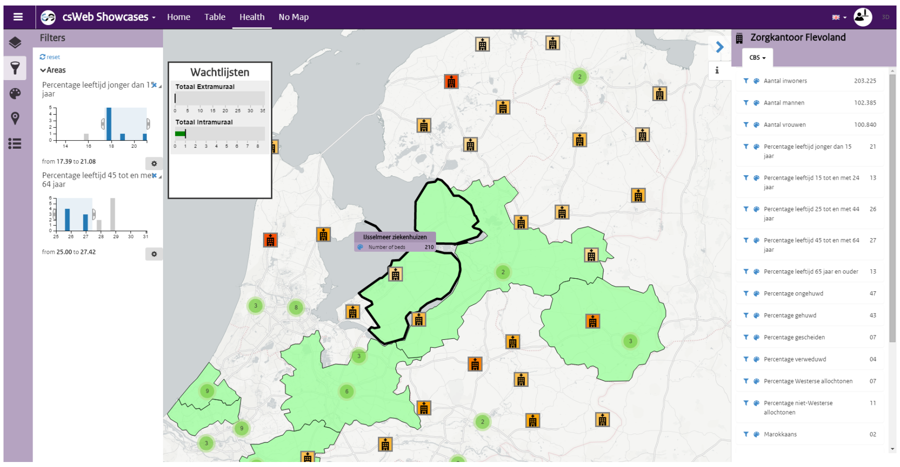


Figure 16: A showcase of the functionality in the CommonSense framework, in which Dutch healthcare information is visualised.

While the framework is intended as a generic visualisation solution, it is still in a relatively early stage of development, and because of this the framework is not quite ready for data on the scale as generated by the ValueFlex project. While some functionality to support temporal data, like that of the ValueFlex project, already exists, it is designed for small datasets with in the order of hundreds of features and timepoints. No functionality is present to allow CommonSense to cope with large quantities of data, nor does it allow such large spatial temporal datasets to be visualised effectively.

As discussed in Chapter 1, any solutions proposed in this work must be integrated into the CommonSense framework. Due to the reusability of the CommonSense framework it is important that any new functionality is designed and implemented within CommonSense framework in such a way, that it too can be reused in future projects. This asks for the chosen solution to be designed and implemented in a generic way so that it will fit the design philosophy of the CommonSense framework. As such it is important that any new components must cooperate with existing functionality, not only ensuring no existing functionality is broken, but also that the result is one unified application.

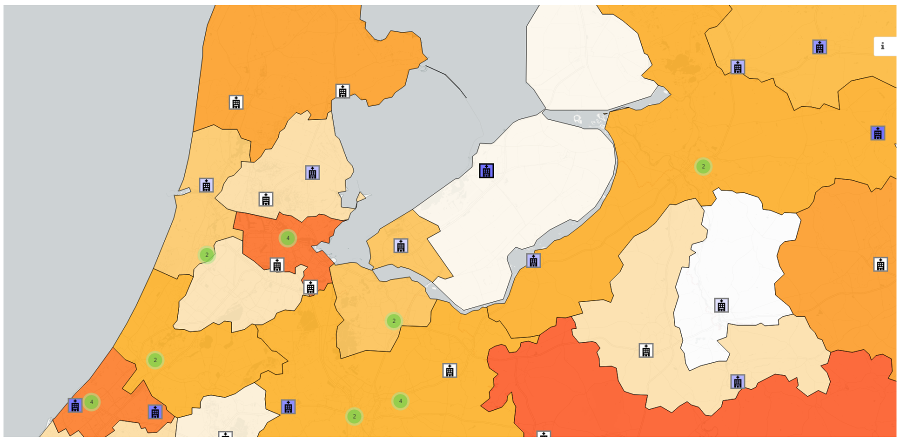


Figure 17: CommonSense interactive map, showing features, in this case polygons and points, coloured by their attributes.

5.1.1 *Existing functionality*

CommonSense already offers a substantial amount of the functionality discussed in Chapter 3. The interactive map is present, and it provides some filtering functionality. The existing functionality is however not always sufficient for the purposes of this work, and some additional functionality is required for CommonSense to be usable as an effective analysis tool for large scale temporal geospatial multivariate graphs. Below we discuss what needed functionality is already present and what is missing, in particular with regard to the two visualisations, the interactive map and the evolution spectrograph.

Interactive Map

The interactive map as discussed is already implemented in CommonSense. It was created using the JavaScript library Leaflet [75]. Leaflet offers many of the standard exploration functionality commonly present in visualisations where exploration is required, such as zooming and panning. On this map any kind of features can be displayed, being simple points and lines, or polygons. The user can choose what attribute each of these features are to be colour mapped by, and what colour map to use. Additionally the user can choose whether to colour the icon itself or its boundary. Alternatively the thickness of the boundary can also be used to indicate the magnitude of an attribute.

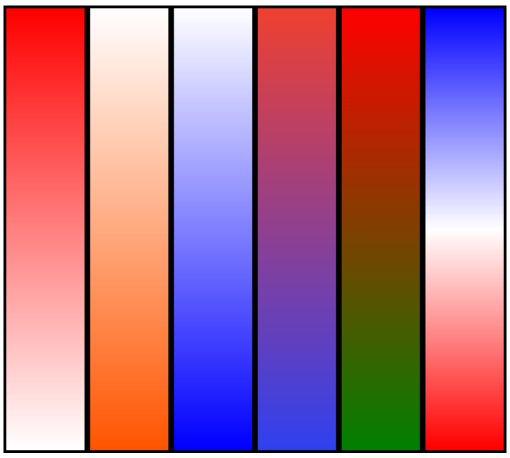
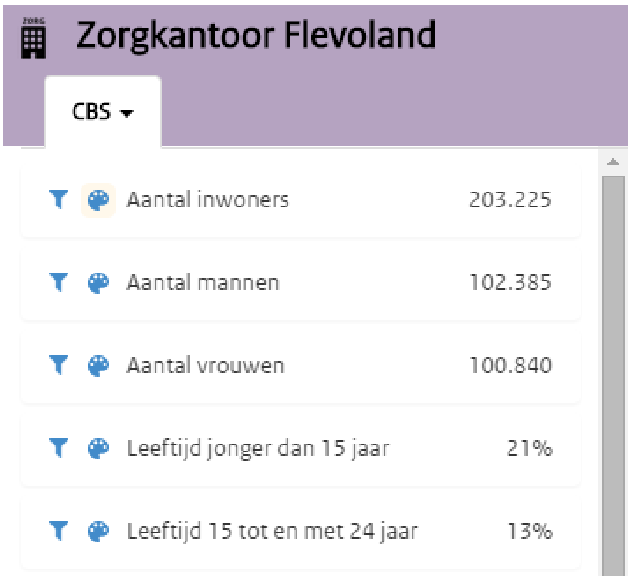


Figure 18: Selection of the colour maps available in CommonSense.

The colour maps currently available in CommonSense are somewhat limited. A selection of the colour map functions available by default are shown in Figure 18. Each consists of two, or in one case three colours between which is interpolated. The interpolation results in mixtures of colours appearing in the middle of the range of these functions. The biggest problem with these colour maps is however that large ranges of the colour map results in colours which are hard to perceptually distinguish. This mean a user will have a hard time performing the reverse mapping of colours using the legend. To improve this, a new colour map has been added as part of our additions, which is shown in Figure 29. This new colour map will be discussed further in Section 5.2.2.



Zorgkantoor Flevoland

CBS

Aantal inwoners	203.225
Aantal mannen	102.385
Aantal vrouwen	100.840
Leeftijd jonger dan 15 jaar	21%
Leeftijd 15 tot en met 24 jaar	13%

Figure 19: Attribute overview in CommonSense. This panel shows the attributes of the currently selected feature, at the currently selected time slice.

Selecting features on the map gives the user a detailed list of all attributes of that feature in the attribute panel, as shown in Figure 19. These are the values of the currently selected time slice. The buttons in front of each attribute allow that attribute to be used for colouring the features on the map, and also allows features to be filtered based on that attribute.

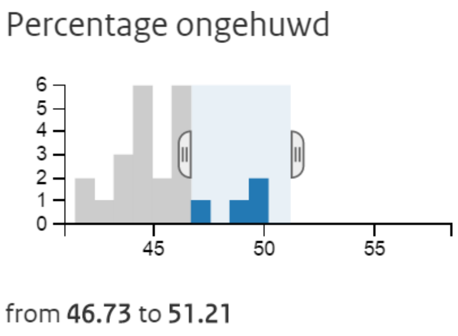


Figure 20: Filter controls in CommonSense, which allow the users to filter the displayed set of features on any of its attributes. In this particular example the features, representing counties, are filtered by the percentage of unmarried inhabitants. As a result only counties with a unmarried inhabitant percentage between 46.73% and 51.21% are displayed.

Pressing the filter button brings up a panel such as the one shown in Figure 20, although the exact filter shown depends on the type of the attribute. This panel allows a user to filter features based on attribute. In this example a bar chart is used to display a histogram of the percentages of unmarried inhabitants of counties in the Netherlands. The user can select a range on this histogram to choose which range of values to show. Features with values for this attribute outside of the selected range will be filtered out, hiding them on the map. Multiple filters like this can be used simultaneously.

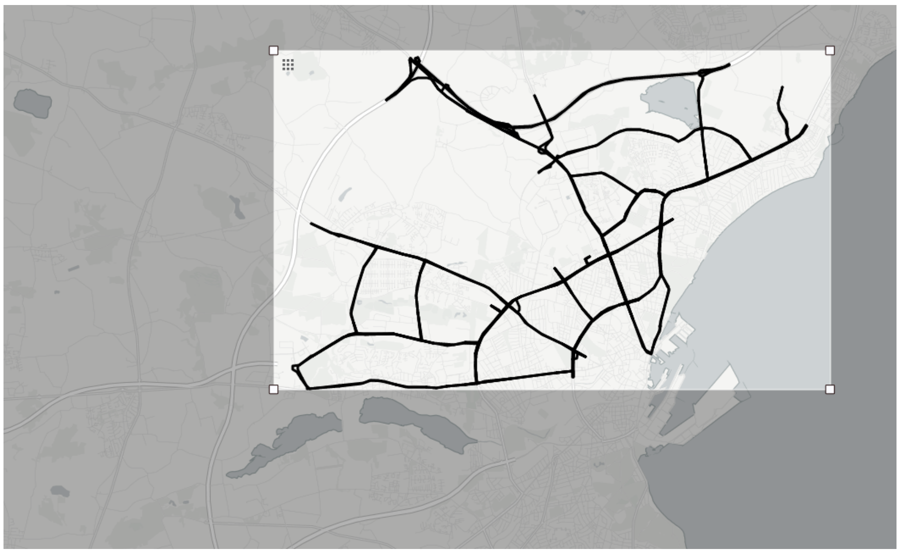


Figure 21: The location filter allows a region to be selected, which exclude any feature outside of the region to be filtered out.

One other special kind of filter available is the location filter, as shown in Figure 21. As the name suggest this filter allows features to be filtered based on their location. Activating it presents a selection box to the user, which can be moved and resized to fit around a region of interest. Any features outside of the box are no longer drawn. While for the map visualisation itself this functionality is not particularly useful, as a user could simply zoom in or ignore features outside of his area of interest, when combined with the linked evolution spectrograph visualisation this becomes a useful tool for filtering.

Evolution spectrograph

While the interactive map is already implemented, the evolution spectrograph has been implemented from scratch. Some support for temporal data is already present however. Internally support is present for features having different attribute values over time. CommonSense also already contains a timeline, which is shown in Figure 22. This timeline component has an indicator. It shows which point in time is currently the focus point, which determines the time slice being displayed on the map. The user can drag this indicator element to manipulate this focus point. The timeline supports zooming and panning, allowing the user to easily manipulate the time range being displayed, and choose whether they wish to work on the level of seconds or milliseconds, or the level of years or decades.

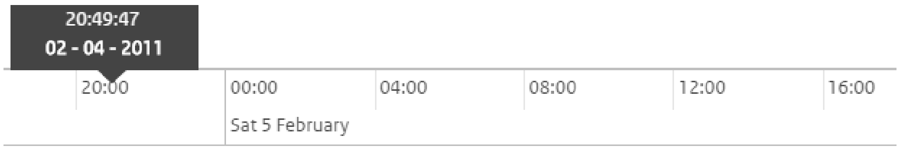


Figure 22: Timeline in CommonSense. The timeline allows the user to manipulate the current temporal focus point and time range, which determines the time slice shown on the interactive map.

Another functionality that CommonSense provides is widgets, an example of which is shown in Figure 23. Widgets are elements which are placed inside separate containers, and shown on top of the map visualisation. They can be moved and resized as the user desires. What these widgets contain exactly could be anything. They can offer additional controls, or new views of the data, such as additional information of a selected feature as is the case in the indicator widget in the example. Another example of a widget is one that shows the current value of some sensor, such as the current temperature. These widgets offer a great place to add the evolution spectrograph, as will be discussed further in Section 5.2.

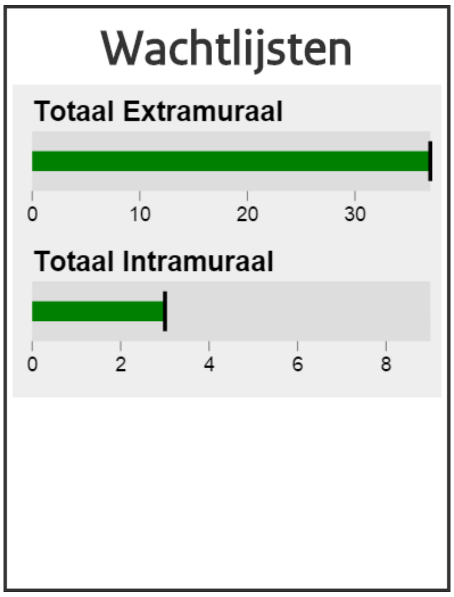
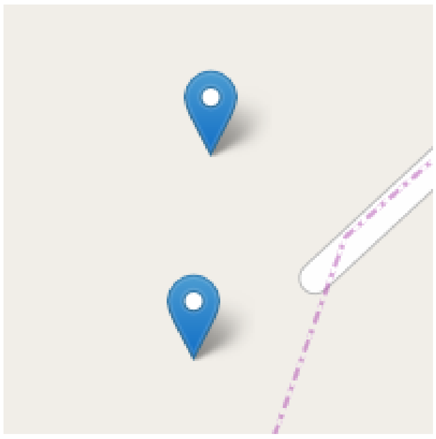


Figure 23: Example of an indicator widget in CommonSense. This particular widget shows additional additional information about the currently selected province.

