MASTER'S THESIS | LUND UNIVERSITY 2018

Algorithmic Approach to Error Correction in Map Data-sets using Conflation Techniques

Linus Röman, Simon Finnman

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-06

Algorithmic Approach to Error Correction in Map Data-sets using Conflation Techniques

Linus Röman
`tpi13lro@student.lu.se`

Simon Finnman
`tfy14sfi@student.lu.se`

June 26, 2018

Master's thesis work carried out at ÅF - Digital Solutions AB.

Supervisors: Thomas Hermansson, `thomas.hermansson@afconsult.com`
Krzysztof Kuchcinski, `krzysztof.kuchcinski@cs.lth.se`

Examiner: Flavius Gruian, `flavius.gruian@cs.lth.se`

Abstract

OpenStreetMap is a crowd-sourced, free and open-source data-set that contains geographical data. As any other data source OpenStreetMap contains a variety of errors, topological, geo-spatial and semantic, to mention a few. This thesis focuses on using algorithms to detect and flag these errors. In particular, focus is put on the concept of conflation, where two data-sets are compared. The first step is to establish a matching between the two data-sets, which is done with an accuracy of above 94% in our test areas. After this match has been established differences between the data-sets can be found. These differences can be used to flag errors which can be forwarded for manual correction. We have looked at attributes such as names, where name dissimilarity can been used to differentiate different types of errors from each other, as well as investigated the correctness of speed limits in the OpenStreetMap.

Keywords: MSc, Matching, Spatial, Conflation, Algorithms, Automation, Geo-Spatial, OSM

Acknowledgements

We would like to thank Thomas Hermansson for his contributions at ÅF to accommodate us during the thesis work. We would like to thank Krzysztof Kuchcinski for his input and guidance with the thesis work and report. We would also like to thank Daniel Palmqvist for his weekly discussions on the more substantive parts of the algorithm production.

Contents

Chapter 1

Introduction

Modern maps provide the foundation for many services we take for granted today. They provide search functionality, efficient routing, location sharing, etc. Modern digital maps need to be precise and correct, which requires frequent updates and maintenance. Many map providers put a lot of effort and money into updating and maintaining their maps, as well as collecting new data. In recent years data sources for **geo-spatial** data have become more abundant, due to e.g. **crowd-sourcing** of geo-spatial data. Geo-spatial data refers to any data that expresses a geometry as coordinates in relation to the earth. One example of a geo-spatial data-set that uses crowd-sourced data is OpenStreetMap (OSM). The quality of such data-sets varies, since it relies on volunteers to collect, correct and maintain the data-set (Thereby crowd-sourced) [10]. A study in Germany found that the OSM data-set has good coverage in urban areas, but it lacks features for rural areas [23]. Much work has been put into trying to automate or semi-automate the maintenance, updating and collecting of geo-spatial data. The task of automation however, is not simple.

One method to automate the handling of geo-spatial data is conflation. The word conflate is derived from latin and means "to blow together" [18]. In the book Geographic Information Systems and Science, Longley proposes the following definition of conflation; “the process of combining geographic information from overlapping sources so as to retain accurate data, minimize redundancy, and reconcile data conflicts.” [6]. There are different types of conflation. In this report we will use the term conflation interchangeably with horizontal conflation, which refers to the matching of **features** in order to eliminate **attribute** value and positional discrepancies in common areas of two data-sets. A feature is a term used to describe a geo-spatial object, i.e. a road, building, wall, park, bridge, etc. Each feature has associated attributes, such as a name, position, etc. Conflation is a very complex field that involves theory from many disciplines, e.g. graph theory, image processing, statistics and pattern recognition [22].

1.1 Purpose and Problem Statement

The master thesis was performed for a client of ÅF – Digital Solutions AB. The client owns and maintains a large set of geographical data which is used to provide a map service. During acquisition and maintenance, the data can become inconsistent, or errors can be introduced. Typically errors occur in the value of attributes of the map data, leading to faulty road names, incorrect speed limits, etc. Spatial and topological errors also occur frequently, especially in rural areas. These kinds of errors are often characterized by roads that do not connect to any intersections, are misplaced, cross through a building/barrier, etc. These types of errors pose potential problems for a so called feature matching step, in which relations between data-sets are established. Since this work focuses on finding incorrect features, this means we specifically focus on matching incorrect data.

The purpose of this work is to attempt to find algorithmic solutions to evaluate the viability of automatically matching two geo-spatial data-sets and the possibility to correct for potential errors in geo-spatial data, using conflation techniques. Specifically we will try to improve OSM data-set road-network quality, by a three step approach. We will first pre-process data-sets in order to work with them in the Atlas framework (See Section 2.4.1). This will be followed by a feature matching step. This matching step will be tested for three matching algorithms. The first matching algorithm (Section 4.2) is a naive approach based on closest distance, the second approach is based on the implementation of another matching algorithm [9] and the third approach is our own matching algorithm based on something we call feature expansion. The matches are then evaluated to find potential errors in an error detection step. We will use open source geo-spatial data collected by danish authorities in order to improve the OSM data-set.

1.1.1 Research Questions

The thesis aims to answer the following questions

1. For what types of errors can correction be automated?
2. How do we detect errors reliably using matching strategies?
3. How well do matching algorithms perform on open source data-sets?
4. How generic is an algorithmic approach to solution suggestion?

1.2 Related Work

The work on conflation has been of large interest for almost 40 years. The earliest work in this field, to our knowledge, is the work of the Bureau of Census and the United States Geological Survey in the 1980's. The focus of their efforts were to consolidate vector data in order to remove geometrical inconsistencies [13].

In 1999 Walter and Fritsch proposed a statistical approach to match two data-sets from different sources of varying detail and attribution. The proposed algorithm matches data by building a **buffer** around a feature, and looking for features that are partly or completely inside the buffer. In conflation the term buffer refers to an area, specifically in relation to a feature. This approach is based on previous work in computer vision. The methodology is broken down into five steps; data pre-processing, computation of potential matches, elimination of unlikely matches, weighting of matches and lastly computation of unique matches. In their work they describe various kinds of matching, and present how their algorithm deals with these scenarios. They further restrict the way the algorithm finds matches by limiting the angles which buffers grow [19]. Our work places emphasis on matching, specifically matching between two disparate data-sets of varying detail and attribution. We take inspiration from Walter and Fritsch in their methodology on matching. Specifically our own algorithm uses a variation of the buffer approach to selectively expand edges in the graph until a match can be established. We will also need to detect errors in the attribute values associated with features, which their work does not cover.

In 2008 Mustière et al. [9] published an article describing a method of matching networks for different detail. Mustière notes that very few articles have attempted matching between networks of different detail. The proposed matcher attempts to match networks in **1:many**, **1:1** and **1:0** relations using geometrical, thematic and topological heuristics. An N:M matching refers to the amount of features matched in either data-set, N features are matched to M features. The matcher implementation is called NetMatcher, and matching is done in four steps. First candidate nodes are found in data-set A, corresponding to nodes in data-set B (this step is called pre-matching of nodes). This is followed by finding candidate edges in data-set A for data-set B (called pre-matching of edges). Then unique node matches are selected by analyzing the topology of the edges incident on the pre-matched nodes. Unique matches for edges are then found by evaluating a shortest path heuristic between matched nodes [9]. Our work is similar in the detail of the input data-sets. One of our data-sets is more detailed than the other. Our report also features an implementation of the NetMatcher. The primary difference in our work compared to that of Mustière is the addition of an error detection and correction step, in order to flag inconsistencies. Moreover our approach is applied to improve the quality of a crowd-sourced data-set using a lower quality open source data-set. In addition three algorithms are presented in our work.

In 2016 Zhang et al. [22] published an article describing a real world application of conflation to add pedestrian walkways to a road network, in order to better support multi-modal navigation (i.e. Biking and Walking). Zhang notes that, "Quite often, one data-set may be superior to other datasets in one, but not all aspects.", hinting that many good data-sets can be improved via conflation using lower quality data. Zhang also writes "Second, most of the former researches have primarily described the general strategy and basic ideas for the task of data conflation. The concrete approaches as well as their automatic conflation results are seldom discussed and evaluated.", pointing out that there is a need to evaluate the real world performance of conflation algorithms [22]. In their work, they describe a five step approach to conflate their two data-sets, matching, identification of pedestrian walkways, transformation of data-sets to eliminate geometric inconsistency, remodeling of the conflated data-set and finally error checking and correction [22]. We have a similar step-wise approach to our work, however the work of Zhang et al. has some transformation steps which are necessary when replacing or moving geo-spatial features from one data-set to another. However our approach does not focus on adding or moving features between data-sets, rather flagging deviations found in established matches. Also our data-sets are of inferior quality to the data-sets used in Zhang's work.

In 2017 Du, H. et al. [3] published an article detailing matching between OpenStreetMap (OSM) data and an authoritative data-source, where features are matched based on location information as well as names and types. The method is a general approach and was tested for matching between OSM and authoritative data in Great Brittain and France. They note that the "crowd-sourced data" of OSM has a high variability in quality due to being collected by volunteering citizens rather than experts. They note that to enrich authoritative data, it is essential to match corresponding features correctly. They also point out the lack of a defined taxonomy in the OSM data-set. Their approach to matching consists of building large buffers to obtain candidates and then filtering out the most suitable candidate. The validation of the approach is to manually classify features as True positive, False positive, False negative, and True negative. They manage a Recall of 0.85 and a Precision of 0.89 as their best result [3]. This work is quite similar to ours, however only covers the matching process with one algorithm. In our work, we present three different algorithms for matching, as well as automated error detection. We, like this article, have manually matched two areas of our data-sets, to provide results of the matching algorithms.

1.3 Disposition

We will begin by describing the quality and content of our data-sets, as well as showing common errors and describing the frameworks we use in chapter 2. Then we will describe the pre-processing steps required in order for our data to interface with our frameworks in chapter 3. In chapter 4 we discuss three different approaches to matching algorithms, which include one naive approach, one approach from literature as well as our own algorithm. In chapter 5 we explain error detection and correction, and how we deal with it in this report. Chapter 6 is our results, including results on all three matchers, as well as statistics for error detection and correction. Chapter 7 is a discussion of the results based on the approach, followed by a conclusion and discussion on future work in chapter 8 & 9.

1.4 Contributions

The responsibility of the different parts of the implementation and report can be seen in the two sections below.

The implementation in general was a collaboration between us both. No major decisions were made individually. Implementation details were discussed collectively beforehand. Weekly strategic meetings were held, where the current status of the project was verified and the next week was planned. Code implemented by Simon was reviewed by Linus and vice versa. The responsibilities of the implementation are seen in section 1.4.1.

The report was for the most part written in collaboration, the content of the report was discussed vigorously. The responsibilities of the report are seen in section 1.4.2.

1.4.1 Implementation

The implementation was divided as follows:

Pipeline Framework Simon Finnman (A framework to modularize the different steps of conflation, making swaps of algorithms easier)

Pre-processing Simon Finnman

Naive Matching Algorithm Linus Röman

NetMatcher Linus Röman

Expansive Edge Matcher Simon Finnman

Error Detection and Correction Simon Finnman

Evalutation Linus Röman

Results Linus Röman

1.4.2 Report

The report was divided as follows:

Introduction Simon Finnman

Data-sets Simon Finnman

Pre-Processing Simon Finnman

Matching Linus Röman and Simon Finnman

Error Detection Simon Finnman

Results Linus Röman

Discussion Linus Röman and Simon Finnman

Conclusions Linus Röman and Simon Finnman

Future Work Simon Finnman

Bibliography Simon Finnman

Chapter 2

Data-sets

In this chapter we will introduce the data-sets that were used. We introduce two data-sets, OSM and OpenDataDK in sections 2.1 and 2.2. In the subsections of these sections we discuss data quality and representations.

Two features that are matched should correspond to the same real world entity, therefore a complete and accurate matching requires that the two input data-sets spatially intersect. Moreover, in order to detect errors and correct some faulty values of attributes, there should also be some intersection of attributes of features. This was kept in mind when we selected our two data-sets. Usually one data-set will be of superior quality to the other, this data-set will be referred to as the **reference** data-set, while the inferior data-set will be called the **appended** data-set. These names are taken from the naming conventions in [22]. This however does not imply that the data of the appended data-set cannot be used to improve the reference data-set. Quite often the case is that the two data-sets include different kinds of attributes, or that the appended data-set has superior quality for the values of some attributes.

The geographical region chosen for the work was Denmark, since there was easily accessible data for large portions of Denmark. Since OSM data is global, areas were chosen for the appended data-set, and then OSM was cut in order to fit that data spatially. The chosen data-sets in Denmark are chosen to have some overlapping attribution with the OSM data-set.

2.1 Reference Data-set

OSM was chosen as the reference data-set. OSM is an open-source global geo-spatial data-set which crowd-sources data from a large number of volunteers.

2.1.1 OSM Data Quality

The quality of OSM data has been an area of extensive research since it was released 2004. Many authoritative users have concerns regarding the credibility of the OSM data, due to the belief that OSM data is sourced by amateurs. Assessments have shown that OSM data quality varies wildly, and is heavily dependent on the population size in the geographical region. A study by Barron et al. used the following metrics to determine OSM quality[2]

Completeness Describes how complete a data-set is.

Logical Consistency Declares the accuracy of relations manifested in the data-set.

Positional Accuracy Describes the relative and absolute accuracy of coordinates.

Temporal Accuracy Describes the accuracy of the history of the data-set

Thematic Accuracy Describes the accuracy of the attribute values.

The study found that OSM has high positional and thematic accuracy for features in and around urban areas, due to the high amount of contributors, however the quality of rural areas are often of a lower level.

2.1.2 OSM Data Representation

OSM data is represented in XML format. All coordinates in OSM are represented in WGS84 coordinate reference system which is the standard coordinate system for GPS. It contains three blocks of elements: **nodes**, **ways** and **relations**. A node can describe either a point along a line, or serve as a standalone feature, but can generally be thought of as a vertex in a graph. A way describes an edge in a graph and connects two nodes via a series of points that describe the geometry of that edge. Relations describe relational metadata between elements. All of the three elements contain a key-value store, containing attribute data describing the element. The OSM wiki defines guidelines for data-collection, as well as standards for naming tags.

In our work we will primarily focus on road-network data, thus nodes are seen as vertices, and edges are seen as roads connecting vertices. Roads express attributes, whilst nodes denote some break points along a road e.g. crossings. Since a road can be broken by several nodes, each segment of that road between the nodes is refered to as **segments**.

2.1.3 OSM Data Acquisition

OSM data can be downloaded from GeoFabrik.de, which provides up to date OSM data for large regions. The data-set that was downloaded contains data for all of Denmark, with a file size of 247 MB.

2.2 Appended Data-set

OpenDataDK was chosen to be the appended data set. This is an open data-set sourced by the Danish government.

2.2.1 OpenDataDK Data Quality

Since there is no published work evaluating the quality of OpenDataDK, we will briefly account for the quality of OpenDataDK. Since data is collected by each municipality independently, the data quality varies depending on municipality. The largest observed variations are positional accuracy, well defined intersections and amount of defined attributes.

