Trajectory Analysis and Road Network Prediction on aSTEP

A Map Matching and Attribute Prediction Toolset

Project Report
SW605f20

Aalborg University
Computer Science

Computer Science
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Trajectory Analysis and Road Network Prediction on aSTEP

Theme:
Developing Complex Software Systems

Project Period:
Spring Semester 2020

Project Group:
SW605f20

Participant(s):
Abiram Mohanaraj
Christoffer Hansen
Fredrik De Frène
Laurits Brøcker
Simon Park Kærgaard

Supervisor(s):
Sean Bin Yang

Copies: 1

Page Numbers: 98

Date of Completion:
May 28, 2020

Abstract:

We develop a Map Matching service and an Attribute Prediction service for the collaborative multi-project, aSTEP. These two solutions are a part of the spatial and spatio-temporal components. To perform map matching, we investigate different techniques. We choose GraphHopper's Routing Engine, which employs a Hidden Markov Model. To adjust the performance of map matching, we have experimented with the Error-σ and transition probability-β. We end up with 43 as the Error-σ and 2 as the transition probability-β. Due to the lack ground truth data for the map matching task, our experiments are evaluated based on our expectation of a given trajectory.
For the Attribute Prediction service, we develop two micro-services. The Graph Attribute micro-service, which is responsible for interacting with end-users and fetching graphs from the database. The Attribute Prediction micro-service has the responsibility of predicting speed limit of any road network. We investigate different graph embedding and prediction algorithms and conclude that node2vec and relational fusion networks will be the main techniques. We utilise node2vec to embed the input graph and on the dual graph to get node and edge attributes. We also use the angle between the edges as our between-edge attributes. With these attributes, we train the network and model with a training loss at 115.68, validation loss at 155.38, and a test loss at 164.35. This model is inadequate for optimally predicting speed limits on road networks. In conclusion, we have developed working micro-services for map matching and missing attributes prediction on road networks. We have implemented the functionality for further training of models.

Preface

This report has been written in the Spring Semester of 2020 in a group bachelor project at the Aalborg University under the supervision of Sean Bin Yang. We thank Tobias Skovgaard Jepsen for his work with Relational Fusion Networks, and for his assistance, when we reached out to him.

Abiram Mohanaraj
<amohan17@student.aau.dk>

Christoffer Hansen
<ch17@student.aau.dk>

Laurits Brøcker
<lbrack17@student.aau.dk>

Fredrik De Frène
<fdefre16@student.aau.dk>

Simon Park Kærgaard
<skarga17@student.aau.dk>

Contents

Resumé

This report is the product of our group's participation in a collaborative project, namely aSTEP. The report will document our work on a map matching service, a graph attribute service, and an attribute prediction service, in addition to explaining our work process.

The project uses Sprints, a Scrum principle, to structure the development process. As a result, the report contains four Sprint chapters, where each sprint lasts approximately four weeks, during which the group work towards a set of specific goals.

Sprint one is dedicated to familiarising ourselves with the aSTEP platform, establishing good communication between the groups, and creating a map matching service for Task 1. We investigate GraphHopper and Hidden Markov Models, which is what GraphHopper uses to perform map matching. It also utilises OpenStreetMap by default, which is compatible with the aSTEP user interface. At the end of Sprint one, a working map matching service is implemented, but some work regarding the service's ability to communicate with users and other services remain.

Sprint two focuses on finishing the map matching service and preparing to start Task 2 regarding missing attributes in a road network. We implement a user interface for the map matching service, which shows the original and map matched GPS-points (trajectories) on a map. We also apply the communication channels between this service and the aSTEP framework, allowing other services to use the service in the way it has been designed by the routing super group. In preparation of our next task, we investigate graph embedding and prediction algorithms, where we find node2vec to be the optimal algorithm for graph embedding, and Relational Fusion Networks the most suitable prediction algorithm.

The fulcrum of Sprint three is to further investigate the algorithms chosen for attribute prediction, as well as implementing our solution for Task 2. Hereto, we choose to implement two services: The Graph Attribute service and the Attribute Prediction service. The Graph Attribute service acts as a front-end to the Attribute Prediction service, with its main task being to fetch a graph from the database based on the requesting user's input. The Attribute Prediction service embeds this graph using node2vec, performs a dual graph analysis, and calculates between-edge features, before predicting attributes using a trained Relational Fusion Network. We implement the services successfully, and additionally improve the map matching service's correctness by tuning GraphHopper's default Hidden Markov Model implementation. We also apply an improvement requested by another group, since they require a 1-to-1 correspondence between original GPS-points and map matched points for their service to function.

In Sprint four, we finalise the Attribute Prediction and Graph Attribute service. More specifically, we focus on implementing the Relational Fusion Network architecture to train a model, ensuring the data is correctly passing between the two services according to the specifications and making sure the visual part of the services is displaying the results correctly. Also, we test and evaluate to the extent possible in the limited time frame that is available.

In conclusion, we have implemented a fully functioning micro-service for map matching, which builds upon a Hidden Markov Model. Under our experiments, we find the standard deviation (Error-σ)

of 43 and a transition probability (β) of 2 to be optimal for the Hidden Markov Model. We have also implemented two working micro-service for the missing attribute prediction on road network. Performance-wise, our relational fusion network is underfit and therefore would require further training.

0 Introduction

With the vast amount of data available, the opportunity to make productive and helpful software is there for those who choose to seize it [1]. The processing of spatio-temporal data, meaning data that involves both time and location, is a relevant subject at the time of writing. To ease the development of software to process these kinds of data, Aalborg University has developed AAU's Spatio-Temporal Data Analytics Platform (aSTEP).

This semester project concerns the further development of aSTEP, a collaborative multi-project that began in 2016. The project focuses more on excellent communication between the groups involved and introduces the students to conventional work methods used in the industry. Developed and maintained by the aSTEP-2019 students, it is handed over to us to continue development. This is an interesting challenge, as we are required to quickly learn and adapt to this type of development, so that the project can be comfortably completed for us and all groups that depend on the work we do.

aSTEP

aSTEP is a platform for data-driven decision making and data analytics on spatial, temporal, and spatio-temporal data. It provides an environment for developers to deploy machine learning based services. aSTEP is a result of the collaborative effort of multiple 6th-semester Software Engineering groups, spanning several years.

aSTEP's development is a multi-project that includes roughly half of the current 6th-semester software students, distributed in standard AAU project groups. The aSTEP-2020 project is further divided into two sub-projects regarding the topics: *Time-series* and *Routing*. Each of these collaboration-projects have to be developed on the aSTEP platform. The two projects are based on sensory data and traffic data, respectively, provided by real-world collaborative partners. Furthermore, the project focuses on establishing and maintaining consistent documentation for implementing the project parts in each of the modules.

aSTEP-2020 consists of a total of 6 student groups. These groups are distributed into the two sub-projects routing and time-series, and thereafter referred to as the *routing super group* and the *time-series super group*.

Tasks of aSTEP-2020

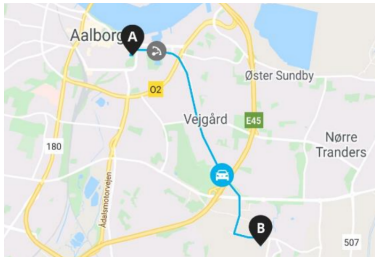
The Time-series sub-project focuses on two main tasks, based on user stories regarding bacteria activity in Aalborg's waste water treatment plant, and exoskeletons. The first task is to use historical bacterial data to predict future bacterial activity, also known as forecasting. The second task is focused on detecting outliers in a set of data that is significantly different from the mean.

The Routing super group's tasks are also based on a user story, which in this case involves Bring, a mail delivery company. Bring has to distribute hundreds of parcels all over the country. For this task, they need to find optimal routes for the parcels to be delivered.

The overall routing task is divided further into three tasks:

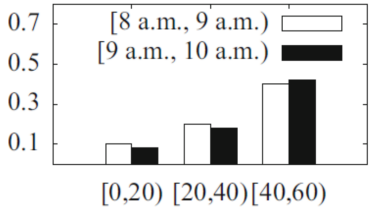
1. Analyse the trajectories provided by Bring, for each of their driver's routes. A trajectory is given by a sequence of GPS-points (t, x, y, v), describing the observed route the driver travelled.

   During the analysis, the given trajectory should be map matched, its speed should be analysed, and the speed data should be aggregated for each road segment. A guiding example for each case is depicted on **Figures 1a**, **1b**, and **1c**, respectively.



(a) A map matched trajectory, travelling from point A to point B.

(b) A speed analysis of a given trajectory.

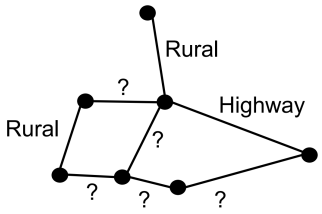


(c) The aggregated speed data for each road segment, at different intervals.

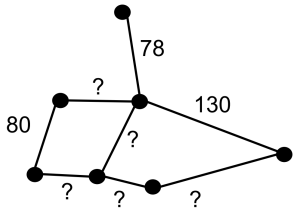
Figure 1: The trajectories should be analysed with the help of map matching, speed analysis, and aggregating the speed data for each road segment.

2. Road networks often consist of incomplete data, and this especially concerns the attributes on the individual road segments also referred to as edges, when modelling with graphs. Filling in the missing attributes on the edges is essential to allow "optimal" routing.

   The attributes can be descriptive characteristics, such as the edge's "type" or speed limit (**Figure 2a** and **Figure 2b**), which makes it a classification or a regression task, when trying to predict either of these.



(a) A road network with missing road type.



(b) A road network with missing speed limits.

Figure 2: The road network should have its attributes predicted, allowing for optimal routing.

3. Perform routing using real traffic information. This allows the drivers to deliver their parcels in a more time efficient and environmentally friendly manner.

   Routing that only considers distance can be inaccurate. Instead, the routing could be enabled by considering time dependencies in the real world traffic, such as peak vs. off peak hours.

The routing super group is expected to complete these three tasks during the semester. How this is accomplished is discovered and documented through the sprints and chapters of the reports. The different tasks are referred to as Task 1, Task 2, and Task 3 going forward.

We have, during the aSTEP kick-off meeting, chosen to take part in the routing sub-project and, more specifically, work with Task 2. However, as a measure of being introduced to the aSTEP platform, and a way of initialising and increasing the collaboration with other groups, it has been decided by our semester coordinator that all routing groups should, at least during Sprint one, focus on solving Task 1 together.

As all routing groups should collaborate on Task 1 during Sprint one, it was necessary to divide the task into multiple, smaller sub-tasks, which are listed here:

1. Matching of GPS-points from trajectories to road segments (Map matching).
2. Speed calculations.
3. Histograms.

Hereto, we were given the responsibility for solving sub-task 1, namely the map matching. We are expected to, given some input that identifies a trajectory, fetch its GPS-points from the aSTEP database and successfully match these to road segments on a map. Additionally, as mentioned earlier, we are also responsible for completing Task 2. This task involves using machine learning technologies to train a model to predict specific attributes for roads, based on existing information in a road network with missing attributes.

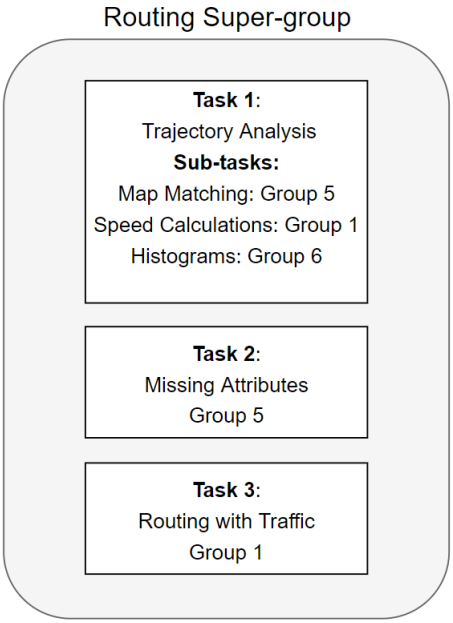


Figure 3: Distribution of tasks between the aSTEP-2020 routing super groups.

Figure 3 shows the distribution of sub-projects and tasks amongst the aSTEP-2020 groups. In this figure, we are referred to as group 5.

Committee Meetings

A committee meeting is held weekly, to structure the entire aSTEP project and keep the level of transparency as high as possible. Hereto one or two representatives from each group are chosen to participate and act as a facilitator of information to the rest of the group.

The purpose of the committee meetings are to update the remaining groups, on the work being carried out by each group. To this end, other groups can help groups that are facing obstacles.

Project Time Schedule

The allocated time frame of the project is divided into *4 Sprints*. The initial kick-off meeting presented the following plan as guidance:

- **Sprint 1 (12/02 - 02/03)**: Get started. What parts of the aSTEP platform can be reused? Make an agreement on design for UI and documentation.
- **Sprint 2 (03/03 - 30/03)**: Design and implementation.
- **Sprint 3 (31/03 - 04/05)**: Design and implementation.
- **Sprint 4 (05/05 - 22/05)**: Test and evaluation.

At the very beginning of Sprint one, Thomas Højriis Knudsen, a former developer and maintainer of aSTEP-2019, held a presentation on the existing architecture and infrastructure of the aSTEP platform. This presentation helped us to identify our sprint goals faster.

Project Management

The goal of this section is to present the defining elements of the group-work-process, and what tools and methods we have utilised to manage the project. Below is a list of the scrum concepts that are involved in the project and sprints.[2]

Product and Sprint Backlog The product backlog is an ordered list of items that describes all tasks to be done during product development, which in this case, spans the entire semester. These tasks may be detailed or abstract, where the more important tasks are first on the list. Before each sprint, a subset of the product backlog is selected and made into smaller and more concrete tasks. These tasks are the focus of that specific sprint and are called the sprint backlog. For this project specifically, we utilise sprint goals in a different way than stated in the Scrum Guide. Instead of a singular goal for each sprint, our sprint goals are more abstract goals that are further refined into a sprint backlog for each sprint.

Sprint Review Sprint review is a section in every sprint chapter. This section concerns itself with reviewing how many and to what level the tasks in the sprint backlog are completed. Incomplete tasks are discussed in further detail regarding the reason it was not completed, and the task is then re-evaluated and potentially added to the sprint backlog of the next sprint.

Sprint Retrospective Our use of the sprint retrospective varies slightly from what is described in the Scrum guide. Instead of focusing solely on what went right during the sprint, we instead focus on both the positives and negatives. We feel this allows us to gain a deeper understanding of what we should continue with during subsequent sprints, or what we can improve on.

Daily Scrum To ensure transparency and to increase the overall understanding of the work that is currently being done at any point in the group, we utilise daily scrum meetings (stand-up meetings). This meeting is held every weekday where each group member briefs the group on what they have worked on the day before, and what they plan to do in the next 24 hours.

Product Backlog

Table 2 contains the project's product backlog. The backlog items are identified through an initial analysis of the work required, but we will also document new tasks that are discovered. We refine and solve these items in terms of sprint goals throughout the sprints.

Index	Description
P1	Implement a map matching micro-service to solve Task 1 on the aSTEP platform.
P2	Implement a graph attribute prediction micro-service to solve Task 2 on the aSTEP platform.
P3	Ensure forward compatibility for both developed micro-services.
P4	Perform a risk analysis and determine risk management.

Table 2: The initial product backlog containing a list of tasks which need to be fulfilled in order for this project to be considered completed.

1 Sprint One

We investigate the aSTEP-2019 platform developed by the accumulative effort of previous students. We establish our sprint goals and thereto a sprint backlog. Following this, a risk management plan is created for the entire project. To this end, we develop a micro-service intended to solve issues regarding Map Matching (MM).

1.1 Sprint Goals

The goals of the first sprint is to familiarise ourselves with the existing aSTEP platform created by aSTEP-2019. On top of interacting with and understanding the underlying architectural structure, we as the routing super group, have to fulfil the task of analysing trajectories presented in **Section 0**.

This can be summarised into the following sprint goals:

Goals
• Understand the structure of the aSTEP project. • Assess a risk management plan. • Understand the underlying theory of map matching. • Implement a MM micro-service for aSTEP that is compatible with Task 1.

Table 1.1: Goals for Sprint one.

1.1.1 Sprint Backlog

Based on the sprint goals, we define more elaborate requirements of our tasks during Sprint one in the sprint backlog. These requirements will also reflect the constraints we face during Sprint one. Some of the constraints are derived from meetings held with group 1 and group 6, as part of the super group meetings.

The requirements are prioritised in the order they are presented, with the most important at the top. They are categorised according to the MoSCoW rules.[3, p. 140] The categories are the following:

- (M) The *Must have*
- (S) The *Should have*
- (C) The *Could have*
- (W) The *Want to have but Won't be this time around*

The highest priority items (M) must be implemented for a system to function. Without these requirements completed the solution cannot be deemed fulfilled. The requirements for sprint one are outlined in **Table 1.2**.

Index	MoSCoW	Requirements
S1.1	(**M**)	Understand and document the server architecture of aSTEP.
S1.2	(**M**)	Understand and document the preconditions for developing micro-services on aSTEP.
S1.3	(**M**)	Map match GPS-points from existing trajectories to a road network.
S1.4	(**M**)	Implement a representation of a road network based on an OpenStreetMap (OSM) file of Denmark.
S1.5	(**M**)	Implement the MM micro-service following the service interface guidelines.
S1.6	(**S**)	Explore existing MM in aSTEP.
S1.7	(**S**)	Explore open source MM frameworks.
S1.8	(**S**)	Assess a risk analysis and create a risk management plan for the project.
S1.9	(**C**)	The system could have a standalone user interface displaying results of MM.

Table 1.2: Requirements for MM micro-service of Task 1.

1.2 Structure of the aSTEP Platform

The understanding of the current structure of the aSTEP platform obtained during Sprint one is encapsulated in the following section. We explore how each part is connected and what they consist of. This analysis is requisite for future development.

1.2.1 Project Architecture

The aSTEP project consists of several micro-services. Each has its own separate codebase that a group is in charge of. This leads to a low coupling of the modules in the project, which in turn allows for more teams to work on the same project in parallel. The low coupling of the modules also makes it easier to add new functionality, as the previous implementations are not entirely dependent of each other.

Low coupling does come at the cost of increased complexity. Anyone that wants to run the project would have to start a great number of micro-services on their system. The compilation of the micro-services would take a great deal of time and any update to one of the services would require the

entire toolchain to restart. Tools like Docker and Kubernetes alleviates most of the complexity by performing the necessary compilations, so that the system becomes manageable, with the benefits of low coupling.

1.2.2 Server Architecture

The current server architecture configuration is a product of aSTEP-2018 and aSTEP-2019, and consists of a Daisy-git server, an aSTEP-run01 server, a Kubernetes cluster, and two databases: DB1 and DB2. The latest server architecture inherited from aSTEP-2019 is shown in **Figure 1.1**.

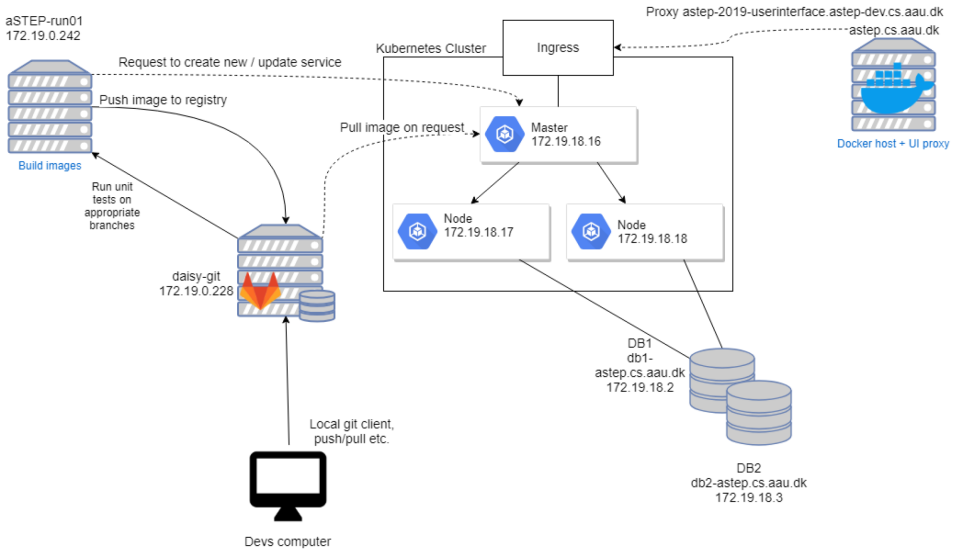


Figure 1.1: Current server architecture of aSTEP.[4]

As the principle of containerisation is deeply rooted in this architecture, we find it natural to give a small introduction to Docker and Kubernetes, before we explain the overall server architecture, as they serve as a tool for containerisation and a tool for orchestration of the containers, respectively.

Docker

Docker allows for development of applications in different isolated environments and can deploy them all on the same host. Docker achieves this by packing each program's code and dependencies into their own container. This is the process known as containerisation.[5]

Docker containers act similar to virtual machines but instead of creating an entire virtual Operating System (OS) for each container, Docker creates a virtual environment for each container to run on the same OS kernel.[5]

Deployment of a container must include two elements: A *Dockerfile* and a *Docker image*. The Docker file contains the instructions from which Docker will automatically build the Docker image of the program. Each instruction will create a layer in the image. This means that if one decides to change parts in the Docker file, the image only need to replace the corresponding layers. From an image, one or multiple containers can be created and deployed.[5]

Kubernetes

Kubernetes is an orchestration tool for managing Docker containers, as well as other container tools. It is used for the automatic deployment of the aSTEP project and its related micro-services. This is achieved through the Docker file in each project-related git repository, as Kubernetes uses it to generate a Docker container[1], which then runs on a Kubernetes node. This automates the deployment process for each micro-service, so that any commit to the master branch of a given service, will cause Kubernetes to start the process of deploying those changes to the service in production.

To sum up and give an overview of the overall server architecture, each component is described below:

- The *Daisy-git server* hosts the repository for the codebase of the different services, including the Docker file. GitLab has built-in integration with Kubernetes.

  As an early initiative from aSTEP-2018, pushes to the master branch needs to be handled by a merge request before it can be deployed into production. This is a safety precaution.[4][6]

- The *aSTEP-run01 running server* follows the principle of containerisation, which includes the use of Docker and Kubernetes. Every service runs inside its own Docker container, thus making it possible to run multiple services simultaneously, on the aSTEP-run01 server. In short, its purpose is to build the Docker images of the services and make create/update requests to the Kubernetes cluster.[4]

- The *Kubernetes cluster* is the production server that runs the production of the different services, which includes keeping all services' Docker images updated. As depicted in **Figure 1.1** the cluster consists of a master node with two slave nodes that connect to DB1 and DB2. Ingress is an API running in the Kubernetes cluster that redirects to the correct service within the cluster when attempting to accessing it.[6]

- The databases *DB1* and *DB2* are used to store necessary information for all services. DB1 contains a road network representation of Denmark extracted from an OSM of Denmark. DB2 contains trajectory data from Bring to be utilised for the Tasks within the routing super group.

1.2.3 aSTEP User Interface

aSTEP has a singular UI system, which means that each service is a dynamically rendered page on the same UI. The UI of aSTEP-2019 consists of a three-pane layout with a navigation bar, an input area,

[1]Kubernetes adds the container to a pod, which is then executed. More than one container can also be in a pod.

and a chart area, all highlighted with different colours in **Figure 1.2**. All services currently running on aSTEP share this layout.