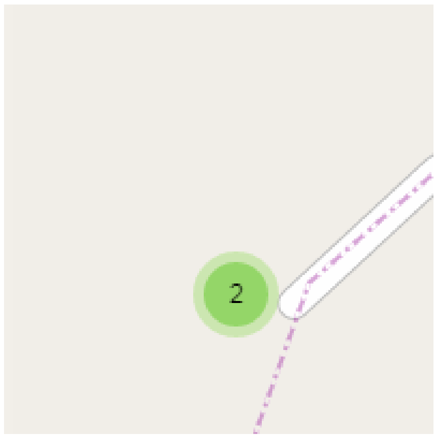
5.1.2 *Limitations*

In Section 2.1.2 three types of analysis were listed which are desirable in the context of ValueFlex. They were value oriented, time oriented, and comparison oriented analysis. In its current form, CommonSense is not particularly well suited to allow for any of these three types of analysis. The first, where users attempt to find specific values such as extremes could theoretically be done, but without any temporal overview, it would require going back and forth through time to manually find these values. Time oriented analysis is practically impossible, since there is no way to see how data behaves over time. This shortcoming also makes dataset comparison hard. While multiple datasets can be shown simultaneously on the map, allowing them to be compared spatially, without a high level overview of the data that includes time, it is hard to compare different datasets thoroughly. Essentially the temporal support of CommonSense is very limited, which is very important for the analysis required for large scale temporal geospatial multivariate graphs.

CommonSense is also poorly suited for coping with large amounts of features. Clearly, in the case of millions of households as simulated by the ValueFlex project, there would be too many features to be able to distinguish them when all the features are all displayed simultaneously on the map. The CommonSense framework already contains a feature that attempts to solve this problem by clustering points, as implemented by the JavaScript library Leaflet.MarkerCluster [76]. This library coalesces points between which the distance is below a given threshold into a single cluster point that represents these individual points. This process is depicted in Figure 24. This process can repeat itself, where a marker representing a cluster of points can be clustered again. By doing this one can set an upper limit to the point density on the map. This approach unfortunately only works well with datasets consisting purely out of points, since it is unable to cluster lines or polygons. It is also required that these points only represent a location without any further attributes, since these attributes becomes inaccessible after clustering, and can only be viewed by zooming in far enough until the individual points reappear. A temporal multivariate graph, such as an electrical grid, does not satisfy either of these requirements. Components of the graph have many attributes, and the edges can not be represented by a simple point. Because of this, this solution will not be satisfactory for the kind of data considered in this project.



(a) Two separate point markers



(b) Two points markers grouped into a single cluster

Figure 24: Point clustering functionality in CommonSense. Once two or more point markers lie too closely together, these points are automatically grouped into a single cluster point.

Another limitation is that the CommonSense framework currently does not support selective retrieval of features. While different datasets are loaded dynamically, these datasets are loaded in their entirety once selected by the user. Loading the complete datasets discussed in this paper would be problematic, as we have shown these can easily grow in the order of terabytes in size. With the average computer having only in the order of gigabytes of memory, it is impossible to send these entire datasets to users at once. This is ignoring even other significant problems such as bandwidth limitations and server capacity issues. To utilize the solutions discussed in the previous chapter, not only must the aggregation and serving of data be implemented, but new functionality is also required in the CommonSense framework.

5.1.3 *Architecture*

A high level overview of the relevant components of the architecture of CommonSense is shown in Figure 25. The CommonSense framework uses the AngularJS [77] toolset, or Angular for short. At its core Angular consists of directives and services. Directives are markers on DOM elements, that add functionality to, or transform these elements. By using directives functionality can be added to raw HTML. CommonSense defines a set of these directives and services which can be used to build a visualisation framework. Services on the other hand offer common functionality that is needed in multiple locations of the application. In the context of CommonSense, directives are mostly used for reusable interface components, such as the map, the timeline and the attribute overview. These directives take care of displaying their respective elements of the application on screen. This way a custom application can be created easily by using a set of these preconstructed elements.

A specific subset of these directives are called widgets within CommonSense, which are elements which are displayed in separate containers on top of the rest of the framework. Examples of services used in CommonSense are the translation service, the layer service and the communication bus service.

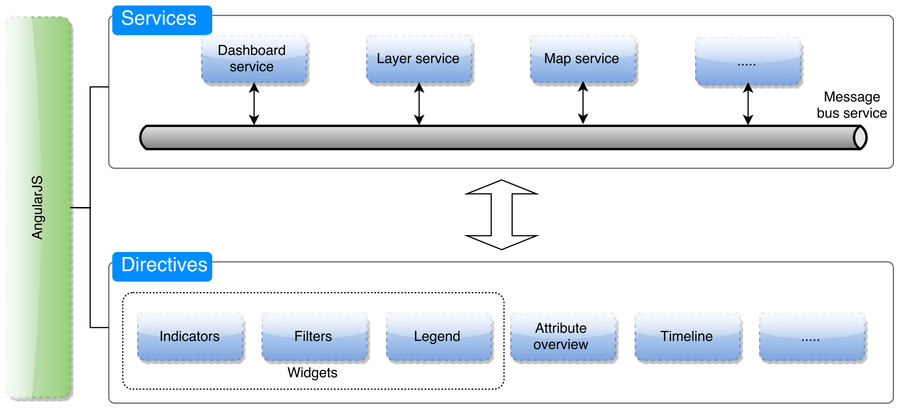


Figure 25: High level view of the relevant components of the architecture of CommonSense.

Most important for this project are the communication bus service and the layer service. While some communication occurs directly between widgets and services, the communication bus service is used to announce events throughout the framework, and allows components such as directives to subscribe for these events. This is used, for example, when the time focus point is changed using the timeline directive. This change has an impact on what is visualised in other directives such as the map. Any component affected by this subscribes for the appropriate event at the message bus, to which the timeline directive posts a message once it receives user input and changes the focus time point. This way individual components can be decoupled from each other, but still respond to events such as changes to viewing parameters.

The layer service is responsible for retrieving and storing data. This service is named as such, due to the fact that data is structured in layers. This naming convention comes from the fact that the CommonSense framework is mostly geared towards GIS data, where different sets of data, such as that of roads, political boundaries, and rivers, are considered as different layers. In the case of our graph infrastructure datasets, one such a dataset would be one layer. The layers are organised in groups, allowing layers with related data to be grouped together in the user interface. These groups in turn belong to a project, in which all application wide information is stored such as the temporal parameters like the current focus time, and the extends of the time range currently being analysed. This structure is shown in the class diagram in Figure 26. In this diagram the type notation present in TypeScript [78] was used, as this is the language in which CommonSense is written.

The most important aspect of this diagram is the way the features are stored, shown in the bottom half. Each feature has both a set of properties, as well as a set of sensors, which are sets of name-value and name-array pairs respectively. Any of the time independent attributes are stored in the properties collection. The storage of time dependent attributes is slightly more complex. The attribute values are stored in the sensors set. The associated time stamps are stored separately in the timestamps array. There are however two locations to store these timestamps, at the level of a feature, or the level of a layer. Thus the same sampling can be used for an entire layer, or an individual sampling for each feature. It does however enforce different attributes of a single feature must be recorded at the same interval.

For each sensor, a property with the same name also exists. At all times, the value of this property is kept equal to the nearest sensor value to the current time focus point. This task is left to the layer service. Understanding this mechanic is important for understanding how new temporal data can be inserted dynamically into CommonSense.

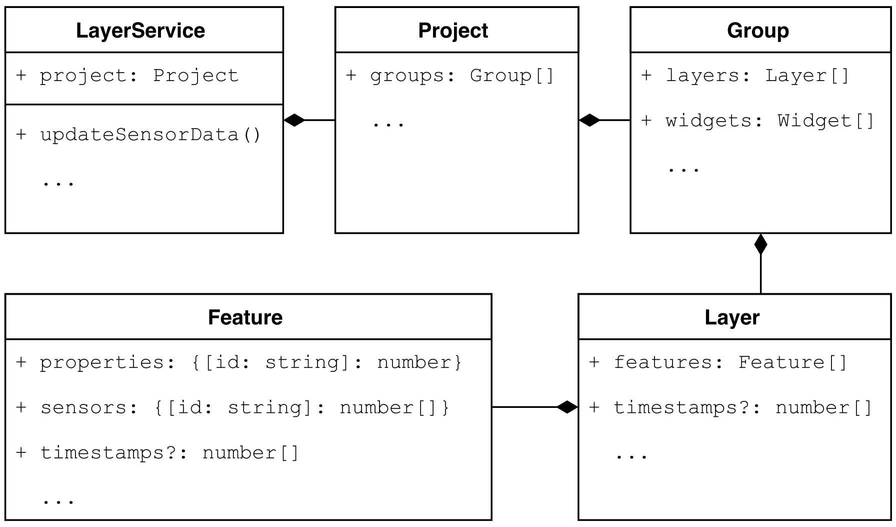


Figure 26: Simplified class diagram of data model in CommonSense.

How solutions proposed in this work are integrated into CommonSense will be discussed next. First the visualisation aspects will be discussed, after which the data solutions will be covered.

5.2 INTEGRATION

This section will discuss how all the functionality discussed in the previous chapters can be integrated into the CommonSense framework. First we discuss the data management. How exactly the spectrograph was realised, and what functionality it offers will be discussed next. After this an augmentation to the timeline will be discussed. Finally the configuration of the map element of CommonSense is explained.

5.2.1 *Data management*

Unfortunately, due to time constraints, the data solutions discussed in the previous chapter could not be completely realised during the course of this project. Work on this functionality has started, but no usable results have been achieved at the time of writing. One factor in this is the documentation of CommonSense, which is still preliminary due to the active development of the framework. As a result, understanding the inner structure of CommonSense and integrating the visualisation additions was more time consuming than expected. In this section ways for integrating the data solutions mentioned in the previous chapter into CommonSense will be discussed. Most important for these solutions is that they integrate well into the CommonSense framework, and allow existing functionality to continue to work. It must also be possible to use only the spatial or the temporal data solutions, so that they can also be used for datasets which only contain large amounts of spatial, or temporal data respectively. Unfortunately these solutions are mere suggestions, and their effectiveness has not been verified.

Spatial

The layer service, as discussed before, is responsible for the management of data. This service however relies on so called layer sources for the actual retrieval of data from different kinds of sources. For example, there is a GeoJSON source, but also a Web Map Service (WMS) source, which is a standard for serving map images. By adding a new type of layer source which can cope with the data services proposed in Section 4.3, it would be possible to easily integrate the spatial data retrieval into the CommonSense framework. Hooking into Leaflet functionality may also be advantageous, as Leaflet already deals with loading the right background tiles as the view changes. As the background tiles use the same system as the data tiles, when a background tile request is made a request for the same tile can be made to the data server. Whichever approach is used, of course it is important all requests are performed asynchronously as to not block the user interface while new data is retrieved. It should be possible to query required data aggressively and letting the browser worry about caching any request. Browsers by default cache a lot of the data they request. If the data server is set up correctly, a `Last-Modified` header can be send along with each tile. The browser can send this information back on future requests, allowing the server to determine whether a file needs to be send again, or the cached version can be used, which would result in a 304 HTTP response.

Temporal

The temporal aspect can be solved in a similar way, by adding some sort of temporal source manager to the layer service. This source manager must make sure the right sensor data is attached to each feature by the time the layer service updates the properties of features. It must also update the timestamp arrays accordingly. In the case of temporal data we did not propose to use a tiling scheme, unlike with the spatial data. Thus we can not rely on the browser to perform the caching in this case. As such the implementation at the client side must keep track of what data it has already requested. In particular when shifting a small amount through time this can save a lot of unnecessary data transfer. In this case it is not needed to retrieve the entire time range selected by the user, but only the part of the time range that was shifted into view.

5.2.2 *Evolution Spectrograph*

The evolution spectrograph visualisation has been implemented by adding a new widget directive, as is indicated in Figure 27. By using a widget the spectrograph can be visible simultaneously with the map, so that a linked view environment is realised. It also allows the visualisation to be easily moved and rescaled as the user desires. At the same time this keeps the design modular, so that the visualisation can be easily added or removed depending on a project its needs. This is important since not every project that will use the CommonSense framework is temporal in nature.

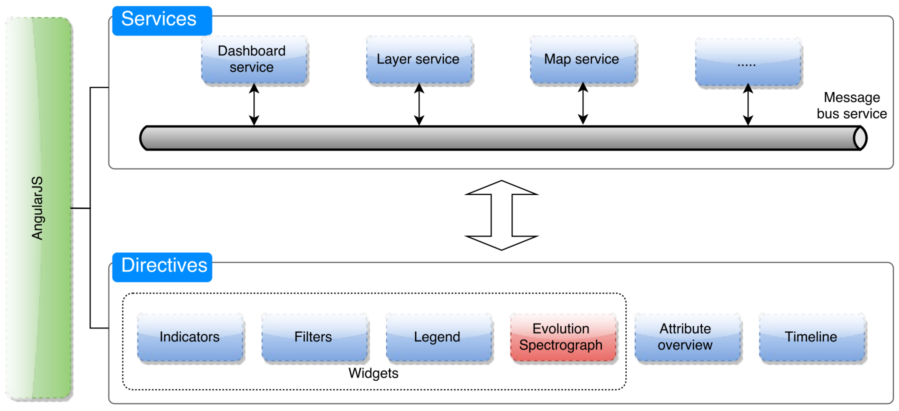


Figure 27: The evolution spectrograph is implemented in CommonSense by means of a widget directive.