Example - Ballerup

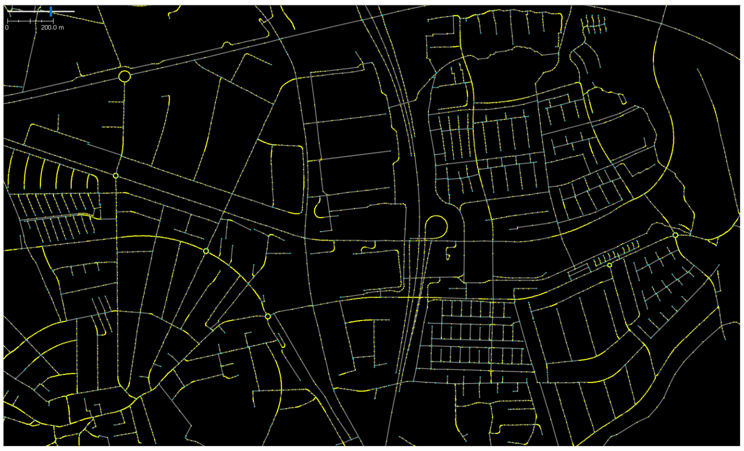


Figure 2.1: Large scale view of the ballerup municipality data-set. On this scale the data-set looks promising, showing fine detail and good coverage.

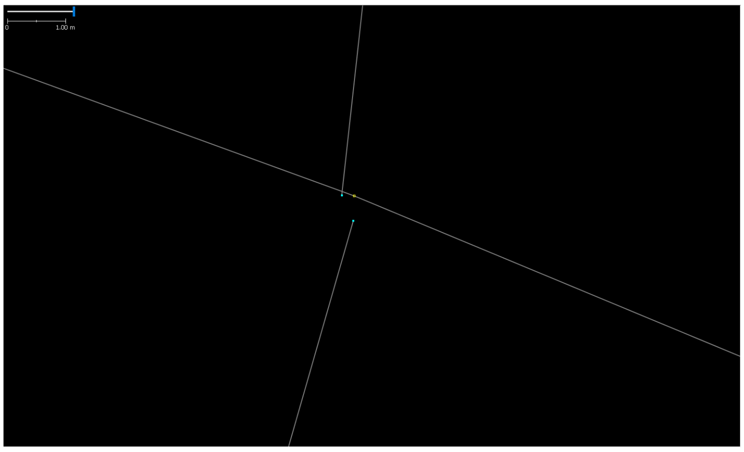


Figure 2.2: Small scale view of the ballerup municipality data-set. This scale reveals a large flaw in the data-set, the intersection is not connected.

Figure 2.2 shows a typical flaw in intersections of the OpenDataDK data-set. The entire figure is 10x6 meters in size and the disconnected nodes are within 50 cm of the intersection node. The vast majority of the intersections in figure 2.1 exhibit the same flaw as seen in figure 2.2. This type of data-set is problematic for road matching algorithms that make use of positional and topological accuracy to match features. They may also present difficulties when validating and correcting errors. Due to the potential problems, data-sets with this type of defect are discarded.

Silkeborg & Hedensted

We chose two municipalities from OpenDataDK, Silkeborg and Hedensted. The two municipalities contain few defects of the sort that was found in Ballerup. Moreover the attribution provides a good foundation for additional matching criteria as well as potential value corrections. Both data-sets cover both rural and urban regions.

2.2.2 OpenDataDK Data Representation

Data-sets provided by OpenDataDK can be downloaded in several different formats. The specific formats for any data-set varies depending on the municipality, however the most frequent and widely used data-format is GeoJSON. GeoJSON is represented as JSON, which has a standardized format, described in RFC 7946 [1]. GeoJSON typically includes an array of features, a type declaration and a declaration of what coordinate reference system (CRS) that is used to express spatial coordinates. Both Silkeborg and Hedensted use a CRS called ESPG:25832, based on a CRS fixed to the European continental plate [11]. These coordinates must be translated to WGS84 in order to be compatible with the OSM data (See Section 3.2.2). A feature contains a key-value store called properties, which contains any attributes of the feature.

2.2.3 OpenDataDK Data Acquisition

OpenDataDK data can be downloaded from the official OpenDataDK website. The data-sets typically cover regions that are 900km^2, with file sizes around 27 MB.

2.3 Spatial, Topological and Detail Conflicts

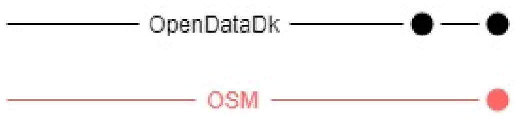
When comparing the absolute amount of nodes and edges in each of the OpenDataDK data-sets to that of the OSM data-set there is a clear difference in detail. In the Hedensted area for OpenDataDK there are around 8900 edges, and 4500 nodes. In the same area of OSM there are around 48000 nodes and over 116000 edges. In the Silkeborg area, OpenDataDK contains around 10000 nodes and 20000 edges, whilst OSM in the same area contains 60000 nodes and 145000 edges. This data hints at some huge differences in detail, which is further confirmed by the total edge lentgh of the two data-sets, seen in table 2.1. On average OSM features are half as long as OpenDataDK features in both municipalities. It is important to note that this does not mean that one should expect 1:2 matches between the OpenDataDK data-sets and the OSM data-sets, since there are probably many missing features in the OpenDataDK data-sets. There is also a large variability in detail for different areas and types of features. Highways in OpenDataDK are usually very long (In the Silkeborg data-set one segment of highway is 10km long.), whereas in urban areas, there is often an overrepresentation of nodes in OpenDataDK (Mostly due to the types of errors seen in figure 2.3). We think, however, from reviewing our data-sets that OSM is the data-set of higher quality.

Table 2.1: Total edge length in Silkeborg and Hedensted

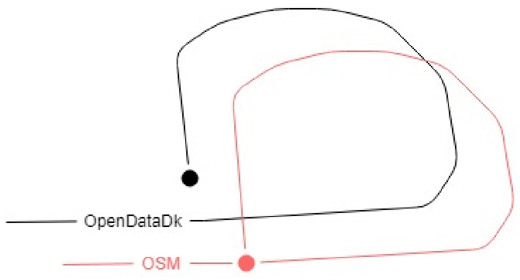
	Hedensted [m]	Silkeborg [m]
OpenDataDK	2368409	4494864
OSM	11459453	19308649

During the course of our thesis work we have observed many recurring inconsistencies in our data-sets. Many of these conflicts are inconsistencies that arise when comparing the reference data-set to the appended data-set. In figure 2.3 some of the errors are shown.

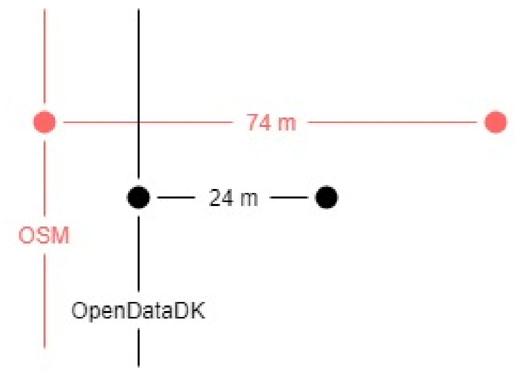
The first three figures (2.3a, 2.3b, 2.3c) show different kinds of end errors. These are errors that are frequently observed at roads with dead ends in the OpenDataDK data-set. Figure 2.3a and figure 2.3d show redundancy errors, where features (in this case a node) have been superfluously placed in the data. Usually short end errors (figure 2.3c) are observed in driveways to private housing, or parking lots. In figure 2.3e, the unique feature error is shown. This error is very common, and it is usually the case that OSM features have no counterpart in the OpenDataDK data-sets. Figure 2.3f shows a disconnected intersection, this kind of error is common in OpenDataDK, however is less frequent in Hedensted and Silkeborg. Figure 2.3g shows a difference in detail between OSM and OpenDataDK, specifically that double roads have single representations in the OpenDataDK data-sets. This kind of error is common among highways. Many of these kinds of errors could potentially be corrected by an additional pre-processing step, however these errors are not corrected in order to maintain the integrity of the data-sets, as well as to provide a better real world scenario.



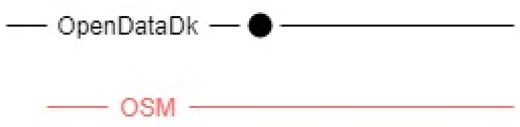
(a) **Double end** error at the end of a road.



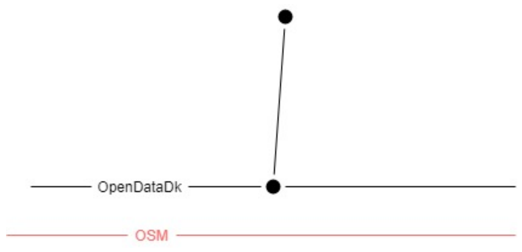
(b) **Unclosed loop** error at the end of a road.



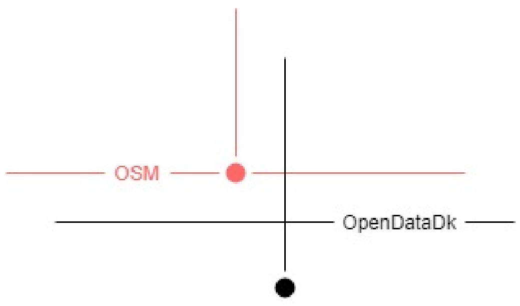
(c) **Short end** error at the end of a road.



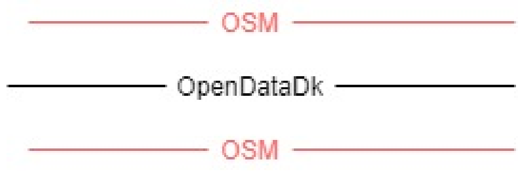
(d) **Additional node** in the middle of a road.



(e) **Unique feature**, there is no corresponding feature.



(f) **Disconnected intersection**, the intersection does not connect to the road.



(g) **Simple doublet**, one road represents two roads.

Figure 2.3: Typical topological & spatial errors found in the data-sets.

2.4 Data Frameworks

Accessing, computing, manipulating and visualizing our data-sets requires frameworks which allows us to store the data in a sensible form in memory. Moreover, in order to assess and validate data, tools that can render the data visually are required.

2.4.1 Atlas

Atlas is an open source project, which loads OSM data into memory in a suitable data structure, and provides functionality for computation, saving, manipulating and generating Atlas data. Atlas also includes tools for geo-spatial computation, e.g. distances, headings etc. Furthermore there is support for high-performance spatial queries, since Atlas spatially indexes all features [7].

Spatial Queries

Spatial queries refer to queries defined by spatial properties, such as finding all features inside a polygon, or finding all nodes outside a box, etc. The spatial indexing of Atlas uses a Quad-Tree implementation, combined with an R-tree implementation from the JTS Topology suite. These methods of spatial indexing allow the Atlas framework to compute spatial queries quickly [7]. Atlas also includes functionality to easily construct polygons. The most used function in this work is the bounding box feature, which constructs a polygon box around a location or geometry. Constructing boxes around large geometrical features is not always desirable, since there is no guarantee of how large the box will become (See figure 2.4). Therefore a separate polygon builder was created in order to better fit a polygon to large and complex features, by traversing them and building a bounding-box along the shape (See figure 2.5).

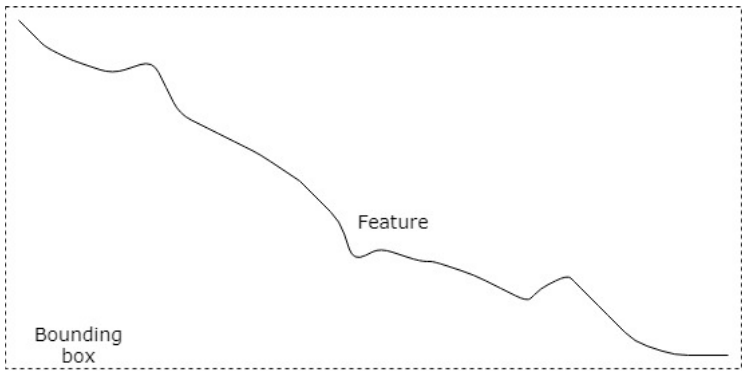


Figure 2.4: A large and complex geo-spatial feature (A road) and its associated bounding box, when using Atlas bounding box feature.

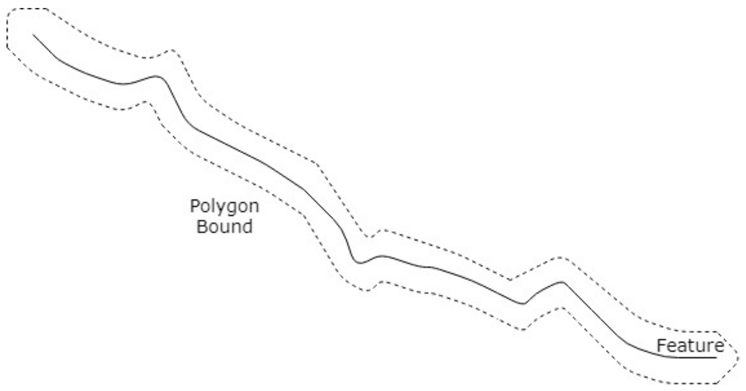


Figure 2.5: A large and complex geo-spatial feature (A road) and its associated polygon bound, when using our polygon bound feature.

Generating an Atlas

The downside to Atlas is the time to generate data and the size of the resulting data. Generating a 30km x 30km area in Copenhagen, with the Denmark OSM source, requires more than three days to compute. This was avoided by the techniques described in section 3.1. Generating atlases that cover all of Denmark results in around 52 GB data. Choosing to generate Atlases that are too large often result in "out of heap space" errors.

Atlas partially supports generating Atlas data from a GeoJSON source. This feature is mainly intended to load GeoJSON that was previously rendered by saving an Atlas as GeoJSON, and does not adhere to the GeoJSON standard. Therefore GeoJSON from external sources cannot be used to generate Atlas data, without a preliminary pre-processing step, which is detailed in section 3.2.2.

2.4.2 Atlas-JOSM

Atlas JOSM is a tool derived from a tool called JOSM. Atlas JOSM allows us to visualize atlas files in a map [8]. A visualization of an Atlas can bee seen in 2.1.

Chapter 3
Pre-processing

In order to use our data-sets with the Atlas framework, we are required to perform some pre-processing steps. These are necessary in order to allow Atlas to parse our non OSM data, so that we can work with both OSM and OpenDataDK within the same framework. Some steps are also taken in order to reduce Atlas generation time in the OSM data-set.