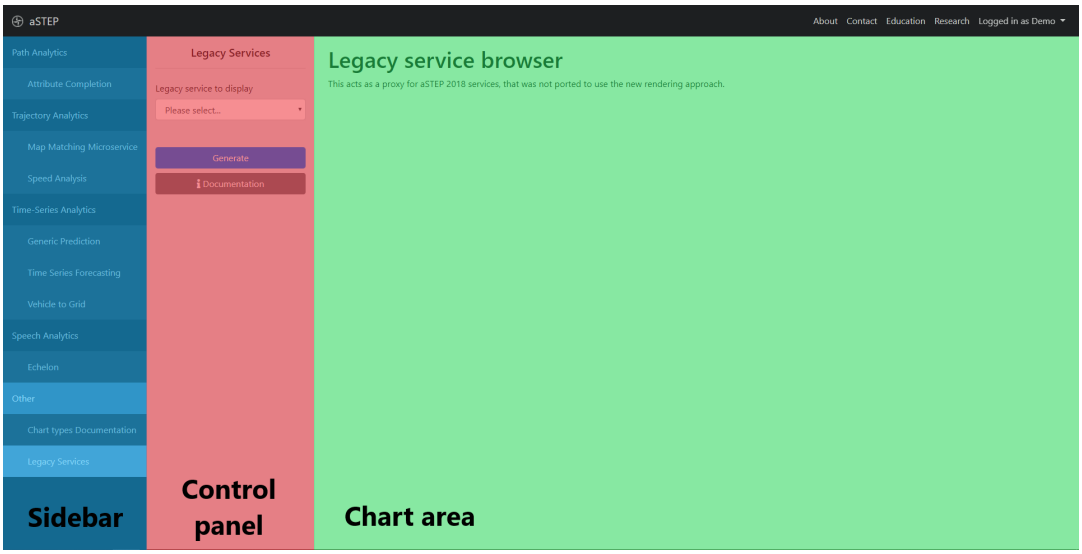


Figure 1.2: The three-pane layout of the aSTEP website highlighted with different colours.

- *Sidebar:* this area functions as a sorted menu of all available services running on aSTEP. The user can select a specific service to access it's UI.[7]
- *Control panel:* the area from which the user interacts with the service to give input for the chart area. Each service has its own control panel. This area allows for various field types i.e. input field (text or numeric), file input, and other selection elements. Mandatory for this area is the 'Generate'-button, 'Documentation'-button, and the user/developer mode toggle button. These buttons, respectively, take the input from the user to generate data for the chart area, show the readme in the chart area, and switch between user and developer input fields[2].[7]
- *Chart:* this area's content varies from service to service, as they use different charts depending on the service's output. Among the currently available chart types supported by the aSTEP interface are text, markdown, geographical clustering, geographical geometry, chart js, and composite.[8]

The process and more detailed requirements for implementing this UI for a service follows in the next section.

1.2.4 aSTEP Service Interfaces

Functionality on the aSTEP platform is implemented in terms of services/tools. Previous aSTEP teams have developed a common interface for the implementation of services.

[2]Developer fields allows the developers to test specific input to a service.

aSTEP-2019 proposed a list of Request For Comments (RFC) to the old aSTEP-2018 interface, where the development of each micro-service reused many of the same chart-types and other HTML elements. These changes increased the system's extendability and modularity, by separating the implementation of each micro-service from the development of the front-end by proposing a standardised set of endpoints.[9]

A shared design of the UI was also attained. The UI was also implemented as a Single Page Application (SPA), which means the UI was changed from being server side to client side. With this change it is now much easier to test the UI during development.[10]

In short, the changes mean that a service need to satisfy these three requirements:

- They must listen on port 5000 for data traffic.
- They must follow the rules of a specified interface version.
- And finally, they must have CORS enabled.

The latest interface version from aSTEP-2019 is version *y19w20*. This version makes the following endpoints mandatory for each service:

- POST */render* is used to render a chart of a certain pre-defined type in the view area of the UI.
- GET */info* specifies the name, category, and which interface version the service is using for communication with other services.
- GET */fields* is used to generate input fields in the *form-area* of the UI.
- GET */readme* specifies the content shown when the documentation button in the form-area of the UI is pressed.[9]

The service is expected to give output in *application/json* format to each endpoint, as it is the MIME[3] media type for JSON text[12]. It should also accept input in the form of *application/x-www-form-urlencoded* and *multipart/form-data*, as these are the standard HTML encodings for POST requests[13]. A service should accept both, as the former is the default type of encoding and is generally more efficient, while the latter allows for file uploading and has increased efficiency encoding binary data.[14][15]

1.2.5 Service Interface Changes

During Sprint one, a handful of new RFCs were proposed to the interface architecture, which resulted in a new interface version called *2020v1*. These changes were motivated by the lack of data communication and interconnectivity between services.

[3] *Multipurpose Internet Mail Extensions[11].*

The idea of letting some services act as preprocessor tools of data for other services was suggested, which meant changes to the */info* endpoint, that is now, no longer a mandatory endpoint. It can now be left out if there is no need for the service to show up in the UI, thus making it a preprocessor tool with no direct interaction from the UI.

As a response to the lack of data communication between services, two new endpoints were introduced together with a few additions to the existing ones. The new endpoints are:

- POST */data*
- POST */combined*

The */info* endpoint now has to specify the name and type of both the input and output data for the service. Together with this change, the new */data* endpoint now has the option to export the raw data instead of only displaying it. */combined* is an endpoint to serve both the */render* and */data* outputs in one. This should lessen the load on the servers as computations will only be done once contrary to before[14]. These changes and additions to the existing three-pane UI layout is aimed to achieve a notebook-like interface.[16]

Other small changes were also made to the mandatory buttons in the UI control panel. In hand with the changes to the */data* endpoint, a "Print Raw Results"-button has been added to show display the raw data being returned from the endpoint.

1.3 Risk Management

As an early part of the project we aim to identify risks for the development process in order to create a mitigation plan, a way to monitor the development of certain risks, and a contingency plan. Together this is called an RMMM plan.

1.3.1 Risk Identification & Analysis

We present the possible risks for this project and analyse them in terms of *probability* of the risk's occurrence, the impact (*loss*), *exposure*, and *consequence*. Some of the risks are identified with the inspiration of Boehm's list of risk items.[17] We are mainly looking to identify risks that affect the project's schedule, resources, quality, and performance in a negative way.[18, p. 644-649]

Probability ($P(r)$) is calculated and given a score from 0 to 1 based on the likelihood that the risk occurs. Loss ($L(r)$) is scored from 1 to 10 based on how severe the risk is for the product.[19, p. 753] The risk exposure ($E(r)$) is then calculated using the following equation:

$$E(r) = P(r) * L(r)$$

We have gathered all the identified risks in **Table 1.3** with a short description of the risk, exposure, and consequence.

	Risk	P(r)	L(r)	E(r)	Consequences
Project scope					
1	Project scope not defined precisely	0.2	8	1.6	Redoing work
1.1	• Changes due to new information	0.2	7	1.4	Redoing work
1.2	• Changes due to unexpected problems	0.2	6	1.2	Added work
Requirements					
2	Continuous changes of requirements	0.5	6.5	3.25	Redoing work
2.1	• Changes due to new information	0.5	4	2	Added work
2.2	• Changes due to unexpected problems	0.5	4	2	Added work
Personnel					
3	Personnel illness	0.5	2.5	1.25	Slowed production
4	Personnel loss	0.05	9	0.45	Higher workload for rest
5	Personnel not collaborating	0.2	4	0.8	Wasted time/Less quality
6	Personnel underperforms	0.2	3.5	0.7	Less quality
Database					
7	Database interaction/access problems	0.2	7	1.4	Wasted time
8	Database data-loss	0.05	9	0.45	Slowed/Halted production
Learning					
9	Learning aSTEP infrastructure(*)	0.1	7	0.7	Increased production time
10	Learning new ML techniques(*)	0.1	7	0.7	Increased production time
11	Learning new programming languages(*)	0.15	2.5	0.375	Increased production time
Equipment					
12	Slow training speed of agents(**)	0.2	1	0.2	Slowed production time
Group Collaboration					
13	Waiting for other group's work	0.5	3	1.5	Wasted time
14	Coordination with other groups	0.8	3	2.4	Slowed production time
External					
15	COVID-19	0.9	5	4.5	Pandemic precautions

Table 1.3: A risk analysis where the probability P(r) of the event occurring, a loss L(r) related to this probability, were it to happen and lastly the exposure E(r) calculated based on these values.
(*) Indicates that we do not reach the learning goals for a certain risk.
(**) Indicates the risk that the hardware performing the agent training is sub-par.

1.3.2 Risk Mitigation, Monitoring, and Management Plan

We further analyse and prepare to handle risks if they occur by developing an RMMM plan. As supervising an RMMM plan for all the identified risks will be too big of a task, we instead create an exposure threshold. This is to ensure the risks with the most impact are in focus. We set this threshold to 20% of the risks with highest exposure. Therefore, we cut the risks down to 4 and sort them by exposure, as shown in **Table 1.4**. For each risk, we refer to the same risk-number as in **Table 1.3**. In the RMMM plan, we define precautions and actions to mitigate, monitor, and manage the prioritised risks, provided that they happen, to reduce the consequences or resolve the risk if possible.[18, p. 650-652]

Risk	E(r)	Mitigation	Monitor	Management
15	4.5	Follow official guidelines	COVID-19 situation	Take necessary precautions
2	3.25	Ensure correct requirements	Monitor project progress	Update requirements
14	2.4	Ensure good communication	Monitor groups	Coordinate with groups
1	1.6	Ensure that scope is correct	Monitor project scope	Expand scope

Table 1.4: Risk, Mitigation, Monitoring, Management (RMMM) plan.

We have identified the COVID-19 situation to have the highest exposure for the project. To monitor this risk we must follow AAU's COVID-19 guidelines and the overall dissemination of the COVID-19 virus. To mitigate and manage the risk we must follow the official guidelines, but also ensure that we can continue the development of the project as a group.

For the other high exposure risks, we employ agile methods that ensure continuous refinement of requirements and scope with the Scrum framework. To this end, we have weekly meetings with the rest of aSTEP-2020 to monitor the other groups' projects and thereby the overall progress of the aSTEP-2020 project.

1.4 Map Matching

To become acquainted with our part in solving Task 1, we will investigate how map matching works. Map matching, also called position snapping, denotes procedures that assign geographical data on a digital map. The typical geographical data are points using the Global Positioning System (GPS).

1.4.1 Map Matching Types

We categorise map matching into two types. These are referred to as *global* or *offline* map matching and *incremental* or *online* map matching. Offline map matching computes the position snapped GPS-points for an entire trajectory at once, and can therefore only be performed when all GPS-points of the trajectory are known ahead of time. Online map matching is localising the input and employing strategies different from offline to create a solution. These are often considered inferior or incomplete compared to the output of offline map matching and are used when only real-time data is available.[20]

The data provided by Bring makes offline map matching ideal for our solution to Task 1.

1.4.2 Existing aSTEP Map Matching

Time and resources allocated for Task 1 are limited, which make the software integration and configuration process model a natural candidate for our development process. This method, as seen in **Figure 1.3**, looks at the specifications of the project and determines if it is possible to reuse existing solutions or if discovering new configurable software is necessary. If neither is possible, the requirements may have to be refined in order to find a satisfying solution.[18, p. 52-54]

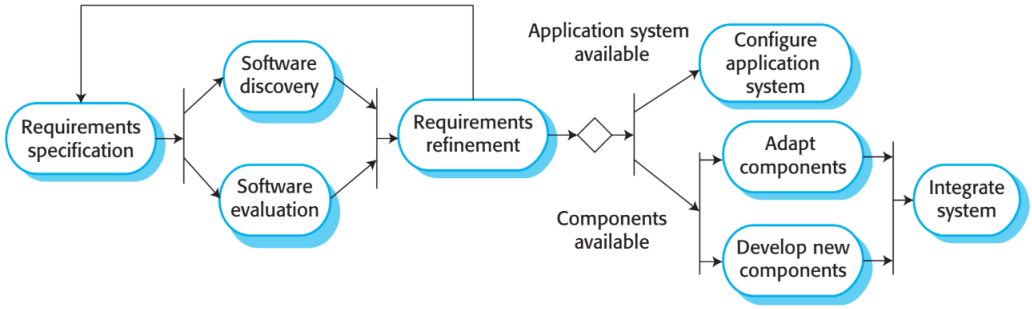


Figure 1.3: The integration and configuration software process model[18, p. 52]

The aSTEP platform already have an existing map matching micro-service that was implemented by aSTEP-2018. However, the project code does not exist on the Daisy-git server, but exists on a proxy server, as shown in **Figure 1.1**, thus making us unable to access and examine it.

Therefore, we have to explore alternative solutions. When looking for software to fit our solution, we look for software that has sophisticated map matching techniques and is compatible with the data supplied by Bring.

Map Matching Implementations

Among existing open source map matching technologies that fit our requirements, we have found OsmApi, GraphHopper Routing Engine, MapBox, and OSMnx. To find the best candidate, we have evaluated and listed them in **Table 1.5**. The API services have constraints in regard to the amount of POST requests that can be performed in a short amount of time, unless a premium service is paid for. On the other hand, GraphHopper has an open source offline map matching tool. The tool is still under development but its functionality is adequate and will fit our solution well. We, therefore, deem it to be the best candidate for a solution.

Name	Map Matching Type	OSM compatible	API
OsmApi	Offline/Online	Yes	Yes
GraphHopper	Offline/Online	Yes	No/Yes
MapBox	Offline/Online	Yes	Yes
OSMnx	Offline/Online	Yes	Yes

Table 1.5: Different map matching API's/Frameworks

1.4.3 GraphHopper

GraphHopper is an open source routing engine, used for route planning and route optimisation. It is Java based and utilises OSM by default, which is compatible with the existing technologies used in aSTEP. GraphHopper uses a Hidden Markov Model (HMM), outlined in [21], with the help of the Viterbi Algorithm (VA), to perform its map matching.[22][23]

Hidden Markov Model

Provided that we need to reconfigure or tweak the parameters of the GraphHopper tool, we need to understand the underlying theory of an HMM. We explore this concept and illustrate how an HMM performs map matching.

When using an HMM, we are concerned with predicting random variables or states that are not directly observable and are therefore called *hidden states*. When a hidden state changes, an *observation* is emitted regarding the new state. This observation is optional for the algorithm to acknowledge, and are produced relative to their *emission probabilities*. Between the hidden states, there are *transition probabilities* that describe the likelihood of transitioning between states.[24]

This concept can be applied to a map matching scenario by substituting observations for GPS-points from the data set, denoted z_t, and hidden states for road segments, denoted r_i. This can be seen in **Figure 1.4**, where each hidden state emits an observation.

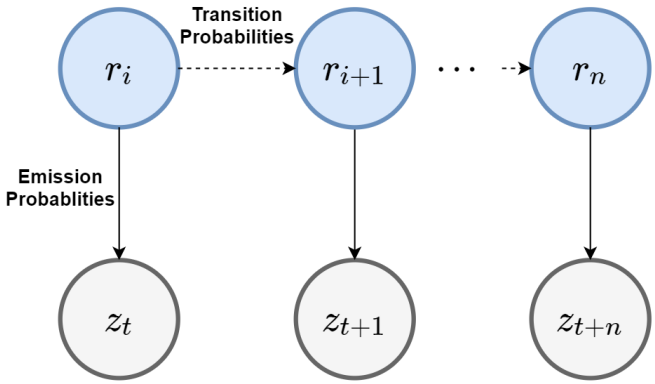


Figure 1.4: A simple hidden markov model with hidden states and their corresponding observations.

We now present the emission probability. Given a location measurement z_t and a road segment r_i we can use the conditional probability:

$$p(z_t|r_i) = \frac{1}{\sqrt{2\pi}} e^{-0.5\left(\frac{\|z_t - x_{t,i}\|_{greatcircle}}{\sigma_z}\right)^2} \tag{1}$$

where $x_{t,i}$ is the closest road segment by a measure of the *great circle distance*.[25] The intuition behind

the emission probability is that the road segments further away are less likely to be the road that the driver is travelling on. σ_z denotes the standard deviation of the GPS-point measurement. To this end, the nearby candidates for map matching are computed within a set radius specified by GraphHopper.[23]

For the transition probabilities, the car should intuitively move between the most plausible road segments, avoiding any unnecessary manoeuvring. To do this, another distance measure is introduced, namely the *route distance*. The distances should be closest to the great circle distance possible, and is given by:

$$d_t = \left| \|z_t - z_{t+1}\|_{greatcircle} - \|x_{t,i} - x_{t,j}\|_{route} \right| \tag{2}$$

This is used for the transition probability formulated as:

$$p(d_t) = \frac{1}{\beta} e^{\frac{-d_t}{\beta}} \tag{3}$$

In this formula, β is the route and great circle distance difference estimated. This is when the VA is used to determine the most likely sequence of candidates.[22][21]

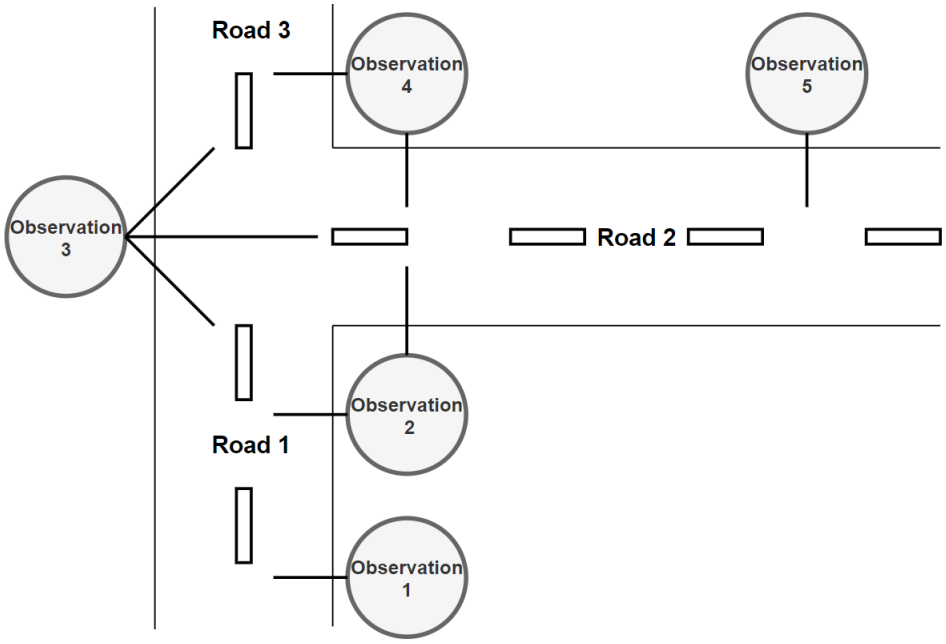


Figure 1.5: An example with GPS-points as observation on a road network.

In **Figure 1.5**, a three-way intersection exhibits a scenario where a driver makes a right turn to *Road-2* coming from *Road 1*. Problems arise when the GPS-point location of the driver are imprecise. It can often happen that recorded GPS-points deviates from the road to different sides with uncertainty of direction. As a consequence, at *Observation 3* it is difficult to determine whether the driver goes straight from *Road 1* to *Road 3* or turns to the right onto *Road 2* from the GPS-points.

We convert the example from **Figure 1.5** into an HMM representation to solve the aforementioned problem, depicted in **Figure 1.6**. The dotted lines between the hidden states are the transition probabilities where the VA will determine the most likely sequence of states. This is the basis of how GraphHopper implements its map matching technology that we use.

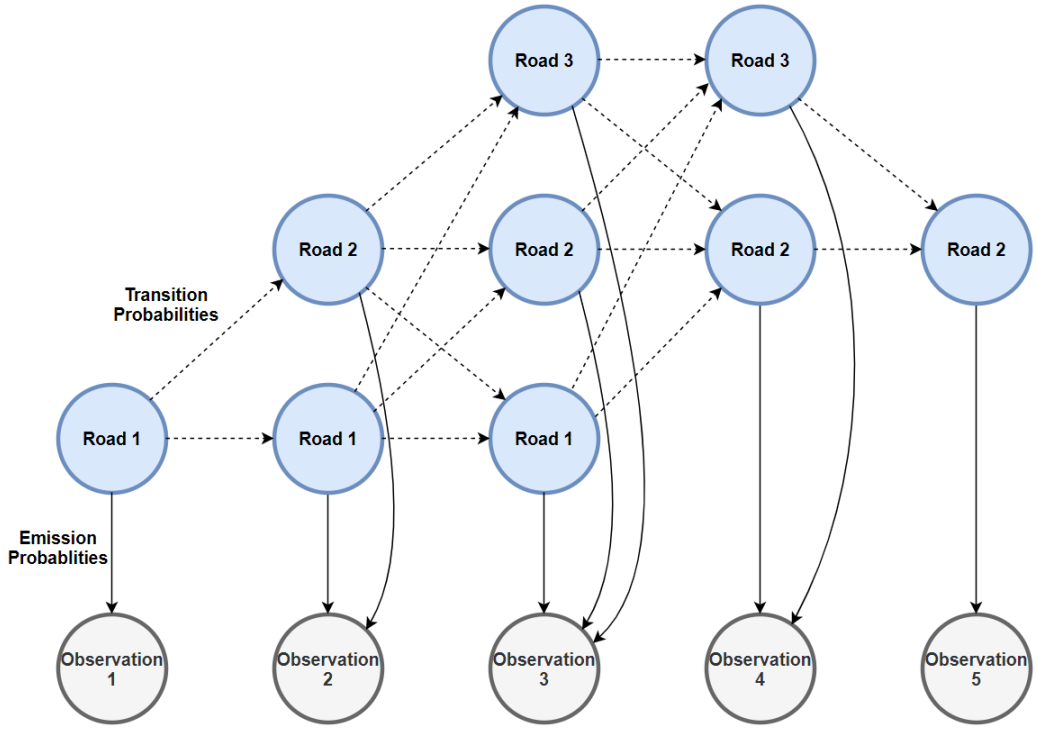


Figure 1.6: We convert **Figure 1.5** to an HMM representation.

1.5 Designing the Map Matching Service

After having explored the majority of the aSTEP architecture and infrastructure in **Section 1.2** together with a suitable tool for implementing a map matching algorithm, the next step is to design and implement our MM service for the aSTEP platform.

1.5.1 Component Design of the Map Matching Service

The MM service is expected to be a stand-alone service. There are two sides to its integration into the system. First, it should allow users to perform map matching on a selected trajectory. Secondly, it should allow for communication internally on the platform with other services (i.e. provide another service upon request with snapped GPS-points from a map matched trajectory).

Figure 1.7 illustrates how the MM service communicates with the UI and DB2. It also shows the

order of execution when a user uses the service. An overview of the data and the parameters that are required for each communication channel is shown at each arrow.

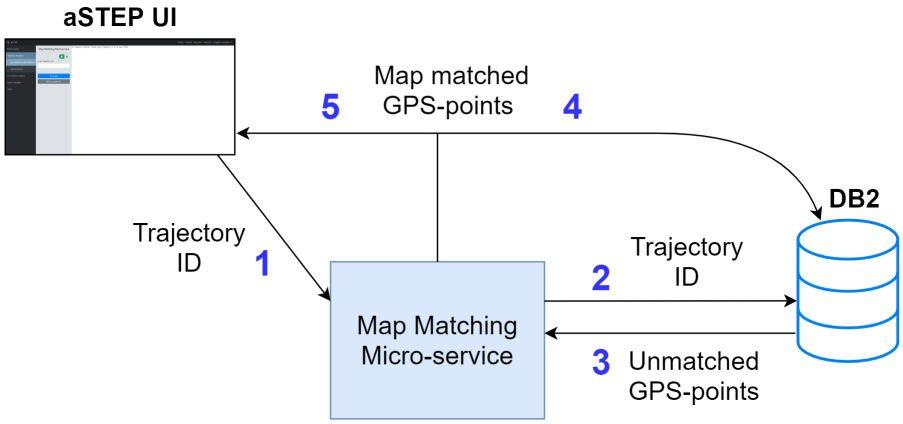


Figure 1.7: Design plan of the user interaction with the MM micro-service, Sprint one.

1. When the user interacts with the aSTEP UI on the website, they need to provide the micro-service with an ID of an existing trajectory kept within DB2.
2. With the provided trajectory ID, the micro-service queries the database for all GPS-points from the corresponding trajectory.
3. From the query, DB2 returns a list of unmatched GPS-points for the service to map match.
4. After map matching the queried GPS-points, the service inserts the snapped GPS-points with their associated edge IDs in a new table in DB2. These GPS-points will be used later when communicating with other micro-services.
5. Simultaneously, it should also show the map matched trajectory in the chart-area of the aSTEP UI.