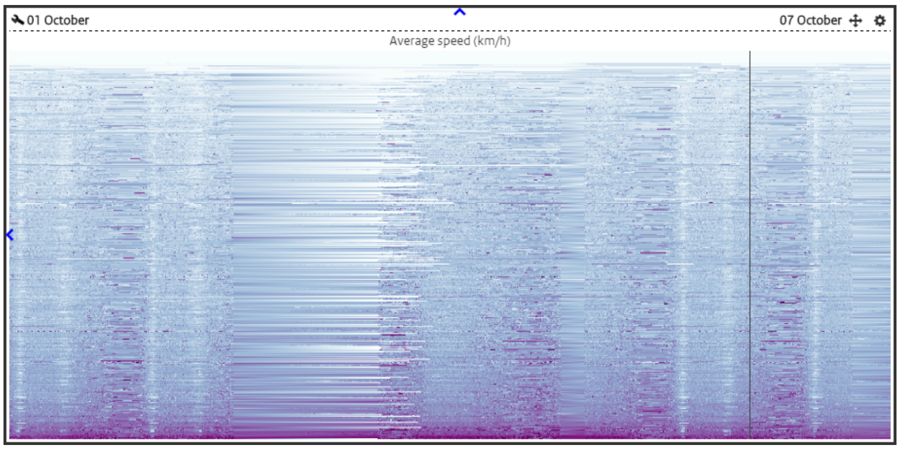


Figure 28: Evolution Spectrograph as realised in CommonSense.

To implement the evolution spectrograph efficiently, it was decided to utilize WebGL [79]. Initially Three.js [80], a lightweight JavaScript library which provides a low level abstraction layer over the WebGL renderer, among others, was used to implement the rendering. OpenGL, and as its derivation, WebGL, are notorious for requiring a large amount of boilerplate code even for the most trivial of renderings. This library could potentially remove large amounts of this boilerplate. Unfortunately, at the time of writing, Three.js does not support triangle strips. As will be explained later this is a desirable method of rendering in our case. In the past Three.js has offered support for this rendering mode, and requests and pull requests have been made to reintroduce the functionality [81]. However, as official support is current lacking, Three.js was abandoned and instead raw WebGL has been used in this work. The decision to use WebGL instead of for example a 2D canvas or SVGs, was primarily made because of the better performance of WebGL due to its acceleration by the GPU [82]. In early stages of the project experimentations were done with SVG rendering, but a large number of SVG elements was required to achieve an acceptable result. Using such a large number of SVG elements lead to significant slow downs, with render times of single images going over a second in some scenarios. These kind of slow downs are unacceptable in an interactive application, since an user no longer feels he is directly interacting with the data if a response takes longer than 0.1 second [83, 84, 85]. Another benefit of using WebGL is that the interpolation of the data can be performed by shaders, allowing the computations to be off-loaded to the GPU instead of the CPU. The end result is shown in Figure 28.

The black line on the evolution spectrograph indicates the current focus time shown on the map. It is linked to the focus time indicator of the timeline, shown in Figure 22, so any movements in the indicator on the timeline translate to movements of the black line on the spectrograph. At the top of the spectrograph an indication is given of the time range being displayed on the spectrograph. It helps the user relate the information shown on the timeline and on the map, to the information shown in the evolution spectrograph. One may also notice a different colour map being used in this figure than those presented previously in Figure 18. The colour map used is based on a suggestion from the tool ColorBrewer.org [86]. This website offers

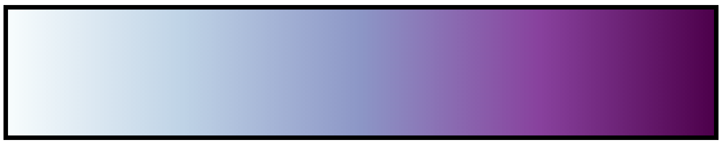


Figure 29: Colour map inspired by ColorBrewer.org [86].

many different colour schemes with easily differentiable colours. While these colour maps are primarily intended for class based data, they work quite well for continuous data as well. These colour maps suffer less from the perceptual issues discusses previously. The colour map used in the spectrograph renderings shown in the remainder of the thesis will be the one shown in Figure 29.

The spectrograph is also able to display multiple attributes, as was discussed in Section 3.1.2. This is achieved by effectively rendering multiple spectrograph side by side, each showing the same time range, but each copy visualises a different attribute. A feature is represented by a row in the same vertical position in each of the spectrographs. The result is shown in Figure 30. Each copy of the spectrograph has its own focus time indicator to help relate the spectrograph to the other views.

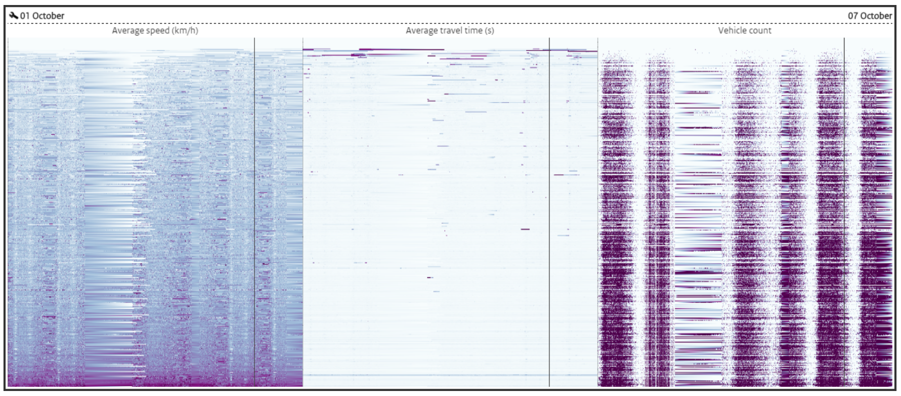
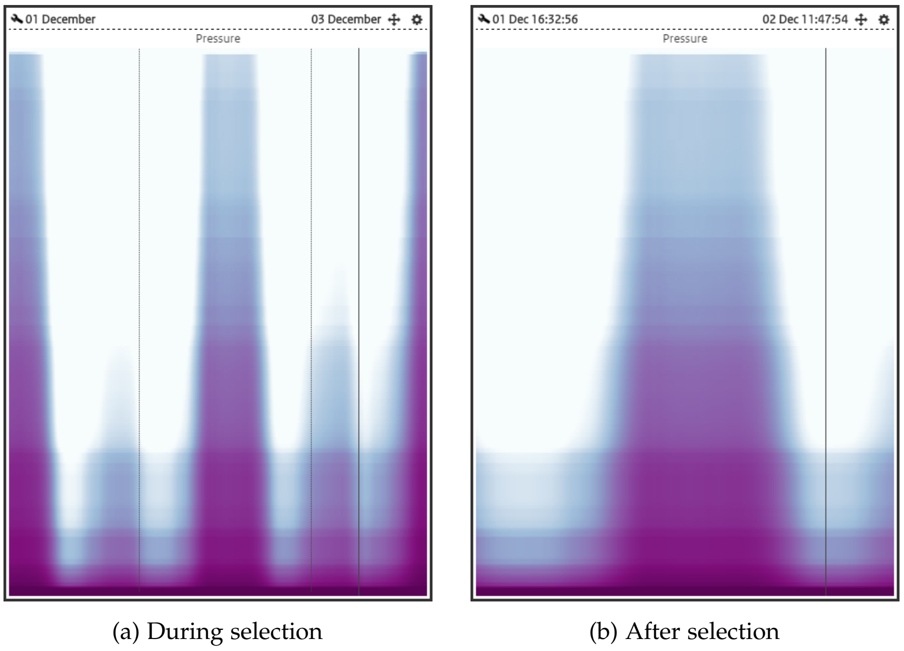


Figure 30: The implemented spectrograph has support for showing multiple attributes by rendering the same time range multiple times, but each time depicting a different attribute of the feature.

Interaction

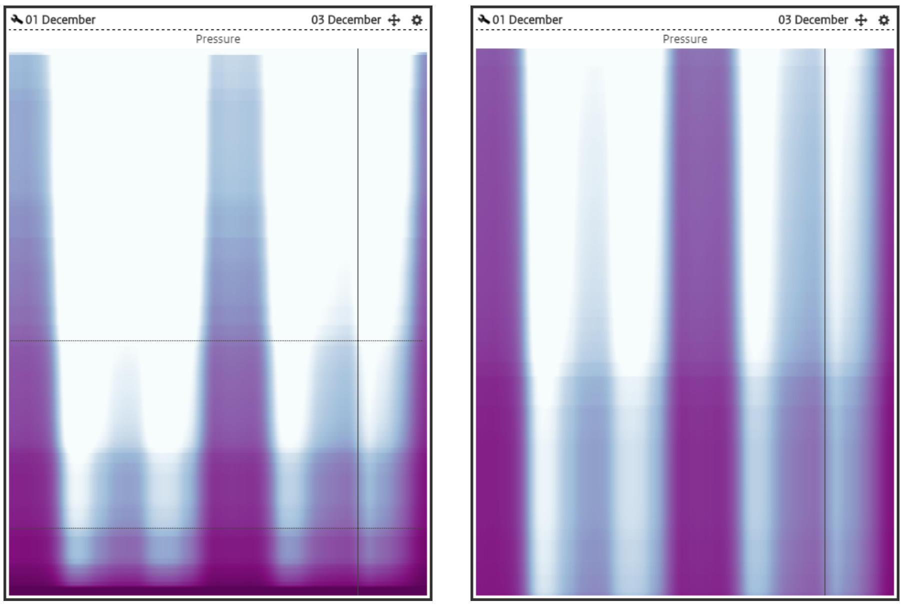
The evolution spectrograph offers a number of interactions. First of all, a user is able to narrow down the time range they want to analyse. This interaction is shown in Figure 31. By dragging over the evolution spectrograph, an



(a) During selection

(b) After selection

Figure 31: The user can select a time range on the spectrograph by making a horizontal selection.



(a) During selection (b) After selection

Figure 32: The user can select a subset of features on the spectrograph by making a vertical selection.

area bounded by dotted lines can be selected that spans the horizontal space of the spectrograph, selecting a time range. The user is also able to select a vertical range, as is illustrated in Figure 32, which results in a subset of features being shown, both on the spectrograph and the map. The user can switch between the two types of selection by holding or releasing a modifier key. This allows the user to easily take a closer look at a specific interesting time range, or set of features. Due to the linked nature of the visualisation the timeline at the bottom of CommonSense is also updated accordingly, as is the set of features displayed on the map. Finally, it is also possible to select a feature through the evolution spectrograph. This will highlight the feature both in the spectrograph and on the map, as well as show the attributes of the feature in the attribute overview, shown in Figure 19. This helps identify the spatial location of a feature shown on the spectrograph

Rendering

To achieve this result triangle strips were used to construct the rows making up the image. Using this technique, given an array of vertices, the vertices at index 0, 1 and 2 form a triangle; 1, 2 and 3 form another triangle; 2, 3 and 4 form the next triangle; etc. An example of this is shown in Figure 33. Such a strip is constructed for each feature being displayed on the evolution spectrograph. In Figure 33, x and v represent the x position of the vertices in the column, and the value attached to those vertices, respectively. t_n and v_n represent the n-th timestamp, and the value of the attribute being displayed

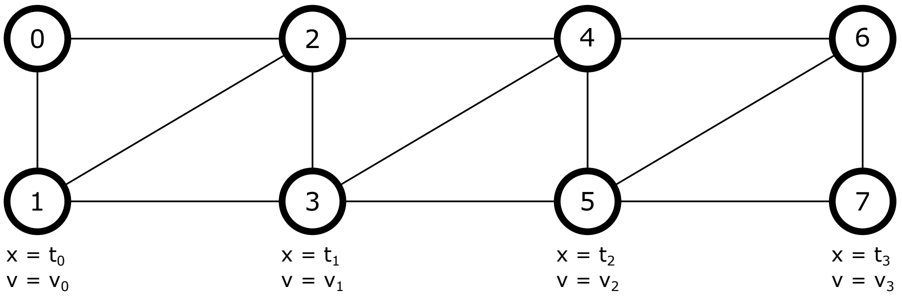


Figure 33: Triangle strip.

at the n-th point in time. By setting the x position of the vertices equal to the timestamp at which their value is defined, the relative position of the vertices is automatically correct, and the time range shown in the spectrograph can be easily manipulated by offsetting the camera by the time point which corresponds to the left side of the current time range.

By assigning the values to visualise to the vertices, an OpenGL Shading Language (GLSL) `varying` variable can be used to get linearly interpolated values. This results in the values between any of pair of horizontal vertices to be computed using the GPU. This way the original data is approximated. One can compare the result to using nearest-linear mipmap filtering, as the nearest version of the data is retrieved from the server, to which linear filtering is applied to acquire the final value that corresponds to each pixel on screen. The colour mapping then occurs in the fragment shader using a one dimensional texture. By using a one dimensional texture to perform the colour mapping, instead of interpolating the colours themselves, false colours are avoided [87, Chapter 5]. False colours are colours which are not present in the colour map used, but are created by interpolating two colours which do exist in the colour map.

Rendering each individual row of triangles using a separate draw call would introduce some additional overhead. To avoid this overhead degenerate triangles are used to stitch the triangle strips together, allowing all of them to be rendered at once. At the end of each strip, four of these degenerate triangles are added by repeating the last vertex of the current strip, and the first vertex of the next strip. Consider the situations sketched in Figure 34. With this approach, a index buffer for these vertices would look like the following:

$$0,1,2,3,4,5,6,7,\underline{7},\underline{8},8,9,10,11,12,13,14,15$$

The four degenerate triangles consist of the vertices

$$\begin{gathered} 6,7,7 \\ 7,7,8 \\ 7,8,8 \\ 8,8,9 \end{gathered}$$

These triangles are shown in Figure 34 in red. Since these triangles are degenerate, they will not be mapped to any pixels in the final result. Only the triangle strips making up the rows will remain.