3.1 OSM Pre-processing

Atlas supports loading OSM files to Atlas natively, however using large source files leads to large computational complexity. It is therefore important to find suitable ways of decreasing the complexity. The method we use to reduce loading times is to first cut the Denmark OSM source to the exact geographical region of interest (Matching the area covered by OpenDataDK). A polygon bounding box is fitted to the boundary of the OpenDataDK data-set and is then used to cut the OSM source. This results in a much smaller OSM file size, typically around 20MB. The smaller file size reduces the time required to generate an Atlas to around 3 minutes.

3.2 OpenDataDK Pre-processing

Since Atlas does not natively support generating Atlas data from external source GeoJSON, the OpenDataDK GeoJSON needs to be manipulated to a format that is parsable by Atlas. Atlas supports the following four feature types:

Point A point represented by a single coordinate

LineString A series of points

Polygon A series of points that enclose an area

MultiPolygon An array of polygons

Atlas also expects properties describing specific metadata to be present in the feature, e.g. a unique identifier, an item type, directionality etc. It is also assumed that coordinates are represented in WGS84, hence all coordinates need to be transformed to WGS84.

3.2.1 JSON Library

In order to support in place mutation of the OpenDataDK data-set, we developed a JSON library with support for Java 8 functional interface, as well as mutability of JSON objects. The library was optimized for quick parsing of large JSON files, in order to reduce loading times.

3.2.2 JSON Transformation

The OpenDataDK data-sets for Hedensted and Silkeborg represent only road network data. These roads are represented as either LineString or as a type called MultiLineString. Hence the first step of the transformation is to split the MultiLineString into several LineStrings while retaining the properties of the MultiLineString.

The second step visits all coordinates in the data-set and transforms them to WGS84 if required. Coordinate transformation is computed by a library in Java called JCoord [20].

The next step is to identify intersections in the data-set. Intersections are identified by marking coordinates which have multiple occurrences. This implicitly means that the coordinate occurs in more than one LineString. This is a common method of finding intersections in GeoJSON data.

All LineStrings are then split at intersections. For example, a LineString with two intersections is then split into three LineStrings with the same properties. This ensures that every LineString represents an edge in a graph (and a segment of the original road), and that all endpoints of LineStrings are nodes in a graph.

The final step is to add Atlas-specific attributes to all LineStrings, such as ID tags, itemTypes etc. The resulting JSON structure is then saved, and is ready to be read by Atlas. A flow chart of this process can be seen in figure 3.1

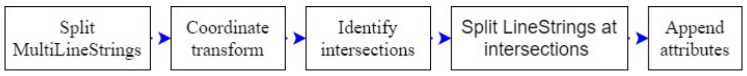


Figure 3.1: This figure shows the steps in the JSON transformation process

Chapter 4

Matching

In this chapter, we will detail our approach to matching the road-network data-sets. Matching is the process of relating entities in one data-set to the corresponding entities in the other data-set. In our case, we will match geo-spatial features between two data-sets. Usually a matching algorithm will select a feature from the **source** data-set, and then search for the corresponding feature in the **target** data-set. The result of a matching algorithm is a mapping of features from the source data-set to features in the target data-set. Features of the source data-set that are not mapped, are assumed to have no counterpart in the target data-set, and vice versa. A good matching can be used to find conflicts between the two data-sets, as well as providing a constant time look-up of corresponding features. This is useful for correcting errors, adding attribute values, adjusting for spatial discrepancies, evaluating data-set completeness etc.

The sections of this chapter will present three different ways of performing automatic matching using different algorithms. The first algorithm is a naive approach in order to provide a baseline for the other algorithms. It will also serve as an opportunity to discover some of the corner cases encountered in matching. Next is our version of an algorithm that is presented in the paper written by Mustière et al. [9]. Lastly we will present an algorithm developed by us tailored to perform well during the circumstances in this master thesis. Their advantages and disadvantages will be discussed and also examples of each algorithms limitations will be shown.

The algorithms presented in this chapter, match road network features based solely on geometrical, spatial, topological factors. If the matching used thematic values, such as names, errors could be disregarded as unlikely matches. Hence the matching process should not rely on thematic attribution.

First we will present theory pertaining to the three matching algorithms.

4.1 Theory

Before we describe the three algorithms we will first discuss a metric that is used in various ways for each of three algorithms. This metric is called Hausdorff distance, and can be a good metric for determining how alike two features are.

4.1.1 Hausdorff Distance

Hausdorff distance is a symmetric distance measure which can be used to determine the longest distance between two shapes [12]. Hausdorff distance is commonly used in computer vision, it is also very useful for feature matching. Many algorithms make use of the Hausdorff distance as a geometric similarity metric for features [5, 15, 9]. Formally it is defined as [12]

$$d_h(A, B) = \max\left\{ \max_{a\in A} \min_{b\in B} d(a, b), \max_{b\in B} \min_{a\in A} d(a, b) \right\} \tag{4.1}$$

Where A is the set of points that define the first shape, and B the second. d is any distance function, most commonly the euclidean distance [12]. We will use the euclidean distance in this thesis.

Non-Symmetric (One-Sided) Hausdorff Distance

Sometimes it is useful to compute the maximum distance from one shape to another shape. This is different than computing the maximum distance between two shapes. This can be accomplished using the non-symmetric Hausdorff distance. The formal definition of the non-symmetric Hausdorff distance is as follows [12]:

$$\vec{h}_d(A, B) = \max_{a\in A} \min_{b\in B} d(a, b) \tag{4.2}$$

Where A is the set of points that define the first shape, and B the second. d is any distance function, most commonly the euclidean distance. We will use the euclidean distance in this thesis.

The non-symmetrical Hausdorff distance is especially useful for linear feature matching when measuring the distance between lines of different length. It can be used to compute the offset of the shorter line on the longer line, without taking into account the difference of length, as shown in figure 4.1. The non-symmetric Hausdorff distance is also referred to as the one-sided Hausdorff distance.

Abstraction and Example

A simple way of thinking about the Hausdorff distance is to imagine placing a point on a shape such that the shortest distance to the other shape is maximal. In the case of the non-symmetrical Hausdorff distance the point is only allowed to be placed on one of the shapes. This is why the non-symmetric Hausdorff distance also is known as the one-sided Hausdorff distance.

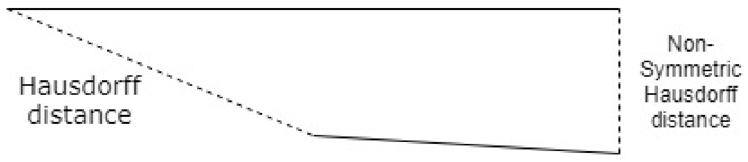
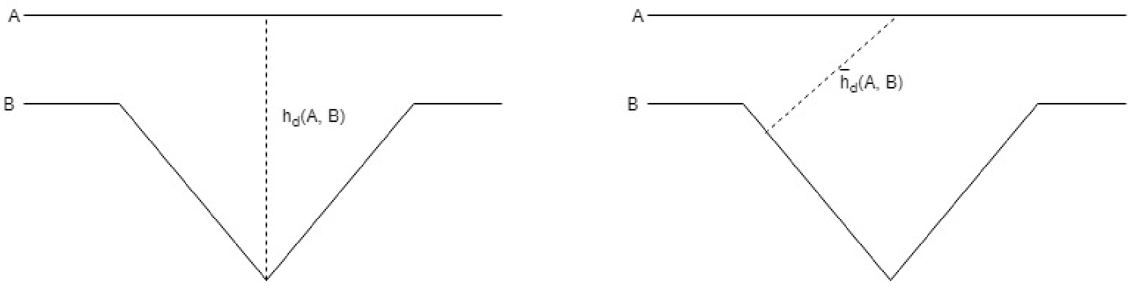


Figure 4.1: Difference between Hausdorff distance and Non-symmetrical Hausdorff distance, when line length is large. The non-symmetric Hausdorff distance is computed from the shorter line to the longer line.

Implementation

Our implementation of the one-sided Hausdorff distance, as well as the Hausdorff distance is very similar to the implementation described in the NetMatcher article [9]. For each point defining the geometry of a PolyLine, the minimum distance is measured to the other PolyLine. The maximum distance among these minima is chosen. If this only is done for one PolyLine, the one-sided Hausdorff distance is computed, but if it is measured mutually the Hausdorff distance is computed. This is a simplification that can be done when measuring Hausdorff distance along linear features (i.e. shapes that are made up of linear segments), allowing the running time to become $O(mn)$ where m is the amount of points in the first PolyLine and n the second.



(a) Two shapes A and B, and the line which corresponds to the Hausdorff distance shown as a dotted line.

(b) Two shapes A and B, and the line which corresponds to the one-sided Hausdorff distance shown as a dotted line.

Figure 4.2: Hausdorff distance (left) and one-sided Hausdorff distance (right).

4.2 Naive Matching Algorithm

In order to provide a baseline matching algorithm, we begin by presenting a naive approach. This approach is simple since it is only capable of 1:1 and 1:0 matches. Since the road network is described using two types of entities, nodes and edges, the algorithm is divided into two independent steps, first a step matching nodes, followed by an edge matching step.

Investigation of our data-sets showed that the geo-spatial displacement between the source data-set and target data-set was very small in terms of rotation. Moreover the offset between the two varies a lot, but in most cases the offset is moderate to small, within 10 meters (See Section 6.2.1). This made us consider that in most cases perhaps the best match is the closest. In the node case, this means measuring the euclidean distance between points, since each node represents a geographical point. In the case of matching edges, the Hausdorff distance provides a good metric for geometrical similarity, as discussed in Section 4.1.1.

4.2.1 Node Matching

This section aims to give a more detailed explanation of the process of matching nodes from source to nodes in target.

The basic idea for naive node matching is to, given a node in the source data-set, find the closest node in the target data-set using euclidean distance. This can be done by globally comparing nodes of the target data-set to the selected node in source to find the target node of minimum distance. However, atlas provides functionality for spatial queries, that are performance optimized. The functionality allows querying of nodes within an area, represented by a polygon. Therefore calculating the euclidean distance pairwise globally can be avoided and instead a query can be made for nodes inside a polygon buffer around any selected source data-set node. Buffering in this way is quite common for many matching algorithms [9, 19, 5, 22]. Although buffers are mainly used for finding potential candidate edges when matching edges, this concept extends to nodes as well, as is the case in the NetMatcher algorithm [9]. Another advantage of querying in this way is that unlikely matches are eliminated, such that, if a node has no counterpart in the target data-set, it is less likely to find one, if the buffer is sufficiently small. Once the likely candidates in the target data-set have been found, the node of smallest distance is chosen as the match. The efficiency of this approach is highly dependent on the implementation of spatial queries in the underlying framework. Assuming that the running time of a spatial query, e.g. nodes inside a bounding box, is $O(x)$, the running time of a global matching using such an implementation should become $O(n_{source}(n_{target} + x))$, where n_{source} is the number of nodes in the source data-set and n_{target} is the maximum number of nodes inside a bounding box of the chosen size in the target data-set. n_{target} will reasonably vary depending on the size of the bounding box. If the side of the bounding box is t, then the size of the box will be t^2, hence performance is also dependant on the chosen size of the bounding box. The impact may or may not be noticeable compared the impact of spatial queries. This gives the user a possibility to make a trade-off between the likelyhood of finding a match and running time. We expect tighter boundaries to run faster and to have a lower probability of incorrectly matching nodes, large boundaries might result in a larger quantity of correct matches, with the trade-off of some incorrect matches, and poorer running time.

4.2.2 Edge Matching

The principle behind matching edges is the same as the principle behind matching nodes. The chosen metric for computing the closest edge was the Hausdorff distance. Since we want to prioritize matches between edges of similar heading, a simple extension was made to the Hausdorff distance. This extension is described in an article on optimizing linear feature matching [5], and includes the angle difference between the two features. Originally the article also uses name similarity to improve the matching heuristic, but since we want to be able to detect potential errors in the name attribute in a later step (See Section 5) this was omitted. By including a factor that puts a penalty on two edges with a big difference in heading, edges with the same heading are scored with a lower relative distance. If we denote the Hausdorff distance with d_h, the new metric becomes $d'_h = \frac{d_h}{cos^2(\alpha)}$ where α denotes the angle between the two headings.

Now that a metric has been found, a polygon can be created around an edge selected from source. When this polygon is created using the atlas framework, a rectangular polygon is created, as can be seen in figure 2.4. This can in some cases create large distances between the edge and polygon. For this reason we will use the tighter polygon bounding box described in Section 2.4.1.

All the edges that intersect this polygon can be queried from target data-set and d'_h can be calculated for each of these edges. The edge with the smallest score is matched. The entire source data-set is matched when this is done for each of the source edges.

Calculating the value of d_h is a computationally heavy operation, $O(mn)$, see section 4.1.1. Therefore an upper limit is set to the allowed difference in heading between the two edges in a potential match to decrease the amount of candidates, thus lowering the running time. The value chosen is 45 degrees. To calculate the time complexity of the algorithm we denote the number of segments in an edge as deg(edge), the maximum number of edges intersecting a polygon around an edge, given a threshold, as p. Also assume that the time complexity of the spatial query performed is $O(y)$. If the edge from source is called k and the edge from target l, The total running time of this implementation, including the naive node matching step becomes $O(n_{source}(n_{target} + x + p \cdot y \cdot deg(k) \cdot deg(l))$. The actual running time will be highly dependent on the time-complexity of the underlying implementation of the spatial-query.

This approach only produces one to one mappings from the source data-set to the target data-set. However, correct matches are rarely represented as one to one mappings. The advantage to this approach is that it is intuitive and robust.

4.3 NetMatcher

To have a better understanding of the existing matching algorithms out there, we have chosen to partially implement an algorithm presented in a paper by Mustière et al. [9]. There is an open-source version of the algorithm implemented in a framework called Geoxygene. This version cannot be used in our framework, it would require a wrapper in order to interface correctly with Atlas. Bugs in this wrapper would be difficult to identify, since they would likely manifest in the Geoxygene framework. To correctly wrap Geoxygene and create an interface to that wrapper would probably take very much time. Therefore we have decided to implement our own matcher based on the principles of NetMatcher. Our implementation does not include the fourth step, edge matching, due to a lack of time. NetMatcher assumes that one of the data-sets in the matching is more detailed than the other data-set. In our case OSM is more detailed than OpenDataDK (As discussed in Chapter 2). NetMatcher consists of a four step process. These steps are as follows:

1. Pre-matching of Nodes

2. Pre-matching of Edges
3. Node Matching
4. Edge Matching

In the four sections below we account for each of these steps and how our implementation differs from the one detailed in the article.

The NetMatcher requires that the source data-set is less detailed than the target data-set, in order to work correctly. This requirement is specific to the second step of the algorithm, pre-matching of edges. This is explained further in section 4.3.2.

4.3.1 Pre-matching of Nodes

The first part of NetMatcher is the pre-matching of nodes. This step simply selects nodes in the source data-set and, for each given source node, builds a list of potential candidate nodes in the target data-set. The candidate nodes are selected from the target data-set within a distance of the source node, the distance is called the threshold [9].