The internal communication between the MM micro-service and other services on aSTEP for Task 1 is depicted in **Figure 1.8**. Similar to the previous figure, we specify the communication channels and our expected output.

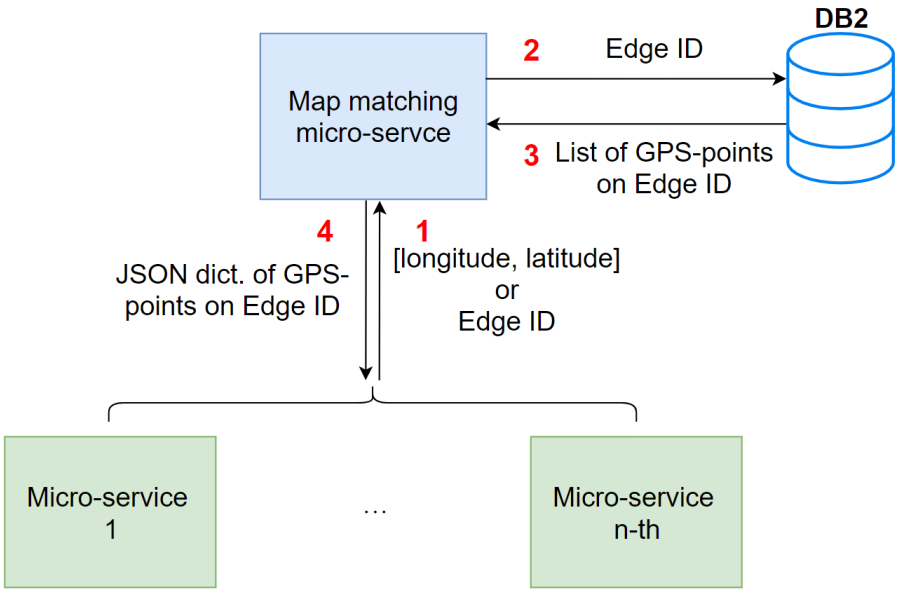


Figure 1.8: Design plan of the internal communication between the MM micro-service and other micro-services.

We refer to it as internal communication as it lets other services request information about, which GPS-points lies on a specific edge on the road network.

1. Another service will provide a set of coordinates ([longitude, latitude]) or the specific edge ID to the MM micro-service. If provided with [longitude, latitude], the MM service will find the closest edge and get the corresponding edge ID, before proceeding to step 2. If provided with an existing edge ID, the MM service will proceed to step 2 immediately.
2. With the newly found edge ID (or provided edge ID), the service will query DB2 to get the corresponding GPS-points.
3. DB2 returns a list of all GPS-points that have been matched to the specific edge.
4. The MM service returns the GPS-points in JSON format to the requesting service.

1.5.2 Processing of Data

The data in DB2 must be processed before it can be returned as map matched GPS-points, or a list of points for the closest edge, to the UI or other micro-services, respectively (**Figure 1.7**). This is done differently for the two cases, as they require different data formatting.

For the UI, a trajectory ID is used to query the database, and all corresponding trajectory points are deserialised into Kotlin objects. These objects are then fed through the map matching algorithm, which produces the map matched GPS-points used by the UI.

For the other micro-services, the coordinate received is fed to the map matching algorithm to extract the closest edge to that point. Thereafter, the database is queried on that edge's ID, and all points are deserialised into Kotlin objects. The objects are then serialised to JSON format, as seen in **Code Snippet 1.1**, and returned to the micro-service that requested it.

```
"points"
    [
        {
            "id": 1226309,
            "lon": 9.62924005026026,
            "lat": 55.66553184921759,
            "edge_id": 886947
        }
    ]
```

Code Snippet 1.1: Format of a single JSON object returned to other micro-services.

1.6 Collaboration

We include this section to outline the collaboration with the other groups during Sprint one. This sprint focused on the internal cooperation within the routing super group. The collaboration focused on planning Task 1, which included the agreement upon the design of the interaction between the micro-services developed by the routing groups. We have through multiple meetings, refined the common solution for Task 1, as we have gained a better understanding of the aSTEP project.

We came up with the following plan for the organisation of Task 1's micro-services and related implementations, for future development, on **Figure 1.9**.

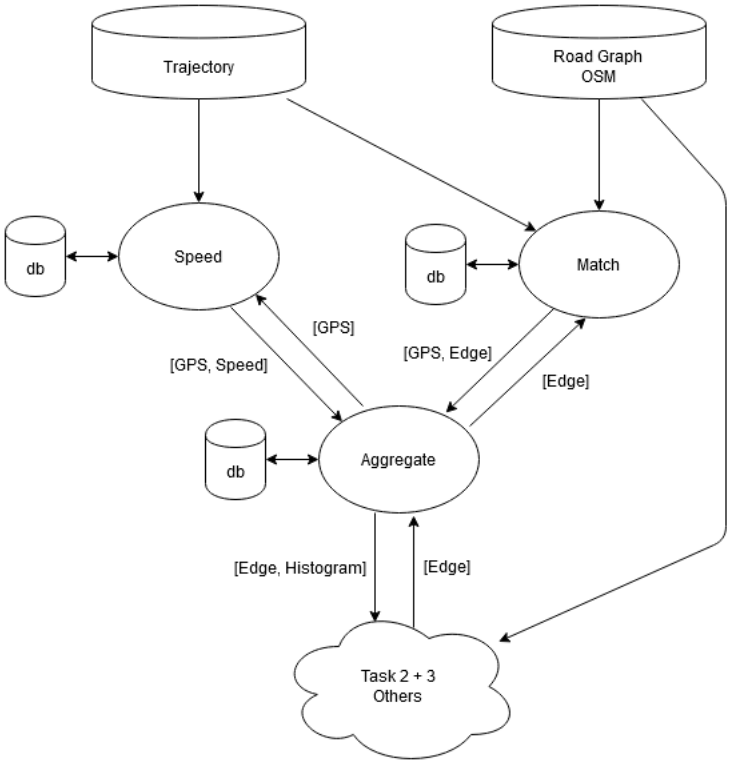


Figure 1.9: Task 1 system planning. The MM service interaction with the aggregated part of the system that creates histograms.

The plan is for group 1 (referred to as *Speed* in the diagram) to select a trajectory from the trajectory database and calculate the speed. We (referred to as *Match*) use the same trajectory and perform map matching on it. To perform map matching, we also use the map from the *Road Graph OSM* database. Lastly, group 6 (referred to as *Aggregate)* can request information from either group to create the histograms that will be used for later tasks.

1.7 Sprint One Review

In the initial part of Sprint one, we established four sprint goals (**Table 1.1**), which have been refined and put in the form of a sprint backlog in **Table 1.2**. We evaluate the sprint backlog in **Table 1.6**.

Backlog Index	Completed	Status
S1.1	✓	An understanding of the aSTEP server architecture has been attained and documented by the initial presentation of the aSTEP-2019 project in conjunction with reading and understanding the RFCs and code of relevant existing services.
S1.2	✓	The preconditions for micro-services in aSTEP has been identified by reading the RFCs and has been documented.
S1.3	✓	The functionality of map matching a trajectory from the database to roads has been implemented.
S1.4	✓	GraphHopper makes an optimised representation of the road network for routing tasks.
S1.5	O	The endpoints for a MM micro-service has not been implemented, but an understanding of how endpoints are implemented is attained.
S1.6	✓	An evaluation of the existing MM service in aSTEP is made, and the conclusion is that it is inadequate, as the source code is not present in GitLab.
S1.7	✓	An analysis of existing MM frameworks is made, and GraphHopper is chosen.
S1.8	✓	A risk management plan was created using the RMMM model to determine the notable problems that should be kept under investigation as the project develops.
S1.9	O	A graphical user interface has not been developed. Note that this has the priority *Could have*.

Table 1.6: Review of the sprint backlog for Sprint one. The completed items are denoted by ✓, while the incomplete ones are marked with O. Partially completed items will be denoted by ⊗.

The majority of the Sprint one goals have been completed. However, some of the goals have not been implemented to the necessary extent and have thereby been deemed incomplete.

Communication between groups when designing the infrastructure of the services that make up Task 1 went well. Relative to this, we have taken on the task of adding a new road network to DB2 to replace the "Road Graph OSM" database on **Figure 1.9**. This will be needed for our, as well as the other groups', individual tasks after the completion of Task 1.

In conclusion, the unfinished sprint backlog items will be implemented in the next sprint, as they are necessary for the completion of Task 1. Furthermore, the task of adding a road network to DB2 will also be a focal point in the next sprint.

Sprint Retrospective

During the retrospective meeting, we deemed the tasks completed during Sprint one adequate. Adjusting to the new environment, in which we are expected to develop our service, took longer than expected. However, creating a robust understanding of said environment was prioritised to achieve a better final result. Although we have implemented the map matching algorithm, we deem **Requirement S1.5** unfulfilled, as the requirement strictly states it must be implemented following the service interface guidelines. With the aforementioned understanding of the development environment, the remaining work on this task during Sprint two and future development of the new service in Task 2 should be easier.

2 | Sprint Two

In correspondence with the goals of Sprint two, we have two focus points in this chapter; documenting the essential parts of the MM service's final implementation and preparing for Task 2. Hereto, we investigate graph embedding strategies and algorithms to find the best-suited candidate for a solution.

2.1 Sprint Goal

To finalise the MM service, we must create a UI for the MM service and connect it with our MM algorithm. On top of this, we must also expose the */data* endpoint to allow other services to fetch the data required when creating histograms from our service.

In regards to Task 2, we need to create a representation of the road network that is compatible with the output of GraphHopper. We also want to research different techniques that allow for the classification of edges and what this requires in terms of preprocessing.

Goals
• Complete the MM service. • Create a common road network to be used by groups for future tasks. • Understand graph embedding techniques. • Understand classification techniques for edge prediction.

Table 2.1: Sprint goals for Sprint two.

2.1.1 Sprint Backlog

Once again we take each sprint goal and specify them into more elaborate requirements in our sprint backlog below.

Index	MoSCoW	Requirements
S2.1	(**M**)	Implement endpoints for the MM micro-service to communicate with the aSTEP platform and other micro-services.
S2.2	(**M**)	Establish connections to the DB2 to fetch the Bring data and to store map matched trajectories.
S2.3	(**M**)	Achieve an understanding of the road network utilised and created by GraphHopper.
S2.4	(**M**)	Store the road network generated by GraphHopper in the database.
S2.5	(**S**)	Analyse graph embedding techniques and its relevance for Task 2.
S2.6	(**S**)	Analyse predictive algorithms and their relevance for Task 2.

Table 2.2: Sprint backlog for Sprint two.

2.2 Finalising the Implementation of the Map Matching Service

With the fundamental map matching functionality in place, we can now direct our focus to implementing it on the aSTEP platform, which was not completed in the previous sprint. We provide an overview of the data returned from the endpoints, and the resulting UI. Moreover, we exhibit a basic example of what a user-interaction with our service might look like. Lastly, we look into the implementation of communication with external components like the database and other services.

2.2.1 User Interface and Interaction

The implementation of the MM service follows the *2020v1* version of the service interface. As the service will be an interactive stand-alone service on the aSTEP website, we first need to expose the */info* endpoint.

/info Endpoint

From our */info* endpoint, we return the JSON dictionary shown in **Code Snippet 2.1**. It specifies the input and output channels for the service, which is a essential part of the *2020v1* service interface. These denote the "type" of the expected input and output data to the service. The input channels also create the input fields in the control panel on the left side of the UI, from which the user can interact with the service. It is, therefore, not necessary to define any additional input fields through the */fields* endpoint.

```
{
    "version" : "2020v1",
    "id" : "mapmatching",
    "name" : "Map Matching",
    "category" : 1,
    "input" : [
          {
              "name" : "trajectoryID",
              "label" : "Trajectory ID",
              "type" : "integer"
          },
          {
              "name" : "coordinate",
              "label" : "Coordinate",
              "type" : "coordinate"
          },
          {
              "name" : "edgeID",
              "label" : "Edge ID",
              "type" : "integer"
          }
     ],
     "output" : [
          {
              "name" to "gpsids",
              "label" to "A list of GPS ID's on the
                          edge/coordinate/trajectory",
              "type" to "points"
          }
     ]
}
```

Code Snippet 2.1: JSON dictionary as input for */info*.

Other than that, the control panel also shows the *Visualise Results*, *Print Raw Results*, and *Documentation* button by default. These are the new buttons specified in the *2020v1* service interface. *Print Raw Results* simply prints the output of the */data* endpoint. In order to use the *Documentation* button, we expose the */readme* endpoint. This endpoint returns a JSON dictionary that specifies a markdown-chart and a markdown text file containing the text to be shown. All the controls in their entirety are shown in **Figure 2.1**.

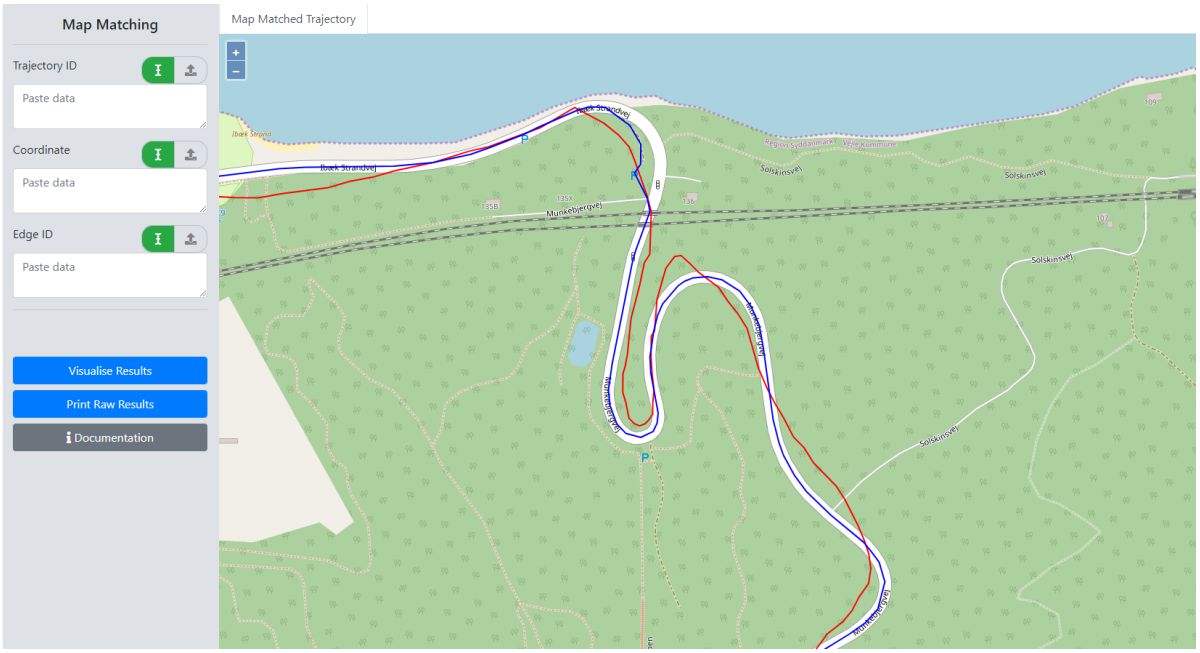


Figure 2.1: The MM service's UI on the aSTEP website showing the original trajectory with red and the map matched trajectory with blue.

/render Endpoint

To provide the user with visual feedback when giving input to the service and then pressing the *Visualise Results* button, we expose the */render* endpoint. Our service gives the */render* endpoint different charts depending on the input provided by the user. It is possible for the user to request results of multiple trajectories, coordinates, or edges at the same time, resulting in multiple charts. These charts are encapsulated into a *composite* chart. This means the user can switch between each chart by selecting the tabs at the top of the chart-area.

For the chart visible on the figure above, we return **Code Snippet 2.2** that specifies a *map-geo* chart containing both the map matched and original trajectory. This is to let the user compare them against each other. Here, red is the original trajectory and blue is the map matched trajectory.

```
{
    "chart_type": "map-geo",
    "content": {
        "view": {
            "center": {
                    "lat": "lat_value",
                    "lon": "lon_value"
                },
            "zoom": 13
            },
         "featureData": {
            "type": "FeatureCollection",
            "features": [
                {
                    MapMatchedTrajectory
                },
                {
                    OriginalTrajectory
                }
            ]
        }
    }
}
```

Code Snippet 2.2: JSON dictionary as input for */render*. *MapMatchedTrajectory* and *OriginalTrajectory* covers the specific information needed to define the geometry type to be shown on the map, as well as the input GPS-points defining each trajectory.

/data Endpoint

To fulfil the design plan of the internal communication with other micro-services, as presented in **Section 1.5.1**, we have implemented the */data* endpoint according to the parameters specified in that section. In practice, this means the */data* endpoint functions as a dynamic dispatcher that accepts either a single or a list of *trajectoryIDs*, *coordinates*, or *edgeIDs*. These are the input channels defined in **Code Snippet 2.1** as well. If provided with a *trajectoryID*, it returns all GPS-points that make up the map matched trajectory. In the case it is supplied a *coordinate*, it returns all GPS-points on the closest edge. If the input is an *edgeID*, it returns all GPS-points that lie on the specific edge. In all three cases, the format of the returned data can be seen in **Code Snippet 2.3**. If it is supplied with incorrect input data, the endpoint returns an error message.

```
{
    "points": [
        {
            "id": "GPS-point id",
            "lon": "lon_value",
            "lat": "lat_value",
            "edge_id": "edgeID"
        },
        ...
    ]
}
```

Code Snippet 2.3: JSON dictionary as output for */data*.

Interaction with a User

We present two examples to demonstrate the interaction with the MM micro-service from another perspective. We show the connection between the MM micro-service and the user, corresponding to the initial design plan **Figures 1.7** and **1.8** back in **Section 1.5.1**.

We exhibit an example on **Figure 2.2**. It illustrates the interaction between the MM service and the aSTEP UI.

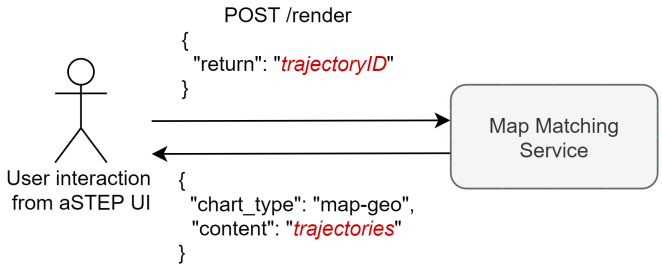


Figure 2.2: Example of user interaction with the MM service from the aSTEP website.

When a user interacts with the MM micro-service from the aSTEP website by entering a *trajectoryID* into the input field, it will be received by the */render* endpoint. The MM micro-service will then, as specified in **Code Snippet 2.2**, return a *map-geo* chart showing the corresponding trajectories to the UI, so that it can be displayed to the user, like on **Figure 2.1**. This interaction is identical when providing the service with an *edgeID* or a *coordinate*.

Interaction with Other Services

Figure 2.3 shows the flow of communication between the MM service and another aSTEP service. Unlike when a user interacts with the MM service, a service would instead return its results through the */data* endpoint. Right now, only the Bucketizing Tool is communicating with our service in this manner.

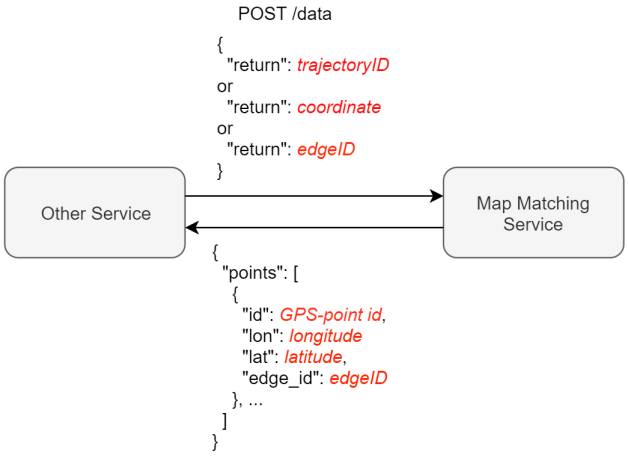


Figure 2.3: Example of another service, like the Bucketizing Tool, interacting with the MM service.

Here we see how the specific data is being returned to the requesting service, when it sends either a *trajectoryID*, *coordinate*, or an *edgeID* through the */data* endpoint, as explained earlier.

2.2.2 Storing the Map Matching Results

Another part to this entire process is to store the results of the map matching. When a user sends a request to the MM service with a trajectoryID, the resulting GPS-points are stored in a new table in DB2 called *mapmatchedpoints*. This table has an id column, a map matched *(lon, lat)* pair, and an edge id. The edge id refers to the road segment that the given point has been matched to. The id column acts as both a primary key and a reference to the *gpspoints*[1] table's primary key.

The other groups are interested in which road segment (*edge_id*) each GPS-point is associated with.

[1]This table contains the original unmatched GPS-points.

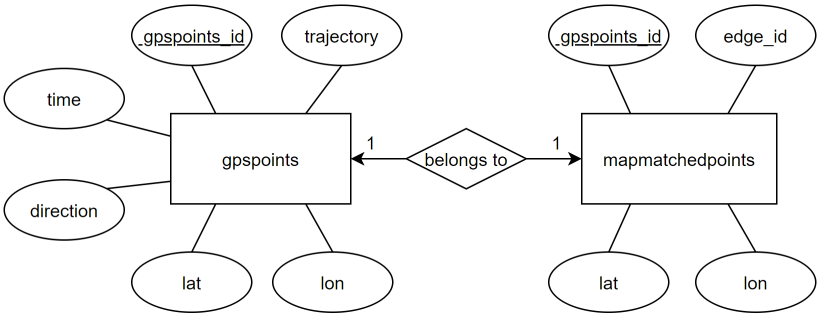


Figure 2.4: Entity Relation (ER) diagram describng the relation between table *gpspoints* and the new *mapmatchedpoints* table in DB2.

2.3 Graph Storage in Database

In order to coherently refer to the same data across all aSTEP routing services, working with an identical road network is essential. Therefore, it is necessary to store this network in the database.

As mentioned in Sprint one, a road network representation of Denmark is already available in DB1. By inspection of the data contained in the database, we deem it unfit in use for future services, as it does not have notion of "edges". Also, it seems to be incompatible with our implementation of GraphHopper.

The currently used map data for the MM micro-service is contained in a *denmark-latest.osm.pbf* file, which is the latest version of the OSM of Denmark (downloaded 29-02-2020).[26]

OSM files contain three different types of data, namely *node*, *way*, and *relation*. *Node* specifies a certain GPS-point on the map. *Way* is a collection of nodes and *relation* is a collection of ways. From these it is possible to create a road network. Using different tools, the road network representation can be altered depending on what task it is being used for.

We have constructed the road network with the use of GraphHopper. GraphHopper creates its own graph representation with two data structures: *Edge* and *Node*. An *Edge* is defined by two nodes (base node and adjacent node), along with other labels (road type and speed limit), which specifies the features of the edge. Furthermore, an edge contains a list of GPS-points, including the two nodes, specifying its geometry.

The road network will consist of a set of edges and a set of nodes. An edge will be a connection from the base node to the adjacent node. This is the road network we will store in the new database, which is a task we volunteered for amidst the routing super group. **Figure 2.5** specifies how we organise it.

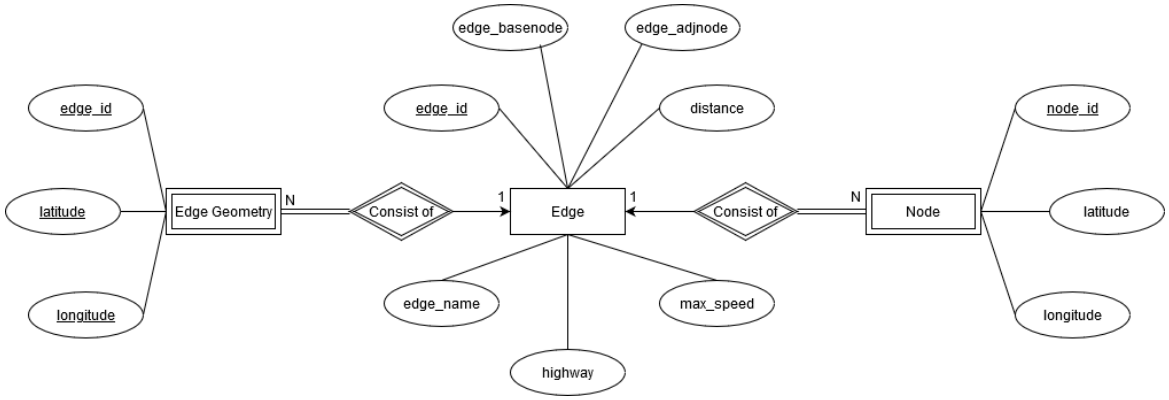


Figure 2.5: ER diagram for the road graph. The highway attribute of the Edge entity specifies the road type.