By rendering the data in this way using triangle strips, values for each feature can be defined at arbitrary points in time. As shown in Figure 34,

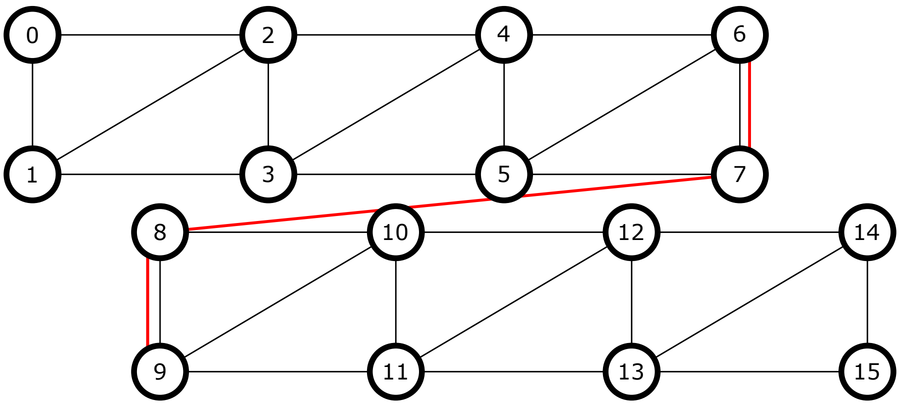


Figure 34: Multiple triangle strips, connected by degenerate triangles, which are shown in red.

different rows can have data starting and ending at different points in time. In this particular example the data is defined at a set interval, but this is also not a requirement for this approach. This freedom is important since, as can be seen in Figure 26, each feature can potentially have a unique set of timestamps at which its sensor values are defined. By using this rendering technique this is fully supported, and no resampling is required prior to rendering.

Precision

Rendering using this approach however leads to a problem. Timestamps within CommonSense are millisecond precision integer numbers, giving numbers in the order of 10^{12}. Using these values as coordinates for vertices leads to these vertices being extremely far away from the origin of the world. Once converted to floats by the graphics pipeline, precision problems occur in these very large values. This problem is extensively discussed by Thorne [88]. In this project the problem resulted in camera translations no longer being fluent when zooming in to the level of minutes. For example, the timestamp 1296934200000, once converted to a 32 bit floating point number on the testing system, results in the value 1296934240256. This is more than a 40 second shift from the original value. As a result, when attempting to shift the camera at the level of seconds, by offsetting it by such a millisecond precision number, nothing happened until the difference became large enough for the millisecond precision number to be mapped to a different floating point number.

To solve this problem, as proposed by Thorne, the data is moved towards the camera, instead of the camera being moved to the data. The vertex positions are translated to the origin before being send to the GPU. Since this translation is performed before the data is sent to the GPU, the high precision of JavaScript numbers is still available, and no error is introduced in the shift. As a result when shifting, even when zoomed in at the level of seconds, movements remain fluent. The downside of this approach is however that this transformation takes place on the CPU, and has to be done again each time the view changes. Normally these transformations are left to the GPU as it is specialised to perform these kind of transformations in a highly parallelised fashion. Fortunately, the performance decreases has not turned out to be particularly problematic, as will be shown in the next section.

Performance

One of the goals of the implementation is to offer a *interactive* platform on which data can be analysed. As discussed before, any action by the user should result in a response within 0.1 seconds for a user to feel like they are directly interacting with the data. To verify this requirement the performance of the rendering of the spectrograph was analysed. We consider the evolution spectrograph specifically since it is the most computationally expensive of the visualisations. To perform this analysis the Google Chrome Developer Tools [89] were used. This set of tools includes a profiler, which can be used to keep track of the run time of all functions in a JavaScript program. As our testing platform, we used a desktop system containing an Intel Quad-Core i5-2500k CPU running at 3.30GHz, and an NVIDIA GeForce GTX 970 GPU. For testing we consider a scenario where the spectrograph is displaying three attributes, of a dataset of 449 features with temporal data over 7 days recorded at 5 minute intervals. As each spectrograph is drawn separately this is roughly three times as intensive as a single spectrograph. On this testing system, the rendering code took on average, over 41 calls, 18.32 ms. About 11.79 ms of this time is spend on shifting the data to avoid the aforementioned precision issues. Clearly this easily falls under the 100 ms requirement discussed earlier, even when including the shifting time.

Even with an indication of the current focus time on the evolution spectrograph, there is a disconnection between the controls used to manipulate temporal parameters, and the view needed to guide these manipulations. A specific event in the evolution spectrograph can not easily be mapped to a date on the timeline. Because of this it was decided to augment the timeline directive by displaying temporal data onto it, in the form of a chart. As vertical space is limited, displaying the spectrograph itself on the timeline as well would force many of the rows to be mapped to the same row of pixels, resulting in a strongly aggregated result. Because this limited space requires an aggregate regardless, it was decided to use a line chart. A line chart is better suited to utilizing this limited vertical space to display values accurately. Using this line chart an aggregate over the entire graph is displayed for each time step. With the line chart the user immediately gets a rough indication of what the data looks like at a specific point in time.

Which aggregate to use is however a though decision to make in the general case. Simply averaging over all the values of all features is the easiest solution, and does give the guidance this feature was implemented for. However, any outliers in the data will of course be lost this way. One must also realise that the data available in the browser is already an aggregate, as was discussed in Section 4.4. If at that aggregation stage an average was used, using a function such as the minimum or maximum will no longer result in the actual minimum and maximum present in the original dataset. While the average was used in the implementation, ideas for other functions will be discussed in Section 7.3.3.

The realised timeline chart is depicted in Figure 35. To create it the JavaScript library dc.js [90] was used. It is a charting library, which supports crossfilter [91] for efficient exploration of large datasets. It uses D3.js [92] for rendering. The line chart is attached to the timeline, displaying a line chart over exactly the same time range as shown on the timeline. It also shifts with the timeline as it is moved by the user. This allows the user to easily see where in the shown time range interesting behaviour may occur. The vertical axis of the chart is automatically scaled to the minimum and maximum values found in the data.

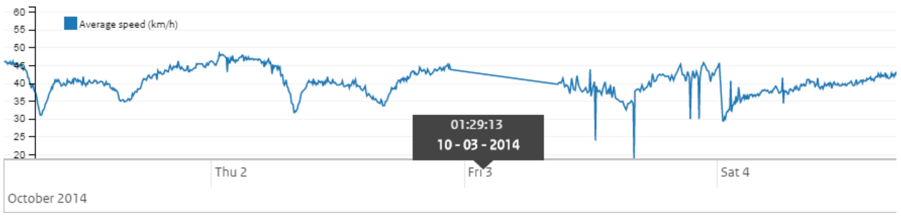


Figure 35: Timeline chart, showing the average average speed over the entire graph.

As with the spectrograph the timeline chart also supports showing multiple attributes, as is shown in Figure 36. Each different attribute is shown as a separate line with a unique colour.

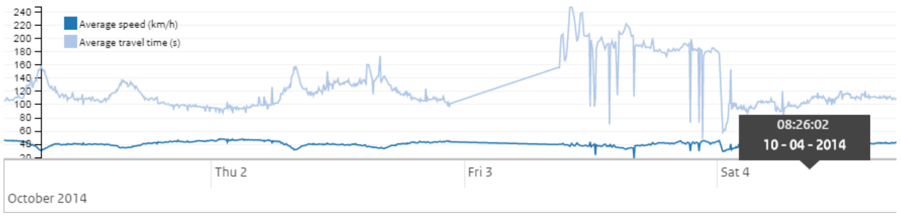


Figure 36: Timeline chart, showing the average average speed over the entire graph.

Interaction

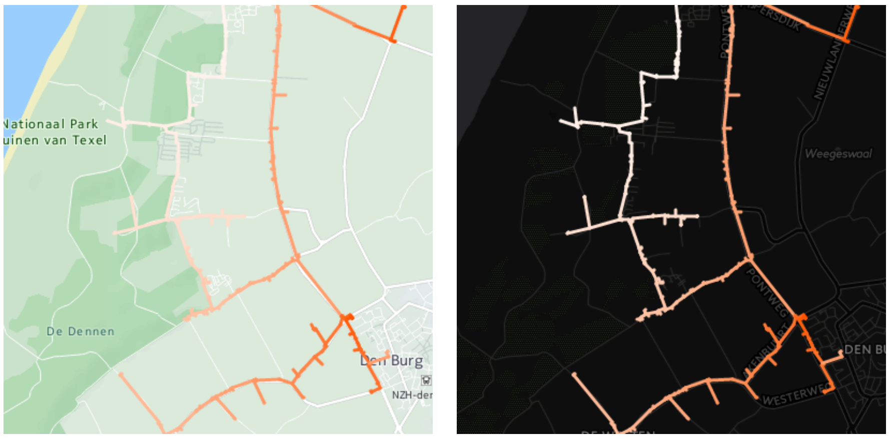
As discussed previously the timeline offers the ability to shift through time, zoom in out and, and select a time slice to visualise on top of the map. No additional interaction has been build into the timeline element of the analysis platform. It does however react accordingly when interacting with for example the spectrograph. Choosing a smaller time range there updates the time range shown on the timeline, and choosing a subset of features on the spectrograph or map changes the set of data on which the timeline chart is based.

5.2.4 *Configuration*

A number of useful features were already present in CommonSense, such as the interactive map and the attribute panel discussed in Section 5.1.1. The implementation of these elements was not modified over the course of this project. For them to function correctly however, they need to be configured. This involves declaring attributes, styling features and choosing background layers.

To use CommonSense effectively, one needs to tell it what kind of attributes a feature can have, and how to display these. For each type of feature, for example buses and branches, one needs to declare a `FeatureType`, which specifies the name of the type, its styling, and its attributes. This styling specifies properties such as the icon to use, opacity, and the fill and stroke colours to use when not colour mapped. For each of the attributes, one needs to specify a label which will be used in for example the attribute panel. CommonSense also needs to know whether an attribute is text, numerical or a date. With this information CommonSense elements can apply appropriate styling when displaying the attribute values. Finally, it is possible to specify whether an attribute should be shown in the attribute panel or not. This is useful for attributes such as the unique identifier discussed in Section 4.4, which are required by the application, but are not important for an end user to view.

Finally a choice of background layer is also important, since the structure of the graphs will be rendered and colour mapped on top of this. If a colour in the colour map is also used in the background layer, this may result in the graph being hard to see. See for example Figure 37a. The parts of the graph which are colour mapped to white are very hard to distinguish from the roads shown on the background layer, which are also shown in white. Many of the background layers found by default in the CommonSense framework



(a) Graph rendering on a light background layer.

(b) Graph rendering on a dark background layer.

Figure 37: The choice of background layer has a large impact on the visibility of the graph rendering.

are very light, causing issues with the colour map function shown in the figure. Because of this a new background layer option was added, Dark Matter [93], which is shown in Figure 37b. This is a background layer created and hosted by CartoDB. On this layer the graph can be clearly distinguished from the roads.

5.3 DISCUSSION

CommonSense already offers some useful functionality, such as the interactive map, attribute panel, a timeline and filters. It is however severely lacking in its ability to cope with large datasets, and its features geared toward temporal exploration are limited. Two temporally oriented features were added successfully to the framework to solve the latter problem, but unfortunately due to time constraints no support for large datasets could be added to CommonSense.

6 EVALUATION

In this chapter, the realised solution is evaluated to determine the success of this work. In Section 2.1, three types of analysis were discussed which are desirable in an analysis solution; value oriented analysis, time oriented analysis and comparison oriented analysis. To evaluate whether the implemented solution succeeds at offering these types of analysis, the solution has been used to analyse two datasets. The observations made during these analyses will be reported next.

6.1 REQUIREMENT SATISFACTION

Unfortunately the development of the ValueFlex project has come to a temporary halt halfway into this project. As a consequence no dataset is available of a large scale electrical grid simulation. However, other datasets can be used to verify the effectiveness of the analysis solution created in this work, since the developed solution has been set up in a generic way to allow its use with future projects. An unfortunate consequence however is that for these different datasets no expert is available to provide questions a potential user would wish to ask, nor can any conclusions be verified. To verify the solution, two datasets have been used. The first is a dataset of the gas distribution network of the island Texel in the Netherlands. The second dataset contains traffic information of major roads around the city of Aarhus in Denmark. Both these datasets will be analysed in the following sections.

According to the work of Telea, the goals of visualisation can be categorised into two groups [87]. The first goal is to find an answer to concrete questions, which generally have a concrete answer such as a single number, while the second is to find facts about a given problem that we were not aware of. The analysis described in the following sections will primarily focus on the analysis of the second type. Instead of trying to answer concrete questions, the approach used in these sections is to view an overview of the dataset, and from there investigate any oddities and patterns that stand out. This may lead to questions of the first category. This way we hope to still prove the validity of the proposed solutions by showing how a wide range of questions can be answered using the implemented solution, even if the originally intended data source is not available.

6.1.1 *Gas distribution network Texel*

The gas distribution network datasets contains simulated data of the pressure within the various pipes of the gas network of Texel over the course of two days, measured at 15 minute intervals. Three different types of pipes are distinguished, high pressure, medium pressure, and low pressure. For each of these pressures the dataset contains, from high to low pressure, 265, 1536 and 18999 pipes. This gives us 50.880, 294.912 and 3.647.808 recorded pressure values respectively, for a total of 4.105.728 pressure values. For each pipe the geographic coordinates of its beginning and end are provided.

This dataset has been split up into three disjoint datasets, where each contains the pipes of a given level of pressure. This was done for two reasons. First of all, without the dynamic data serving solution, the CommonSense application is not capable of visualising this amount of data all at once. On our testing setup which uses the browser Google Chrome, when loading just the low pressure parts of the network, the browser aborts the web application as a result of memory issues. Because of this during analysis only the medium and high pressure parts of the network have been used.

Secondly, the pressure values found in the different subsets have relatively small variation, but the median pressure is very different. Therefore, when attempting to display these values simultaneously using a single simple linear colour map, all pressure values of the same subset will be effectively mapped to the same colour. This is because the differences in pressure between elements of one subset are negligible when compared to the differences in pressure between the subsets. While solutions exist to circumvent this problem, such as the use of a non-linear colour mapping function, currently the implementation does not support such functionality.