In our implementation a bounding box is built around a selected source node using the bounding box feature (See Section 2.4.1). We then query the target data-set for any nodes inside that bounding box, thus giving us a list of node candidates. The way this threshold is chosen, does not seem to affect the algorithm much, as long as it is a high value. The authors write: "In our case, the purpose of this pre-matching step is to make an over-selection of candidates. [...] thresholds are better over-evaluated than under-evaluated."[9].

4.3.2 Pre-matching of Edges

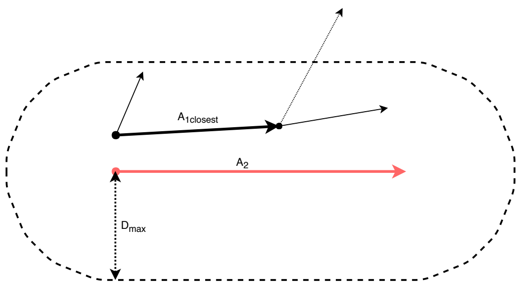
The second step of NetMatcher is a necessary precursor to the third step: node matching. The second step establishes possible relations between edges, generating a list of candidate edges. The node matching step relies heavily on this list of candidates [9].

The pre-matching of edges is done in a similar fashion to that of the pre-matching of nodes. NetMatcher establishes candidates by making use of the one-sided Hausdorff distance from the more detailed data-set to the less detailed one. The situation that arises is similar to the one in Figure 4.1, where the one-sided Hausdorff distance better describes the likelyhood of a match between the edges than the Hausdorff distance. Given an edge of the less detailed data-set, denoted A_2, NetMatcher finds the closest edge in the more detailed data-set, denoted $A_{1closest}$. Candidates for A_2 are then selected by finding all edges A_1 from the more detailed data-set fulfilling [9].

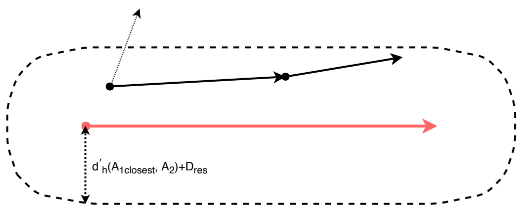
$$d'_h(A_1, A_2) < \min\left\{D_{max}, d'_h(A_{1closest}, A_2) + D_{res}\right\} \tag{4.3}$$

where D_{max} is the maximum distance where candidates are considered, D_{res} is the relative Hausdorff distance to the closest match where candidates are considered [9]. d'_h is the one-sided Hausdorff distance discussed in Section 4.1.1. A visualization of the process can bee seen in figure 4.3.

Our implementation uses the same heuristic that NetMatcher uses for pre-matching of edges. We begin by selected an edge A_2 in the source data-set. A polygon bound is then



(a) Edges which produce one-sided Hausdorff distances larger than D_{max} are filtered. This can be represented by the area with a boundary at uniform distance D_{max} from the source edge. Edges partially or completely outside of this area are filtered. The closest edge is identified as the thick dark line. Dotted edges are filtered.



(b) Edges which produce one-sided Hausdorff distances less than $d'_h(A_{1closest}, A_2) + D_{res}$ are pre-matched. This can be represented by the area with a boundary at uniform distance $d'_h(A_{1closest}, A_2) + D_{res}$ from the source edge. Edges completely inside of this area are pre-matched. Dotted edges are not pre-matched.

Figure 4.3: An example of the pre-matching of edges step of the NetMatcher. The figures visualize equation 4.3.

fitted to this edge (See Figure 2.5), set at a buffer distance similar to that of our D_{max}. This size of buffer ensures that any edges with h'_d intersects the polygon. Then Atlas is queried for any edges A_1 intersecting or within this bound. Subsequently the minimum distance $A_{1closest}$ is computed by iteration of the queried edges. Candidates are then found by a second iteration of the queried edges, constrained to Equation 4.3.

Potential downsides to this approach is the risk of pre-matching edges perpendicular to the source edge.

4.3.3 Matching of Nodes

NetMatchers third step establishes matches for nodes. Since the spatial relation between the source and target nodes have already been established in the pre-matching step and likely candidates for edges have been established in the pre-matching of edges, the topology of node candidates can be analyzed in this step. First NetMatcher categorizes nodes in three Categories based on the pre-matched edges of the nodes [9].

- A pre-matched node is said to be **complete** if a correspondence is found for all edges out of the source node to the outgoing edges of the target node.
- A pre-matched node is said to be **incomplete** if a correspondence is found for some of the edges out of the source node to the outgoing edges of the target node.
- A pre-matched node is said to be **impossible** if there is no correspondence between the edges of the source and target node.

A correspondence is defined by two metrics. The first is the so-called "turning criterion", and the second is the heading (one-way, two-way) of edges in cases where that information is provided. The turning criterion states that the order of edges must be preserved, such that if the order of edges around the source node are A_1, B_1 and C_1 then the order of their pre-matched counterparts should be the same around the target node. Secondly the headings of edges must conform [9].

A selection is then done. If there are no pre-matched nodes, or if all pre-matched nodes are impossible, the node does not become matched. If there is only one complete pre-match, this match is selected. If there are several complete pre-matches, the closest is selected. If only one of the pre-matched edges are incomplete, this is selected. If none of the above criteria are fulfilled, pre-matched nodes are grouped and more criteria are applied [9]. However the details of grouping nodes and evaluating the additional criteria are not described in the article, therefore omitted here.

Our implementation closely mimics that of NetMatcher, except for the evaluation of node groups, meaning that only 1:1 mappings are considered (in the original NetMatcher article, 1:n matches are also considered, where one node is matched to several nodes). By pairwise comparison between the source node and a candidate node, the outgoing edges of both are analyzed. The outgoing edges of the source node are verified to have a pre-matched edge connected to the target node. A high number of matching edges is considered to increase the probability of the match contributing to a good global match. To ensure topological correctness, the order of the edges are also verified. The most complete node is matched. If there are several nodes that are considered to be equally complete, the closest is matched.

The advantage of performing more analysis on the surrounding of each node is that it reduces the possibility of false positives being matched. By omitting the grouping step, we cannot account for 1:n matches, where one node is matched to several nodes in the target data-set. This will make our implementation less versatile than the original implementation.

4.3.4 Edge Matching

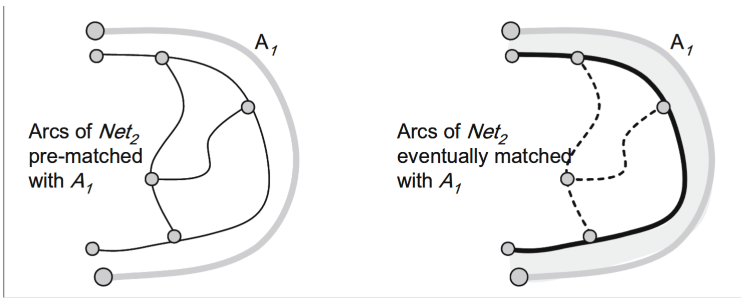

A1
Arcs of Net2 pre-matched with A1
A1
Arcs of Net2 eventually matched with A1

Figure 4.4: The area between each pre-matched edge and the source edge is computed. This area is used as a weight for the edge. The shortest distance is found using e.g. Dijkstra's algorithm, and that path is selected as the match [9].

The final step of NetMacher, the edge matching relies on finding paths between matched nodes through the network of pre-matched edges. These edges are weighed depending on the surface spanned by the area in between the pre-matched edge and the source edge, and then the shortest path between the two matched nodes is selected as the best possible match. An example of this step is shown in figure 4.4 [9]. However, we chose to not implement this step due to a lack of time, and considerations after seeing results of node matching in NetMatcher. This is instead left as possible future work to this work, in order to compare a full implementation of NetMatcher to the other edge matching algorithms proposed in this thesis.

4.4 Expansive Edge Matcher

In addition to the other two algorithms presented, we also created our own algorithm, based on our own knowledge and experience of the domain. The **EEM** (Expansive Edge Matcher) attempts to match road features based on context, topology and spatial values. It is inspired by several other matching algorithms and methods. Surprisingly naive node matching works incredibly well for matching nodes, in both Hedensted and Silkeborg (See Section 6.2.1). Therefore a variation of the naive approach to matching nodes is used as a way to obtain a set of candidate edges. Also, since buffers are widely used in related work [9, 19, 5, 22], buffers are used to gradually expand so called matching groups. Additionally potential correct matches are evaluated based on the Hausdorff distance as is done in other articles [5, 15, 9]. It also takes into account the topology of edges, by selectively pruning expansions that drift too far from the source edge, and preserves context throughout the matching process, by matching groups of edges, instead of matching edges individually. It supports 1:1, 1:0, 1:n and n:m matches.

The EEM uses a recursive algorithm. Given an initial condition the EEM expands features in an effort to produce matching groups that have similar shape, spatial attributes and length. It consists of three steps, firstly an initial condition must be selected for the recursion, in a step called feature selection. This step is followed by the expansive recursion algorithm, which uses depth first search in order to establish a matching. The final step is an evaluation step, where the best of the produced matching groups is chosen and inserted as a match.

4.4.1 Feature Selection

The goal of the first step is to provide the recursive algorithm one or many potential initial conditions. First an edge in the source data-set is chosen, which is called the **initial edge**. A bounding box is built around the starting position of the initial edge and the target data-set is queried for **candidate nodes** within this box. The set of incident edges on the candidate nodes become the **candidate edges** (See figure 4.5).

The recursive step is then called for each of the candidate edges, along with the initial edge. Thus this step does not attempt to naively select the closest edge or node, instead attempting to match any of the features found within the starting area to the source edge. We assume that any spatial displacement of the two data-sets are small or moderate, such that, if there is a correct matching for the source feature, a corresponding feature will be found within the bounding box. This is a reasonable assumption to make for our data-sets given the results of the naive node matching. If there are no candidate edges, the initial edge is not the start of any matching group. However it may still be a part of another

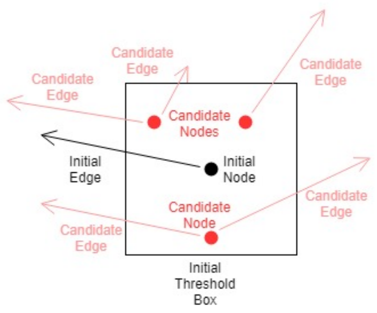


Figure 4.5: Node candidates and edge candidates are found within a buffer of the start of the initial edge. Edge candidates are any edges incident on the nodes within the buffer.

matching group, since another initial edge might lead to a match that includes this source edge.

4.4.2 Recursive Expansion

The inputs to the recursive expansion are a source edge, a target edge and a **threshold**. In the initialization of the recursion, two sets are created in order to keep track of already visited nodes, thus preventing stack overflows as result of cycles in the road network. The visited sets are passed to the core recursion, together with the other inputs.

The core recursion **annotates** one of the input edges as the **goal edge** and the other **current edge**. The goal edges end node is called the **goal node** and the current edges end node is called the **current node**. The **threshold box** refers to a bounding square centered in a node, the side length is determined by an input called the **threshold**. The following base cases are accounted for in the following order:

Already Visited If the current node was already submitted in the visited set, the recursion returns a non-match.

Match Found If the current node is within the threshold box of the goal node, the recursion returns a list containing one match. For an example of the match found base case, see figure 4.6.

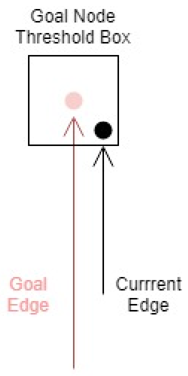


Figure 4.6: The current node is within the Threshold box of the goal node.

Switch Annotation If the threshold box of the current node does not intersect the goal edge, but the threshold box of the goal node intersects the current edge, annotations are swapped (the current edge becomes the goal edge, vice versa). For an example of the switch annotation base case, see figure 4.7

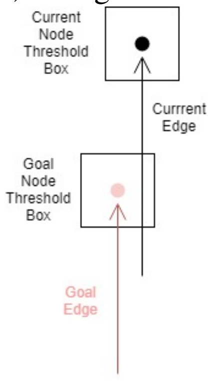


Figure 4.7: The current node is within the Threshold box of the goal node.

Prune current If the threshold box of the current node does not intersect the goal edge a non-match is returned. For an example of the prune current base case, see figure 4.8

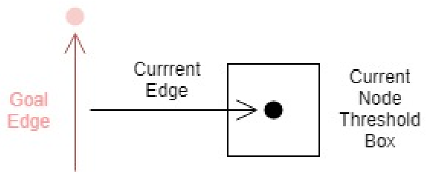
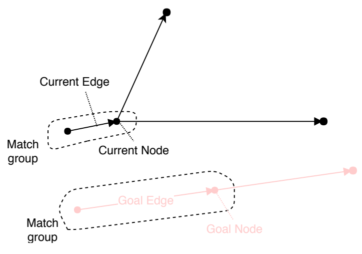


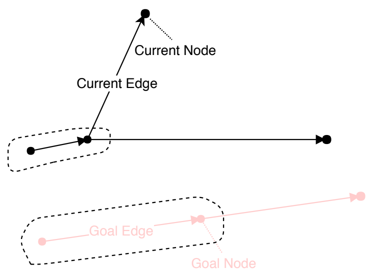
Figure 4.8: The threshold of the current node does not intersect the goal edge.

After the base cases, we are ensured that the threshold box of the current node intersects the goal edge (See the Prune Current step). Therefore the second part of recursion is the "expansive" part. The expansion is done by selecting a new edge incident on the current node and recursing with that edge as the current edge. This is done for all incident edges on the current node. If none of the indicdent edges return a match, a non-match is returned. Otherwise all lists of matches are concatenated and updated with a relation between the current and goal edge. Any potential matches are stored as a list and returned as a match. Since these relations are added throughout the call stack, the final returned list of matches contains all the matched edges and the relations between them. Figure 4.9, shows an example of the EEM recursive step on a basic example.

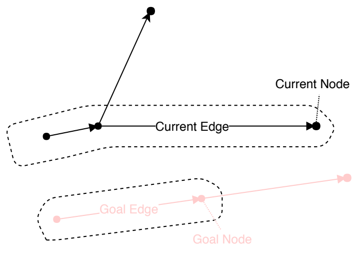
A match returned by the expansive recursion contains all edges of that are matched, and all relations between them. The edges belonging to the source data-set are called the matches **source edges**, and the edges belonging to the target data-set are called the matches **target edges**. The algorithm ensures that the starting node of the source edges correlate with the starting node of the target edges, since this is specified by the initial condition. Moreover it also ensures that the end node of the source edges correlates to that of the target nodes (See Match Found base case). This should make the algorithm good at matching groups of edges between intersections, since intersections are most likely to



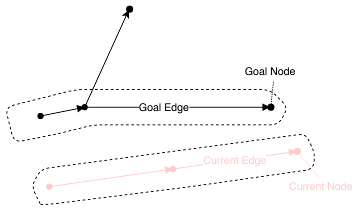
(a) Step one An initial condition is input from feature selection, this initial condition does not fulfill any base case.