The GraphHopper data is incomplete[2] with regards to the edge features. Features, like speed limit, concerns only 321,561 out of the total 1,282,785 edges extracted from the file, resulting in a graph with a large number of featureless edges. This reaffirms the application of Task 2, which we will develop next.

The new *mapdata* database will contain three tables: One for the nodes, one for the edges, and lastly one for the geometry or structure of the edges. An entry in the *Edge* table will contain the following data:

- A unique id (*edge_id*)
- A source node id (*edge_basenode*)
- A target node id (*edge_adjnode*)
- The length of the edge (*distance*)
- Speed limit (*max_speed*)
- Road type (*highway*)
- Road Name (*edge_name*)

An entry in the *Node* table is defined as a unique id for the node, a latitude, and a longitude. Entries in the *Node* table can only exist if a corresponding base node id or an adjacent node id exists in the *Edge* table. For each edge in the *Edge* table, a number of entries will exist in the *Edge_Geometry* table. These entries will contain a latitude and a longitude along with an id for specifying which edge the coordinate is part of.

[2]The OSM data is also incomplete

2.4 Graph Embedding and Prediction Algorithms

With this section, we begin the preparation of Task 2 in the project, that is the task of filling in the missing attributes on a road network. To perform this task, there is a selection of graph embedding techniques and prediction algorithms we wish to investigate.

2.4.1 Graph Embeddings

Graph embedding is the act of converting a graph to a low-dimensional vector space while preserving all properties of the graph. This is important as graph analytics is resource-demanding, and only a subset of mathematics and machine learning is usable for this representation. Having a vector representation allows for faster and simpler calculations, but also a bigger toolset of applicable mathematics. For example, using cosine or euclidean distance for similarity is straight forward with vectors.[27][28]

DeepWalk DeepWalk is an online graph embedding algorithm suitable for multi-label graph classification tasks. The DeepWalk algorithm performs a random walk using each node in the graph as the root and, thereafter, updates the representations with an update procedure. The output of DeepWalk is a low dimensional vector space of the features for each node. The authors of DeepWalk claim the algorithm can "create meaningful representations for graphs too large to run spectral methods on", albeit with reduced accuracy.[29]

Node2vec Node2vec is a feature learning algorithmic framework for arbitrary networks. It uses the structure of a graph to extract features regarding the nodes which then can be used in conjunction with a machine learning algorithm to predict certain information about the edges. Node2vec learns a mapping of nodes to a low-dimensional vector space of features that preserves the locality of the nodes. Node2vec implements a flexible notion of a node's neighbourhood by using a biased random walk procedure, which explores various neighbourhoods of the node.[30]

Graph2vec Graph2vec is a neural embedding framework, created to learn a data-driven distributed representation of arbitrarily sized graphs. Graph2vec differentiates itself from the previously mentioned approaches by embedding the entire graph into a single vector, instead of embedding each node. It builds on the principles of doc2vec, as the entire graph and its sub-graphs can be seen as a document made up of sentences. The use cases for this algorithm is mostly graph-classification and clustering. [31]

2.4.2 Prediction Algorithms

Support Vector Machines Support Vector Machines (SVMs) are models used for classification or regression. The basic idea of SVMs is to find a hyperplane in a vector space which classifies the training data with an error of at most ϵ (a predefined deviation). This hyperplane can afterwards be used as a model to classify data. SVM is not suitable with overlapping target classes and for large datasets.[32]

Neural Networks Neural Networks (NN) are designed to be pattern recognisers, or function approximaters, and have a wide variety of applications ranging from image recognition to speech recognition. In theory they can approximate any function given enough training data while having a finite number of neurons in their hidden layered structure.[33, p. 308-310][34]

***k*-Means** The *k-means* algorithm is an unsupervised approach for learning features of within a dataset. It is a hard clustering algorithm, which means it divides data into k amount of clusters, defined by the learned features. To use the algorithm we have to specify k amount of classes for it to divide the data into.[33, p. 499]

Graph Convolutional Networks Graph Convolutional Networks (GCNs) is a neural network approach for extracting features from graphs while using parameterised filters. The back propagated parameters are usually shared over all areas in the graph, hence why it's convolutional. GCN's learn a graph representation much like a Multilayer Perceptron (MLP), by using forward-propagation, meaning the *k*-th layer's input is the *k-1*-th layer's output. The difference in the feature propagation is that GCN's stimulate similar predictions by taking the hidden representations of each node and averaging it with the neighbouring nodes at the start of each layer. Thereby, locally connected nodes in the graph will have smoother representations.[35][36]

Relational Fusion Network Relational Fusion Networks (RFN) is an extension to GCNs, explicitly designed with sparse road networks in mind. It is proposed in Jepsen et al.[37], and outperforms GCNs in both regression and classification tasks, with regards to road segments. The reason for this is GCN's feature propagation described above. GCNs use the average representations of neighbouring nodes, which in this case would be road segments. However, road networks contain attributes which can not be represented in nodes. In addition, road networks have a lower density, in comparison to state of the art GCN tasks such as biological or social networks, meaning there are fewer neighbouring road segments.

2.4.3 Evaluation

The graph embedding techniques explored in this section offer distinct outputs. This is important to consider in terms of how we want to represent our graph for the chosen prediction algorithm.

Node2vec and DeepWalk create similar outputs, which have the same dimensions of a $d \times n$ vector[3]. Therefore, we have to evaluate them based on their performance presented in their papers, and as seen on **Figure 2.6**, the node2vec graph embedding outperforms DeepWalk on three datasets.

Algorithm	**Dataset**		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

Figure 2.6: Node2vec compared with other graph embedding techniques on three different datasets.[30]

The output of graph2vec being a 1-dimensional vector for the entire graph is a concern, as it does not represent features for each node or edge within the graph. It is commonly used for task where a comparison of graphs is necessary. Based on this, we deem that node2vec is the superior technique for the task.

For the prediction algorithms, we investigated both supervised and unsupervised approaches. As we have the access to a sizeable amount of labelled features within the *mapdata* database for training, we believe a supervised approach is achievable. For deciding on the specific algorithm, we base our choice of what makes the most sense in the context of the task. Here, RFN's which specialises themselves for road networks is the obvious choice. It also has compatibility with node2vec, as it can be used for both the input graph but also the dual graph used in the algorithm.

With these considerations in mind, node2vec in conjunction with a RFN classifier is deemed most relevant for solving Task 2 in Sprint three.

2.5 Collaboration

Collaboration within the routing super group during Sprint two mainly focused on the internal communication between the services for Task 1. As our service is in charge of fetching data from the database and distribute it to other services, we needed to make sure the implementation follows the initial design plan from Sprint one. This included the expected data from both the receiving and sending side of each service. This is documented in **Section 2.2**.

A missed detail regarding the output of our service is that we need to ensure a 1-to-1 correspondence between the original GPS-points and the map matched GPS-points from our service. This is an important requirement from the group performing speed calculations, as they need all GPS-points to get more precise results.

Furthermore, meetings have been held with group 1 regarding the road network. During these meetings, the structure of the road network in the database was communicated to the other group. The demands from group 1 regarding the road network was also made known.

[3] d represents the number of dimensions in the feature representation, and n is the amount of nodes in the graph.

2.6 Sprint Two Review

The entirety of the backlog for Sprint two has been completed including the incomplete backlog items from Sprint one. A full overview is shown in the table below.

Backlog Index	Completed	Status
S2.1	✓	Implementation of the mandatory endpoints for the MM micro-service has been documented in **Section 2.2**. These follows the service interface version *2020v1*. Our service is now deployed in the Kubernetes cluster and functioning on the aSTEP website.
S2.2	✓	We managed to establish connection to the DB2 database. Through this connection, we run the queries to fetch the GPS-data from Bring and afterwards to store the map matched GPS-points and edge IDs.
S2.3	✓	An understanding of the way GraphHopper represent and use a road network has been documented and used to extract the necessary information from an .osm file of Denmark to convert into a road network.
S2.4	✓	A road network generated by GraphHopper has been stored in DB2 that stores information of edges and their associated nodes. This is all stored in three new tables.
S2.5	✓	We investigated DeepWalk, node2vec, and graph2vec as techniques to embed the graph. We determined that embedding the graph and converting it to a low-dimensional vector space is important when performing predictive tasks. Hereto, node2vec was chosen for our application.
S2.6	✓	We investigated Neural Networks, k-means, Support-Vector Machines, and Graph Convolutional Networks for predicting the edges of our road network. To this end, Relational Fusion Network was deemed relevant for Task 2.

Table 2.5: Review of the sprint backlog for Sprint two. The completed items are denoted by ✓, while the incomplete ones are marked with O. Partially completed items will be denoted by ⊗.

A consideration regarding the overall completeness of the MM micro-service is concerned with the actual results we achieve. Many of the map matched trajectories show undesirable behaviour. An example is shown in **Figure 2.7**, where the map matched trajectory deviates from the expected route. Thus, we deem it necessary to improve the GraphHopper algorithm to achieve better results. This is a part of Sprint three.

Figure 2.7: Example showing the map matched trajectory (blue) deviating onto a pathway besides the road.

Sprint Retrospective

We were early in the sprint faced with the challenge of the COVID-19 situation; a risk that was identified in our risk analysis. Thus, firstly we were unable to maintain the same level of communication we had internally in the group and with other project groups, and secondly the level of productivity. Therefore, as an addition to the RMMM plan for the COVID-19 risk, we arrange daily online group meetings during the lockdown as an effort to preserve some level of communication and work sharing.

3 Sprint Three

We fill firstly document the improvements to the MM service. Thereafter, we shift our primary focus towards completing Task 2. Hereto, we dive into the more detailed theory of node2vec and the specialised GCN: RFN. On the basis of the aforementioned investigation and an analysis of the existing Attribute Completion aSTEP service, we also propose a design plan for our new Attribute Prediction (AP) service. Furthermore, we document the implementation accomplished on this service during this sprint. The topics are all defined in the sprint goals and further refined in the backlog.

3.1 Sprint Goals

We present the sprint goals, which we focus on during Sprint 3 in **Table 3.2** below.

Goals
• Improve the map matching algorithm used in the MM service. • Achieve a deeper understanding of the chosen attribute prediction techniques and algorithms. • Implement an attribute prediction service for aSTEP.

Table 3.1: Goals for Sprint Three.

3.1.1 Sprint Backlog

We introduce the sprint backlog for Sprint 3. We must first extend the MM service implementation and afterwards investigate node2vec and RFN in order to begin the implementation of the new AP service.

Index	MoSCoW	Requirements
S3.1	(**M**)	Ensure a 1-to-1 correspondence between the original GPS-points and the map matched GPS-points.
S3.2	(**M**)	Implement endpoints for the AP service to communicate with the aSTEP platform.
S3.3	(**M**)	With node2vec, embed a graph with the speed limit as its weights, and use this to get node representations.
S3.4	(**M**)	With the graph and the feature matrices, train an RFN.
S3.5	(**M**)	Predict the speed limit on the unlabelled edges of a graph.
S3.6	(**S**)	Through an investigation, document the theory of node2vec, and RFN.
S3.7	(**S**)	Make a design plan for both the attribute prediction service's UI and architecture.
S3.8	(**C**)	Improve map matching by adjusting the standard deviation (error-σ) and transition probability (β).

Table 3.2: Backlog for Sprint three.

3.2 Improving Map Matching

During the second sprint, we discovered that GraphHopper, with the current implementation of the HMM, did not achieve satisfactory results. We, therefore, address the issue with a few assumptions to improve its performance.

The root of the problem is how GraphHopper searches for potential candidate states, referring to the HMM. GraphHopper uses the designated *Error-*σ (Standard deviation) and multiplies it by 2 for its range searches. This means that with a lower σ value, the algorithm chooses its closest road segments. We observed that GraphHopper would select GPS-points from all side roads, and even the opposite traffic lane, of the actual road in the map matched trajectory. Two examples showcasing these issues are depicted in **Figure 3.1** and **Figure 3.2**. Again, the original trajectory is shown in red and the map matched in blue.

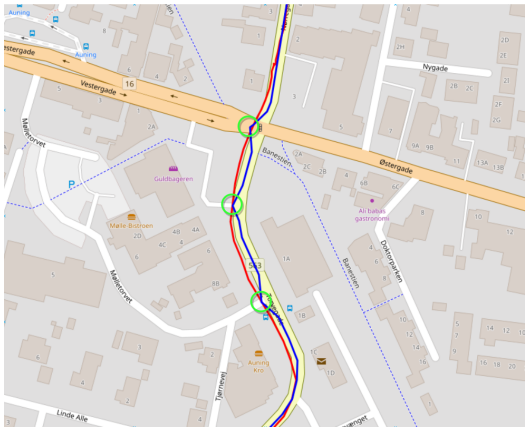


Figure 3.1: GPS-points being snapped onto all side roads that the trajectory passes, with Error-σ = 0.01 and β = 2.

Figure 3.2: GPS-points being snapped to the traffic lane going in the opposite direction of the trajectory, with Error-σ = 0.01 and β = 2.

The results shown on the figures are unsatisfactory, as these points are not a part of our expected trajectory. The reason for this is the low range search caused by the Error-σ. When we increase the Error-σ, the algorithm matches far fewer points, as the Error-σ is also directly tied to the selection of candidate nodes. A trajectory with 600 GPS-points could potentially be reduced to 60 GPS-points by increasing the Error-σ. While these few matched GPS-points would be snapped correctly, this result is not useful for the other groups to work with, as we would no longer have 1-to-1 correspondence between the original GPS-points and the map matched GPS-points. This 1-to-1 correspondence is necessary for speed calculations performed by Group 1.

Despite this, the improved version uses a higher σ-value, as we rely on the correctness of the newly matched points. On **Figure 3.3**, we demonstrate an example of what a map matched trajectory looks like with a high σ-value. Here, we have the scenario of very few points being matched. Given the original red trajectory, we can deduce that GraphHopper only map matches the four distinct points along with the entire blue map matched trajectory.

With the assumption, that the car has only followed the road segments between the map matched GPS-points, we can, to combat this problem, infer the geometry of the edges, that have been map matched, through the road graph itself. We then match all the points on the original red trajectory to the closest points on the actual edge from the *mapdata* database, by the great circle distance. The results of this approach is shown on **Figure 3.4**. Thereby, we avoid matching points onto wrong roads that that is closest by a distance measure. This way we solve the 1-to-1 correspondence problem while achieving more correct map matching.

We also tried adjusting the transition probability-β, this, however, did not to have an impact on the path GraphHopper chose, unless an unnaturally high value was selected, which resulted in worse map matching. Therefore we use the default transition probability β of 2.

Figure 3.3: Very few GPS-points get matched to a road with Error-σ = 43 and β = 2.

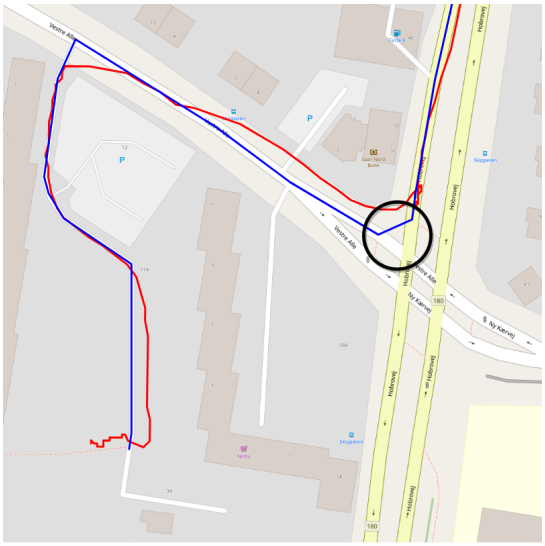


Figure 3.4: Snapping all GPS-points to be on the road with Error-σ = 43 and β = 2.

However, this solution is not perfect, as seen on **Figure 3.4**. The blue trajectory cuts the corner of the intersection, highlighted by the black circle. This problem, however, is out of our hands, as it's caused by a lack of GPS-points in the road graph itself. It could also be the case that the driver's GPS failed to record the location in the very moment it went into the intersection, resulting in no GPS-points in the original trajectory that were close enough to be snapped to the GPS-point inside the intersection.

Nonetheless, the 1-to-1 correspondence and higher accuracy allows other groups to create better histograms, that can be used for the future tasks of routing and classification. Furthermore, these issues are to be addressed by GraphHopper by separating the filtering of candidates and the Error-σ.

3.3 Node2vec Theory

In **Section 2.4**, we explored graph embedding techniques, and we determined that node2vec is a suitable technique to embed a graph's nodes into a low dimensional vector. Here, we present the underlying theory behind the method outlined by Grover et al.[30].

Given a graph $G = (V, E)$, node2vec learns a feature representation for each node $n \in V$ as a mapping function $f : V \mapsto \mathbb{R}^d$ where d represents the parameter corresponding to the number of dimensions in the feature representation. The matrix of f is, therefore, a matrix of size $|V| \times d$.

Node2vec builds on the fundamentals of the skip-gram model created for word2vec, where the model associates words that are within the same context window. With text, the notion of order and neighbourhood of words is trivial, as they are processed linearly, in the form of sentences. This notion of neighbourhoods does not exist in graphs in the same intuitive manner, and therefore we need a way

to model these sentences or so called neighbourhoods within the graph. These neighbourhoods are denoted as $N_S(u)$ for each node $u \in V$ with the search strategy S[30], which we will firstly uncover below before we can return to the functionality of the skip-gram model.

3.3.1 Random Walks and Search Bias α

Next, the random walk strategy of node2vec is introduced to sample the surrounding neighbourhood from a node $v \in V$. The following conditional probability distribution simulates the random walk:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

To perform the walk, given a starting node $c_0 = u$ for each subsequent node c_i, **Equation 4** presented above is used. It should be noted that these walks are of fixed length, which is user-defined. If an edge $(v,x) \in E$ exists, then the probability is defined by π_{vx}, which is the transition probability between the nodes, divided by Z, a normalisation constant.

While it is called a *'random'* walk, it is possible to influence how the graph is traversed through different hyper parameters. To do this, α is introduced as a search bias. For the α bias, two parameters are utilised: The *Return* parameter p and the *In-out* parameter q. p controls the likelihood of returning to the previously visited node. Having a high p value would cause the random walk to explore and traverse the graph more, while a low value would make the traversal return more often. With a low p value, exploring the local structure of the graph is emphasised. For the q value, one should consider the two traversal algorithms Breadth-First Search (BFS) and Depth-First Search (DFS). Having a high q value would be equivalent to having a low probability of going deeper into the graph. A high q value is, therefore, similar to BFS. On the other hand, having a low q value would mean the global structure of the graph is emphasised.

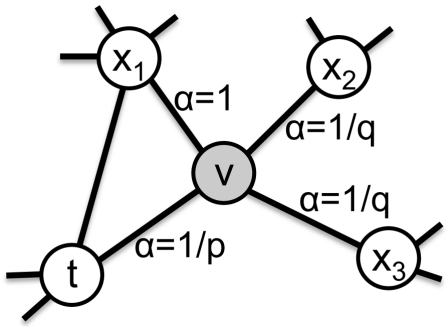


Figure 3.5: An example of a graph where we traverse from t to the current node v. v now has to determine which node to visit next, depending on the search bias and parameters p and q.

With this information in mind, the search bias, α, can now be defined:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{5}$$

When sampling the random walks, the transition probabilities (π_{vx}) are initialised to the weights of the edges, and afterwards will result in $w_{vx} = \pi_{vx}$ with the bias $\alpha(t,x)$ added. In the formula, d_{tx} denotes the distance between t and x, an illustration of this is shown on **Figure 3.5**.

Node2vec solves this optimisation problem by defining an objective function based on the maximum likelihood principle. The goal of the objective function is to optimise the log-likelihood of observing a *network neighbourhood* $N_S(u)$ for a node u, given a feature representation f:

$$\max_f \left\{ \sum_{u \in V} \log Pr(N_S(u)|f(u)) \right\} \tag{6}$$

3.3.2 Skip-gram Model

The skip-gram model is created for word2vec, and in that context, the model is used to find word representations that are useful for predicting the surrounding words. As mentioned, the node2vec implementation uses this exact model, but with the walks generated as its input. The skip-gram model follows the structure of a single hidden layer neural network, with an input layer, a hidden layer, and an output layer. A characteristic of this model, however, is that the hidden layer does not have any activation function, and projects the values of the input layer. Additionally, the output layer uses hierarchical softmax.[38]

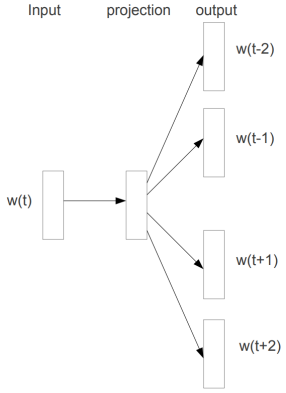


Figure 3.6: The skip-gram architecture for learning the word vector representations.[38]

3.3.3 Node2vec Algorithm

The pseudo-code for the node2vec algorithm is presented in **Figure 3.7** which summarises this section. The main parameter of the algorithm is the graph *G*. The remaining parameters are the user-defined hyper parameters discussed above, which are used to adjust the node embeddings. The node2vec algorithm shows how the neighbourhoods for the nodes are created. For each node in the graph, a random walk is performed. This is repeated *r* times, as *r* is the parameter specifying the number of walks per node. By performing random walks, neighbourhoods for each node is created, which are then used to maximise the objective function. The process of maximising the objective function (stochastic gradient descent) is equivalent to the process in word2vec.

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent(k, d, $walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to $[u]$
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr$, G')
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$

Figure 3.7: The node2vec algorithm.[30]

3.4 Relational Fusion Network

RFNs, based on Jepsen et al.[37], are designed to address certain shortcomings in state of the art GCNs in relation to road networks. It is capable of utilising node attributes, edge attributes, and between-edge attributes when predicting. Between-edge attributes are characteristics between road segments or edges. An example of a relevant between-edge attribute could be the angle between edges.

A road network is modelled as an attributed directed graph $G = (V, E, A^V, A^E, A^B)$, where V is a set of nodes, and E is a set of edges. A^V and A^E are mappings from the nodes and edges to their attributes, respectively. A^B is a mapping of a pair of edges (between-edge) to its attributes. The graph G, referred to as the primal graph, can also have a dual graph representation $G^D = (E, B)$ where $B = \{((u, v), (v, w))|(u, v) \in E, (v, w) \in E\}$ is the set of between-edges. This would make the edge set and between-edge set of the primal graph, be the node set and edge set in the dual graph.