Figure 38 contains a screenshot of CommonSense showing an overview of the medium pressure elements of the gas network. The spectrograph in this figure is sorted by the average pressure of each pipe, with the lowest average pressure pipes at the top and the highest pressure pipes at the bottom. A clear pattern immediately becomes apparent in this dataset by looking at the spectrograph and the timeline chart. In particular on the timeline chart, in the morning of both the days present in the dataset we can see the pressure over the entire network going down, creating a minimum around 8 AM. This drop can be explained by people getting up for work, at which point they turn on the heating and take a shower. After this initial spike the pressure goes up a bit again, although not quite to the same level which is reached during the nights. In the evenings when people come back home after work a similar drop is visible as the one occurring in the morning, as consumption again increases as people turn on the heating and start cooking dinner. The pressure clearly shows a pattern that repeats on both days, which is explained by the behaviour of the island its inhabitants.

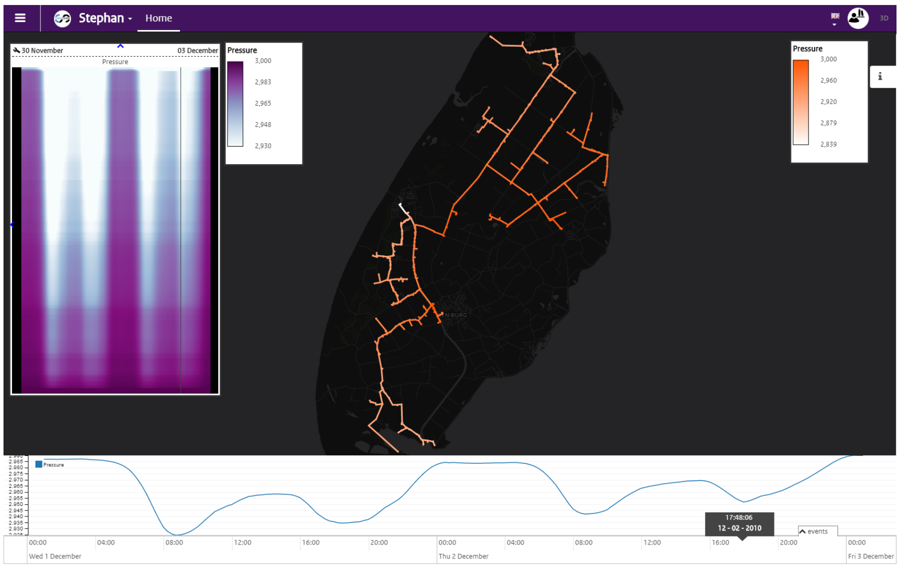
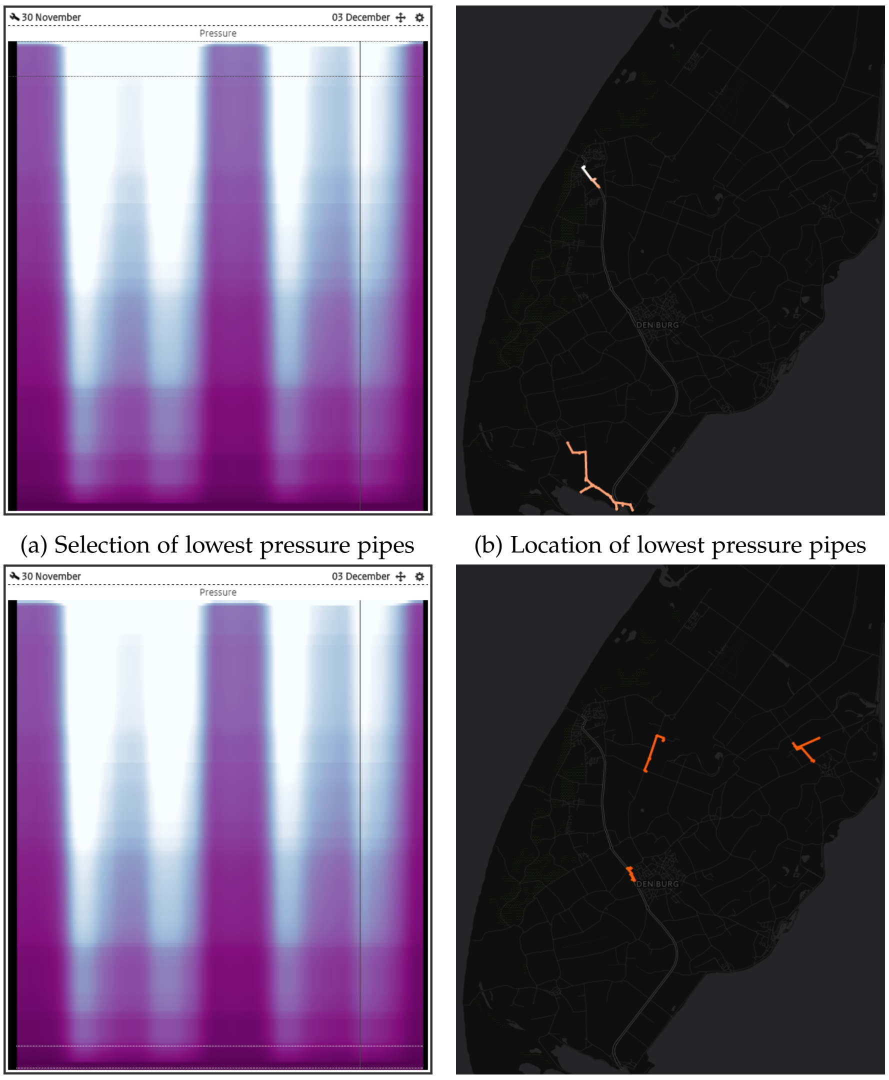


Figure 38: An overview of the medium pressure gas network dataset of the island Texel visualised using CommonSense. The spectrograph is sorted by the average pressure of each pipe, with the lowest average pressure pipes at the top and the highest pressure pipes at the bottom.

On the spectrograph it becomes apparent this pattern occurs over the entire network, but the local minima and maxima are not equal across all elements of the network. In particular during the day, elements that appear near the top of the spectrograph show a drop in pressure, while the pressure for those elements that appear at the bottom of the spectrograph remains almost constant. The map gives a rough indication of where the elements with the highest and lowest pressure appear on the island as the pressure of the pipes is colour mapped onto them. The pressure appears to be higher centrally and lower near the edges of the island. By selecting a subset of the dataset on the spectrograph, the subset that contains only features from the top or the bottom of the spectrograph, we can easily see where exactly the



(a) Selection of lowest pressure pipes (b) Location of lowest pressure pipes

(c) Selection of highest pressure pipes (d) Location of highest pressure pipes

Figure 39: The location of the lowest and highest pressure pipes on the spectrograph can be identified by selecting them.

low and high pressure parts are located. The selections and resulting map views are shown in Figure 39. The fact that the high pressure parts have a more constant pressure than the rest of the medium pressure network can be explained by viewing the high pressure part of the network, as is shown in Figure 40. The medium pressure parts that remain constant are close to the points where the high and medium pressure networks are connected. As new gas is fed into the medium pressure network from the high pressure network, the pressure hardly drops at all.

From the spectrograph it becomes apparent there are no significant temporal outliers or events, as there appear to be no sudden changes in the pressure of any of the features in this dataset. Each feature behaves very similarly over time, with pressure values changing smoothly. This is most likely because the measurements in this dataset do not originate from actual sensors, but from a simulation. This results in the data being very clean, with pressure values almost always strictly increasing or decreasing towards the next minimum or maximum.

6.1.2 *Traffic data Aarhus*

The second dataset analysed using the tool developed in this thesis is a traffic sensor dataset of the CityPulse project [94]. The main objective of the

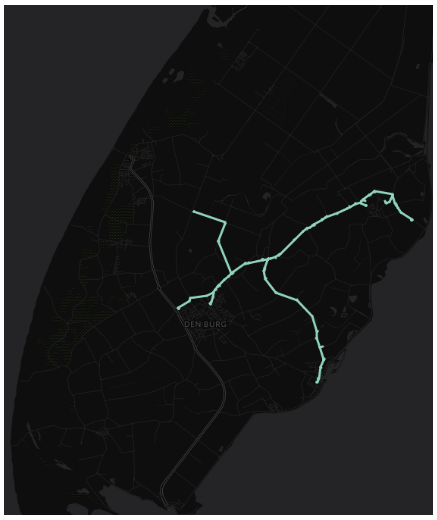


Figure 40: Location of the pipes of the high pressure network.

CityPulse project, as described by their website, is to *"develop, build and test a distributed framework for the semantic discovery and processing of large-scale real-time IoT [internet of things] and relevant social data streams for knowledge extraction in a city environment"*. They provide a number of datasets on their website for use under the Creative Commons Attribution 4.0 International license. Among these datasets are three datasets which contain the traffic sensor data of major roads around the city of Aarhus, Denmark. Each dataset describes a different time range. In this work the dataset describing the months October and November in 2014 is used.

This dataset contains the average speed, average travel time and vehicle count for 449 road segments, where both sides of the road are considered separate segments. These attributes are recorded every 5 minutes over this two month interval, although as will be demonstrated shortly data is not always recorded. Once again the entire dataset proves to be too much for the CommonSense application without the dynamic retrieval system in place, thus a subset was used. Because of this only the first week of this dataset was used, which describes the days of Wednesday October 1st 2014 till Tuesday October 7th 2014. For comparison, this subset has 646.496 recorded values for each of the individual attributes, whereas the whole dataset has 4.383.048 values. This means these datasets have 1.939.488 and 13.149.144 total data values respectively.

The meta data of the dataset describes where the start and end points of each road segment are located in the real world. Unfortunately, no accurate geometry was provided. Simply drawing straight lines between the start and end points results in lines that poorly match the real world geometry of the roads they describe, as shown in Figure 41a. Real world roads often do not follow straight lines, especially near cities. The problem is aggravated because many of the road segments measure traffic across junctions, which appear as sharp corners on a map. However, by querying the Google Directions API [95], which provides a route between any two given points, a more accurate geometry has been retrieved. The result is shown in Figure 41b. Which form is preferred depends of course on the user. Some user may be simply interested in the rough structure of the network, and less so in the exact location of all the pieces. In this case the former layout may be preferred. Personally, we believe the more accurate representation is favourable in this situation, as the map is used as a tool to relate the data back to the real world, which is easier when the drawing most accurately represents its real world counterpart. Because of this the accurate roads are used in the following images.

An overview of the Aarhus dataset is shown in Figure 42. In that screenshot the spectrograph and timeline chart show the average speed attribute of

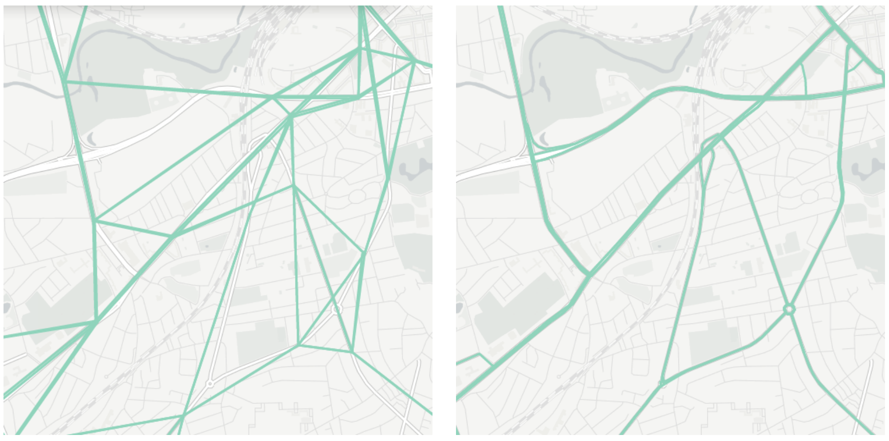

(a) Straight lines between road segment start and end points (b) Line geometries retrieved from the Google Directions API

Figure 41: The CityPulse dataset only describes the start and end points of road segment, which when directly connected results in poorly the drawn lines poorly matching the roads they describe. By using the Google Directions API more accurate road geometries were realised.

the road segments. The spectrograph is sorted by average speed, arranged in ascending order from top to bottom. It immediately becomes apparent this dataset is much more noisy than the gas dataset, as the values shown in the spectrograph and timeline chart show many small fluctuations. This is most likely due to the fact that this is a real world dataset, measured using sensors, as opposed to being generated by a simulation. It does however also show similarities with the gas dataset. In particular on the timeline chart, it becomes apparent this dataset also contains a daily repeating pattern. The average speed drops twice throughout the day, around 8 AM and 4 PM, which obviously correlate to the peak hours during which most people commute between home and work. Annotation A in Figure 42 highlights these drops in speed for the first two days. The same behaviour can be seen in vertical lines in the spectrograph, but it is easier to identify on the timeline

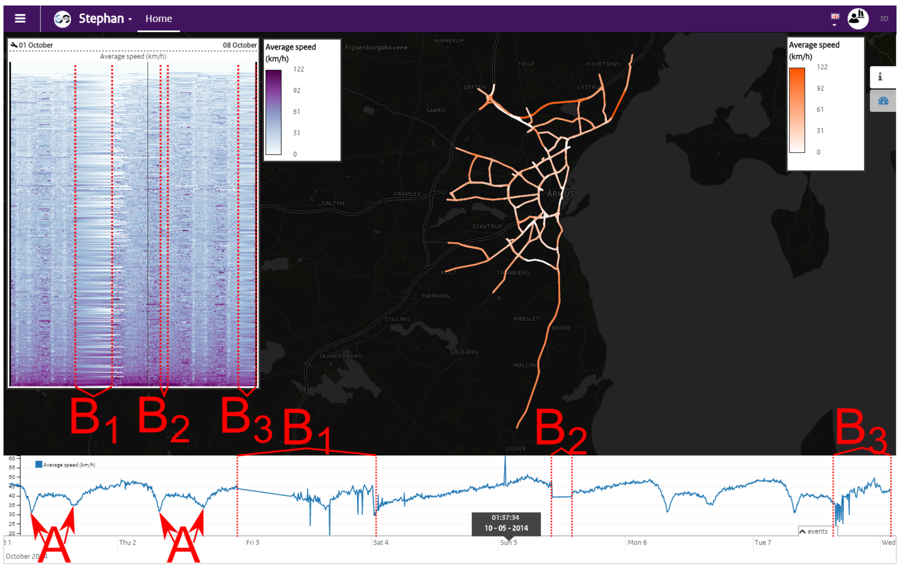


Figure 42: An overview of the traffic datasets of major roads around the city of Aarhus. On the spectrograph and timeline chart the speed attribute of the road segments is shown. The spectrograph is sorted by average speed, arranged in ascending order from top to bottom. Interesting patterns in the visualisations are annotated in red.

chart. Unexpectedly, these drops do not appear during the weekend. The average speed still clearly fluctuates, but there are no large sudden drops at any hour of the day. This is of course because most people do not work during the weekend, thus resulting in less traffic on the roads which allows for better traffic flow.