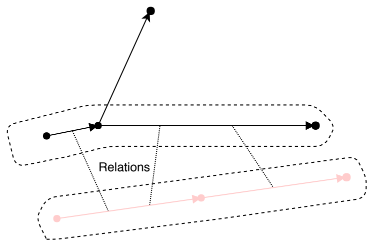
(b) Step two The next edge incident on the current node is expanded. This edge is pruned, since it fulfills the **prune current** base base.



(c) Step three The other incident edge on the current node is included in the match group. The current edge now fulfills the **switch annotation** base case, annotations are swapped.



(d) Step four The next edge of the current node is included in the match group. Note that in step three, annotations were swapped. The match found case is now fulfilled and a match is returned.



(e) Step five As the recursion returns the match, relations are added in each step between the current edge and goal edge, forming the relations represented by the dotted lines.

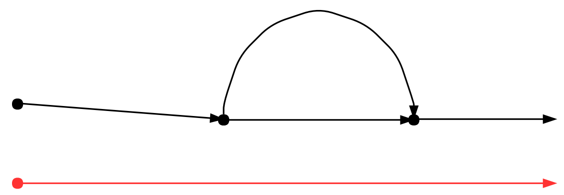
Figure 4.9: A five step example of the recursive step of the EEM. Match groups are represented by the area delimited by the dotted line. The target data-set is represented by dark lines and the source data-set is represented by the light lines.

meet these two conditions. The algorithm also guarantees that any intermediate nodes are sufficiently close (within the threshold box) to the corresponding list of edges to be considered a match.

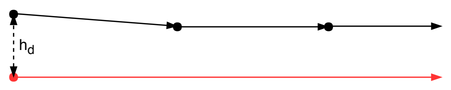
The recursive step is potentially the most time consuming, since it attempts to visit all incident edges on nodes along the matches list of edges. For every node that is visited spatial queries are done, which increases computational complexity. Moreover, in cases where one of the data-sets features a long road, without intermediate nodes and the corresponding features are rich in intersections and have multiple representations (as is the case of a Simple Doublet error, See Figure 2.3g), time complexity can become exponential. In order to deal with such cases, the returned list of matches is capped to 200 matches. Improved heuristics can probably get rid of the exponential time case altogether, by e.g. dynamic programming or some local optimization with memoization (This is left as future work). Another drawback of the recursion is in the base case **Match Found**, since it is unclear if there can be better matches by further expanding the node where the match was found. In the cases of Figure 2.3a and Figure 2.3b, the recursion stops early and cannot progress to the entire shape, thus obtaining a bad evaluation in the next step. A better alternative would be a way to check incident edges of the end node in order to see if there are any potentially better matches, and let the recursion continue for those candidates, this is also left for future work.

4.4.3 Match Evaluation

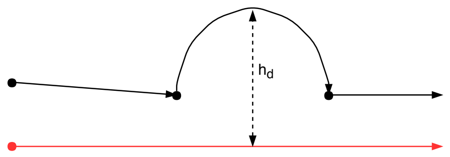
In the final match evaluation step, all matching groups that were found for the particular initial edge are evaluated in order to find the most suitable matching group. Thus the input is a list of matching groups, returned by the recursive step, for each of the candidate target edges.



(a) An example edge configuration that will lead to two match groups. One match group will contain the half-circle edge and one will not. The two match groups are seen in figure 4.10b and 4.10c



(b) The first match group. The Hausdorff distance can be seen as the dotted line. This match group is the best candidate of the two match groups.



(c) The second match group. The Hausdorff distance can be seen as the dotted line. This match group is the worst candidate of the two match groups.

Figure 4.10: An example of an edge configuration leading to multiple match groups. The match evaluation step will select the match group represented in figure 4.10b, since it produces the minimum Hausdorff distance.

Each matching group is evaluated by Hausdorff distance, and the group with lowest Hausdorff distance is selected as the best match. When calculating Hausdorff distance along a match group, two accumulated lines are constructed from the matched edges, one line representing the source edges and one representing the target edges. Thus the Hausdorff distance is evaluated for the entire match, and not once for every included edge, thus avoiding having to use the one-sided Hausdorff heuristic used by NetMatcher (See Section 4.3). The full Hausdorff distance, provides a more strict evaluation than that of the one-sided, since it evaluates the mutual maximum distance between edges. This step selects the most correct match and filters inferior matches, for example roads which meet match conditions, even though they should not constitute a match. An example of the match evaluation step is seen in figure 4.10.

Chapter 5
Error Detection and Correction

In the domain of conflation, error detection refers to the discovery of conflicts between related features, thus relies on some shared attributes of the input data-sets. Usually attributes of geo-spatial data-sets are tailored to fit a certain need [9]. The needs of danish municipalities are probably different from the needs of OSM users, therefore the expected overlap of attribution is small. The common attributes can be compared for every relation, in this way errors can be found and inconsistencies accounted for. We will use the term **flag** to signify an object describing a discrepancy between the two matched data-sets. The name flag was inspired by the open source library atlas-checks. Flags contain the detected error, as well as some contextual data pertaining to the error, such as nearby matches etc.

In the Hedensted data-set there are two attributes which overlap the attributes of the OSM data-set, names and speed limits. However in Silkeborg there is little overlap of attribution. The Silkeborg data-set contains an attribute called designation. The designation of a feature usually contains information about the name of that feature, but can also contain additional metadata, such as: municipality, addresses etc. Unfortunately there is no clear standard as to how the metadata is represented, therefore parsing the name poses a problem.

The problem posed by error detected is further compounded by the matching itself. A good matching between two data-sets provides the foundation of error detection, since false matches will almost always lead to false errors being detected. These matches can of course be flagged anyway, since the underlying reason for the false match could indicate a geometrical/topological inconsistency in the input data-sets.

Given a set of error flags, a correction could be attempted. The most difficult part of correction is knowing which of the data-sets contain the correct value of an attribute. Even if one of the data-sets has attribute values of lower quality, there is no clear way to be certain about which of the two attribute values is the correct one. Uncertainties in correction are compounded by the risk of trying to correct false matches. Attempting to automatically correct a false flag poses a risk of decreasing overall data-set quality.

5.1 Name Error Detection

In order to detect conflicts on names, a similarity metric should be established. A similarity measure can allow further analysis of the type of conflict. A reasonable expectation of the similarity measure, is that it should be able to differentiate typos and misspellings from entirely different names all together. For this task we used the Levenshtein distance between the names of matched features, which has previously been recommended in an article on conflation by Samal et al. [14]. Samal notes that "the Levenshtein distance is particularly well suited to accommodate minor spelling errors.". Samal also proposes another string similarity metric, based on the phonetics of the words, however adds that it should be used for errors in transcription, for example when audio is written to text. The Levenshtein distance returns an integer value representing the amount of insertions, deletions and replacements required for the two strings to match [4]. The expectation is that typos and misspellings result in small distances, whilst different words will exhibit a large distance. We think that name similarity in the match could be used as a metric to evaluate the overall validity of the match, since we think it is likely that correctly matched features will contain similar names. However it would not be a strict validation. For the most part it might indicate that two related road segments of the source and the target belong to the same road, however the actual relation of the two segments cannot be verified this way, for example see figure 5.1.

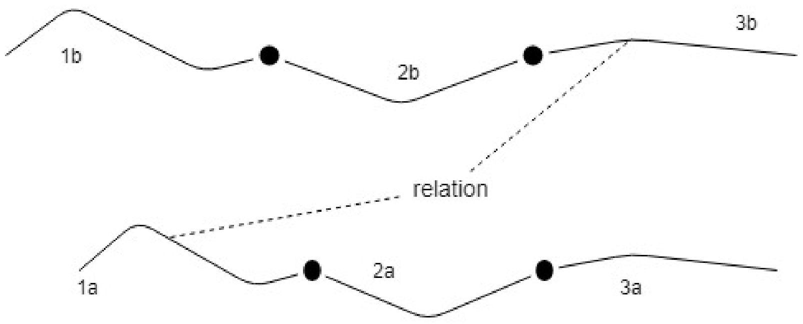
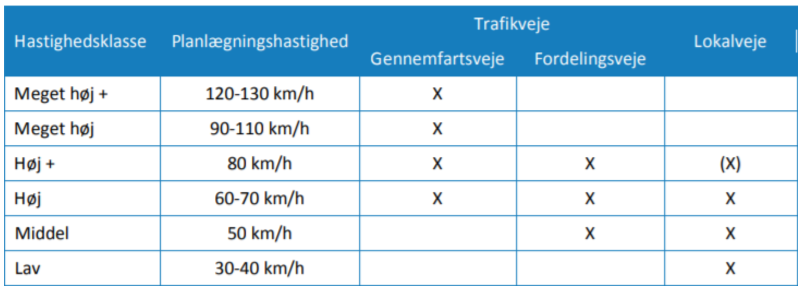


Figure 5.1: An example of a faulty relation between two road segments of the same road.

It could be possible to try to eliminate false flags, by discarding relations with high Levenshtien distance, but there would be a risk of discarding true flags, thus this is not done.

5.2 Speed Error Detection

Speeds are straightforward to compare. However, in our case, the Hedensted data-set does not provide explicit speed limits. It does however provide an attribute called Vejklasse, which roughly translates to Road Class. The value of this attribute usually consists of three tokens, each providing some context to a definition of the road. These tokens have formal definitions given by the danish authority on road regulation and safety, Vejdirektoratet [17]. In a document from 2012 Vejdirektoratet provides the following table concerning speed limits and Road Class identifiers for road features in whats called "open land". What is meant by Open Land is practically any feature located outside villages, towns or cities, according to a review of the data, along with both articles from Vejdirektoratet [16, 17].



Hastighedsklasse	Planlægningshastighed	Trafikveje		Lokalveje
		Gennemfartsveje	Fordelingsveje	
Meget høj +	120-130 km/h	X		
Meget høj	90-110 km/h	X		
Høj +	80 km/h	X	X	(X)
Høj	60-70 km/h	X	X	X
Middel	50 km/h		X	X
Lav	30-40 km/h			X

Figure 5.2: Speed limits for three different road classes in open landscapes in the OpenDataDK data-set [17]

This information specifies the speed classes for three types of roads. However a fourth road class is unaccounted for, Trafikvej. This road class is not specified at all in the 2012 document, however a document from 2008 shows that the class is used in villages and cities, and also provides a more fine grained definition of Lokalvej by providing more specific speed limits within cities and villages [16], which can be seen in table 5.1.

Table 5.1: Speed limits for though different road classes in cities, towns and villages in the OpenDataDK data-set [16]

Hastighedsklasse		**Funktionel**
Hastighedsklasse	**Ønsket hastighed (km/h)**	**Vejklasse**
Høg	60-70	Trafikveje
Middel	50	Trafikveje/Lokalveje
Lav	30-40	Trafikveje/Lokalveje
Meget Lav	10-20	Lokalveje

Using this information the speed limits in OSM can be checked if they conform to the speed limit bounds specified by the Road Classes defined by Vejdirektoratet.

5.3 Error Correction

For the reasons stated in the introduction of this chapter, as well as time constraints, error corrections have not been implemented as a part of this thesis work, instead they are left for future work.

Chapter 6
Results

This chapter will present our approach to evaluation and the results of our algorithms. In Section 6.1 we present the different ways that the algorithms will be evaluated, using different metrics. In Section 6.2 we present the results of our matching algorithms, using the evaluation techniques discussed in section 6.1. Finally in Section 6.3 we present the results of Speed limit error detection in Hedensted using the Expansive Edge Matcher.

6.1 Evaluation

This section presents how we evaluate our algorithms. Node matches is evaluated using a manually matched reference and the distance between matched nodes. Edge matches is evaluated using a manually matched reference. Edge matches are also evaluated by length difference, Hausdorff distance and name similarity metrics. We will also present how parameter values are chosen for the different algorithms. Moreover we will present the hardware used when bench-marking algorithm execution times.

6.1.1 Evaluation of Matching Algorithms

There are many different ways of evaluating the quality of a matching algorithm. One of the ways of evaluating the quality of a matching algorithm, is by comparison to a manually matched area. This approach was used in Du, H. et al. [3], where matches were evaluated manually.

We have provided a manually matched reference to two areas in our data-sets, one in Hedensted and one in Silkeborg. These manually matched areas can be used in order to validate matches established by the matching algorithms. The area in Hedensted is a rural area and the area in Silkeborg is an urban area. These two types of areas where chosen since the quality of OSM varies between these areas, according to Barron et al.

[2]. Moreover, by visual inspection, these two areas contain many different topological structures and some errors of the types seen in figure 2.3.

The two different manually matched areas are presented below.

Village Area in Silkeborg

The matched data in this data-set is a part of a village/small town in Silkeborg municipality. It was chosen since there seemed to be good coverage between OSM and the reference data-set. The selected area also seamed to contain a lot of different types of streets and crossings. These conclusions were made by visual inspection of the data-sets in JOSM. The area is shown for OpenDataDK in figure 6.2 and for OSM in figure 6.1.

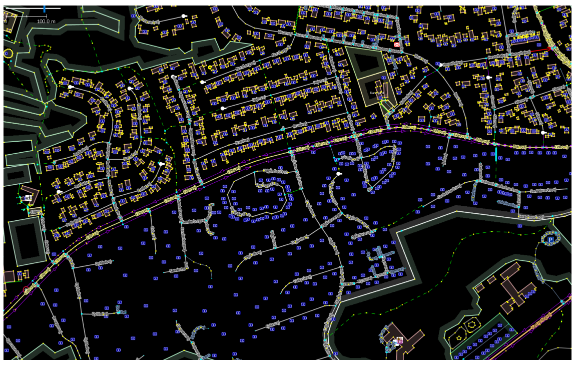


Figure 6.1: This is a visual representation of the OSM data in the matched area in Silkeborg.

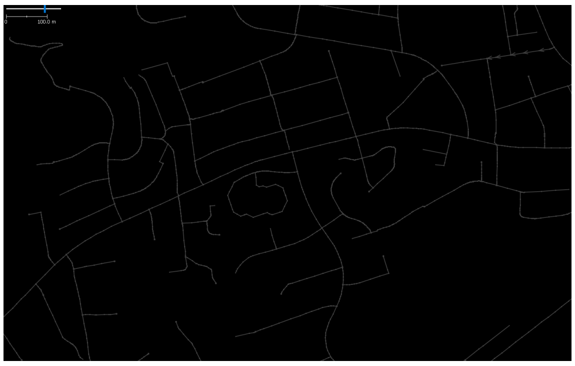


Figure 6.2: This is a visual representation of the reference data in the matched area in Silkeborg.

Rural Area in Hedensted

The matched data in this data-set is a rural area in Hednested in Denmark. This data-set was chosen since it has good preservation of connectivity in the road network and also features a variety of roads and crossings.

Evaluation of Correct Edge Matches

Since a correct mapping of nodes to nodes is provided by the manual match, evaluation of node matches from the algorithms are trivial.