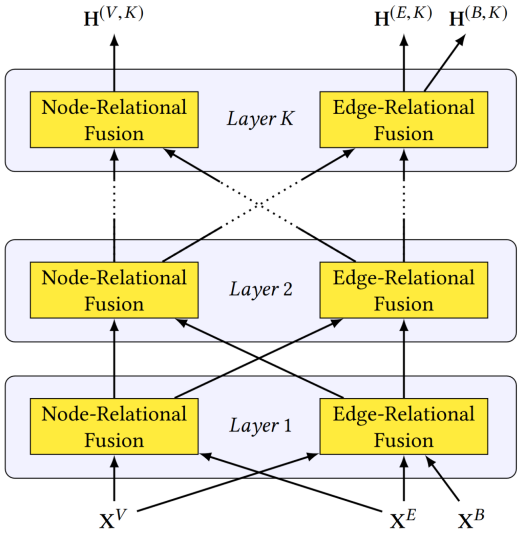


Figure 3.8: The architecture of a relational fusion network.[37]

An RFN consists of k relational fusion layers where $k \geq 1$. It takes the matrices $\boldsymbol{X}^V \in \mathbb{R}^{|V| \times d^V}$ for the numerical encoding of the node attributes, $\boldsymbol{X}^E \in \mathbb{R}^{|E| \times d^E}$ for the numerical encoding of the edge attributes, and $\boldsymbol{X}^B \in \mathbb{R}^{|B| \times d^B}$ for the numerical encoding of the between-edge attributes as input. The input is propagated through the k layers by performing node-relational and edge-relational fusion to learn representations (hyper parameters of the RFN) from the node-relational and edge-relational views. Node-relation fusion is performed by calculating the relation fusion operation on the primal graph. Edge-relational fusion is performed by using relation fusion on the dual graph. The relation fusion operation for the edge-relational view requires node attributes, edge attributes, and between-edge attributes due to the definition of between-edges.

3.4.1 Relational Fusion

The relation fusion operation takes a graph $G' = (V', E')$ along with the necessary feature matrices $\boldsymbol{H}^{(V',k-1)}$ and $\boldsymbol{H}^{(E',k-1)}$ as input. The relational fusion uses a FUSE^k function to compute the relational representations for each relation by fusing the input representation of source node v' ($H_{v'}^{V',k-1}$) and the target node n' ($H_{n'}^{V',k-1}$) of the relation along with the representation describing the relation between v' and n' ($H_{v',n'}^{E',k-1}$) for layer k. After computing the relational representations, they will be aggregated into a single representation of v' with an AGGREGATE^k function. After the aggregation, an optional normalisation step could follow. This is necessary if the representations have different scales among

the neighbourhood sizes. Different options for the FUSE^k and AGGREGATE^k functions exist. They will be analysed in this section.

Fusion Functions

There are two types of fusion functions that are responsible for extracting the information of each relation, namely ADDITIVEFUSE and INTERACTIONALFUSE. The purpose of these functions is to establish the sharp boundaries of the highly homophilic regions, which are regions where the features change drastically. This occurs often in road networks, where the speed limit of one edge can change to a significantly different value on another adjacent edge. With the ADDITIVEFUSE, the source node v', the target node n', and their edge, $(v', n') \in E$ all of layer $k-1$, are added together. Each of these representations will be multiplied by their respective weight matrix, for then to have a bias term added. In the end, it is encapsulated in a non-linear activation function σ.

$$\text{ADDITIVEFUSE}^k(\boldsymbol{H}_{v'}^{(V',k-1)}, \boldsymbol{H}_{n'}^{(V',k-1)}, \boldsymbol{H}_{v',n'}^{(E',k-1)}) = \sigma\left(\boldsymbol{H}_{v'}^{(V',k-1)}\boldsymbol{W}^s + \boldsymbol{H}_{n'}^{(V',k-1)}\boldsymbol{W}^t + \boldsymbol{H}_{v',n'}^{(E',k-1)}\boldsymbol{W}^r + \boldsymbol{b}\right) \tag{7}$$

where $\boldsymbol{W}^s$ is the weight matrix of the source nodes, $\boldsymbol{W}^t$ is the weight matrix of the target nodes, and $\boldsymbol{W}^r$ is the weight matrix of the relational weights. Furthermore, the same concepts apply when performing the fusion function with INTERACTIONALFUSE. The difference lies in how the different representations and their weight matrices are put together. INTERACTIONALFUSE uses the Hardamad product of the representations and encapsulates that in an activation function σ, and then adds the bias afterwards.

$$\text{INTERACTIONALFUSE}^k(\boldsymbol{H}_{v'}^{(V',k-1)}, \boldsymbol{H}_{n'}^{(V',k-1)}, \boldsymbol{H}_{v',n'}^{(E',k-1)}) = \sigma\left((\boldsymbol{H}^{(R,k-1)}\boldsymbol{W}^I \odot \boldsymbol{H}^{(R,k-1)})\boldsymbol{W}^r\right) + \boldsymbol{b} \tag{8}$$

where $\boldsymbol{W}^I$ is a trainable interaction matrix, and $\boldsymbol{W}^r$ is the relational weight matrix. The differences of ADDITIVEFUSE and INTERACTIONALFUSE, is that ADDITIVEFUSE summarises the relation of v' and n', while INTERACTIONALFUSE more explicitly model the nodes interaction and therefore has a improved modelling capacity.

Relational Aggregators

Among the existing aggregators for graphs, none are compatible with RFNs as they use common transformation of each neighbour, while the FUSE operation performs context dependant transformations. Therefore, the attentional aggregator needs to be applied. The attentional aggregator computes a weighted mean over the relational representations. The equation for it is:

$$\text{AGGREGATE}(F_v) = \sum_{Z_N \in F_v} A(v, n) f_n \tag{9}$$

where F_v is the set of fused relational representation of a node v. $A(v, n)$ is an attention weight that encapsulates the influence of neighbour node n on the aggregate of node v. The attention weight depends on the relationship between the nodes n and v in the input graph, and this relation is given

in the feature matrix $H_{v',n'}^{R,k-1} \in \mathbb{R}^{d^R}$. It is computed as:

$$A(v,n) = \frac{exp(C(H_{(v,n)}^{(R,k-1)})}{\sum_{n' \in N(v)} exp(C(H_{(v,n')}^{(R,k-1)}))} \tag{10}$$

where the attention coefficient function $C : \mathbb{R}^{d^R} \longrightarrow \mathbb{R}$ is defined as:

$$C(H_{(v,n)}^{R,k-1}) = \sigma(H_{v,n}^{R,k-1}\mathbf{W^C}) \tag{11}$$

W^C is the weight matrix, while σ is an activation function.

3.4.2 Forward Propagation

Forward propagation is the process of propagating the input graph through the k fusion layers and thereby calculating the predicted attributes for the input graph. The pseudocode for forward propagation in RFNs is shown in **Figure 3.9**.

Algorithm 2 Forward Propagation Algorithm

1: **function** ForwardPropagation($\mathbf{X}^V$, $\mathbf{X}^E$, $\mathbf{X}^B$)
2: **let** $\mathbf{H}^{(V,0)} = \mathbf{X}^V$, $\mathbf{H}^{(E,0)} = \mathbf{X}^E$, and $\mathbf{H}^{(B,0)} = \mathbf{X}^B$
3: **for** $k = 1$ to K **do**
4: $\mathbf{H}^{(V,k)} \leftarrow \text{RelationalFusion}^k(G^P, \mathbf{H}^{(V,k-1)}, \mathbf{H}^{(E,k-1)})$
5: $\mathbf{H}^{(B',k-1)} \leftarrow \text{Join}(\mathbf{H}^{(V,k-1)}, \mathbf{H}^{(B,k-1)})$
6: $\mathbf{H}^{(E,k)} \leftarrow \text{RelationalFusion}^k(G^D, \mathbf{H}^{(E,k-1)}, \mathbf{H}^{(B',k-1)})$
7: $\mathbf{H}^{(B,k)} \leftarrow \text{FeedForward}^k(\mathbf{H}^{(B,k-1)})$
8: **return** $\mathbf{H}^{(V,K)}, \mathbf{H}^{(E,K)}, \mathbf{H}^{(B,K)}$
9: **function** Join($\mathbf{H}^V$, $\mathbf{H}^E$)
10: **for all** $\big((u,v),(v,w)\big) \in B$ **do**
11: $\mathbf{H}^{B'}_{\big((u,v),(v,w)\big)} \leftarrow \mathbf{H}^{B}_{\big((u,v),(v,w)\big)} \oplus \mathbf{H}^V_v$
12: **return** $\mathbf{H}^{B'}$

Figure 3.9: Forward propagation in RFNs.[37]

When performing forward propagation at each layer, the nodes are first transformed by using relational fusion on the primal graph as shown on line 4 in the algorithm. In order to transform the edges, it is necessary to combine the features regarding between-edge and nodes, as an edge is characterised by these features as well. This is performed by the join operation on line 5, which calculates the direct sum of node feature matrix and the between-edge feature matrix for each pair of adjacent edges. The edges are then transformed on line 6 by performing the relational fusion operation on the dual graph with the edge feature matrix, and the feature matrix from the join operation on line 5. The between-edge representation is then transformed by a feed forward operation. The forward propagation algorithm produces three outputs, namely node, edge, and between-edge classifications.

However, if only some of the outputs are necessary, it is possible to "switch" off some of the operations in the last layer to save computational resources. For example whenever $k = K$ on line 4 and 7, it can be replaced with $H(V,k) = H(V,k-1)$ and $H(B,k) = H(B,k-1)$, respectively.

3.4.3 Training and Backpropagation

As mentioned in the fusion function and relational aggregator paragraphs, the weights of the RFN are the hyper parameters which can be adjusted to improve the performance of the network. To determine how incorrect the predictions of the RFN are during the training phase, a notion of loss is necessary. This notion is implemented through an objective function called a loss function, which calculates the loss in terms of the predicted values from the RFN and the expected values which is known beforehand. In order to calculate the loss, a forward pass is necessary through the forward propagation algorithm explained earlier in this section. This loss would then be needed to be propagated back through the layers of the RFN, and adjustments would be made to the weight to minimise this loss. The propagation of the loss backwards through the layers can be accomplished by calculating the partial derivative of the loss with respect to the weights, and afterwards applying a learning rule with the partial derivatives. The derivatives can be calculated and propagated through the layers with the help of a variation of the chain rule for differential calculus.

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \frac{\partial L}{\partial o} * \left(\sum_{[h_r,h_{r+1},..,h_k,h_o] \in \mathcal{P}} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{h_i} \right) \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \tag{12}$$

where $h_1, h_2, .., h_k$ are hidden units, o is the output, the weight between an arbitrary h_r to h_{r+1} is denoted as $w_{(h_r,h_{r+1})}$, and $\mathcal{P}$ is the set of all paths from h_r to o. The gradients will be calculated by calculating the gradient of the output layer first, and then last hidden layer in a backward fashion of the network.[39][p.21-24]

3.5 SW604F19 - Tag Classification on OpenStreetMap

aSTEP-2019 already has a similar service to the service for Task 2, that we intended to develop. We will analyse the Attribute Completion service to determine if our new service should adapt any of its design choices or implementations.

By inspection of the controls and UI, the service lets the user choose between two classification algorithms (KNN and GCN) from the control panel. Here there is also the option to choose from a list of predefined cities to show the predicted attributes on with the selected algorithm. From these two selections, the service will show the results in the chart-area. Here a map-geo chart is used to show all edges within the chosen city. Each edge are classified by a certain speed limit, that each has its own specific colour. A screenshot from the service on the aSTEP website is shown on **Figure 3.10**. The service also provides other results to be shown like a map showing the training data, test data, and accuracy.

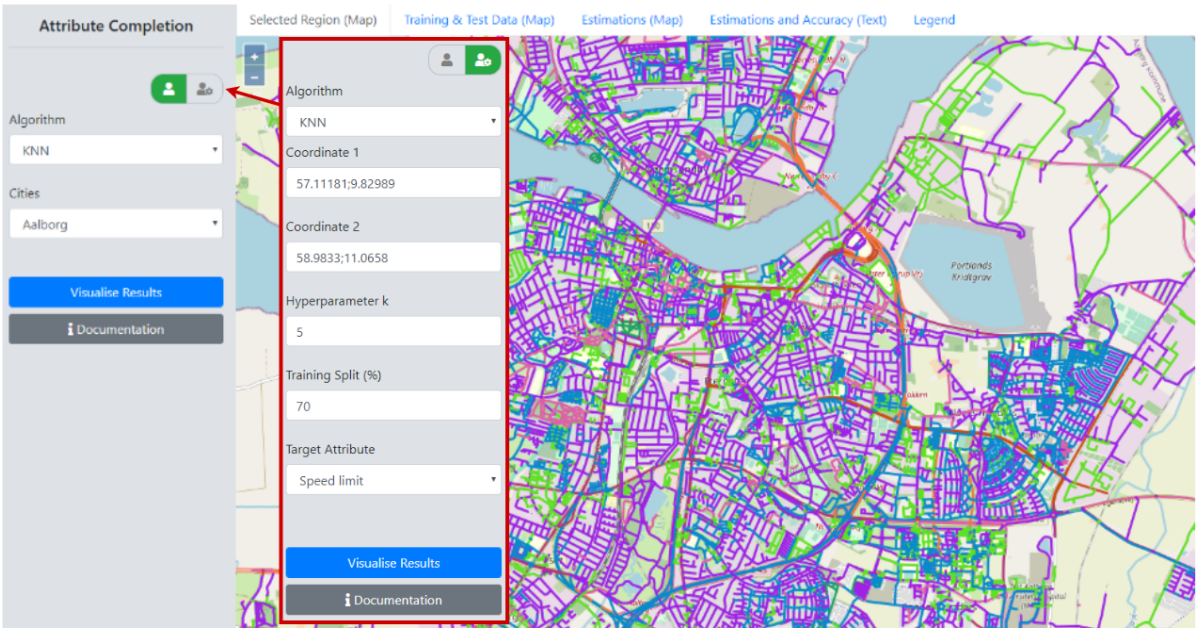


Figure 3.10: Screenshot of the Attribute Completion Service from aSTEP-2019, showing the UI with both the user and developer fields.

In order to get more options to change the visualised graph on the map, the service has a few additional controls within the *developer fields*, which are visualised in the red squared box on the figure. For the KNN algorithm, there is the option to define the bounds of the graph, as well as a few other options to change the training and outcome of the predictions. This functionality, however, does not seem to work, which could be caused by the service interface changes made in aSTEP-2020. Besides, the functionality of the developer fields for the GCN algorithm are not fully implemented either.

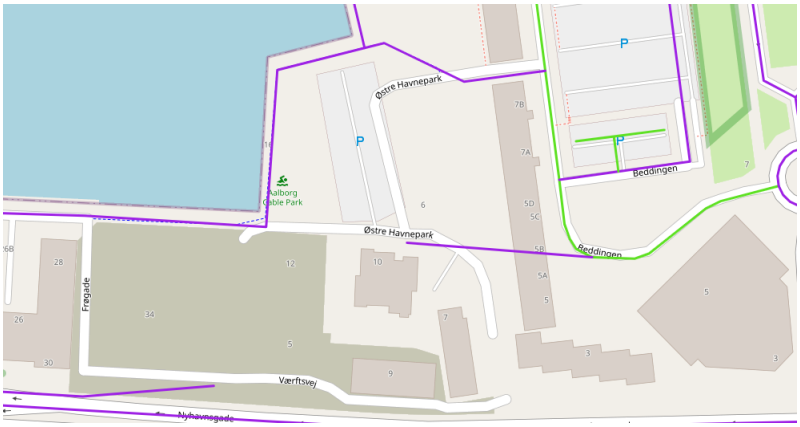


Figure 3.11: Example of incorrect road segments (edges) shown on the map in Aalborg from the aSTEP-2019 Attribute Completion micro-service using KNN. The same issue applies to GCN.

Also, when taking a closer look the predicted road segments, they often do not correspond to the roads on the map it self. An example is shown on **Figure 3.11**. The service seems to make predictions on roads that does not exist on the map-geo chart. In that way, this service will not be able to give predictions on the map data, that all routing services uses in the aSTEP-2020 project.

From the analysis, we have concluded, that the most viable solution is to implement our new service without using the any code of the existing service. However, we have deemed certain design aspects of the aSTEP-2019 Attribute Completion fit to be used in our new micro-service. It will have very similar front-end (UI and controls) with some changes and additions, together with a new back-end prediction algorithm. This includes the way the user can choose both predefined and user defined areas to get predictions on. For the prediction itself, we have chosen a more specialised algorithm, namely RFN, rather than using the KNN or GCN algorithm that also lacks a full implementation in the existing service.

3.6 Designing the Attribute Prediction Service

We now determine the overall architectural design of the implementation for Task 2. Similar to the design plan of the MM micro-service, we will provide a diagram of the internal and external flow of the service. On top of that, we will also propose a design for the graphical user interface, to give our self additional guidelines for the implementation. This design plan, however, is made with the possibility of changes.

To enforce modularity, for future services also, we have made a decision of dividing the functionality of the initial AP service into two separate micro-services. The first service will be referred to as the Graph Attribute (GA) service. It will first and foremost be used as the front-end part of our solution for Task 2, where users from the aSTEP website can interact with and can take a look at the predicted attributes on the road network. Secondly, the GA service will also, from a given input, be able to fetch a corresponding graph from the *mapdata* database. A user will be able to give input to the service from the aSTEP website through various input fields. Similarly, other services will also be able to send requests to the GA service with same type of input, thus making the GA service serving as a "subgraph fetcher" tool for other service as well.

The second service will act as the back-end, regarding our solution to Task 2. This means it will perform the actual attribute prediction, hence this service will go under the original name of the Attribute Prediction (AP) Service. We also aim to make this service serve as a general tool to predict attributes on a given road network, that is not only able to make predictions on the road network that will be provided by the GA service, but any road network with missing attributes. Exactly how the GA and AP services are expected to be connected to complete Task 2 will be explained in the component design below.

3.6.1 Component Design for Task 2 Services

The component design of the two services encapsulates the communication between the GA and AP service as well as the communication between external micro-services plus the user from the aSTEP website. This is depicted on **Figure 3.12**. Below the figure is a step by step explanation of the internal flow, that follows enumerations on the figure.

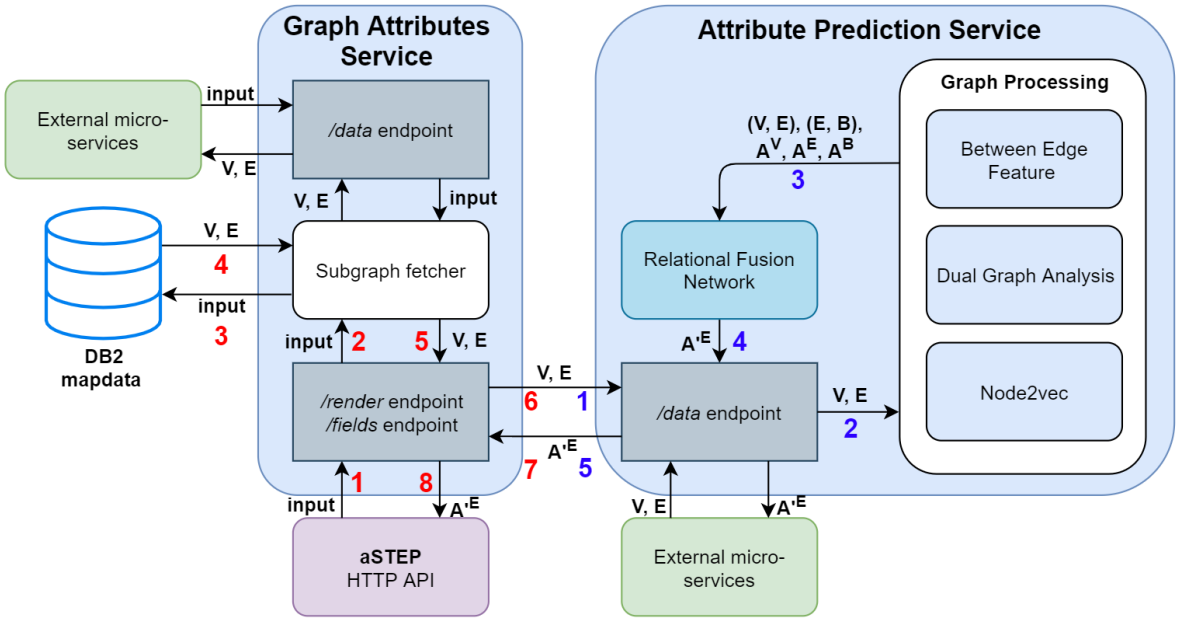


Figure 3.12: Component design of the two services aimed to solve *Task 2*. It shows the internal flow of the data being provided, either by the aSTEP website or an external micro-service, to the GA service and afterwards passed through to the AP service to finally arrive back at the GA service again.

Graph Attributes Service

The enumeration below describes the steps highlighted in red on the left side of the figure. This is the GA service.

1. From the aSTEP website UI, the user provides the GA service with an input defining a graph to be predicted on, using the input methods from the previous section.
2. The */render* forwards the input information to the *subgraph fetcher*, so a graph can be fetched.
3. The *subgraph fetcher* defines the boundaries of the subgraph from the input and queries the *mapdata* database with this information.
4. From the mapdata database, a tuple containing the corresponding vertices V and edges E, that defines the subgraph, is extracted.
5. The *subgraph fetcher* passes the extracted subgraph back to the */render* endpoint.

6. */render* sends a POST request to the AP service with the subgraph (V, E).
7. After receiving the subgraph from the GA service, the AP service will perform all its internal actions (these are all explained in the steps for the AP service, below), and finally return the predicted attributes A'^E for each edge in the subgraph.
8. In the last step, the */render* endpoint handles the visualisation of the the labelled subgraph to the user on the aSTEP website.

Additionally, the GA service can be used as a subgraph fetcher tool by other external services. This scenario is illustrated in the top of the GA service. The procedure is identical to step 1-4.

Attribute Prediction Service

Finally, we will explain the steps highlighted on the right side of the figure regarding the AP service. These steps correspond to the numbers highlighted in blue.

1. The AP service receives a graph from the GA service in form of a set of nodes V and a set of edges E.
2. This tuple is then passed through the three processes encapsulated in the *Graph processing* box. Here, the input graph is embedded, dual graph analysed, and between edge features are calculated. As a result, *Graph processing* returns a 6-tuple with the input graph (V, E), node attributes A^V, edge attributes A^E, dual graph (E, B), and between edge attributes A^B.
3. The 6-tuple is then fed into to a pre-trained RFN model.
4. The RFN model returns a corresponding set of edge attributes A'^E to the */data* endpoint.
5. */data* returns said edge attributes to the respective service whom send the request.

3.6.2 Graphical User Interface Design

We will primarily focus on the GA service regarding the GUI. We propose a number of options for letting the user visualise the predictions returned from the AP service. The chart-area will be inspired by the existing Attribute Completion service, where a *map-geo* chart is exploited. Here, the idea is to highlight the roads in different colours, to illustrate the road segments' predicted attribute(s).

To realise this design, the graph data must be compatible with the map used by the *map-geo* chart. It is also necessary to use a subgraph of the *mapdata*, as it is not possible, computational wise, to give predictions on the entire graph of Denmark, all at once.

The existing Attribute Completion service provides the option to choose between predefined cities. This seems like a viable solution, as we then can ensure that the input from the predefined cities corresponds to a subgraph taken from the mapdata database. We also expressed in the analysis of the Attribute Completion service, that we also wanted to provide the user with more freedom to customise the input. Therefore, we imagine to let the user choose a subgraph from the *mapdata* database, through three options:

1. We could provide fields for entering a GPS-point and a radius (double). From this information, it would then be possible to calculate a circle and determine the set of edges and nodes from the *mapdata* database, which is contained within this circle. These edges and nodes will then function as our subgraph.

2. Another solution could be to copy the design of the drop down menu, with selected cities in Denmark as choices, from the Attribute Completion service, and as an addition provide a field to let the user determine the radius (double). From this information, it would then be possible to retrieve a graph from the database which contains all edges within the circle defined by the selected city's centre and the radius.

3. The last option would be to provide two fields for two files specifying the graph; one file containing a list of edges and one containing a list of nodes. Then from this information, we can construct a graph and use it as input to our service.

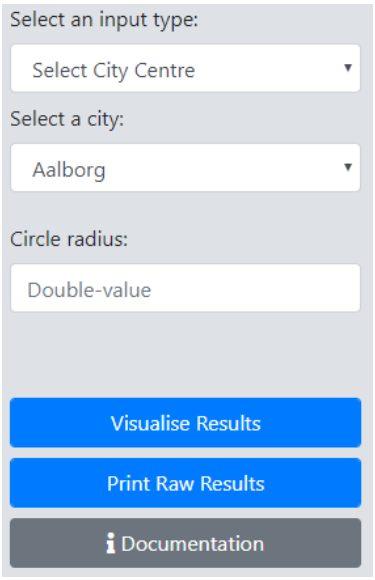


Figure 3.13: Two input fields to let the user select a specific city and the radius of the circle from the city's centre. The defined circle will act as the subgraph.