Both in the spectrograph and the timeline chart three time ranges appear where the values change suspiciously smoothly relative to the rest of the data. These ranges are indicated by annotation B in Figure 42 on both visualisations, where B_1, B_2 and B_3 refer to the same time range on both the spectrograph and the timeline chart. While over the entire time range both visualisation show constant small fluctuations, here the values suddenly change linearly over time. The first time range where this happens actually covers the 8 AM rush hour, so clearly a significant change similar as shown on the other days would be expected. A quick peek at the raw data informs us there is actually no data available at these time ranges. Apparently there was some issue which resulted in the loss of this data. Both the spectrograph and timeline chart visualisation tried to reconstruct the missing data through interpolation. As simple linear interpolation is used, this results in smooth transitions which stand out from the other data which is constantly fluctuating. Some parts of these ranges, such as the second half of the first smooth period, and the entire third smooth period, do contain fluctuations on the timeline chart, while the spectrograph still appears smooth. This is because just a few of the sensors start reporting values again. Thus a new graph wide average can be computed, while most part of the graph will still show the linearly interpolated values.

A seemingly related observation is some constantly coloured horizontal bars in the spectrograph. An image highlighting a number of these horizontal bars is shown in Figure 43. The spectrographs in this image are again sorted by speed. These bars seem to suggest the average speed remains perfectly constant on some road segment. This is however caused by another peculiar behaviour in the data. The bottom spectrograph in Figure 43 gives an enlarged view of a small section of the top spectrograph, giving a detailed view of one of such horizontal bars. Clearly the speed remains perfectly constant for a significant portion of time. When also viewing the vehicle count attribute, which is shown on right hand side of the spectrograph, a strong correlation appears. Each time the speed remains constant, the vehicle count part of the spectrograph is white, which corresponds to the value zero. While one might expect the recorded speed to become zero when there is no traffic, the sensor appears to keep reporting the same speed over and over until a new car passes by. Only then the average speed is updated. As a result these horizontal bars appear where the speed remains perfectly constant. A keen observer may have noted these bars all appear in vertical columns. Unsurprisingly, these columns correlate to the nights. Especially in the vehicle count part of the spectrograph the nights show a clear contrast to the days. It is unsurprising these period where no cars are registered primarily occur during the nights. As a result, these horizontal bars mostly appear during the night time.

The spectrograph can also show other correlations between attributes. In the case of the CityPulse traffic dataset, all attributes are strongly correlated, as can be seen in Figure 44. This figure shows all three attributes present in the CityPulse traffic dataset; the average speed, the average travel time and the vehicle count; both on the evolution spectrograph and the timeline chart. While the use of a shared y axis on the timeline chart does reduce the visibility of the correlations somewhat, the timeline chart does shows a strong correlation between the three attributes. The speed is inversely correlated with the vehicle count and travel time, and the vehicle count and travel time are positively correlated. This is of course to be expected. As the vehicle count goes up, roads becomes busier, which reduces the average speed. As the speed decreases, naturally the travel time increases. Thus all

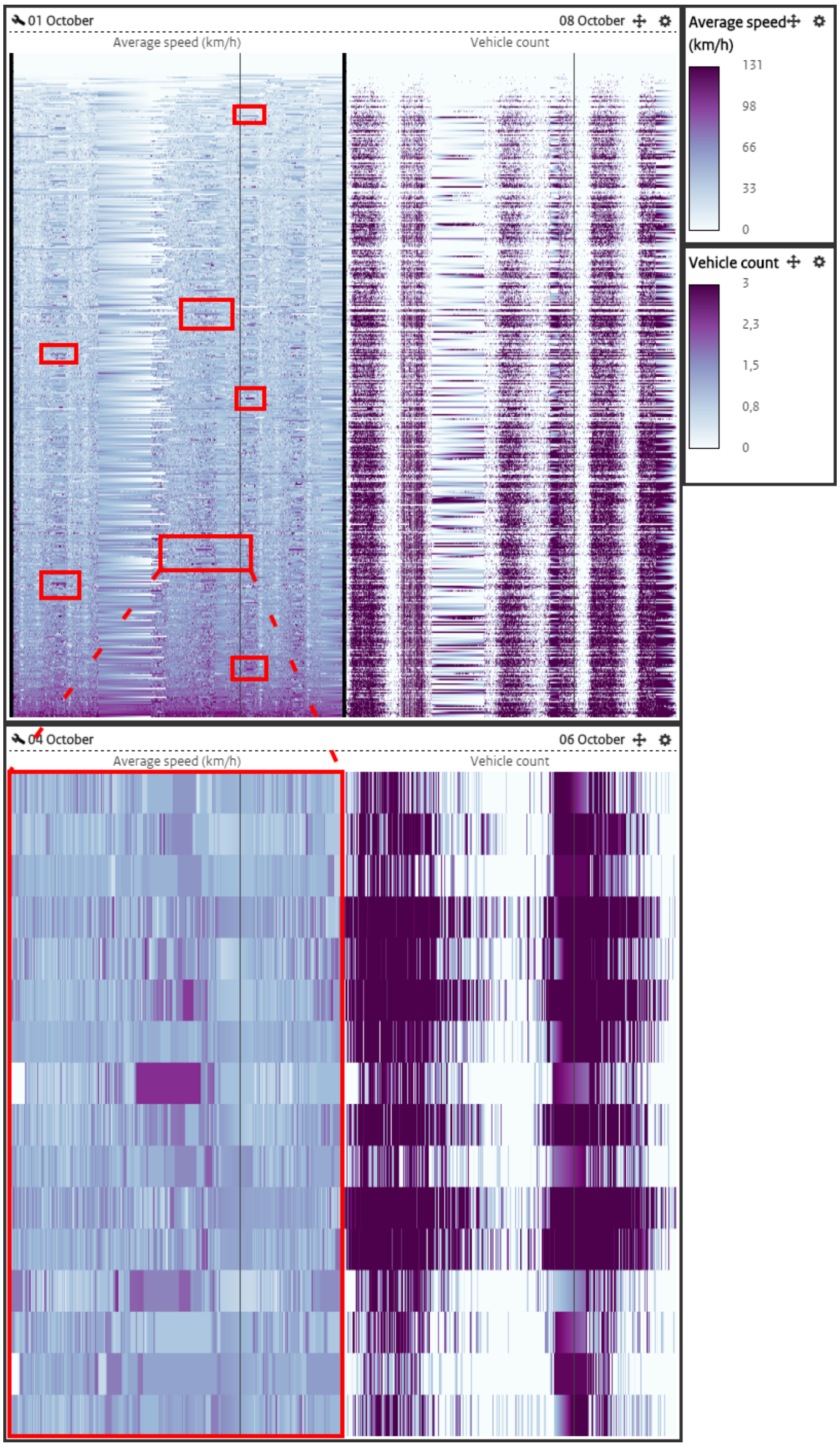


Figure 43: Two snapshots of spectrographs showing the CityPulse traffic dataset, both sorted by average speed, with rows arranged in ascending order from top to bottom. Both spectrographs show two attributes; the average speed on the left, and the vehicle count on the right. On the top spectrograph horizontal bars exhibiting constant colour are highlighted in red. The bottom spectrograph shows a zoomed in view of one of these horizontal bars.

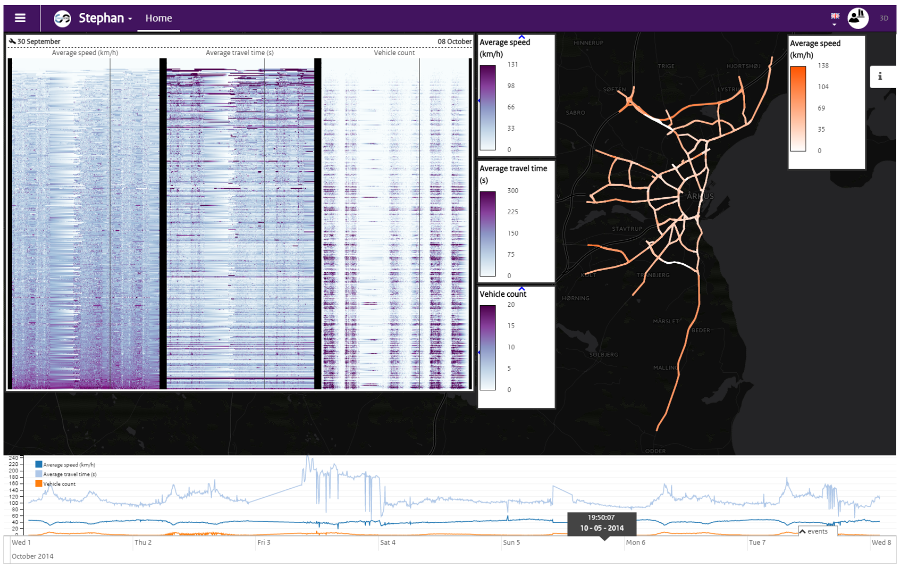


Figure 44: Overview of the CityPulse traffic dataset showing all three of the attribute, average speed, average travel time and vehicle count, on the evolution spectrograph and timeline chart. Rows are sorted in ascending order by average speed, including those visualising the travel time and vehicle count.

three attributes are expected to be correlated. The existence of this correlation can also be confirmed using the spectrograph. As the darkest rows appear near the bottom in the average speed column of the spectrograph, where the speed is highest, these same rows are the lightest in the case of the average travel time, indicating a relatively low travel time. These same rows are also dark in the vehicle count column, as higher speeds allow for more throughput. However, not all individual rows show this same pattern. For example, one dark purple line appears near the bottom of the average travel time column. This is of course because the travel time does not depend solely on the speed at which vehicles drive on a given road segment, but also its length. This dark line in particular belongs to the long road which lies south of Aarhus. This road forms the longest road segment in this dataset, thus naturally the average travel time will be above average on this road.

One more oddity that appears in the visualisation is some sharp spikes on the timeline chart. One may notice these spikes happen primarily near the areas with missing data, as indicated in Figure 42. In particular near the first of these time periods where data is missing one may notice on the evolution spectrograph that the sensors in some of the road segments have started reporting values again sooner than others. This means that at some of the timestamps, only a limited number of values is recorded. Unfortunately, due to the way the timeline chart is constructed, this leads to this erratic behaviour in the chart. It is constructed by first finding all the time points in the dataset at which a value is recorded. The algorithm then simply takes all the values recorded at each time point, and averages them. This clearly leads to values that misrepresent the average of the network, as during these periods with missing data, only a small number of values are recorded. Thus the resulting average is computed using only a small subset of the network. As more sensors start reporting data again, the average stabilizes again.

Thus these spikes are caused by a limitation of the implementation. The timeline chart is not currently designed to cope with missing data. A non-trivial question is however what the expected behaviour is in the case of missing data. The answer will depend on the specific dataset being visualised, as the cause and meaning of missing data depends strongly on the context of the data, and the goals of a user. Does a user wish to know which sensors are failing to report data, then the visualisation should attempt to highlight parts of the dataset where this problem occurs. Does the user however expect a best approximation of the missing data, the application should instead attempt to reconstruct the data. However even then many different techniques could be used to perform this reconstruction. The quantity of missing data may also be a factor. While here the entire network of sensors stops reporting values, which may be useful to be informed about, a single missing value from one sensor may be less important. Thus context is important. In the general case, it will even be impossible to decide whether data is in fact missing, or that the frequency at which values are recorded is simply strongly erratic. A feature to highlight areas with missing data may be desirable, for example by shading the timeline chart by which percentage of the features contributed to an average, but even then it must be left to the user to define exactly when data is in fact missing.

6.2 DISCUSSION

In this chapter we have shown how the solution discussed in this thesis can be used to analyse temporal geospatial multivariate graph datasets. While we were unable to test the solution on datasets of the scale we originally intended, we have clearly shown how the visualisations can be used to perform both value oriented and time oriented analysis on moderately sized datasets. The addition of the data management system should however not introduce significantly different results, due to the simple fact that no larger amount of data could be viewed simultaneously than was done during the evaluation. The introduction of the data management system would merely result in the ability to view differently detailed versions of the data. We identified different patterns that emerged over time, such a daily repeating patterns. We also explained observations by looking at specific vehicle count values. The third type of analysis, comparison oriented, has however not been evaluated. Unfortunately, we found no appropriate alternative dataset which allows us to perform a meaningful comparison analysis. While functionality is present to allow for this, the effectiveness of it remains unproven. The implementation also showed some limitations, such as its inability to support non-linear colour mapping functions, and the lack of functionality to highlight missing data.

7 DISCUSSION AND CONCLUSIONS

This chapter first discusses the results achieved in this work in Section 7.1, and compares the results to what was out lined in Chapter 1. We will also answer the research questions in Section 7.2, and share some ideas for future work in Section 7.3.

7.1 DISCUSSION

We set out to create an interactive analysis platform for large scale temporal geospatial multivariate graphs in a web-based environment, using the CommonSense framework. While we succeeded at some of these goals, we unfortunately did not achieve everything we hoped to. In particular the support for large scale graphs, while thoroughly investigated and a promising solution was found, has not been realised in the actual implementation. As such we could not verify the success of our proposed data management solutions. As such the datasets used during the evaluation were of a smaller size than originally intended, but we believe it to be likely similar results would be achieved using larger datasets. We successfully integrated our visualisation ideas into the CommonSense framework, even though this often proved challenging due to preliminary documentation. These ideas include a highly detailed temporal view called the evolution spectrograph, which to our knowledge is a novel approach for large scale temporal geospatial multivariate graphs visualisation, in particular when combined with an interactive map. While we were unable to test our solution on the originally intended use case, nor have we tested the solution at the scale we originally intended, we have shown that the additions we proposed can be used effectively to analyse temporal geospatial multivariate graphs of up to a moderate size in a web-based environment, proving the solutions can be applied on these types of graphs in general, not just electrical grids.