Evaluation of edges is more complicated since the manually matched areas only contain mappings of the nodes. An edge has one start and one end node, delimiting the edge. Evaluating an edge match is done by evaluating the delimiting nodes of that edge, similar to how node matches are evaluated.

Spatial Similarity

Matched features can be evaluated by their spatial similarities. This can be done for both node matches and edge matches.

Since nodes do not express a geometry, rather a point, the spatial similarity metric used for node matches is the euclidean distance. Zhang et al. [21] notes that the offset between two data-sets approximately corresponds to a normal distribution, thus we expect distances between matched nodes to roughly resemble a normal distribution.

As mentioned in section 4.1.1, the Hausdorff distance has been used in other studies as a shape similarity metric. Therefore we use the Hausdorff distance as a spatial similarity metric between matched edges.

We also measure the difference in length between two matched edges. This metric is designed to show the overall coverage of matched features, similar length edge matches contribute to similar coverage in both source and target data-sets.

Histograms are used to plot the results of the spatial similarity metrics. The Hausdorff metric, as well as the length difference metric of edges are plotted with a logarithmic x-axis scale, in order to provide better resolution of low scores.

Evaluation of Matches using Name Dissimilarity

By using a binary metric like the manual match validation, information from partially correct matches are lost. To know how good the algorithms are at obtaining partially correct information, a more flexible metric could be used. As explained in chapter 5, it is expected that two edges with similar names represent segments of the same road. This does not mean that they represent the same segments of the road. Since this information could help evaluate the validity of a global match, it will be presented in this chapter. This metric cannot be used to evaluate the number of correct matches, but acts as a measurement of how many matches are partially correct. This metric is displayed in the form of histograms in every matching algorithm. All dissimilar names also constitute errors detected and generate flags. Hence the name dissimilarity results are both a measure of the matching algorithms relative performance (based on name similarity), as well as the output flags from error detection (based on name dissimilarity).

Statistics on the Manually Matched Areas

Table 6.1 shows statistics on the distances between the manually matched nodes between the reference data-set and the appended data-set. The last three columns display information about the euclidean distance between the matched nodes. Notably, the largest distance, seen in the last column, is due to the short end error shown in figure 2.3c.

Table 6.1: Statistics on manually matched areas. Distances are between the manually matched nodes of the two municipalities.

Municipality	Number of nodes	mean distance (m)	median distance (m)	longest distance (m)
Silkeborg	82	2.93	1.29	36.57
Hedensted	59	4.65	1.75	63.37

6.1.2 Parameter values for algorithms

The input values to the algorithms plotted in the histograms are those input values which gave the best results in the test areas for each algorithm, and are presented in table 6.2.

These threshold are found through trial and error, using the manually matched areas. This can seem contrived in a general case, but manually matching an area of a data-set requires little effort in comparison to attempting manual conflation. Furthermore, our implementation allows multiple thresholds to be run concurrently, and results are automatically compiled and saved. In addition, the running times of the algorithms are sufficiently low, making the task of finding the best threshold easier. Furthermore the thresholds presented in the tables are selected from a wide spectrum in order to capture the effects of choosing too small threshold values, and also of choosing too large threshold values.

Table 6.2: Threshold parameter values used for histograms in the three tested algorithms. These values are the values which scored best in the test areas for each municipality.

Algorithm	Threshold Hedensted [m]	Threshold Silkeborg [m]
Naive Node Matching	55	40
Naive Edge Matching	2	2
NetMatcher	20	20
EEM	5	5

Moreover additional parameter values for all results are presented in the list below.

Naive Node Matching None.

Naive Edge Matching Matches are cut if angle between the edges exceed 45 degrees.

NetMatcher The pre-matching of edges are done with a three meter buffer around an edge, with D_{res} = 5m and D_{max} = 20m (See equation 4.3).

Expansive Edge Matcher None.

6.1.3 Evaluation of execution times

The hardware used for evaluation of execution times for the matching algorithms is presented in the list below

Processor Intel core i7-7600U (TDP-up) 2.8 GHz base, 3.9 GHz turbo.

Ram 16 GB DDR-4 2400 MHz.

Execution times are measured as the mean of ten runs, with the given threshold for each algorithm. The algorithms are run 11 times, but the first measurement is disregarded, due to the warm-up time of the Atlas spatial indexing.

6.2 Statistics of Matching Algorithms

In this section we present the results, according to the evaluation techniques in section 6.1, of our matching algorithms. First we will present the results of the naive approach (naive node matching and naive edge matching), followed by the results of node matching by the NetMatcher and finally the results of the expansive edge matcher.

6.2.1 Evaluation of Naive Algorithm

Below, we present the results of the naive approach. This approach is broken into two parts, the node matching followed by the edge matching. The threshold of the naive node matching refers to the side length of the box used for querying corresponding features, as explained in section 4.2.1. The threshold of the naive edge matching algorithm refers to the distance a polygon bound is expanded around the road feature, as explained in section 4.2.2. The thresholds used to produce our results were chosen from a large spectrum in order to see the effects of choosing too small thresholds, bu also to see the effects of choosing too large thresholds.

Naive Node Matching

In table 6.3 and table 6.4, we can see the results of the naive node matching step in the Hedensted and Silkeborg municipalities. These results show the amount of correctly and incorrectly matched nodes in the test area of the data-set. It also present the total amount of matched nodes globally, these include the matches established in the test area, as well as the rest of the data-set. The execution time is also presented in these tables for each threshold.

The naive node matching algorithm matches nodes in the test area very accurately. Since the algorithm relies on the closest feature heuristic, this indicates that the two input data-sets have quite small offset. Furthermore, increasing the threshold value does not negatively affect the correctness of the algorithm within the test areas. Matches which have already been established will not be affected by a larger threshold, since the new nodes introduced by a larger threshold will be discarded by the smallest distance metric. A larger threshold will potentially introduce new node candidates in the target data-set, to unmatched nodes in the source data-set. These candidates could constitute the correct match (see threshold 55m in table 6.3), but sometimes will give rise to an incorrect match (see threshold 3m in table 6.4).

An interesting result is that running time is not affected as much as we thought by larger threshold values. We suspect that the running time of the spatial queries overshadow that of the distance measures of nodes, see section 4.2.1. It is possible to measure specifically the impact of spatial queries in relation to the impact of distance evaluations, however we did not measure this.

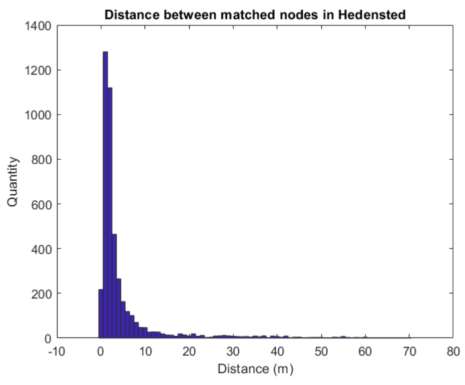
The histograms in figure 6.3a and figure 6.3b show the distance between matched nodes. These histograms indicate that distances between nodes in the two data-sets are exponentially distributed. Specifically they are approximately normal distributed, which aligns well with the findings of Zhang et al. [21]. Zhang noted that the offset between two data-sets are generally approximately normal distributed. The peak quantity of matches is around one meter for both municipalities, based on the two histograms figure 6.3a and figure 6.3b. The histograms also show that there are few matches past 25 meters.

Table 6.3: Results of the naive node matching step in Hedensted. The table shows correctly and incorrectly matched nodes in the test area, the percentage of matched nodes globally and execution times for the different threshold values.

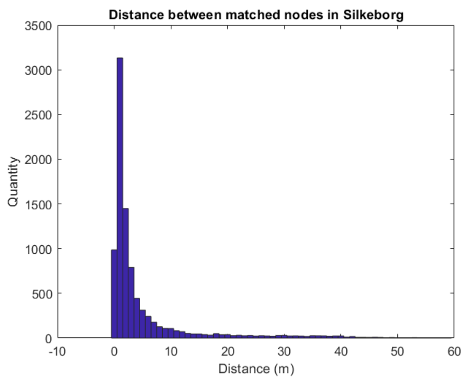
Threshold (m)	Correctly matched nodes (Test Area)	Incorrectly matched nodes (Test Area)	Matched nodes (Globally)	Execution time (ms)
0.1	0	0	0.5%	36
0.5	7	1	6.4%	33
1	15	1	21.9%	29
1.5	26	1	39.1%	29
2	38	1	53.2%	31
3	49	1	68.1%	31
4	50	1	75.2%	31
6	53	1	82.5%	34
8	53	1	86.1%	34
10	53	1	88.4%	32
15	54	1	90.8%	33
20	55	1	92.5%	38
55	58	1	97.2%	45

Table 6.4: Results of the naive node matching step in Silkeborg. The table shows correctly and incorrectly matched nodes in the test area, the percentage of matched nodes globally and execution times for the different threshold values.

Threshold (m)	Correctly matched nodes (Test Area)	Incorrectly matched nodes (Test Area)	Matched nodes (Globally)	Execution time (ms)
0.1	0	0	0.6%	60
0.5	14	0	12.4%	60
1	37	0	31.7%	62
1.5	51	0	45.1%	62
2	59	0	53.2%	66
3	67	2	62.9%	62
4	69	2	68.4%	67
6	71	2	74.2%	67
8	73	2	77.5%	68
10	75	2	79.6%	72
15	77	2	82.8%	73
20	78	2	84.8%	80
40	79	2	90.2%	86



(a) Histogram of distances between matched nodes in the Hedensted data-sets



(b) Histogram of distances between matched nodes in the Silkeborg data-sets

Figure 6.3: Histograms of distances between matched nodes using the naive node matching algorithm.

Naive Edge Matching

In table 6.5 and table 6.6, we can see the results of the naive edge matching step in the Hedensted and Silkeborg municipalities. These results include the number of correct matches in the test areas, as well as the number of global matches. The execution time is also presented in these tables. The edge matching step performs worse than the node matching step. Notably the running time is worse. This is likely due to the large number of calls to the Hausdorff distance computation.

The edge matching does not seem to benefit much from thresholds higher than 4 in either of the two municipalities. The number of correct matches in the test area does not increase, nor does the number of incorrect matches decrease past this point (seen in table 6.5 and table 6.6). Moreover there is little impact on the number of global matches past 4 meters threshold (seen in table 6.5 and table 6.6).

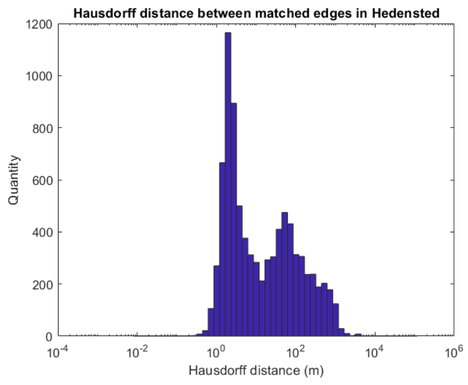
The histograms in figure 6.4a and figure 6.4b show the Hausdorff distances between matched edges in the naive edge matcher. Although many edges are matched with a Hausdorff distance within 10 meters, about 50% are matched with a Hausdorff distance above 50 meters. The reason for this is that the two data-sets are very unlike in detail, as discussed in Section 2.3. The difference in detail means that a matcher which is only capable of 1:1 matches is not suitable for matching edges. The naive algorithm is a 1:1 and 1:0 matcher, and cannot account of the difference in detail, e.g. when one road is represented by three segments in OSM, but only one segment in OpenDataDK.

Table 6.5: Results of the naive edge matching step in Hedensted. The table shows correctly and incorrectly matched edges in the test area, the percentage of matched edges globally and execution times for the different threshold values.

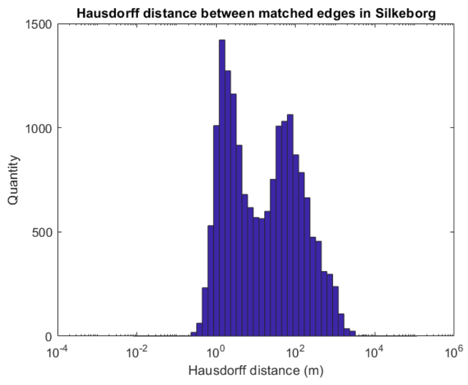
Threshold (m)	Correctly matched edges (Test Area)	Incorrectly matched edges (Test Area)	Matched edges (Globally)	Execution time (ms)
0.1	26	58	61.1%	4184
1	34	62	87.7%	5185
2	40	64	96.1%	5583
4	40	64	98.0%	5484
8	40	64	98.3%	5773

Table 6.6: Results of the naive edge matching step in Silkeborg. The table shows correctly and incorrectly matched edges in the test area, the percentage of matched edges globally and execution times for the different threshold values.

Threshold (m)	Correctly matched edges (Test Area)	Incorrectly matched edges (Test Area)	Matched edges (Globally)	Execution time (ms)
0.1	64	34	68.1%	30689
1	84	34	86.2%	43690
2	88	34	88.9%	49942
4	88	34	90.2%	55244
8	88	34	92.6%	57698



(a) Histogram of the Hausdorff distances between matched edges globally in the Hedensted data-sets.



(b) Histogram of the Hausdorff distances between matched edges globally in the Silkeborg data-sets.

Figure 6.4: Histograms of Hausdorff distances between matched edges using the naive edge matching algorithm.

In table 6.7 and table 6.8, we can see the cumulative length of the matched edges in both data-sets. It is clear that the matched length of the source data-set, OpenDataDK, is much larger. In Silkeborg we see that the matched length of the source data-set is about 1.5 times larger than the matched length of the target data-set. In Hedensted we see that the matched length of the source data-set is about 2 times larger than the matched length of the target data-set. In Section 2.3, we presented that the average length of edges in OpenDataDK is twice that of OSM in the same area. Thus it is reasonable that the matched length in OpenDataDK is larger than that of OSM for a 1:1 matching, since on average the edge in OpenDataDK is twice the length of the edge in OSM. The naive edge matcher produces 1:1 matchings, see section 4.2.2.

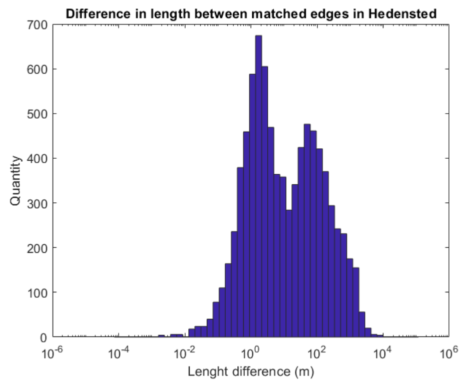
We can see the edge length difference of the matches in figure 6.5a and 6.5b. Notably these two figures have the same overall appearance to that of figure 6.4a and figure 6.4b, which indicates that they are linked. Often, if the length of the two matched edges are very dissimilar, the Hausdorff distance metric will mostly measure the difference in length, which is exemplified in figure 4.1. Matched edges of similar length do not always produce a low Hausdorff distance score, in these cases the largest offset of the two lines are measured, as seen in figure 4.2a.