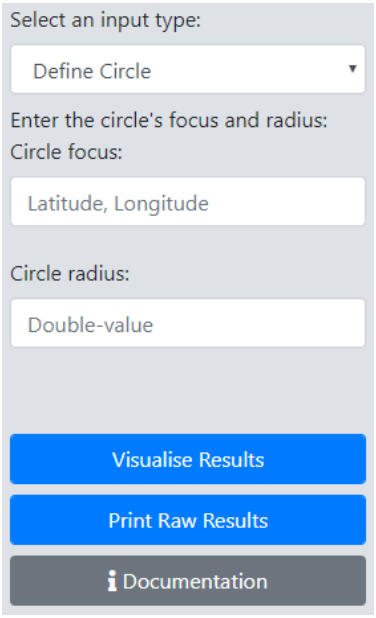


Figure 3.14: Two input-number fields for a focus and radius to define a subgraph kept within the circle they form.

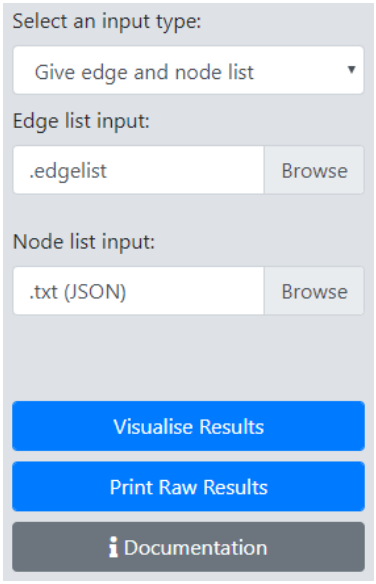


Figure 3.15: Two file input fields to let the user provide an .edgelist and a list of GPS-points in a file format, to define the subgraph.

Figure 3.16: The three options for user interaction with the GA micro-service.

After evaluating the design plan, and the fact that none of these options are mutual exclusive, we determine all three options should be implemented. The prototypes of the design input field designs are shown on **Figure 3.13**, **Figure 3.14**, and **Figure 3.15**.

3.6.3 Feature Selection for the Prediction

When selecting which feature we should try and predict, we have a few choices to consider. To select a feature, we have to consider how training should be performed, what data is available, and how it correlates with the goals we are trying to achieve with the service.

The following data is available in the *edge* table of the *mapdata* database:

1. Distance (description of the length if an edge in meters).
2. Highway (categorisation of the road type, i.e. "Residential" or "Secondary").
3. Max speed (the speed limit for the road segment).
4. Edge Name (name of the road, which the edge is part of).

Of these choices, *Distance* and *Edge Name*, seem inadequate to use as features. We assume that there is little correlation between the distance of surrounding edges and the speed limit of an edge, due to how edges is modelled on road networks. Regarding *Edge Name*, we also do not see any correlation between the name of a road and its speed limit. These characteristics are therefore discarded.

The feature *Highway* would be suitable for classification of the edges. The other potential feature, *Max speed*, is also possible to use in a classification or regression task. However, from the implementation of the *mapdata* database, described back in **Section 2.3**, we know that *Max speed* and *Highway* are not represented on all edges. Hence, it would not be possible to use these features. Therefore, the aforementioned features are not plausible as RFN requires attributes for all edges. One solution to edge features that are present for all edges, while also providing meaningful features, would be the node embeddings of the dual graph. The node embeddings of the dual graph are embeddings of the edges in the primal graph due to the definition of the dual graph.

3.6.4 Selecting a Between-Edge Feature

To utilise the RFN architecture, we also have to decide on a *between-edge* feature to propagate through the layers. We have to consider the available information for any road network to ensure modularity of the AP service.

We use the *latitude* and *longitude* of the three nodes that define the between-edge to calculate the angle between edges, inspired by Jepsen et al. [37]. We deem this between-edge attribute to be the one used for our predictions, as GPS-points are a common feature that exists in all road networks.

3.6.5 Node2vec Graph Processing

As we want the possibility for the AP micro-service to accept any road network, making node2vec an integral part of the design is necessary. We can not assume that all graphs will have equivalent

numerical embedding of their nodes, and we will, therefore, have to use node2vec to ensure nodes have this type of embedding. This will add complexity to the runtime of the service, as performing the random walks of node2vec can be cumbersome on larger graphs. Still, it also allows users to not consider this step as part of their interaction with the service. When embedding each input graph individually, we create a unique representation for each graph, which we hope will improve the predictions as well.

3.7 Implementation of the Task 2 Services

Based on the component design of the GA and AP services from the previous section, we will now document the implementation of the *subgraph fetcher*, *graph processing*, and *RFN*.

3.7.1 User Interface and Interaction

Like the MM service, the GA and AP micro-services are both implemented following the *2020v1* interface guidelines. The two services serve different purposes, with one serving as a subgraph fetcher tool, managing the user interaction, while the other is a graph processing and attribute predicting tool. As such, we will document the implementation of each micro-service's components separately.

As an addition to the aSTEP UserInterface micro-service, we have added a legend to the *map-geo* chart to give an overview of the *OpenLayers-features* being plotted on the map. More details about the implementation can be found in **Appendix A**.

The Graph Attribute Service

Amongst the two micro-services, only the GA micro-service has a GUI. It displays the set of edges with their predicted attributes provided by the AP micro-service on a map in the chart-area. Hereto, we once again expose a *map-geo* chart, with the new legend feature, to the */render* endpoint.

As input options for this micro-service via the aSTEP website, we have chosen to implement two input types, based on the design in **Figures 3.13** and **3.14**. Additionally, instead of calculating the bounding circle using the centre and radius, we instead calculate a bounding box. This makes it easier to utilise the *BETWEEN* operator when querying the database for nodes and edges within the bounds, and decreases the runtime complexity. Each of the above input options has been encapsulated in a *form-select* field, which is a drop-down field. These two input types are defined through the */fields* endpoint and the */info* endpoint. The *application/json* format returned from these endpoints can be found in **Appendix B**.

Subgraph Fetcher

When a user posts a request to the GA micro-service through the aSTEP website and provides a centre-coordinate in *latitude,longitude* format, and a float that describes the *radius* in kilometres, the *subgraph fetcher* will calculate the two corner-coordinates that define the bounding box. Afterwards, it will query the *mapdata* database and return a JSON dictionary containing all nodes and edges within the bounding box. A more elaborate documentation of this functionality can be found in **Appendix C**.

Similarly, when another micro-service sends a similar request using POST through the GA micro-service's input-channels, the same JSON dictionary will be returned to the micro-service through the */data* endpoint.

3.7.2 The Attribute Prediction Micro-Service

The AP micro-service has two input-channels called *nodes* and *edges*, and the channels allow for both raw data and files to be received. *Nodes* will receive a list of nodes and *edges* will receive a list of edges. This corresponds to the input-type design in **Figure 3.15**. The */info* endpoint, that defined these input channels, can be found in **Appendix D**

The received edges must correspond to the received nodes, as they will be used to construct a graph. If the input is a disconnected graph, the largest weakly connected component will instead be selected. This is calculated through the *networkx* library, which provides the functionality to find weakly connected component in a graph. For the edges, an optional speed limit feature can be provided. The input must follow the JSON format shown on the code snippets below. This makes the service able to receive any road network and make predictions on its features, as long as it follows the described format and has the needed spatial dependencies.

```
[
    {
        "node_id": node_id,
        "lon": float,
        "lat": float
    },
    ...
]
```

Code Snippet 3.1: Expected JSON format of the list of nodes given as input to the AP service.

```
[
    {
        "edge_basenode": node_id,
        "edge_adjacentnode": node_id,
        "feature": "edge feature"
    },
    ...
]
```

Code Snippet 3.2: Expected JSON format of the list of edges given as input to the AP service.

After receiving a satisfactory graph, a set of nodes, and a set of edges and optional speed limits for the edges, the AP service will perform the actions encapsulated in the *Graph processing* box in the AP service on **Figure 3.12**, where node2vec will be run on both the primal and dual graphs, and afterwards give the results as input to the RFN. To use the graph as input for the RFN, node attributes, edge attributes, and between-edge attributes along with the primal graph and dual graph

are required. Therefore, three steps are necessary before propagating through the RFN, which are:

1. Node attribute and edge attribute extraction using node2vec.
2. Between-edge attribute extraction in terms of calculating the angle between edges
3. Dual graph analysis.

Node2vec Library

If the input graph is a primal graph, the first step is to embed the graph with the optional speed limits. On the other hand, if the input graph is a dual graph, then this step is unnecessary. After this optional step, we need to run the node2vec algorithm on the graph, which is accomplished by using the node2vec library[40], which assigns attributes to all nodes in the input graph.

In our implementation, this includes constructing a directed networkx graph and running the node2vec algorithm on the graph. Once that is accomplished, and the algorithm has produced a model with a vocabulary of neighbourhoods[1] of recognised nodes, every node in this vocabulary is fit to a vector-representation, that then describes all nodes. The matrix that contains all these node-vectors is what constitutes the node attributes of the input graph.

For the utilisation of node2vec, we initialised it as shown in **Code Snippet 3.3**. The amount of *dimensions* is inspired by Jepsen et al.[37], while the *walk_length* along with the number of walks are the default values of the node2vec library. We decided to set our q value to be three times the p value[2], to emphasise exploring the global structure of the input graphs. After the initialisation, the model is trained using the *fit* method. With the trained model, the *word* vectors can be extracted and used as the node features.

```
pre_process = node2vec.Node2Vec(G, dimensions=1, walk_length=80,
                                num_walks=10, workers=1, p=1, q=3)
node2vec_model = pre_process.fit()
node_features = mxnet.nd.array(node2vec_model.wv.vectors)
```

Code Snippet 3.3: Implementation of the node embedding using node2vec.

Between-Edge Attributes

We define between-edge attributes as the angle between two edges, calculated from the location of the nodes, described by three coordinates. As mentioned in **Section 3.4**, a between-edge is defined by two edges, where one node is part of the two edges. This results in three coordinates for the between edge, which can be used to construct vectors (in 2D). The vectors can then be used in **Equation 13** to calculate the angle between the road segments.

$$cos(\alpha) = \frac{\bar{a} \cdot \bar{b}}{|\bar{a}| \cdot |\bar{b}|} \iff \alpha = \cos^{-1}\left(\frac{\bar{a} \cdot \bar{b}}{|\bar{a}| \cdot |\bar{b}|}\right) \tag{13}$$

[1]Node2vec is based on word2vec, which uses this term to describe recognised words
[2]Recall the meaning of the p (return parameter) and q (In-out parameter) in **Section 3.3.1**

where α is the angle between the road segments, and the vectors $\bar{a}$ and $\bar{b}$ represent road segments.

```
import numpy as np
edge1 = coord1 - coord2
edge2 = coord3 - coord2

cos_v = np.dot(edge1, edge2) / (np.linalg.norm(edge1) * np.linalg.norm(edge2))
angle = np.degrees(np.arccos(cos_v))/180
```

Code Snippet 3.4: Implementation of the between-edge angle calculation.

Code Snippet 3.4 shows how the between-edge angle is calculated in practice. The angle value is furthermore normalised in the interval of 0 and 1 as large values would result too excessive loss during the training phase of the RFN, to be represented by a single integer and would result in a "nan" value.

Dual Graph Analysis

We construct the dual graph from the given primal graph $G = (V,E)$, by taking each edge $e \in E$ and converting them to the node set of the dual graph. The edges of the new dual graph is the between-edge set of the primal graph, given by $B = \{((u,v),(v,w))|(u,v) \in E,(v,w) \in E\}$. After creating the node set of (u,v), we iterate through the successors of v, $\{i_1,...,i_n\}$, in the primal graph. We add edges to the dual graph with (u,v) as the source node and (v,i_k) as the target node for $1 \leq k \leq n$. An illustration of the primal graph and the corresponding dual graph is shown in **Figure 3.18** and **Figure 3.19**.

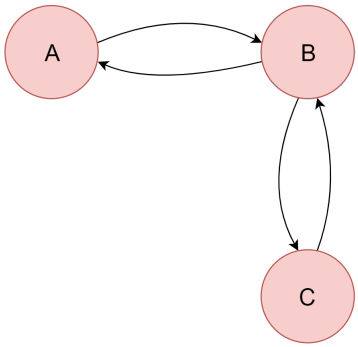


Figure 3.18: An illustration of a primal graph.

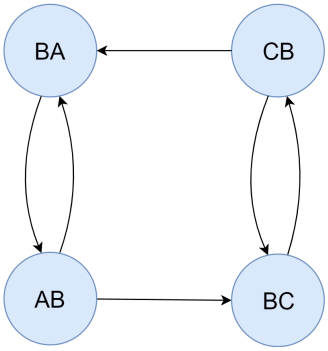


Figure 3.19: The dual graph constructed from the primal graph on **Figure 3.18**.

Relational Fusion Network Library

To implement an RFN model, we can utilise the RFN library produced by Tobias Skovgaard Jepsen on GitHub[41]. A model is instantiated by defining the structure of the input features along with

a specification containing the structures of the relational layers of the network and the specification of the output layer. In the latter specification, we also state that our output will only be the edge attributes.

After instantiating the model, it can be used to make predictions by forward propagating through its layers. The forward propagation takes the feature matrices X^V, X^E, and X^B along with the representations of the primal and dual graph. Instead of directly using the graphs, matrix representations are used. A node adjacency matrix N_{node}, edge adjacency matrix N_{edge}, and a matrix representing whether an entry into N_{node} or N_{edge} exists, will be used instead of the primal graph. For the dual graph, besides the aforementioned matrices, a matrix N_{common_node} will used to represent nodes in the primal graph that are connected by two edges (between edge), and another matrix N_{common_mask} to indicate whether an entry in N_{common_node} exists. The output of the forward propagation is a $|E| \cdot 1$ dimensional vector representing the predicted speed limits. This vector is then converted to a mapping from the edge (edge_basenode and edge_adjacent node) to the predicted attribute, and returned to the caller.

However, as the weights of the RFN are initialised based on a weight initialisation technique like the normal distribution, we will not get satisfactory results on the predictions. Instead, we need to train an RFN model. The implementation of the RFN model training will be documented in the next sprint.

3.8 Collaboration

The collaboration of Sprint three focuses on the routing groups doing Task 2 and 3. Group 1, who does the routing using real traffic information, requested ways to obtain predictions of the road network's edges that has missing features. This meant, that when designing the AP micro-service, it should be made possible for other groups to interact with the service and obtain predictions easily.

We considered two options for solving this issue: One was to allow external micro-services to supply the AP micro-service with an arbitrary road network, in which they specify the feature they wish to predict. The other solution we considered was to enter all predictions in the database for the edges that had features missing. This way, they could look up each edge themselves and retrieve the predicted information. The solution that made the most sense was to allow users to supply the AP micro-service with an arbitrary road network. One reason was, predicting edge features for a road network the size of Denmark would not be feasible computation wise. Another reason is the predictions differ depending on what road network is being used, despite it being a sub-graph. Having the inter-connectivity with external micro-services also allow future micro-services to utilise the service.

3.9 Sprint Three Review

Sprint three has been extensive, in regards to the amount of sprint backlog items and the proportion of work associated with completing them. An unforeseen obstacle during this Sprint was the change to the *2020v1* service interface, meaning that we had to make changes in our otherwise finished MM micro-service.

Backlog Index	Completed	Status
S3.1	✓	We ensure a 1-to-1 correspondence between the original GPS-points and the map matched GPS-points by using GPS-points from the road network itself in places where the Error-σ have caused a less amount of map matched GPS-points than original GPS-points.
S3.2	⊗	The endpoints have been implemented in accordance with the aSTEP micro-service *2020v1* interface specification at the end of Sprint three. However, instead of implementing a single service to solve the attribute prediction task, we chose to divide the functionality into two separate micro-services, the GA and AP micro-service. In this way, the two services can act as tools on the aSTEP platform, that also can be utilised by other services in the future. Naturally, this decision also increased the complexity of the implementation; we did not entirely complete every aspect of the implementation in this sprint as intended. The key components in each micro-service are implemented, but we still need to ensure correct data parsing between the services through the defined input and output channels. Hence, we deem this backlog item partially completed.
S3.3	✓	We have successfully implemented the node2vec library in the AP micro-service, in order to embed the nodes in the graph given as input.
S3.4	⊗	We have implemented the RFN library, but the ability to train the network is not functional yet, as an issue occurred during the relational fusion operation. As such, we cannot consider this backlog item complete.
S3.5	⊗	The prediction implementation for the RFN has not been fully implemented, as the current prediction functionality only works on certain graphs.
S3.6	✓	An investigation of the node2vec and RFN theory was conducted. Hereby, we illustrate understanding of the concepts used in the implementation.
S3.7	✓	From experience with implementing the MM service and by investigating the existing Attribute Completion service, we have made design plans for both the GUI and the internal components of the two services. To the component design plan, we made a chart illustrating the data flow both internally and externally from our service.
S3.8	✓	We have, by adjusting Error-σ, achieved improved map matching results compared to our initial implementation in previous sprint. We also tried to adjust the transition probability, which did not have any significant effect.

Table 3.3: Review of the sprint backlog for Sprint three. The completed items are denoted by ✓, while the incomplete ones are marked with O. Partially completed items will be denoted by ⊗.

We hoped to commence the final sprint being able to train a model fit to produce usable predictions in the AP service. However, due to the issue regarding the relational fusion operation, we must bring this task into the next sprint to be solved. We will thereby also be able to utilise the predictions in the GA service.

Sprint Retrospective

To ensure that we made steady progress during this sprint, we saw it necessary to make several new guidelines for the group regarding how the project work should function during the time in which the university premises are closed. These guidelines were a part of the new RMMM plan for the COVID-19 risk. Following these guidelines, we were able to maintain a more constant workflow, although it also entailed the separation of a group member. We did, in the initial risk analysis, deduce the risk of personnel loss. Thus, to mitigate these risks, and looking forward, we aim to continue following our project work guidelines moving into the last sprint.

4 Sprint Four

The main objective of the final sprint relates to the testing and evaluation of our micro-services. However, as our GA and AP micro-services are still incomplete, we must first implement the RFN algorithm correctly and ensure the correct format of the predictions. The documentation of this implementation will also be recorded in this section. Albeit necessary, this will result in less time to do the actual testing and evaluation.

4.1 Sprint Goals

The final sprint can be encapsulated in the two goals below:

Goals
• Finalise the GA and AP micro-services. • Test and evaluate the results of our services.

Table 4.1: Goals for Sprint four.

4.1.1 Sprint Backlog

Once again, we take each sprint goal and specify them into more elaborate requirements. To finalise the GA and AP service, we revisit aspects of three partial completed backlog items from the previous sprint.

Index	MoSCoW	Requirements
S4.1	(**M**)	With the primal and dual graphs and feature matrices, train an RFN.
S4.2	(**M**)	Predict the speed limit on the unlabelled edges of a graph.
S4.3	(**M**)	Ensure correct data passing between the micro-services through the defined input and output channels.
S4.4	(**S**)	On the GA micro-service, ensure correct visualisation of the road segments and their predicted attribute on the chart map.
S4.5	(**S**)	Partition the data for the training of the RFN model into separate training, and validation sets.
S4.6	(**C**)	Train the model by adjusting on the hyper parameters.

Table 4.2: Sprint backlog for Sprint four.

4.2 Finalising the Implementation of the Two Services

To finalise the implementation of the AP micro-service, we first have to settle on an initial architecture for the model, before training it. We determine whether the training was successful by evaluating our results in terms of training loss, validation loss, and test loss. The backlog items (**S3.4** & **3.5**) that were deemed partially fulfilled in Sprint three, have to be resolved as well for the AP micro-service to function.

After finishing the AP micro-service, we will deploy it together with the GA micro-service. By securing correct communication between the two micro-services, we can consider the sprint goals complete.

4.2.1 The RFN Architecture

The RFN architecture heavily influences the representational power of the RFN. This means that having more layers and units will not necessarily result in better predictions. Finding an optimal architecture, where the model efficiently learns the features of the data, is the goal.

For the architecture of our RFN, we have decided to use two hidden layers, each with a specification as shown in **Code Snippet 4.1**. Unique to RFNs, we have to decide which fusion function and aggregator we wish to utilise for each layer, on top of the amount of units and which activation function. The fusion function used in the hidden layers is of *interactactional* type, presented in **Section 3.4**. This is said to have increased model capacity over the *additive* type. The aggregator used for the hidden layers is the *attentional* aggregator, as it was specifically designed for RFNs, and other aggregators are incompatible.

Regarding units, normalisation, and activation, we use standard practices. The units are set to be a value of $k > 1$ and 2^k, in this case 128. The activation function is Exponential Linear Unit (ELU), defined as:

$$R(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha.(e^z - 1) & \text{if } z \leq 0 \end{cases} \tag{14}$$

where z is the input and α is a multiplier. With this activation function, it allows negative values to pass through the layer, which in a regular ReLU layer, would just be set to 0. Lastly, the normalisation *L2Normalisation*, takes the input vector and normalises it by the square root of the sum of the squared vector values.

```
RFNLayerSpecification(
            units=128,
            fusion='interactional',
            aggregator='attentional',
            normalization=L2Normalization(),
            activation=ELU()
        )
```

Code Snippet 4.1: An initialisation of a single hidden layer in the RFN architecture.

The output layer will have a different specification. Here, there is only one output unit, which specifies the output dimension. The additive fusion function, non-attentional aggregator, and ReLU as the activation function will be utilised as well, demonstrated by [41].

4.2.2 Training of the Relational Fusion Network

To train the model, a dataset is necessary. We first describe how we process our data into three datasets, namely the training, validation, and test set. After the dataset process, we explain the actual training of the model through backpropagation. Then we evaluate on the results of the training by plotting the training and validation loss, and compare these values with the loss from the test set.

Processing of dataset

Before training the RFN, we need to acquire a dataset to train on. For this purpose, we will be using the data, which we stored in the *mapdata* database. In the training phase, it is necessary to know the ground truth of the edges, which we want to predict, as the loss between actual and predicted attributes will be used to adjust the weights of the RFN. Therefore, we query the *mapdata* database for edges where the speed limit (*max_speed*) attribute is known.

After querying the edges from the edge table in the *mapdata* database, they are converted to a directed graph using the networkx library. This graph will contain weakly disconnected components, which we use to make a list of subgraphs from. Some of these weakly connected components consist of a few nodes, which might not be extensive enough to represent real road network data. Therefore, we discard subgraphs with 20 or fewer nodes, as we deem this type of graphs inadequate. Among the subgraphs, we choose one graph with 29,008 nodes as the validation set, another graph with

6,091 nodes as the test set, and the rest of the graphs will be considered the training set. The data processing work flow is illustrated on **Figure 4.1**.

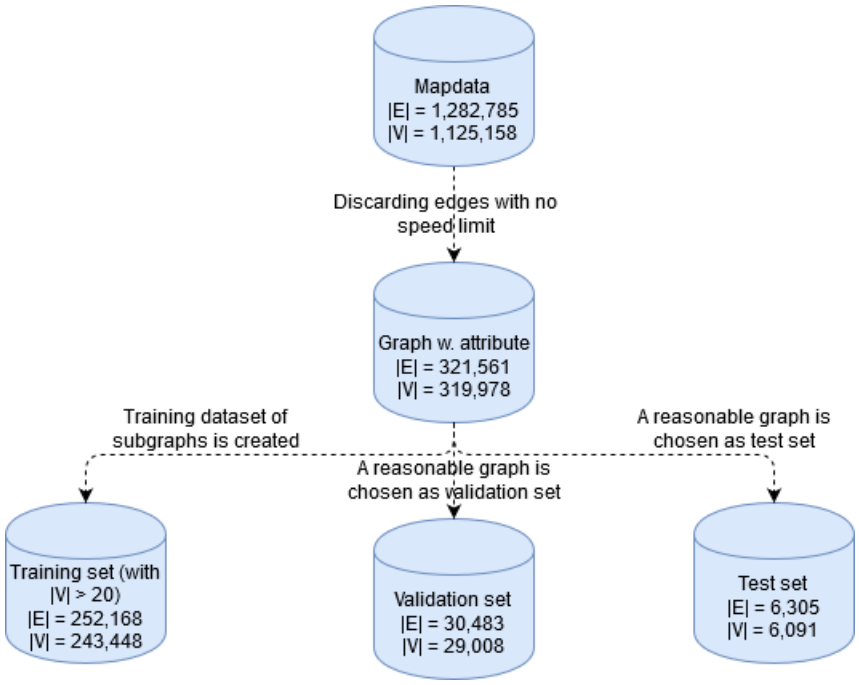


Figure 4.1: Illustration of how the graph in the *mapdata* database is partitioned into a training, validation and test sets for the RFN. The training set is the dataset, we will use to train our model, while the validation set is a separate dataset used to evaluate objectively the performance of the model throughout the epochs. The test set is a third separate dataset used to evaluate the final model.