7.2 CONCLUSION

In this work we have shown an answer to the main research question: **"How can a large scale geospatial temporal multivariate graph be visualised, allowing interesting behaviours and pattern to be analysed, in a real time, interactive and generic way in a web based environment?"**.

To answer this question, we first discussed a number of different visualisation solutions, each of which had their own advantages and disadvantages. As there are many aspects to large scale geospatial temporal multivariate graph datasets, we found it best to have visualisation focus on only a few of these aspects. Some are more focused on finding correlations, while others show the relations of the graph. With the spatial aspect of the graph being covered by functionality in the existing visualisation framework, this work focused more visualisations that help gain insights in other aspects of these graph datasets. We proposed to use the evolution spectrograph, since it gives a very detailed view of the temporal behaviour of individual components of a network, and with some tweaks could even support multiple attributes and the detection of correlations.

We also investigated the size of the datasets we wanted to analyse with these solutions, and how to cope with such large volumes of data in a web environment. The graphs considered in this work are too large to display in a browser directly, being able to grow into the terabytes in size. They contain too many features, and too large time ranges, for which there simply are not enough pixels available on a conventional screen to display it all. Transferring these amounts of data is also impossible in a real time setting, nor can it be stored in a web environment. By aggregating, storing and serving the spatial and temporal aspect of the datasets separately, these problems were dealt with. Many different options exist for these steps, and the exact solution that should be used depends on the data.

To verify the effectiveness of the proposed solutions, they were implemented. Integrating these proposed solutions into CommonSense turned out to be challenging and time consuming. Working with new unfamiliar frameworks, libraries and languages, as well as preliminary documentation meant the realisation took more time than initially expected. As a result only the visualisations discussed in this work were implemented. A way to integrate the data management part is proposed, but the implementation is left to future work.

The success of the proposed visualisation solution has been shown in this work by analysing two real-world datasets which fit the description of temporal geospatial multivariate graphs. The realised visualisations allowed for the discovery of several interesting and potentially important unknown facts in the analysed datasets. While the realised solution does not cater equally well to all the types of analysis that are desired, it does add significant analysis options with respect to temporal data to the CommonSense framework.

7.3 FUTURE WORK

While working on this project many opportunities presented itself to add more functionality, or improve the functionality discussed throughout the thesis. Unfortunately, there was not enough time to explore each of these opportunities. In this section some possibilities for future work, including some of the most promising additions and improvements are discussed.

7.3.1 *Verification*

Unfortunately we were unable to verify all proposed functionality in this work. In future work it would be important to verify some of the suggestions made. Primarily this of course means verifying the solutions proposed to support large datasets in CommonSense should be implemented and tested. Significant amounts of work went into the research of these solutions, thus it is very unfortunately these were never realised. We were also unable to verify how well the implemented solutions can be used to perform comparison oriented analysis. Unfortunately we did not have the right datasets to perform any kind of meaningful comparison oriented analysis. Future work could search further for an appropriate dataset.

7.3.2 *Additions*

One way to further improve the analysis capabilities of the framework would be to add additional visualisation options. These additions could focus on any aspects of the data. An example would be to extend the map visualisation, by computing an estimate of a scalar field of an attribute of the graph. This opens up a lot of possibilities for visualisations, such as a heat map, contour lines, or glyphs. These techniques may be interesting when the dataset contains observations of a set of sensors. This is different from the datasets considered in this thesis, in that attributes measured at a pipe or cable make no sense when interpolated in the middle of an empty field. In the case of sensors which, for example, measure the quality of the air, the sensors give exact measurements at those locations, but the measured attributes are also well defined in other locations, just not measured.

As part of this project the company Phase To Phase was visited, which is based in Arnhem, the Netherlands. This company specialises in developing tools for the analysis, design and monitoring of electrical grids. They were kind enough to show some of their latest developments for the visualisation of electrical grid data. While none of the ideas were directly implemented in the CommonSense framework during this project, some of their ideas are very interesting and could be implemented in a future work. For example, they provide functionality that improves the users ability to relate a location on the map to a real world location. Given a highly detailed spatial dataset of, for example, an electricity or gas network, it may be interesting to visit the real world locations which appear problematic during analysis in CommonSense. They integrated a street view service such as Google Street View [97], which results in a more accurate mapping between the data and the physical location it refers to can be achieved. They improve on this even further by augmenting the street view by a precise indication of the the physical location of the object, as shown in Figure 45.

The visualisation could also be extended to use a three dimensional rendering as opposed to the flat two dimensional map currently being used. This could allow the third dimension to be used to display temporal information, for example by using 3D icons, as shown in Figure 46 from the work of Tominski et al. These 3D icons could be used to display the temporal behaviour of a single feature at the location of the feature, so the user does not have to first identify the feature in the evolution spectrograph. The pencil shown in Figure 46a can visualise one attribute per face of the pencil, encoding time along the length of the pencil. The helix icon, shown in Figure 46b, is constructed similarly, where time progresses along the ribbon. This icon is particularly well suited for cyclic data, such as that shown in the calendar views in Section 2.2.5. With that type of data, each revolution of the ribbon could represent a single day, lining up the same point of time of each day in a vertical column. Another master thesis project at TNO actually already looked into the possibility of adding three dimensional visualisations to the CommonSense framework. Some of his results are shown in Figure 47, in which he utilized the third dimension to visualise an additional attribute of polygons by using height.

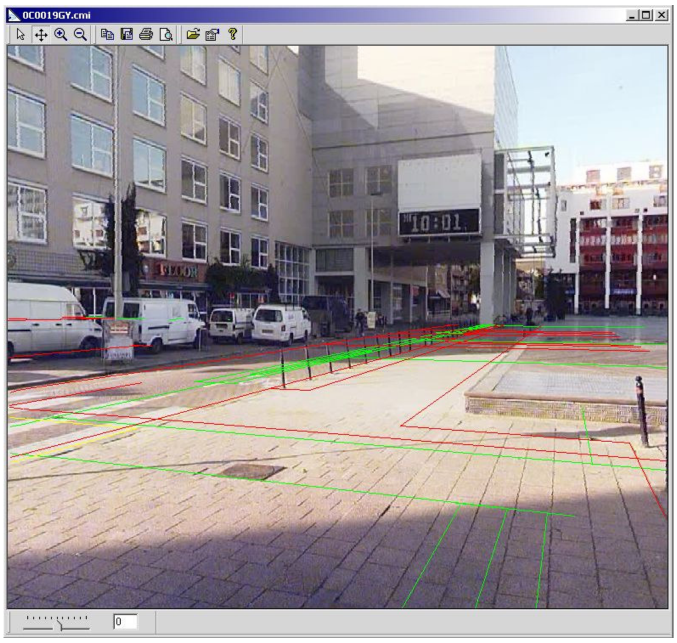
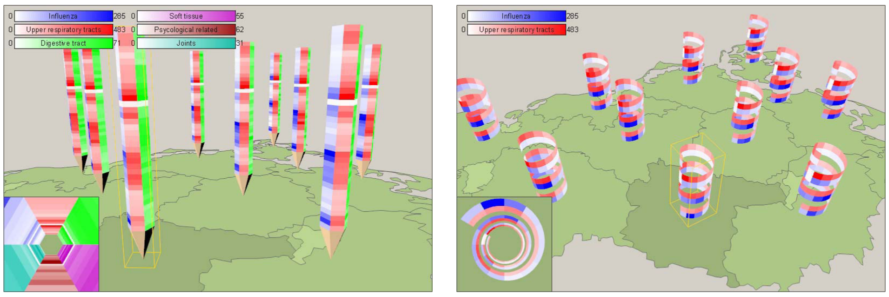


Figure 45: Example of an augmented street view rendering. In this image, the location of power lines is indicated by coloured lines. *Source: Verbree et al. "Interactive navigation services through value-added CycloMedia panoramic images"* [96]



(a) Pencil icons (b) Helix icons

Figure 46: 3D icons can be used to display additional information regarding a location on the map, where time is encoded along the length of the icon. *Source: Tominski et al. "3d information visualization for time dependent data on maps"* [98] © 2005 IEEE

7.3.3 *Improvements*

The implementations discussed in this thesis also have some room for improvement. The user could be given more control over the visualisation, and more interactions could be added.

While exploring, queries need to be kept small to keep the application interactive, but while the user is leaving the view stationary, this time could be used by the application to retrieve more detailed information. Spatially this must be done very carefully, since using a too detailed representation of the network can easily result in the map being cluttered. However temporally, especially in the case of the timeline graph, there is more screen estate for additional information. This can improve the preservation of details in the data.

This work did not focus on creating a user friendly solution. There are plenty of opportunities to improve the user experience user experience. One aspect is consistency. For example, currently the user is only able to select a time range on the evolution spectrograph, not on the timeline or timeline graph. Similarly, the user is able to drag the timeline to shift the focus time range through time. This same behaviour is lacking in the evolution

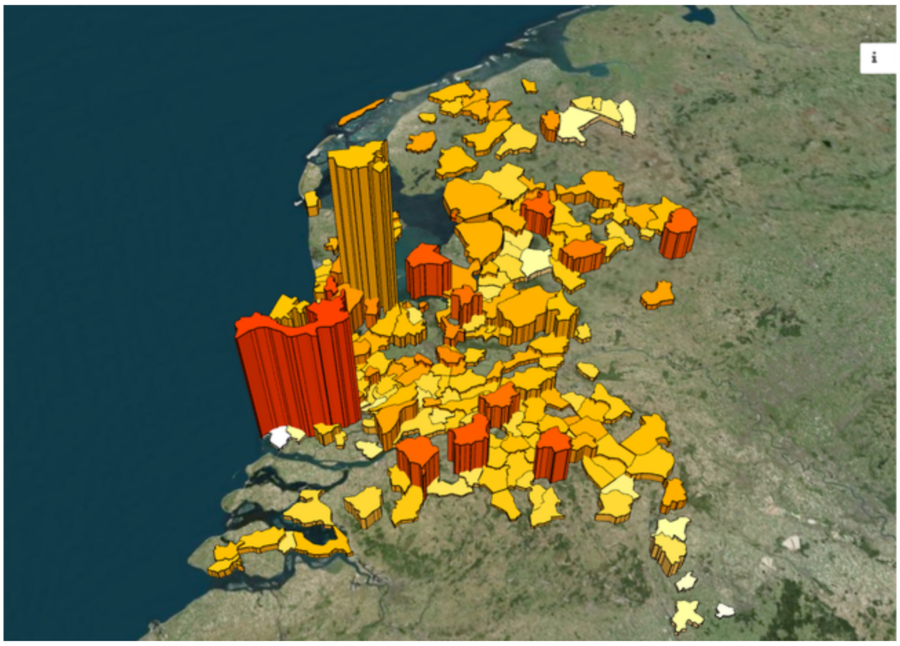


Figure 47: Example of the three dimensional rendering capabilities added to CommonSense by Lukas de Boer.

spectrograph. Also the ability to change the temporal focus point, present on the timeline, is missing on the spectrograph. By making these interactions more consistent, using the application becomes more intuitive, as opposed to having to learn how to use each view from scratch.

By allowing the user to customize more aspects of the visualisation, it could also be tailored more specifically to the users needs. For example, the colour mapping function is currently set by CommonSense itself, using the minimum and maximum values of a dataset. By allowing the user to customize the minimum and maximum values used to map values to the colour map, clamping any values outside of the range to the minimum or maximum, details of interest to the user can be made more pronounced by increasing the colour differences between them. For example, consider a scenario where all values between 0 and 1 signify everything is operating normally, and any values higher than 1 indicate a problem. The range of values which are most interesting then starts at 1, instead of the value of 0 chosen by CommonSense. Part of the colour map is then essentially wasted on uninteresting data. Allowing the user to choose how values are mapped to colours would solve this problem.

Another addition would be to compute multiple version of the temporal data using different aggregation functions, such as the minimum, the maximum, the variance, or any other metric of interest in a particular project. In the current implementation only one aggregation function is provided, resulting in only one version of the temporal data. By computing multiple aggregations, the user could then be given the option to choose between these aggregates dynamically within CommonSense. This could improve the ability to find time points or ranges of interest.

Another potential area of improvement is the spatial aggregation algorithm. Currently cluster selection purely looks at spatial proximity. The resulting clusters represent a cell of the spatial quad tree. The locations of these cells are however very arbitrary, as they have no connection to any real infrastructure. If for example a city happens to lie exactly on the boundary of two cells on which the graph is simplified, the city will be represented by two separate clusters, while a single cluster for the city would be more intuitive. By selecting clusters based on real world infrastructures, this problem could be avoided. For example, clustering could be done on the level of countries, provinces, towns, villages, district, streets or even households. By clustering this way the resulting clusters represent well defined physical regions, resulting in a more natural simplified infrastructure. Some challenges need to be overcome however. By aggregating this way it becomes harder to acquire a desirable level aggregation, as there are only so many natural simplification 'levels' available, with nothing in between. Some cities or streets may also be very elongated, or even crescent shaped, which makes describing them well by a single point hard. This solution also requires accurate descriptions of the regions to simplify by. While datasets such as CitySDK [99] exist, acquiring such data for the entire world will be challenging. Finally, not all areas are equal in size, some countries are much smaller than others. This may result in many more aggregated points in some areas than in others.

Another aspect of spatial aggregation that could be investigated further is the different types of sub networks present in infrastructures. Gas networks use different pressures for different parts of the network, high pressure for long distance high volume transport, and lower pressures at the consumers. Similar concepts consist in infrastructures such as electrical grids, water supply networks and road systems. Generally, the higher throughput connections are less common, but most important for the overall structure of the graph. At a high level they describe how producers and consumers are connected. One could argue these connections are more important than lower throughput connections when viewing an overview of a network. Potentially aggregation could prioritise preserving these high throughput connections in the aggregated result, and focus primarily on aggregating the lower throughput elements.