Table 6.7: Results of the naive edge matching step in Hedensted. The table shows the total length of matched edges globally for the different threshold values.

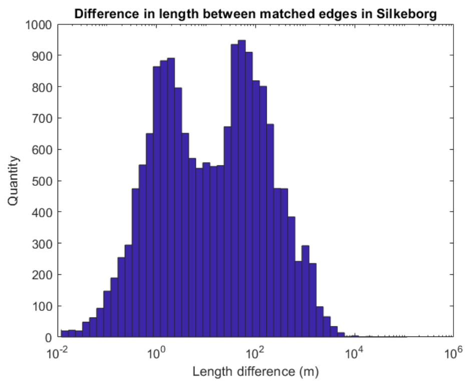
Threshold (m)	Total length matched (OpenDataDK)	Total length matched (OSM)
0.1	2060647	1020250
1	2296147	1229402
2	2347192	1279396
4	2356140	1283819
8	2357680	1291374

Table 6.8: Results of the naive edge matching step in Silkeborg. The table shows the total length of matched edges globally for the different threshold values.

Threshold (m)	Total length matched (OpenDataDK)	Total length matched (OSM)
0.1	4116017	2445400
1	4359252	2849515
2	4384600	2919501
4	4392817	2932680
8	4420850	2958698



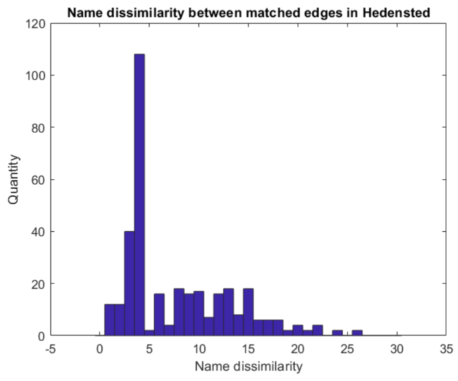
(a) Histogram of the difference in length between matched edges globally in Hedensted.



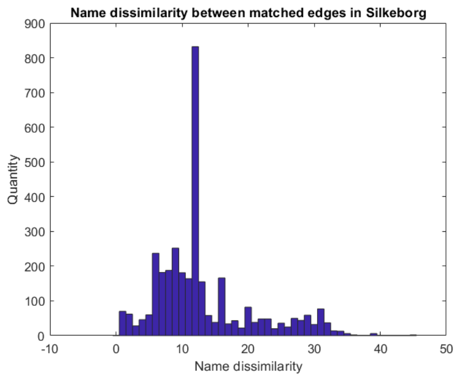
(b) Histogram of the difference in length between matched edges globally in Silkeborg.

Figure 6.5: Histograms of the length differences between matched edges using the naive edge matching algorithm.

Figure 6.6a and figure 6.6b, show histograms of name dissimilarity in the Hedensted and Silkeborg municipalities. In Hedensted there seems to be a large peak at around name dissimilarity score 4, of which a large part is due to an abbreviated term, "Gammel" becomes "Gl". In Silkeborg there is a large peak at name dissimilarity 12, which is due to the name tag including a postfix " - Silkeborg". These cases were verified by manually inspecting flags. Some of the name dissimilarity in both municipalities are due to false matches, and are usually characterized by high name dissimilarity scores. In total 4% of edges in Hedensted are flagged and 19% of edges in Silkeborg are flagged. The reason why there are many more flags in the Silkeborg data-set is that the Silkeborg data-set does not include a name tag, rather a tag called 'designation', which can include more data than only the name, such as country, municipality, city, address range, etc. This additional data is not included in the OSM name tag, and therefore constitutes a conflict in the match.



(a) Histogram of name dissimilarity for matches globally in Hedensted.



(b) Histogram of name dissimilarity for matches globally in Silkeborg.

Figure 6.6: Histogram of name dissimilarity for matches globally using the naive edge matching algorithm.

6.2.2 Evaluation of NetMatcher

This section presents the result of our NetMatcher node matching step. Table 6.10 and table 6.9 present the results of the NetMatcher node matching step. These tables show the amount of correctly and incorrectly matched nodes in the test areas, as well as the total number of matched nodes and execution times for both municipalities. In these tables we note that the execution times are very similar to those of the naive edge matching step, due to the dependence on many Hausdorff distance calculations. The threshold distance refers to the size of the bounding box used in the pre-matching of nodes step. This has little affect on the running time in comparison to the Hausdorff distance calculation. The NetMatcher produces fewer correct matches than the naive node matching step in the test area in both the Hedensted and the Silkeborg data-set. The amount of incorrect matches are the same for both of the algorithms in the test area. Globally the NetMatcher produces fewer matches than the naive node matching algorithm. The reason why the NetMatcher produces fewer matches is likely due to the variance in detail between urban and rural areas in the OpenDataDK data-set, the topological errors described in figure 2.3 and the restrictions placed on matches by the NetMatcher algorithm. Since the NetMatcher relies on one of the data-sets being of consistently finer detail than the other (see section 4.3), the variance in detail is likely a large factor. The topological errors in the OpenDataDK dataset is also the reason why some correct matches in the test area are disregarded, because the NetMatcher algorithm puts constraints on the topology of a match.

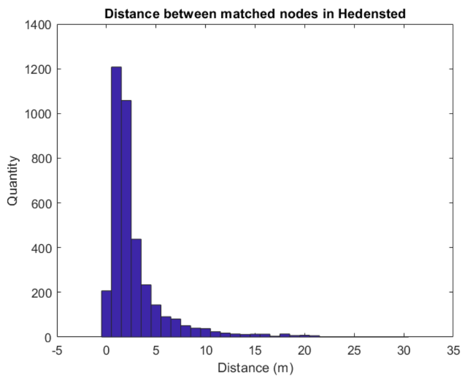
The histograms shown in figure 6.7a and figure 6.7b, show that the distance distributions of NetMatcher is very similar to the distance distributions of the naive node matching algorithm (figure 6.3a and 6.3b). Alike the naive node matchings distance histograms, the NetMatcher distance histograms are approximately normal distributed and aligns well with the findings of Zhang et al. [21]. The NetMatcher distance distribution has a shorter tail than that of the naive node matching since the thresholds tested for NetMatcher were lower.

Table 6.9: Results of the NetMatcher node matching step in Hedensted. The table shows correctly and incorrectly matched nodes in the test area, the percentage of matched nodes globally and execution times for the different pre-matching of node threshold values.

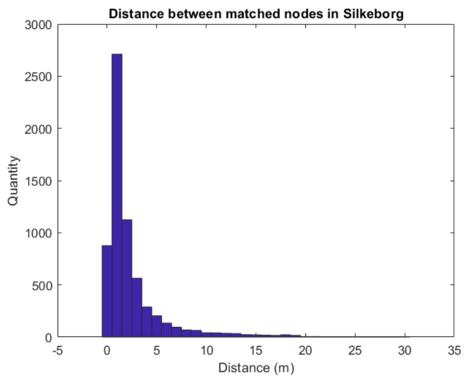
Threshold (m)	Correctly matched nodes (Test Area)	Incorrectly matched nodes (Test Area)	Matched nodes (Globally)	Execution time (ms)
0.1	0	0	0.5%	5383
0.5	7	1	6.1%	5361
1	14	1	20.7%	5606
1.5	24	1	37.0%	5463
2	36	1	50.3%	5450
3	47	1	68.9%	5488
4	48	1	70.8%	5724
6	50	1	76.7%	5393
8	50	1	80.1%	6204
10	50	1	81.4%	5533
15	50	1	83.2%	6064
20	51	1	84.3%	5546
55	52	1	86.4%	5907

Table 6.10: Results of the NetMatcher node matching step in Silkeborg. The table shows correctly and incorrectly matched nodes in the test area, the percentage of matched nodes globally and execution times for the different pre-matching of node threshold values.

Threshold (m)	Correctly matched nodes (Test Area)	Incorrectly matched nodes (Test Area)	Matched nodes (Globally)	Execution time (ms)
0.1	0	0	0.6%	55474
0.5	14	0	11.1%	54871
1	33	0	27.8%	55467
1.5	46	0	39.1%	56306
2	51	0	45.4%	55999
3	55	2	52.2%	55604
4	56	2	55.9%	54686
6	56	2	59.5%	62191
8	57	2	61.3%	60760
10	58	2	62.3%	55148
15	59	2	63.9%	56054
20	60	2	64.5%	54920
40	60	2	65.9%	62937



(a) Histogram of distances between matched nodes in the Hedensted data-sets.



(b) Histogram of distances between matched nodes in the Silkeborg data-sets.

Figure 6.7: Histograms of distances between nodes in the NetMatcher algorithm.

6.2.3 Evaluation of Expansive Edge Matcher

In this section we will present the results of the Expansive Edge Matcher. We have measured results in the test areas, as well as measured Hausdorff distance for the matching groups, the total length of the matches, the difference in length of matches, as well as name dissimilarity.

In table 6.11 and table 6.12 we can see the results of the EEM inside the test areas in Silkeborg and Hedensted. We can see that the ratio of correct matches to incorrect matches is high. The best result was obtained at a threshold of five meters in both data-sets, and produces around 5% incorrect matches in the test areas of both municipalities according to the tables. In Hedensted, the running times are good, the EEM achieves a maximum running time of below six seconds for the largest threshold value. In Silkeborg, the running times are much worse. For low thresholds, up to three meters, the running times are comparable to that of the Hedensted data-set. For thresholds above six meters, running times are approximately 20 times worse than the running times in Hedensted. This is likely due to the exponential time case in recursion, described in section 4.4.2. Aside from the exponential time case, the running time is very good, especially compared to the naive edge matcher. We think the reason for the lower running time of the EEM (Aside from exponential time case), is that fewer Hausdorff distances are computed in total.

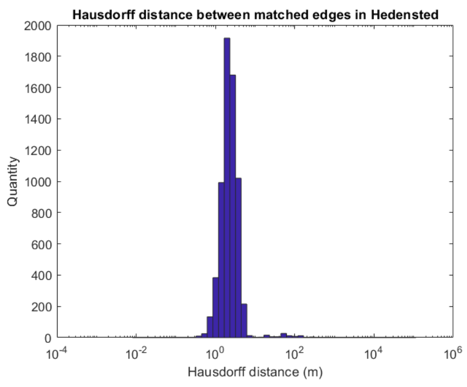
The histograms in figure 6.8a and figure 6.8b are very interesting. What we can see is that the Hausdorff distance between two matched groups is generally very low compared to that of the naive edge matcher. The large majority of matches are found at a Hausdorff distance less than 10 meters, as seen in figure 6.8a and 6.8b. This means that the offset between matched groups are mostly within 10 meters, since Hausdorff distance measures the largest distance between two shapes. The offset of 10 meters can be compared to the average edge length in OpenDataDK, 266 meters (according to section 2.3). This indicates that the matched groups are very similar to one another.

Table 6.11: Results of EEM in Hedensted. The table shows correctly and incorrectly matched edges in the test area, the percentage of matched edges globally and execution times for the different threshold values.

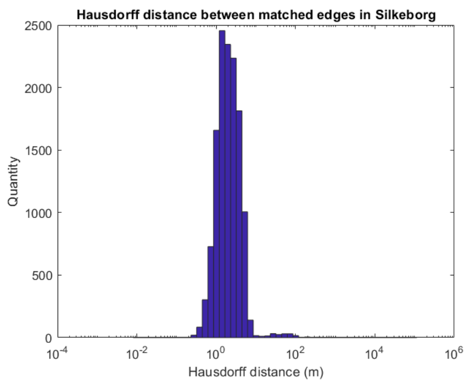
Threshold (m)	Correctly matched match groups (Test Area)	Incorrectly matched match groups (Test Area)	Matched edges (Globally)	Execution time (ms)
0.1	0	0	0%	52
0.5	2	0	0.9%	139
1	10	0	13.9%	408
2	65	2	53.6%	1244
3	82	4	69.3%	1778
4	85	4	76.4%	2109
5	88	4	80.5%	2628
6	89	5	83.8%	4282
7	89	5	85.8%	5861
8	89	5	87.5%	5828

Table 6.12: Results of EEM in Silkeborg. The table shows correctly and incorrectly matched edges in the test area, the percentage of matched edge groups globally and execution times for the different threshold values.

Threshold (m)	Correctly matched match groups (Test Area)	Incorrectly matched match groups (Test Area)	Matched edges (Globally)	Execution time (ms)
0.1	0	0	0%	122
0.5	6	0	5.3%	487
1	56	0	27.1%	2689
2	97	0	53.3%	3831
3	111	6	62.0%	4448
4	115	6	67.3%	36677
5	116	6	70.1%	98426
6	114	7	72.1%	126217
7	113	10	73.7%	125751
8	113	10	75.0%	129084



(a) Histogram of the Hausdorff distances between matched groups globally in the Hedensted data-sets.



(b) Histogram of the Hausdorff distances between matched groups globally in the Silkeborg data-sets.

Figure 6.8: Histograms of Hausdorff distances between matched groups using the EEM matching algorithm.

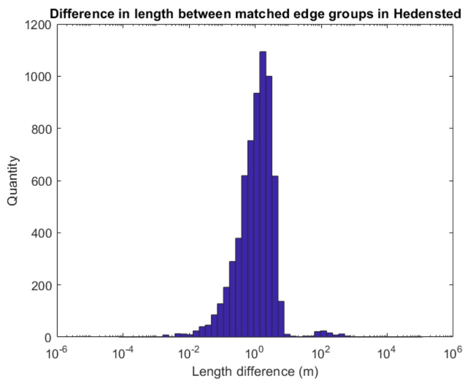
In table 6.13 and table 6.14 below we can see the cumulative length of the matched edges in both data-sets. The cumulative matched length of both data-sets is very similar. This indicates that the EEM works as we intended, by the approach described in Section 4.4. In general we see that the matched length of the source data-set (OpenDataDK) is about the same as the matched length of the target data-set (OSM), within 2%. A cumulative distance difference of 2% between the matched groups can be attributed to the actual difference in length of segments between the two data-sets. Moreover the amount of matched edges in OSM is about twice that of the matched edges in OpenDataDK. This correlates very well with the average length of edges in the two data-sets, described in Section 2.3. The reason why the lengths can be so close in the EEM matches is that EEM supports 1:many and many:many matches, as well as 1:1 and 1:0 in contrast to the naive approach. Similar to the naive approach, we once again observe the similarity between the length difference of edges, seen in figure 6.9a and figure 6.9b and the Hausdorff distances in figure 6.8a and figure 6.8b. This link is expected, and discussed in Section 4.1.1. However it is notable that the difference in length can be very low, as is the case in both Hedensted and Silkeborg, however this is not reflected in the Hausdorff distances. The Hausdorff distance measures the maximum distance between two lines. If one of the lines is longer than the other, the difference in length will be captured by the Hausdorff distance metric, an example of this can be seen in figure 4.1. If the lines are of very similar length, the Hausdorff distance will no longer be affected by this length difference and instead captures the maximum offset between the two lines.