Training

We train our RFN as a regression model. For the training phase, we iterate through a predefined number of epochs, where we in each epoch also iterate through each of our weakly connected graphs from the training dataset. For each graph, we perform the operations in the *Graph processing* box from the component design (**Figure 3.12**). This means that we perform node attribute extraction with node2vec on the primal graph, edge attribute extraction by performing node2vec on the dual graph, and between-edge attribute extraction by calculating the angle between edges as described in **Section 3.7.2**.

The next step is to forward propagate through the RFN. The forward propagation is how the RFN predicts, meaning that this prediction functionality is explained in **Section 3.7.2**. After getting the prediction through forward propagation, we calculate the loss in the prediction of the RFN by using a loss function. We use the mean squared error as our loss function. The forward propagation is performed with autograd, which is a functionality from mxnet to record gradients. This means that when the calculation of loss is performed, the gradients of the loss, with respect to the weights of the RFN, will be stored.

The last step is to backpropagate the loss, which is accomplished by the aforementioned functionality from mxnet. When backpropagating to adjust the weights, an optimisations method is necessary to specify how the gradients should be used to adjust the weights. We use adam as our optimisation method. **Appendix E** elaborates further on the implementation of the training.

Issues in Relational Fusion Network Implementation

An issue arose when we created a dual graph from our primal graph. In our implementation, we only create a node in the dual graph, if a between-edge exists in the primal graph. This means that if we have an edge (u, v) in the primal graph where v does not have a direct successor, then the dual graph node (u, v) will not be represented in our implementation. The core of the issue is how the road network is modelled (by GraphHopper), as it is possible that the aforementioned issue occurs when performing this operation with our particular road network. A concrete example of such a scenario is exhibited on **Figure 4.2**, where the edge *(A,B)* is not part of any between-edge. The edge *(D,C)* is part of the between-edge *((D,C),(C,B))*, and is therefore valid.

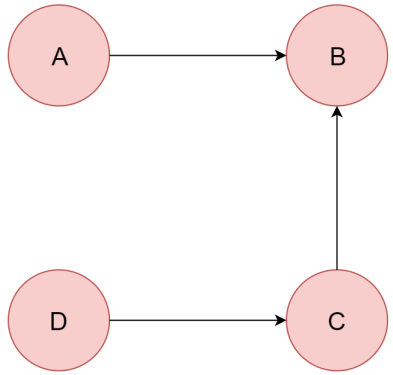


Figure 4.2: An example of a primal graph, where the dual graph construction will cause an issue, as *(A,B)* is not part of any between-edge.

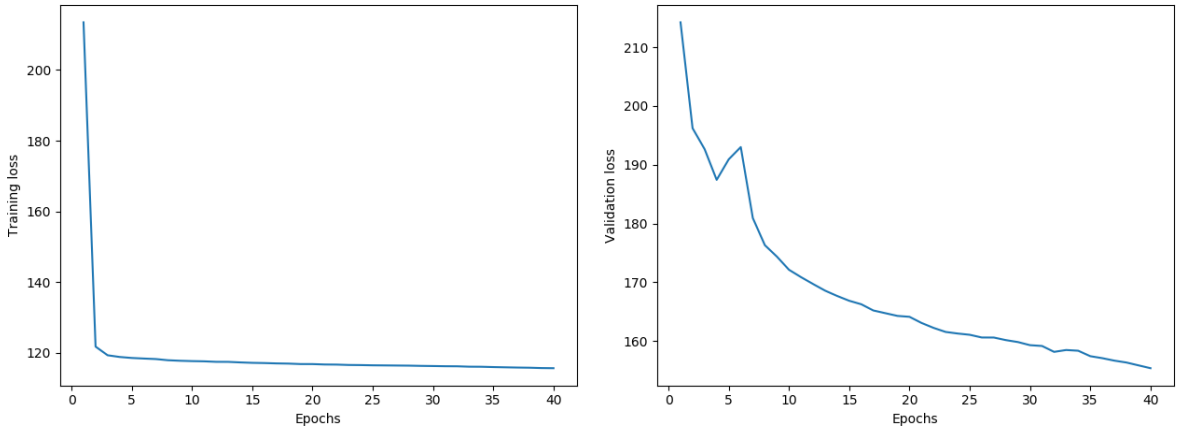
To solve this issue, we removed the conflicting edges and nodes, in this case, the node A, and the edge (A, B), from the primal graph, as they are not present in the dual graph. We deem this a appropriate solution as unreachable nodes are unlikely to be present in a road network, and the implementation of the dual graph in the RFN tutorial is created with this assumption. As a consequence of this, not all edges in the primal graph will necessarily get a predicted attribute.

Results of Training

During the training phase, we record the total loss induced by the training dataset, validation set, and test set, and use this information as our evaluation metric to measure the performance of the RFN model throughout the epochs. The model is trained for 40 epochs. The loss for the training dataset is plotted on **Figure 4.3a**, while the loss for the validation dataset is plotted on **Figure 4.3b**. The training loss are in the interval of 115.68 and 213.47, while the validation loss is the interval of 155.38 and 214.25.

Considering the training and validation loss are both decreasing as seen on the figures, the model seems to be underfit, meaning that the model has not captured extensive patterns in the training set yet. This is especially backed by the development of the validation loss, which seems to keep improving significantly throughout the epochs. When comparing the performance of the model

in terms training and validation loss from the initial iteration to the last, we observe a significant decrease in loss. The decrease is more apparent for the training loss compared to the validation loss. Besides the training and validation loss, we have also recorded the loss in the test set, which is at 164.35, which is not significantly different from the validation loss. We deem the model inadequate as the model is underfit, which will require more training.



(a) Plot of training loss and epochs

(b) Plot of validation loss and epochs

Figure 4.3: The plots show results of the RFN training phase.

4.2.3 Deployment of the Two Services

Both services are put into production on the Kubernetes cluster and is shown on the aSTEP website. External communication between them has also been established, just as explained in the component design in **Section 3.6.1**.

```
predictions = get_input_from_other_service(
              "https://astep-2020-attribute-prediction.astep-dev.cs.aau.dk/",
              {"nodes": nodes.to_json(orient='records'),
               "edges": edges.to_json(orient='records')})
```

Code Snippet 4.2

As mentioned, the communication happens through the input/output channels and the */data* endpoints. **Code Snippet 4.2** above describes how the GA service sends POST requests to the AP service's */data* endpoint and more specifically, the two input channels *"nodes"* and *"edges"*. In the request, the GA service will send two dataframes, one containing the nodes and the other edges, that are queried from the *mapdata* database in the GA's *subgraph fetcher*[1]. In return, the GA service will receive a JSON dictionary of edges and their predicted speed limit as shown in **Code Snippet 4.3** below.

[1]More documentation about the subgraph fetcher can be found in **Appendix C**

```
{
"edges": {
    "(1049046,715859)" : 43.5576,
    "(579606,640350)" : 76.459,
    ...
    }
}
```

Code Snippet 4.3: Each edge is represented by the ID of its base and adjacent node as key and the predicted speed limit as value.

For each edge, we fetch its geometry points from the *mapdata* database, which is used in the construction of a list of *LineString OL.features*. This list is provided to the */render* endpoint, that will display them on a *map-geo* chart. An example is shown in **Figure 4.4**, that basically encapsulates the results of the entire implementation of the services for Task 2.

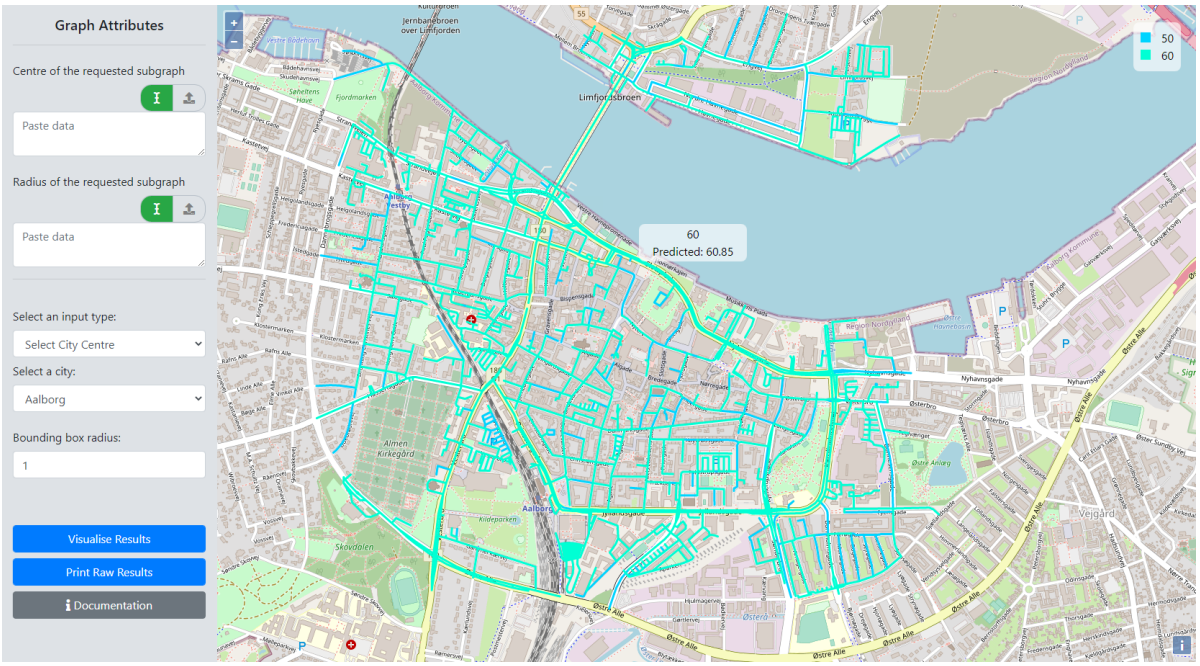


Figure 4.4: A look at the final GUI of the GA service from the aSTEP website. It shows predictions on a graph with centre in Aalborg with a radius of one km.

4.3 Collaboration

Sprint four's collaboration with the other groups were mostly centred around adjusting the communication of the services, as they were being finalised in this sprint. Being able to communicate what kind of and in what format the data needs to be for a service to properly function was of paramount importance. The communication we had with the other groups were exclusively on the aSTEP Dis-

cord server, where discussions were held. As a result, the format of the input and output for some micro-services was adjusted to accommodate different needs. Because the routing group as a whole is dependant on each other's services cyclically, we had to ensure that group 6 could quickly access all important information of our MM service. In turn, this would allow for group 1 to perform its routing using the historical data that group 6's micro-service produces.

4.4 Sprint Four Review

The fourth and final sprint started at a hectic point in the development of the GA and AP service. Remaining from the previous sprint was three partially fulfilled requirements, which we have worked on in tandem with finalising the implementation, as well as training and evaluating the RFN model.

Backlog Index	Completed	Status
S4.1	✓	This was achieved after fixing the issue regarding the dual graph construction. Once all training graphs were embedded with node2vec, they were sequentially trained on the RFN. After 40 epochs, the model achieved a 155.38 validation loss, and a test loss of 164.35.
S4.2	✓	With the dual graph construction issue resolved, it is possible to predict the missing attribute of edges on any road network.
S4.3	✓	Following the *2020v1* service interface's protocol of data parsing, we have between the GA and AP micro-service ensured correct correspondence between their respective input and output channels. Each channel's "type" and the outer key of the JSON dictionary being provided to them coincides.
S4.4	✓	Before sending the nodes and edges to the AP service, we construct each edge by taking the base node, geometry of the road, and the adjacent node. Each of these edges use their base node and adjacent node as a key to their GPS-points. The AP service returns a dictionary of predictions with the base and adjacent node as key. These two dictionary are then matched and plotted.
S4.5	✓	The labelled data available concerns 300,000 edges, which is divided into a validation and training set. The validation set is a weakly connected graph of 30,000 edges, corresponding to 10% of the total labelled data available. The rest of the data is sub-graphed into smaller graphs, ranging from 30,000 edges and below.
S4.6	⊗	Due to time constraints it was not possible to train a wide variety of models, with different amount of layers, units, and so on. That is why we settled on a model inspired by the architecture proposed in Jepsen et al.[37].

Table 4.3: Review of the sprint backlog for Sprint four. The completed items are denoted by ✓, while the incomplete ones are marked with O. Partially completed items will be denoted by ⊗.

As mentioned in the introduction (**Section 0**), we had hoped to reserve this sprint for testing & evaluation. We have performed manual system and integration tests throughout the development phase. However, we have not managed to implement any automated tests (unit tests). Although we have not extensively tested the RFN model architecture, we have evaluated the one architecture that we have proposed. In the end, we have deployed two functional services.

Sprint Retrospective

The new working guidelines we applied in Sprint three have been successful. We have continued with daily meetings to maintain good communication internally in the group. Although we had to focus most of our efforts this sprint to finishing the services, and we did not fully complete all sprint backlog items, we are still satisfied with the results that have been produced.

5 Evaluation

We have now reached the final part of the report, where we will summarise our work and reflect upon it. To this, we will evaluate the overall project completeness by assessing the project backlog, that has been refined throughout the four sprints. Also, we will make a discussion where we will touch some of the challenges we faced and our solutions hereto.

5.1 Evaluation of Product Backlog

To determine whether the problem definition of this project is solved, we need to evaluate on the product backlog. In order to clearly illustrate the progression on the product backlog item, we present **Table 5.1**, where each entry specifies a product backlog item along with a list of sprint goals that has been fulfilled in order to deem the product backlog item complete.

Product Backlog	Sprint Goals
P1: Implement a map matching micro-service to solve Task 1 on the aSTEP platform.	• Understand the structure of the aSTEP project (Sprint 1) • Understand the underlying theory of map matching (Sprint 1) • Implement a MM micro-service for aSTEP that is compatible with Task 1. (Sprint 1) • Complete the MM service (Sprint 2) • Create a common road network to be used by groups for future tasks. (Sprint 2) • Ensure a 1-to-1 correspondence between the original GPS-points and the map matched GPS-points. (Sprint 3) • Improve map matching by adjusting the standard deviation (error-σ) and the transition probability (β). (Sprint 3)
P2: Implement a graph attribute prediction micro-service to solve Task 2 on the aSTEP platform.	• Create a common road network to be use by groups for future tasks. (Sprint 2) • Understand graph embedding techniques. (Sprint 2) • Understand classification techniques for edge prediction (Sprint 2) • Implement endpoints for the AP service to communicate with the aSTEP platform (Sprint 3) • With node2vec, embed a graph with the speed limit as its weights, and use this to get node representations. (Sprint 3) • With the embedded graph, train a RFN. (Sprint 3) • Predict the speed limit on the unlabelled edges of a graph. (Sprint 3) • By an investigation, document the theory of node2vec and RFN. (Sprint 3) • Make a design plan for both the attribute prediction service's UI and architecture. (Sprint 3) • Finalise the GA and AP micro-services. (Sprint 4) • Test and evaluate the results of our services. (Sprint 4)
P3: Ensure forward compatibility for both developed micro-services.	• Implement a MM micro-service for aSTEP that is compatible with Task 1. (Sprint 1) • Implement endpoints for the AP service to communicate with the aSTEP platform (Sprint 3)
P4: Perform a risk analysis and determine risk management.	• Assess a risk management plan (Sprint 1)

Table 5.1: Overview of the progression on the product backlog tasks throughout the sprints.

For **P1**, the implementation of the MM micro-service was mostly completed in the first two sprints. In the first sprint, we attained a basic understanding of how micro-services are implemented and deployed. By the principles of the established service interface on aSTEP, we created the framework of the MM service. Furthermore, the underlying theory behind map matching was understood, and the open source library, GraphHopper, was found to perform the map matching. In the second sprint, the MM service was completed. However, due to accuracy issues, we needed to improve our use of the GraphHopper algorithm in Sprint three.

To solve **P2**, an initial task of storing a road network to be used by all routing groups, and service as a dataset to train the model of the AP service, had to be completed. Hereto, we created a database containing the graph representation of Denmark, that was created by GraphHopper. Furthermore, graph embedding techniques and prediction techniques were also explored in the second sprint, and we chose to implement node2vec for graph embedding and RFN for predictions.
In the third sprint, we elaborated on our chosen algorithms from the second sprint. Additionally, we prepared a component design, that involved the implementation of the GA and AP services. We implemented the framework of the two micro-services during this sprint. The implementation of the AP and GA service was finished in Sprint four. We deployed the them on the aSTEP platform and trained our RFN, to lastly evaluate its results.

P3 was solved by implementing the three services following the *2020v1* service interface guidelines. Together with this, we also focused our design of the GA and AP micro-services on modularity, to let them serve as tools for future projects.

For **P4**, we conducted a risk analysis and management plan at the beginning of the project. The precautions proved to be useful as the RMMM plan was used when certain risks occurred. Risk 5 (Personnel not collaborating) from **Table 1.3**, did not reach the level of exposure to be included in the RMMM plan, but became a reality in Sprint 3 and resulted in the removal of a group member. To Risk 14 (Coordination with other groups), we have followed the RMMM plan, and throughout the sprints we have documented the collaboration with the other groups and its influence on our work.

With this evaluation, we have presented the work completed in the project. To deem the tasks of the project backlog fulfilled, we first have to discuss difficulties and results during development.

5.2 Discussion

During the development of the project certain challenges and considerations have been met. Each of these adheres to a different subject, which we have placed under the topics throughout the section.

5.2.1 Development Using Scrum

Regarding the report, we had the challenge of documenting our project in parallel with the progress on the tasks we made during each sprint, while still making the report's narrative understandable. However, this approach might have added additional complexity for the reader when understanding the implementation of each service, as it has been documented throughout different chapters.

Furthermore, we also lacked a key role in the Scrum environment, namely an apparent product owner. This meant that we had to come up with and prioritise items in the product backlog based on given user stories, instead of a product owner specifying the product backlog items. As a consequence of not having a product owner, it also meant that we had to make decisions that, in a traditional Scrum environment, should not be made by the development team. In general, we lacked clearness of an overall project goal. This meant we might have inadvertently made decisions that was not aligned with the project's intended direction.

5.2.2 The Routing Group Tasks

As described in **Section 0**, we were at the beginning of the semester assigned an additional task to be completed during the project. The purpose of the additional task (the map matching) was to get acquainted with the development process of the aSTEP platform and that this should be the focus during the first sprint.

Task 1 definitely made us familiar with the aSTEP architecture and development process, as we were working closely together with the other routing groups to complete the task of analysing trajectories. Collaboration was a focal point during this process.

However, this task proved to be far more extensive than anticipated and planned for. The subtask of creating a micro-service for map matching could have been an independent task for an entire project. As a result, we were left with less time to implement and fine tune our primary task of predicting missing attributes in a road network. Despite having completed the majority of our backlog items, we still do not think that our solution to Task 2 is completed to the extent that we had envisioned.

The intention with the initial task was good, however, the process of getting familiar with aSTEP and the development process would also have come through the completion of a single task, similar to how the time series super group did it.

Super Group Collaboration

Collaborating in a super group was a new experience for us. We were dependent on other group's plans and needs when making design choices, and we had to had to facilitate our own proposals as well. Many informal meetings were held weekly, while the university was still open prior to the COVID-19 shutdown. The convenience of having a face to face communication when facilitating the common direction for the project disappeared after the lock down. We continued with the communication on online medias such as Discord, but this approach never proved to be as effective as face to face communication.

Working on a larger collaborative project, and as a super group was an educative experience. It proved how challenging it can be to work in tandem with other groups to reach a common goal. It also taught us, that having a common understanding of the design plan is vital to ensure a smooth development phase. We experienced this during the development of the MM service, as we had to correct key components, which was the result of a lack of communication.

5.2.3 Usability Testing

When a service that is designed to be used by other people, like both our MM and GA service are, it is important to ensure that they allow the users to experience their full potential. In the design phase of each task we aimed to find the best and most intuitive solution to display our results, and especially for the GA service, tried to give provide the user with different options, to change the results being shown on the map chart.

Whether we have fulfilled this goal or not, is yet to be determined, as we haven't conducted any usability tests during our project. Due to time constraints, we decided not to prioritise usability testing in our project.

5.2.4 Ground Truth Data for Map Matching

An issue we faced when evaluating the results of the MM service, was the lack of ground truth data. From the trajectories provided by Bring, we can only assume and estimate their spatial correctness. Many factors can have disrupted the observed GPS-points, i.e. connection interference. If we had a way to determine the actual route, it would be possible to choose on an "optimal" Error-σ and transition probability-β for all results. Back in **Section 3.2**, we adjusted these two parameters, however, we can still not prove that these yields the best results, for the above reason.

5.2.5 The Training Data

We described in **Section 4.2.2** that we use the *mapdata* database as our training set for the RFN model. We are aware that this is not the most optimal solution, as data from the same database is also used as input for the model to make predictions on.

As we see it, there is not an obvious solution to this problem. For the RFN to give reliable predictions, it must be trained on data that is representative for the Danish road network. The only data we have that meets this standard is the *mapdata* database. This is also the data we are required to use in order to show the edges on the map chart correctly. Alternatively, we could have trained our RFN on a road network that is not in Denmark. However, we fear this solution will train the model to give predictions that are not representative of the Danish road system. Other countries might have different infrastructure and speed regulations, which could potentially make the predictions worse.

The Labelled Data

Another concern is the labelled data that we use for both training, testing, and validation. We have chosen all edges with a defined "max speed" value in order to maximise the amount of training data to achieve the best results. However, as seen in **Figure 5.1**, the distribution of the labelled data shows an excessive amount of edges labelled with "50 km/h". We are concerned that this distribution will cause our model to overfit. Another concern is that the model learns to predict closer to "50 km/h"

than the other values due to the data distribution, which is not ideal. The figure also shows a low representation of "high speed" labelled data i.e. from motorways. It is therefore likely, that our model will not be able to predict this type of road correctly, as well as other low-represented road types.

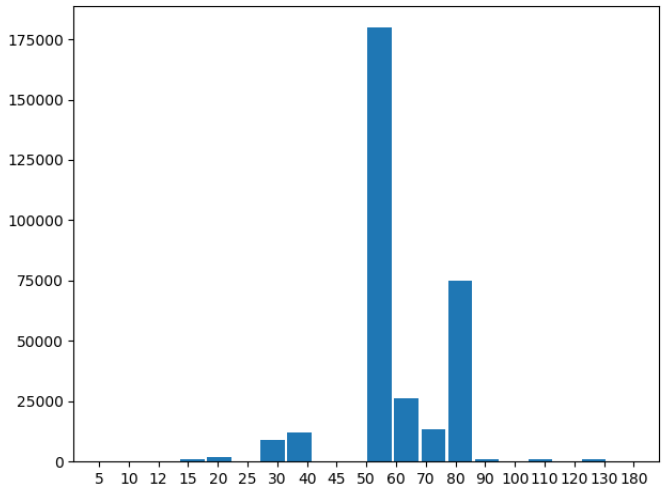


Figure 5.1: The distribution of the labelled dataset, where the majority of the edges contain the label "50".

We should in general have spend more time on analysing our data beforehand, as we have discovered, that our dataset has registered speed limits for railways (180 km/h) as well. These are not representative for the road speed limits and will therefore affect the RFN training negatively.