BIBLIOGRAPHY

[1] Daniel Archambault, James Abello, Jessie Kennedy, Stephen Kobourov, Kwan-Liu Ma, Silvia Miksch, Chris Muelder, and Alexandru C Telea. Temporal multivariate networks. In *Multivariate Network Visualization*, pages 151–174. Springer, 2014.

[2] A Johannes Pretorius and Jarke J Van Wijk. Visual inspection of multivariate graphs. In *Computer Graphics Forum*, volume 27, pages 967–974. Wiley Online Library, 2008.

[3] Nederlandse Aardolie Maatschappij. Gebouwsensoren. `http://www.namplatform.nl/bouwkundig-versterken/gebouwsensoren.html`, march 2014. [Accessed 2015-09-02].

[4] TNO. CommonSense Map (csMap). `https://github.com/TNOCS/csMap`.

[5] Esri. ArcGIS Help - Features. `http://resources.arcgis.com/EN/HELP/MAIN/10.1/index.html#//000n00000070000000`.

[6] StatCounter. Global stats, July 2014 - July 2015. URL `http://gs.statcounter.com/#desktop-resolution-ww-monthly-201407-201507-bar`. [Accessed August 5th, 2015].

[7] Günther Brauner, Wiliam D'Haeseleer, Willy Gehrer, Wolfgang Glaunsinger, Thilo Krause, Henning Kaul, Martin Kleimaier, W L Kling, Horst Michael Prasser, Ireneus Pyc, Wolfgang Schröppel, and Waldemar Skomudek. Electrical power vision 2040 for europe. *A Document from the EUREL Task Force*, 2013.

[8] Omroep Gelderland. Het is gelukt: Lochem zit zonder stroom. `http://www.omroepgelderland.nl/nieuws/2088183/Het-is-gelukt-Lochem-zit-zonder-stroom`, 2015. [Accessed 2015-08-19].

[9] PowerMatcherSuite. The flexible power application infrastructure. URL `http://flexiblepower.github.io/technology/fpai/`. [Accessed 2015-08-19].

[10] Roger C Dugan, Mark F McGranaghan, and H Wayne Beaty. Electrical power system quality. *New York, NY: McGraw-Hill,| c1996*, 1, 1996.

[11] IEEE recommended practice for monitoring electric power quality. *IEEE Std 1159-2009 (Revision of IEEE Std 1159-1995)*, pages c1–81, June 2009. doi: 10.1109/IEEESTD.2009.5154067.

[12] Christophe Hurter, Ozan Ersoy, Sara Irina Fabrikant, Tijmen R Klein, and Alexandru C Telea. Bundled visualization of dynamicgraph and trail data. *Visualization and Computer Graphics, IEEE Transactions on*, 20 (8):1141–1157, 2014.

[13] Cesim Erten, Philip J Harding, Stephen G Kobourov, Kevin Wampler, and Gary Yee. Exploring the computing literature using temporal graph visualization. In *Electronic Imaging 2004*, pages 45–56. International Society for Optics and Photonics, 2004.

[14] Ramana Rao and Stuart K Card. The table lens: merging graphical and symbolic representations in an interactive focus+ context visualization for tabular information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 318–322. ACM, 1994.

[15] Peter Pirolli and Ramana Rao. Table lens as a tool for making sense of data. In *Proceedings of the workshop on Advanced visual interfaces*, pages 67–80. ACM, 1996.

[16] Tichomir Tenev and Ramana Rao. Managing multiple focal levels in table lens. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 59–63. IEEE, 1997.

[17] Alexandru Telea. Combining Extended Table Lens and Treemap Techniques for Visualizing Tabular Data. In Beatriz Sousa Santos, Thomas Ertl, and Ken Joy, editors, *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*, pages 51–58. The Eurographics Association, 2006. ISBN 3-905673-31-2. doi: 10.2312/VisSym/EuroVis06/051-058.

[18] Danny Holten, Bas Cornelissen, and Jarke J Van Wijk. Trace visualization using hierarchical edge bundles and massive sequence views. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 47–54. IEEE, 2007.

[19] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, 1985.

[20] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates. In *Human-Machine Interactive Systems*, pages 199–233. Springer, 1991.

[21] Alfred Inselberg. *Parallel coordinates*. Springer, 2009.

[22] Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *Visualization and Computer Graphics, IEEE Transactions on*, 14 (6):1539–1148, 2008.

[23] Fernando V Paulovich, Luis Gustavo Nonato, Rosane Minghim, and Haim Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *Visualization and Computer Graphics, IEEE Transactions on*, 14 (3):564–575, 2008.

[24] Rafael Messias Martins, Danilo Barbosa Coimbra, Rosane Minghim, and Alexandru C Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[25] Renato R O da Silva, Paulo E Rauber, Rafael M Martins, Rosane Minghim, and Alexandru C Telea. Attribute-based visual explanation of multidimensional projections. 2015.

[26] Jarke J Van Wijk and Edward R Van Selow. Cluster and calendar based visualization of time series data. In *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*, pages 4–9. IEEE, 1999.

[27] Jingwei Wu, Claus W Spitzer, Ahmed E Hassan, and Richard C Holt. Evolution spectrographs: Visualizing punctuated change in software evolution. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pages 57–66. IEEE, 2004.

[28] Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[29] Richard A Becker, William S Cleveland, and Allan R Wilks. Dynamic graphics for data analysis. *Statistical Science*, pages 355–383, 1987.

[30] Mark Derthick, John Kolojejchick, and Steven F Roth. An interactive visual query environment for exploring data. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 189–198. ACM, 1997.

[31] Jonathan C Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Coordinated and Multiple Views in Exploratory Visualization, 2007. CMV'07. Fifth International Conference on*, pages 61–71. IEEE, 2007.

[32] Bas Cornelissen, Andy Zaidman, Danny Holten, Leon Moonen, Arie van Deursen, and Jarke J van Wijk. Execution trace analysis through massive sequence and circular bundle views. *Journal of Systems and Software*, 81(12):2252–2268, 2008.

[33] Graham Wills. Linked data views. In *Handbook of Data Visualization*, pages 217–241. Springer, 2008.

[34] Institute of Electrical and Electronics Engineers. IEEE Std 91a-1991 and IEEE Std 91-1984: "Graphic Symbols for Logic Functions", 1991.

[35] International Electrotechnical Commission. IEC 60617: "Graphical Symbols for Diagrams".

[36] American National Standards Institute. ANSI Y32.2-1975: "Graphic Symbols for Electrical and Electronics Diagrams", 1975.

[37] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343, Sep 1996. doi: 10.1109/VL.1996.545307.

[38] Google. Google Maps. `https://www.google.nl/maps/`, .

[39] Microsoft. Bing Maps. `https://www.bing.com/maps/`.

[40] Nokia. Here. `https://www.here.com/`.

[41] Eric Gundersen. Visualizing 3 billion tweets, june 2013. URL `https://www.mapbox.com/blog/visualizing-3-billion-tweets/`. [Accessed August 5th, 2015].

[42] Joan Maso, Keith Pomakis, and Nuria Julia. Opengis web map tile service implementation standard. *Open Geospatial Consortium Inc*, pages 04–06, 2010.

[43] Ning Ruan, Ruoming Jin, and Yan Huang. Distance preserving graph simplification. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1200–1205. IEEE, 2011.

[44] Paul S Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. Technical report, DTIC Document, 1997.

[45] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[46] Filip Beć. A blazingly fast open source algorithm for POI clustering on iOS. `https://infinum.co/the-capsized-eight/articles/a-blazingly-fast-open-source-algorithm-for-poi-clustering-on-ios`, september 2014. [Accessed 2015-09-08].

[47] David M. Mount and Sunil Arya. ANN: A Library for Approximate Nearest Neighbor Searching. `http://cs.umd.edu/~mount/ANN/`, 2010.

[48] Parallel Universe. SpaceBase. `http://www.paralleluniverse.co/spacebase/`.

[49] James N Hughes, Andrew Annex, Christopher N Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. Geomesa: a distributed architecture for spatio-temporal fusion. In *SPIE Defense+ Security*, pages 94730F–94730F. International Society for Optics and Photonics, 2015.

[50] Kristina Chodorow. *MongoDB: the definitive guide*. "O'Reilly Media, Inc.", 2013.

[51] Jim Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, Programming, and Applications: Software for Humanity*, pages 217–218. ACM, 2012.

[52] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[53] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide*. "O'Reilly Media, Inc.", 2010.

[54] D Richard Hipp and D Kennedy. Sqlite, 2007.

[55] Ricardo García, Juan Pablo de Castro, Elena Verdú, María Jesús Verdú, and Luisa María Regueras. *Web Map Tile Services for Spatial Data Infrastructures: Management and Optimization*. InTech, 2012.

[56] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. The geojson format specification, 2008.

[57] Douglas Crockford. The application/json media type for javascript object notation (json). `https://tools.ietf.org/html/rfc4627`, 2006.

[58] Mike Bostock. TopoJSON. `https://github.com/mbostock/topojson/`, 2013.

[59] Dane Springmeyer, Konstantin Käfer, and Artem Pavlenko. Mapbox vector tile specification, 2014. URL `https://github.com/mapbox/vector-tile-spec`.

[60] Kenton Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*, 2008.

[61] Robert Marianski and Harish Krishna. Mapbox vector tile: Python package for encoding & decoding mapbox vector tiles, 2014. URL `https://github.com/mapzen/mapbox-vector-tile`.

[62] Tore Halset. Java vector tiles: A java encoder and decoder for vector tiles according to mapbox vector tile spec, 2014. URL `https://github.com/ElectronicChartCentre/java-vector-tile`.

[63] Mapbox. Mapbox studio: Vector tile driven map design, 2013. URL `https://www.mapbox.com/mapbox-studio/`.

[64] Eric Fischer. tippecanoe: Build vector tilesets from large collections of geojson features, 2014. URL `https://github.com/mapbox/tippecanoe`.

[65] Vladimir Agafonkin and John Firebaugh. vector-tile: Parses vector tiles with javascript, 2014. URL `https://github.com/mapbox/vector-tile-js`.

[66] Nicholas Hallahan. Leaflet.mapboxvectortile: A leaflet plugin that renders mapbox vector tiles on html5 canvas, 2014. URL `https://github.com/SpatialServer/Leaflet.MapboxVectorTile`.

[67] Claudio Tesoriero. *Getting Started with OrientDB*. Packt Publishing Ltd, 2013.

[68] Robey Pointer, N Kallen, E Ceaser, and J Kalucki. Introducing flockdb, 2010.

[69] Ching Avery. Giraph: Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 2011.

[70] Shay Banon. ElasticSearch. https://www.elastic.co/products/elasticsearch, 2015.

[71] Lars George. *HBase: the definitive guide*. "O'Reilly Media, Inc.", 2011.

[72] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44 (2):35–40, 2010.

[73] Jan Sipke Van der Veen, Bram Van der Waaij, and Robert J Meijer. Sensor data storage performance: Sql or nosql, physical or virtual. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 431–438. IEEE, 2012.

[74] InfluxDB - Open Source Time Series, Metrics, and Analytics Database. https://influxdb.com/, 2014.

[75] Vladimir Agafonkin. Leaflet: an open-source javascript library for mobile-friendly interactive maps, 2013. URL http://leafletjs.com/.

[76] Dave Leaver. Leaflet.markercluster: Marker clustering plugin for leaflet, 2012. URL https://github.com/Leaflet/Leaflet.markercluster.

[77] Brad Green and Shyam Seshadri. *AngularJS*. O'Reilly Media, Inc., 2013. ISBN 978-1-449-34485-6.

[78] Microsoft. TypeScript Language Specification. http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf, february 2015.

[79] Chris Marrin. Webgl specification. *Khronos WebGL Working Group*, 2011.

[80] Ricardo Cabello. Three.js, 2010. URL https://github.com/mrdoob/three.js.

[81] Github Three.js issue #4738: TRIANGLE_STRIP support back. https://github.com/mrdoob/three.js/issues/4738, 2014.

[82] Rama C Hoetzlein. Graphics performance in rich internet applications. *Computer Graphics and Applications, IEEE*, 32(5):98–104, 2012.

[83] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM, 1968.

[84] Stuart K Card, George G Robertson, and Jock D Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 181–186. ACM, 1991.

[85] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[86] Mark Harrower and Cynthia A Brewer. Colorbrewer. org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1): 27–37, 2003.

[87] Alexandru C Telea. *Data visualization: principles and practice*. CRC Press, 2014.

[88] Christian Thorne. Using a floating origin to improve fidelity and performance of large, distributed virtual worlds. In *Cyberworlds, 2005. International Conference on*, pages 8–pp. IEEE, 2005.

[89] Google. Chrome Developer Tools. https://developer.chrome.com/devtools, .

[90] Stephen Levine. dc.js - dimensional charting javascript library, 2012. URL `http://dc-js.github.io/dc.js/`.

[91] Inc Square. Crossfilter: Fast multidimensional filtering for coordinated views, 2013. URL `http://github.com/square/crossfilter`.

[92] Michael Bostock. Data-driven documents (d3.js), a visualization framework for internet browsers running javascript. 2012. URL `http://d3js.org/`.

[93] CartoDB. Dark Matter base map. `https://cartodb.com/basemaps/`, 2014. Data by OpenStreetMap.

[94] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor, D. Puiu, A. Sheth, V. Tsiatsis, and L. Vestergaard. Real time iot stream processing and large-scale data analytics for smart city applications, 2014. Poster presented at European Conference on Networks and Communications.

[95] Google. The Google Maps Directions API. `https://developers.google.com/maps/documentation/directions/intro`, . [Accessed September 29th, 2015].

[96] Edward Verbree, Siyka Zlatanova, and Kees Smit. Interactive navigation services through value-added cyclomedia panoramic images. In *Proceedings of the 6th international conference on Electronic commerce*, pages 591–595. ACM, 2004.

[97] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, (6):32–38, 2010.

[98] Christian Tominski, Petra Schulze-Wollgast, and Heidrun Schumann. 3d information visualization for time dependent data on maps. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, pages 175–181. IEEE, 2005.

[99] Waag Society. CitySDK. `http://dev.citysdk.waag.org/`, 2015.