Yet another concern with the data, is that the registered speed limits often deviates from the conventional speed limits. For instance, the Road Directorate or municipalities often regulates the speed limit in areas with schools, kindergartens, and day nurseries. These "characteristics" are not modelled in the road network, and we can therefore not assume the model will learn these.

5.2.6 The Mapdata Database

One could argue that creating a database from a static version of the OSM of Denmark is a short-lived solution that lacks modularity. Since we ran into issues with the old database (DB1), used by aSTEP-2018 and 2019, during our project, so next years project groups might run into the same issues with our newly created database (*mapdata* database).

Instead of using a static version of the map, we could have used the public OSM database. This would have allowed us to work with the newest road network data at all times, and as such, prevent us from running into the problem of deprecated traffic information. On the other hand, API access is usually restricted to a certain number of requests per day, which is not practical if many services are connected to the same database, as it is in our case.

Another problem is that many services are dependant on the edge representation of the road graph,

which does not exist in the default OSM data. As explained in **Section 2.3**, we construct this representation by running GraphHopper on the osm file. This would not be possible to do as every time a service needs the road network, we would need to query the public OSM database and afterwards run GraphHopper on the OSM data, which is costly. Therefore, we decided against using the OSM database and instead create our own, despite the fact that it might become deprecated as well at some point. However, it is relatively easy to update the database using the code for creating the initial DB2, which is available for the next year's students. All it requires is an updated version of the OSM file for Denmark.

5.2.7 Service Run Time Optimisation

After deploying the GA and AP service to production on the aSTEP server, we encountered several issues that we did not experience locally during development and testing. These issues all had the common denominator that limitations on the Kubernetes cluster caused them. The first issue regarded data passing between the two services, which would cause an error when handling larger graphs. Due to undescriptive error messages, solving this issue took longer than expected. Together with the server group, we managed to find the real issue and a solution to ensure the data passed correctly between the services. If the AP service received graphs with >1500 edges, the request would be rejected as it exceeded 2MB (the default maximum request body size limit). We increased this limit for the two services, but encountered another problem afterwards. The service would take too long to process the graph and would thereby cause the request to timeout. The solution was to increase the timeout limit, which allows request to be serviced for longer before being timed out by the server. This meant that they could now function the way we envisioned in our design plan.

Admittedly, the solution allowed our services to to communicate correctly with each other, however, the run time of the service had multiplied, going from a local environment to production. To give a picture of the problem's magnitude, a user has to wait approximately 16 minutes after pressing the "Visualise Results" button on the GA service (with Aalborg selected as the centre point and a 5km radius). Compared to a local environment, where this process took around two minutes, the latter result is by far ideal, and the response times we now have in production are unsatisfactory.

After troubleshooting, we identified the GA service itself to cause the extended runtime. When it constructs each edge that has to plotted, it has to iterate over three DataFrames to find all coordinates on the edge. The amount of look ups in each DataFrame becomes exponentially larger with the amount of edges in the graph. We attempted to optimise this process, but never found an optimal solution.

6 Conclusion

Between the various analytic modules in the aSTEP framework, aSTEP-2020 requested solutions to both the spatial, temporal, and spatio-temporal module. We chose to participate in the spatial and spatio-temporal collaboration group also known as the routing super group. As part of this collaboration group, we set out to solve tasks regarding routing and road network analysis. Related to this, we have contributed by solving two tasks:

1. Map match trajectories provided by Bring
2. Predict missing road network attributes

Our proposed solution to the first task utilises GraphHopper to create an offline map matching micro-service. This micro-service is able to provide a user or other micro-services with GPS-points that are map matched, which is the operation of snapping GPS-points to a digital map. The GPS-points are visualised to the user on an interactive map, showing both the original trajectory and the map matched one. Other requesting services are given all information related to the map matched trajectory. We improved the micro-service by adjusting the Error-σ to 43 and transition probability β to 2. The development of this service was essential to solve subsequent routing tasks.

For the second task, our proposed solution utilises a Relational Fusion Network to predict the speed limits of edges in a road network. Our solution consists of two separate micro-services, each serving its specific purpose. The Attribute Prediction micro-service must process all input graphs by embedding the graph, creating a dual graph, embedding the dual graph, and analysing its between-edge attributes, to finally be able to perform its prediction. For the Attribute Prediction micro-service, we have trained a model with a training loss at 115.68, validation loss at 155.38, and a test loss at 164.35. This model is underfit as the model keeps learning and the loss decreases as the epochs increases. Therefore, the model is not optimal for speed limit prediction on road networks and needs more training. The Graph Attribute micro-service is responsible for the database interaction and visualising the results of the Attribute Prediction micro-service.

In short, we have developed a solution for map matching, and while the micro-service for predicting speed limits on road networks is not accurate to a satisfactory degree, this can be attributed to a lack of training.

Besides having worked on solving the two aforementioned tasks, we have also contributed to the com-

mon aSTEP infrastructure by implementing the new map database, that contains the road network used by all routing groups. To the UserInterface micro-service, we have also made improvements to the map-geo chart.

For each task we established a set of backlog items, to ensure a sufficient solution was achieved. With all backlog items completed to a degree, where we can consider the overall product backlog fulfilled. By these measures, we have contributed to the spatio-temporal data analytics platform with three additional tools.

7 | Future Works

In the future works of the project, we reflect upon improvements, and additions to our services, that we did not implement due to various constraints.

7.1 Improvements to the MM Service

Initially when designing and implementing the MM service, we were concerned that tuning the Error-σ and transition probability-β, would be an extensive procedure. However, after testing, we discovered that the correct values purely depended on the trajectory, that we tried to map match. When certain values would be perfect for one trajectory, they would make the other be off course. During the implementation and later in the improvement of the service **Section 3.2**, we settled on values that performed well on average.

To improve the service, we think it would be sensible to allow the user to adjust the Error-σ and transition probability-β. Hereto, we would add additional input fields in the control panel.

7.1.1 Trajectory Input

Despite of being out of the scope of Task 1, we still think it would be ideal to allow the user to provide the MM service with their own trajectories to be map matched. This would also emphasise the idea of implementing each service as a tool, that has a wider application.

To realise this addition, it must be ensured that the input trajectory follows the format defined in the service. This includes the JSON formatting of each GPS-point. This would be implemented by an additional input field.

7.2 Expanding the Predictive Capabilities of the AP Service

The RFN in the AP service is currently focused on predicting the speed limit of road networks. Alas, it is only capable of predicting a single attribute. We see two options in order to expand the service's prediction capability.

One option would be to include multiple features to predict on, i.e. including all features presented in **Section 3.6.3** and **3.6.4**. Additional features could also be provided by other services, that works with road analysis. With this approach the model would take multiple features into consideration for when predicting.

The other option would be to separate each feature and have a uniquely trained model for each of them. Thereby, when a user or external service interacts with the AP service, they would have to specify the feature they wish to get predictions for.

7.3 Performance Feedback

We have in **Section 4.2.2** presented our results of the RFN model's performance. It would also be ideal to display these results in our service. The two services are aimed to be an analytic tool, whereas the a measurement of the performance are important. Similar to the existing Attribute Completion service, we could include a way to show these results to the user.

Bibliography

[1] Nicolaus Henke et al. *The Age of Analytics: Competing in a Data-Driven World*. McKinsey&Company, 2016.

[2] Ken Schwaber and Jeff Sutherland. *The Scrum Guide™*. URL: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

[3] David Benyon. *Designing Interactive Systems*. 3. edition. Pearson, 2018, p. 140.

[4] aSTEP-2020. *Server Architecture*. URL: https://wiki.astep-dev.cs.aau.dk/server.

[5] Docker. *Docker overview*. URL: https://docs.docker.com/engine/docker-overview/.

[6] aSTEP-2020. *RFC 0009 - Kubernetes*. URL: https://wiki.astep-dev.cs.aau.dk/rfc/0009.

[7] aSTEP-2019. *0005 - UI Style Guide*. URL: https://wiki.astep-dev.cs.aau.dk/rfc/0005.

[8] aSTEP-2020. *0006 - Definition of chart types.* URL: https://wiki.astep-dev.cs.aau.dk/rfc/0006.

[9] aSTEP-2019. *RFC 0004 - Interface of Microservices*. URL: https://wiki.astep-dev.cs.aau.dk/rfc/0004.

[10] aSTEP-2020. *0007 - Single Page Application*. URL: https://wiki.astep-dev.cs.aau.dk/rfc/0007.

[11] Network Working Group. *Media types (MIME)*. URL: https://tools.ietf.org/html/rfc2046.

[12] Network Working Group. *IETF RFC 4627*. URL: https://www.ietf.org/rfc/rfc4627.txt.

[13] W3C. *HTML Form Content Types*. URL: https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.

[14] aSTEP-2020. *Service Interface*. URL: https://wiki.astep-dev.cs.aau.dk/service-interface#data.

[15] IANA. *The IANA Media Types*. URL: https://www.iana.org/assignments/media-types/media-types.xhtml.

[16] aSTEP-2020. *RFC 0015 - Interface of Microservices - 2020v1*. URL: https://wiki.astep-dev.cs.aau.dk/rfc/0015.

[17] Stern Robert and José Arias. "Review of Risk Management Methods". In: *Business Intelligence Journal* (Jan. 2011).

[18] Ian Sommerville. *Software Engineering 10th Edition*. Pearson, 2016.

[19] Roger S. Pressman. *Software Engineering*. 7. edition. McGraw-Hill Science/Engineering/Math, 2004, p. 793.

[20] C. Y. Goh et al. "Online map-matching based on Hidden Markov model for real-time traffic sensing applications". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. 2012, pp. 776–781.

[21] Paul Newson and John Krumm. "Hidden Markov Map Matching Through Noise and Sparseness". In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2009).

[22] Andrew Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* vol. 13.2 (1967), pp. 260–269.

[23] Graphhopper. *Map matching based on Graphhopper*. URL: https://github.com/graphhopper/map-matching.

[24] Lawrence R. Rabiner. "A tutorial on Hidden Markov Models and selected applications in speech recognition". In: *Proceedings of the IEEE* vol 77 (1989), pp. 257–286.

[25] Wikipedia. *Great-circle distance*. URL: https://en.wikipedia.org/wiki/Great-circle_distance.

[26] Geofabrik. *Geofabrik Download Server*. URL: https://download.geofabrik.de/europe/denmark.html.

[27] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. "A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications". In: (2017). eprint: arXiv:1709.07604.

[28] Palash Goyal and Emilio Ferrara. "Graph Embedding Techniques, Applications, and Performance: A Survey". In: (2017). DOI: 10.1016/j.knosys.2018.03.022. eprint: arXiv:1705.02801.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: (Mar. 2014). URL: https://arxiv.org/pdf/1403.6652.pdf.

[30] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016.

[31] Annamalai Narayanan et al. "graph2vec: Learning Distributed Representations of Graphs". In: (2017). arXiv: 1707.05005.

[32] Alex J. Smola and Bernhard Schölkopf. "A tutorial on support vector regression". In: (2003).

[33] David L. Poole and Alan K. Mackworth. *Artificial Intelligence*. Second Edition. Cambridge Universty Press, 2017.

[34] Balázs Csanád Csáji. "Approximation with Artificial Neural Networks". In: (2001).

[35] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[36] Felix Wu et al. *Simplifying Graph Convolutional Networks*. 2019. arXiv: 1902.07153 [cs.LG].

[37] Tobias Skovgaard Jepsen, Christian S. Jensen, and Thomas Dyhre Nielsen. "Graph Convolutional Networks for Road Networks". In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19)*. 2019.

[38] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: (2013). arXiv: 1310.4546.

[39] Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook*. 2018.

[40] Elior Cohen. *Node2Vec*. https://github.com/eliorc/node2vec. 2018.

[41] Tobias Skovgaard Jepsen. *Relational Fusion Networks*. https://github.com/TobiasSkovgaardJepsen/relational-fusion-networks. 2019.

[42] OpenLayers. *OpenLayers*. URL: https://openlayers.org/.

Appendices

A User Interface Service Legend

We have added a legend to be shown on the *map-geo* chart for the features that are plotted. This was needed as both the MM and GA service have multiple features shown on in the chart area at the same time. It might also be useful for future services that will use the *map-geo* chart. This was implemented during Sprint three.

This addition was made to the existing UserInterface service. This user interface is using the Angular framework, and is built up by multiple components. The charts component and more specifically the map-geo component, in which the legend has been added, is made from the OpenLayers library [42]. The source code to this implementation can be found in the the daisy-git aSTEP-2020 UserInterface repository. Here, the path to the *map-geo* chart is UserInterface/app/src/app/charts/map-geo.

As a brief description, this new legend feature will iterate through the list of features that is provided by a service to the UserInterface service. It will then:

- For each feature add an entry to the *legend-container*. Each feature's name in the legend is dependant on the feature having a declared "*name*" within its "*properties*" in the JSON dict. This is shown in **Code Snippet A.1**.
- Show each feature's chosen colour in the legend right next to the feature's name.
- Make sure that no feature with the exact same colour is shown in the legend twice.

Referring to the wiki's information about the Geographical Geometry (map-geo) chart, the JSON object given to "*features*" is still dependent on OL.js and more specifically its GeoJSON format. However, to show a fitting name for each feature in the legend, one must under "*properties*" define a name with the key "*name*". Otherwise the feature will just be shown with the name "undefined". A full example

of a map-geo JSON object to be interpreted by OpenLayers is shown below:

```
{
    "chart_type": "map-geo"
    "content": {
        "view": [geo-view],
        "featureData": {
            "type": "FeatureCollection",
            "features": [
                {
                    "type": "Feature",
                    "style": {
                        "fill": {color: red, "width": 3},
                        "stroke": {color: red, "width": 3}
                    },
                    "geometry": {
                        "type": "LineString",
                        "coordinates": [listOfCoordinates]
                    },
                    "properties": {
                        "name": "FirstLegendEntry"
                    }
                },
                {
                    "type": "Feature",
                    "style": {
                        "fill": {color: blue, "width": 3},
                        "stroke": {color: blue, "width": 3}
                    },
                    "geometry": {
                        "type": "LineString",
                        "coordinates": [listOfCoordinates]
                    },
                    "properties": {
                        "name": "SecondLegendEntry"
                    }
                }
            ]
        }
    }
}
```

Code Snippet A.1: Example of a JSON object of a map-geo chart with two features.

From the JSON object above, the legend on **Figure A.1** will be shown. The legend-container is always shown in the top-right corner of the chart-area.

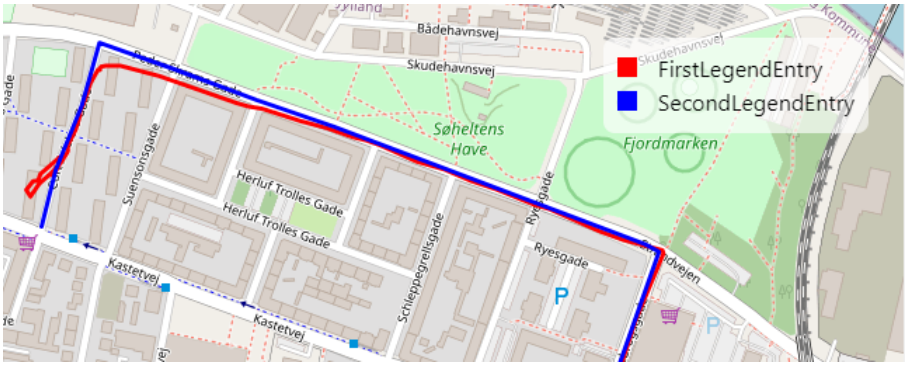


Figure A.1: Legend showing each feature's name and colour.

In addition to this a simple click-listener has also been added to the map-geo chart to let the user click on a feature to show its name.

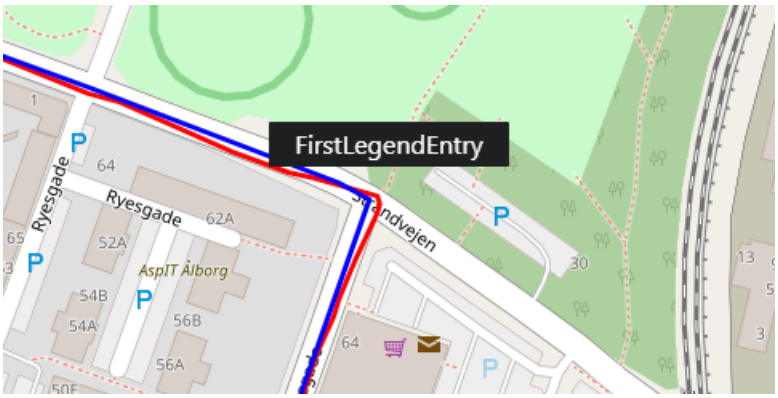


Figure A.2: Feature's name shown when clicked on.

B Graph Attributes Service Endpoints

B.0.1 /info Endpoint

```
{
    "id": "graph_attributes",
    "name": "Graph Attributes",
    "version": "2020v1",
    "category": 1,
    "input": [
        {
            "name": "centre_coordinate",
            "label": "Centre of the requested subgraph",
            "type": "coordinate"
        },
        {
            "name": "radius",
            "label": "Radius of the requested subgraph",
            "type": "radius"
        }
    ],
    "output": [
        {
            "name": "nodes",
            "label": "Nodes of the requested subgraph",
            "type": "nodes"
        },
        {
            "name": "edges",
            "label": "Edges of the requested subgraph",
            "type": "edges"
        }]
}
```

Code Snippet B.1: JSON dictionary as input for */info*

B.0.2 /fields Endpoint

```
{
    "user_fields": [
        {
            "name": "input_type",
            "label": "Select an input type:",
            "default": "None Selected",
            "type": "formset-select",
            "options": [
                {
                    "name": "None Selected",
                    "value": "None Selected",
                    "fields": []
                },
                {
                    "name": "Select City Centre",
                    "value": "Select City Centre",
                    "fields": [
                        {
                            "name": "selected_city",
                            "label": "Select a city:",
                            "default": "Aalborg",
                            "type": "select",
                            "options": [
                                {
                                    "name": "Aalborg",
                                    "value": "Aalborg"
                                },
                                {
                                    "name": "Århus",
                                    "value": "Århus"
                                },
                                {
                                    "name": "København",
                                    "value": "København"
                                }..
                        {
                            "name": "city_box_radius",
                            "label": "Box radius:",
                            "placeholder": "Double-value",
                            "type": "input-number"
                        }
```

Code Snippet B.2: JSON dictionary as input for */fields* Part 1/2

```
                {
                    "name": "Define Bounding Box",
                    "value": "Define Bounding Box",
                    "fields": [
                        {
                            "name": "bounding_box_centre",
                            "label": "Enter the box centre and radius:\n"
                                     "Bounding box centre:",
                            "placeholder": "Latitude, Longitude",
                            "type": "input"
                        },
                        {
                            "name": "bounding_box_radius",
                            "label": "Bounding box radius:",
                            "placeholder": "Double-value",
                            "type": "input-number"
                        }
                    ]
                }
            ]
        }
    ],
    "developer_fields": []
}
```

Code Snippet B.3: JSON dictionary as input for */fields* Part 2/2

C | The GA service's Subgraph Fetcher

We refer it as the "subgraph fetcher" in the report and in the component design in **Figure 3.12**. We use this term to encapsulate all the functionality that, from a given input, returns a graph from the *mapdata* database. A more detailed description will follow.

The subgraph fetcher in the GA service is firstly responsible for receiving the users input to calculate the bounding box from the centre-coordinate and radius. This is done via the function defined in **Code Snippet C.1**.

```
def get_subgraph_from_bounding_box(box_centre, radius):
    lat = float(box_centre[0])
    lon = float(box_centre[1])

    dlat = radius / 111.11
    dlon = dlat / math.cos(math.radians(lat))

    corner_one = [lat - dlat, lon - dlon]
    corner_two = [lat + dlat, lon + dlon]

    return [corner_one, corner_two]
```

Code Snippet C.1: JSON dictionary as input for */info*

The *get_subgraph_from_bounding_box* function returns a list of two coordinates, from which it is possible to form a bounding box. On other words they define the corners of the box. With the corner

coordinates it is possible to define the postgresql query string defined in **Code Snippet C.2**.

```
def get_edges_from_nodes(corners):
    query = "SELECT edge.edge_basenode, edge.edge_adj, edge.edge_id, " \
            "edge.highway, edge.distance, edge.edge_name, " \
            "edge.maxspeed, node.node_id, node.lon, node.lat, " \
            "node2.node_id AS node2id, node2.lon AS node2lon, node2.lat AS node2lat " \
            "FROM node JOIN edge JOIN node AS node2 " \
            "ON edge.edge_basenode = node2.node_id " \
            "ON edge.edge_adj = node.node_id " \
            "WHERE node.lat BETWEEN %s and %s and node.lon BETWEEN %s and %s" % (
                str(corners[0][0]), str(corners[1][0]),
                str(corners[0][1]), str(corners[1][1]))
```

Code Snippet C.2: JSON dictionary as input for */info*

With this query, we get all edges within the calculated bounding box corners by utilising the WHERE operator. However, there is a problem with this exact approach, as all edges with a base node within the the box and adjacency node outside would be cut in half and give us an incomplete graph. Our fix to this was to add the adjacent node afterwards to all edges that had it missing.

D Attribute Predictions Service Endpoints

D.0.1 /info Endpoint

```
{
    "id": "attribute_predicter",
    "name": "Attribute Prediction",
    "version": "2020v1",
    "category": 1,
    "input": [
        {
            'name': 'nodes',
            'label': 'Nodes in the graph',
            'type': 'list'
        },
        {
            'name': 'edges',
            'label': "Edges in the graph",
            'type': 'list'
        }
    ],
    "output": [{
        'name': 'edges',
        'label': 'Labelled edges',
        'type': 'edges'
    }]
}
```

Code Snippet D.1: JSON dictionary as input for */info*

E | Training of the RFN

We present **Code Snippet E.1** which encapsulates how we train our RFN model. As the function that shows the entire training contains details which are already explained, only the relevant and unexplained part will be shown.

```
1  rfn = None
2  trainer = None
3
4  for epoch in range(1, no_epochs + 1):
5      for G in training_dataset:
6          G, G_dual, nodes = get_primal_dual_and_nodes(G, nodes_w_gps)
7          # parameters for RFN
8          edge_indicies_primal = {node: idx for idx, node in enumerate(G.nodes())}
9          node_indices_primal, N_node_primal, N_edge_primal, N_mask_primal =
10             get_primal_representation(G)
11         N_node_dual, N_edge_dual, N_common_node, N_mask_dual =
12             get_dual_representation(G_dual, edge_indicies_primal)
13         node2vec_model = get_node2vec(G)
14
15         # Features of graph
16         edge_features = get_edge_characteristics(edge_dict, G)
17         node_features = mxnet.nd.array(node2vec_model.wv.vectors)
18         between_features = features_for_between_edges(G_dual, nodes)
19
20         if rfn is None:
21             rfn = rfn_model(node_features, edge_features, between_features)
22             trainer = Trainer(rfn.collect_params(), optimizer='adam',
23                               {'learning_rate': 0.01})
24
25         # expected values
26         y = edge_features
27
28         with autograd.record():
29             y_pred = rfn(node_features, edge_features, between_features,
30                          N_node_primal, N_edge_primal, N_mask_primal,
31                          N_node_dual, N_edge_dual, N_common_node, N_mask_dual)
32             loss = loss_function(y_pred, y)
33         loss.backward()
```

Code Snippet E.1: Selected parts of the training is shown

On line 2, the specification concerning the backpropagation is given. This includes the optimization technique, and the learning rate. Afterwards, we iterate through the number of epochs. For each epoch, we iterate through the graphs in the training dataset. Firstly, we get the matrix representation of our primal and dual graphs. This is shown on the lines 8 to 12. On line 13, we create a node2vec model as described in **Section 3.7.2**. Then we get the the feature matrices of the primal and dual graphs on the lines 16 through 18, which is also described in detail in **Section 3.7.2**. In our case, our expected values are the same as the edge feature matrix, which is shown on line 26. On the line 28 to 33, the actual backpropagtion is shown. In the autograd context, the forward propagation step along with the backpropagation of the loss is shown.