Table 6.13: Results for the EEM in Hedensted. The table shows the amount of matched edges in OpenDataDK and OSM aswell as the total length of matched edges in OpenDataDK and OSM for the different threshold values.

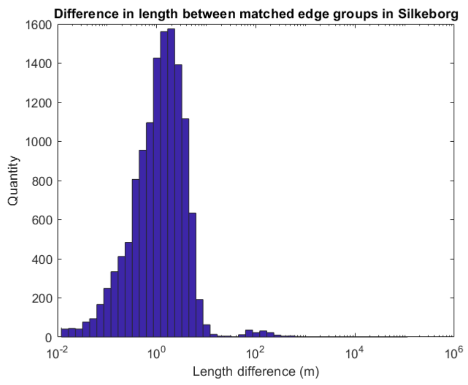
Threshold (m)	Matched edges (OpenDataDK)	Matched edges (OSM)	Total length matched (OpenDataDK)	Total length matched (OSM)
0.1	0	0	0	0
0.5	82	109	8781	8678
1	1239	1975	209967	206809
2	4778	9199	1072726	1059950
3	6184	12588	1509479	1492693
4	6812	14061	1723784	1705240
5	7178	14960	1839645	1821103
6	7469	15809	1970513	1950970
7	7649	16181	2027852	2007587
8	7795	16440	2065087	2044013

Table 6.14: Results for the EEM in Silkeborg. The table shows the amount of matched edges in OpenDataDK and OSM aswell as the total length of matched edges in OpenDataDK and OSM for the different threshold values.

Threshold (m)	Matched edges (OpenDataDK)	Matched edges (OSM)	Total length matched (OpenDataDK)	Total length matched (OSM)
0.1	4	4	311	310
0.5	1069	1487	118432	117485
1	5416	8611	912876	905099
2	10454	18236	2219792	2201218
3	12401	22341	2809621	2785014
4	13454	24547	3075805	3050933
5	14017	25741	3243880	3216863
6	14414	26756	3386840	3358720
7	14743	27481	3494777	3465757
8	15003	28039	3593181	3562861



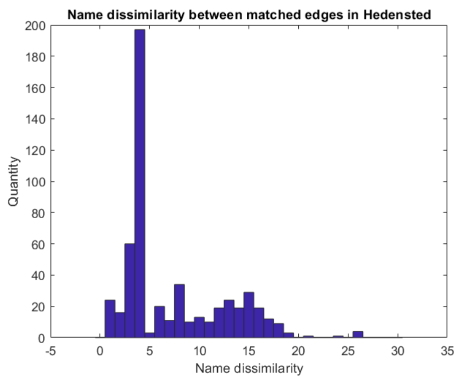
(a) Histograms of the difference in length between matched groups globally in Hedensted.



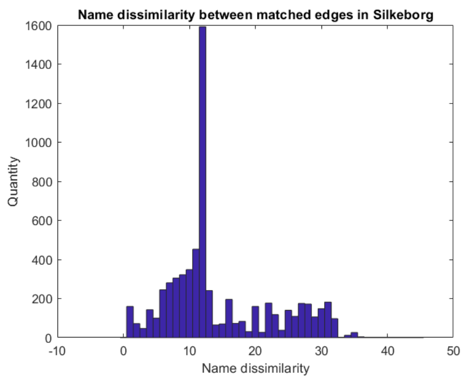
(b) Histograms of the difference in length between matched groups globally in Silkeborg.

Figure 6.9: Histograms of the length differences between matched groups using the EEM algorithm.

In figure 6.10a and figure 6.10b histograms of name dissimilarity of matched groups in EEM are shown. Importantly a matched group can give rise to several flags, since it is the internal relations of the edges that are checked. Also since EEM can produce several matching groups that contain a single edge, flags are over represented in EEM. Most notable is that the shape, and relative size of peaks are very close to that of the naive edge matching step. Once again we can see the peaks at name dissimilarity score 4 and score 12. About 3% of edges have flagged names in the Hedensted data-set and about 26% of edges have flagged names in the Silkeborg data-set. The higher amount of flagged names in the Silkeborg data-set is attributed to the inclusion of meta data in the name attribute as discussed in Chapter 5.



(a) Histogram of name dissimilarity for matches globally in Hedensted.

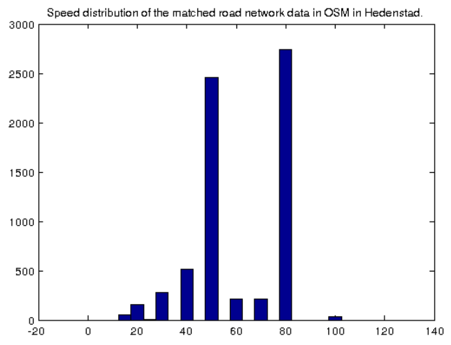


(b) Histogram of name dissimilarity for matches globally in Silkeborg.

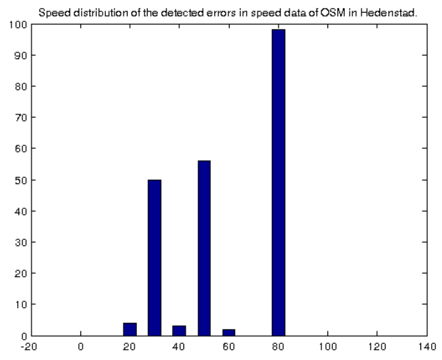
Figure 6.10: Histogram of name dissimilarity for matches globally using the EEM algorithm.

6.3 Speed Limit Errors in Hedensted using the EEM

In figure 6.11a the distribution of speed limits in OSM is shown. We can see that the majority of roads have speed limits 80 and 50. Long roads such as highways, are represented by fewer features, and therefore large speed limits could be underrepresented. In figure 6.11b, we can see the distribution of flagged speeds. Many of the flagged roads have a speed limit in OSM of 30 km/h. We believe that these roads are dis-proportionally flagged due to road boundaries between highways and local town roads. When these boundaries are not aligned in the OpenDataDK and OSM data-sets, the expansive edge matcher will form an incorrect relation between the boundary segments, leading to the speed flag.



(a) Histogram of the distribution of max speeds in the OSM data-set. The Y-axis is the quantity and the X-axis is the max-speed in OSM



(b) Histogram of the distribution of flagged edges in OSM using the speed flag. The Y-axis is the quantity and the X-axis is the max-speed in OSM

Figure 6.11: Histogram of the speed flag error detection using EEM

Chapter 7

Discussion

7.1 The Impact of Data

This section will discuss the importance of good data quality in regards to spatial and thematic data.

7.1.1 Spatial Data

When reviewing results in general for the matching algorithms, it becomes clear that cases of incorrect matching probably have a strong correlation to the topological and spatial errors shown in figure 2.3, as well as the difference in detail between the two data-sets. In general for the naive edge matching algorithm there is a large disparity in matched edge length difference seen in figure 6.5a and figure 6.5b, also supported by data of overall edge length in table 6.7. We observed that the difference in length between matches is typically very large, in some cases surpassing 10 000 meters. Large edge features are more often found in OpenDataDK, as stated in section 2.3. Moreover errors described in Figure 2.3 are quite prominent in the OpenDataDK data-sets, and are also present in the manually matched areas, which evaluate the matching algorithms. The OSM data-set is not always consistent either, but in our experience it has much better quality and coverage than the OpenDataDK data-set. This is quite surprising since much of the work referenced in this thesis expresses concerns that crowd-sourced data, specifically OSM, has worse credibility and quality than authoritative data. This leads us to believe that the OpenDataDK data-sets are of really poor quality. This could help explain some of the poor matching results.

Provided better input data-sets we would expect all matchers to perform significantly better, since the edge cases that they represent no longer need to be dealt with explicitly. However these types of errors represent real world data, and therefore it is important they are represented in the data-sets. Algorithms for any map data-set must therefore be equipped to specifically handle these kinds of errors or, depending on the intention, might detect these flaws and disregard matching them entirely.

7.1.2 Thematic Data

Overall the overlap of attributes in OpenDataDK and OSM is low. As previously stated this can be blamed on the different purposes of these two data-sets. We suspect that OpenDataDK is a data-set acquired by the municipalities in order to know which roads are their responsibility to maintain. Hence there are road names and sufficient spatial accuracy. However features such as speed limits etc. are outside of the scope of this purpose. Many of the values to thematic attributes of the OpenDataDK data-sets in other municipalities than the ones we chose, were riddled with fields containing no information. Therefore we think that perhaps few resources were spent acquiring this data.

Generally the names of the Hedensted data-set are good, meaning that, in a majority of cases they conform exactly to the names of the OSM features. The percentage of exact name similarity matched in the data-set was 96.5%. Moreover the majority of errors are at name dissimilarity 4. This lends credibility to the cases where names are mismatched, and we think it should be a good way of flagging inconsistencies. However name data in the Silkeborg data-set is not of the same quality. Quite often we encountered flagged names that included municipality names, addresses, country name, city name etc. This is why the name similarity metric only obtains 73.5% exact name similarity in Silkeborg. The optimal case is to be able to filter out the street name, and this is done to some extent. However this is a difficult task since the manner in which these values are represented has no clear format. The speed data in Hedensted is very unspecific, meaning that speed limits have large intervals. This makes it difficult to accurately flag for incorrect max speeds in OSM, and also makes the task of correcting OSM impossible in regards to speed in the Hedensted data-set. Due to the unspecificity of speed data in Hedensted only around 220 errors are flagged according to Figure 6.11b

7.2 Matching

This section discusses the results of the matching algorithms we tested. Section 7.2.1 discusses the naive algorithm, section 7.2.2 discusses the NetMatcher and section 7.2.3 discusses the EEM. Finally section 7.2.4 discusses matching in general.

7.2.1 Naive Algorithm

The results of using the naive approach for matching, specifically edge matches which can be seen in table 6.6, are not as good compared to the more advanced approaches. The naive edge matching approach is expected to produce good results when the reference and the appended data-sets are similar, e.g. low offset, similar number of nodes, similar length of edges, since these are optimal conditions for matching using the two provided metrics. But when the data-set contains inconsistencies like the ones presented in figure 2.3, the naive edge matching fails to deliver good results. The node matching step performed above our expectations, being able to match the majority of the nodes in the manually matched test data (see table 6.4). We attribute this to a low amount of spatial discrepancies between the two data-sets. Matches can become ambiguous and probably require more than a 1:1 mapping if the disparity of node quantity between the data-sets are higher.

In areas where the OpenDataDK data-set has poor detail, the naive edge matching algorithm produces worse results. In these areas, using the Hausdorff distance provides a poor metric for the similarity of the edges. Hausdorff becomes a poor metric in any case where the difference in length is large between two edges. Large differences in length arise from e.g. the naive edge matcher. The reason for this is that in general, the edges of OpenDataDK are longer than those of the OSM data-set. Since the naive edge matcher matches edges 1:1, it is often the case that two edges of very different length are matched.

7.2.2 NetMatcher

It is important to point out that our implementation differs slightly from the original NetMatcher. The first two steps of NetMatcher are completely implemented. In the third step, we have implemented 1:1 matching of nodes however omitted the grouping of nodes, since the details of grouping was not detailed in the original article. The omission of the fourth step, edge matching, does not affect the node matching at all. Hence the reason for a poor matching could be blamed on the omission of the grouping of nodes from the node matching step. However we believe that there are more reasons as well.

The results of our implementation of the NetMatcher algorithm are presented in table 6.10 and table 6.9. NetMatcher produces fewer correct matches than the naive algorithm. The NetMatcher also includes less matches in total. This is partly due to the topological constraints put on nodes during the node selection process. In our case this does not contribute to a more correct matching in our test areas. It could be the case that the NetMatcher performs better over the global data, but we cannot verify this using our results. Based on the results from the test areas, NetMatcher could not filter the incorrect matches. We suspect that, since NetMatcher only regards topology when selecting final node matches, NetMatcher will have trouble matching areas with topological errors, since it relies heavily on the topology of incident edges to pre-matched edges in the node matching step. Also since the edge matching step relies heavily on that one data-set is of lesser detail than the other, variations in data-set detail may play a role in the poor matching capabilities when matching nodes using NetMatcher in our thesis. Specifically we observed that the detail of OpenDataDK in urban areas, is significantly better than on open roads. Since our test data is in a city, this might explain some of the poor results obtained for Silkeborg.

7.2.3 EEM

The matches computed by EEM show relatively good results, for a variety of the metrics that were used. Specifically, the EEM manages to match in the test areas with similar accuracy to that of the naive node matching approach, as can be seen in table 6.11. Perhaps this is not surprising, since the heuristic EEM uses for finding potential candidates is very similar to how the naive approach uses a buffer to find a match. The important distinction is that the EEM tests all potential candidates found inside a buffer, thus it is not guaranteed to find the same match as that of the naive node matching algorithm. For example, EEM can discard candidates which do not produce a match in the expansive recursion. It may also choose a match where the corresponding start node is not the closest node. By corresponding the start and end nodes between two matched groups, node matching is implicit by the EEM. This is a consequence of the base cases of recursive expansion (start and end nodes are ensured to be within the threshold distance). Potentially some of the incorrect matches, in the test area, attributed to the EEM are cases that it was not designed to handle, e.g. the double end error where a match is found early.

According to the results, specifically the Hausdorff distance between matches, the EEM shows great potential, see figure 6.8a, figure 6.8b. The relative length of the two matched groups are very similar as can be seen in figure 6.9a and figure 6.9b, and the global matched length is also very similar, see table 6.14 and table 6.13. The largest downside to the EEM as implemented in this thesis is the exponential time case. If this case can be eliminated, the EEM would be considerably faster. The low Hausdorff distance scores, the similar data-set coverage and the similar relative length of matched groups lends credibility to the matching of the EEM. The reasons why the measured lengths are so similar are also due to the start and end conditions placed on any matched group. Moreover the nodes of edges that constitute a match are ensured to be sufficiently close to each other, otherwise they would be pruned in the recursion. Some **current edges** might start and end on the **goal edge**, even though the edge itself might fork from the **goal edge**, see figure 7.1. These edges are still considered a match. The match evaluation step is biased towards selecting the matches of lower Hausdorff distance, hence if there is a more correct edge (By correct meaning contributing less Hausdorff distance), this will be selected instead see section 4.4.2.

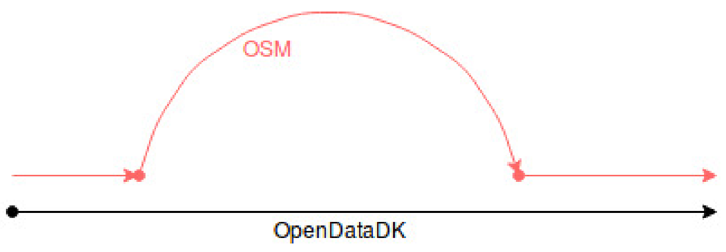


Figure 7.1: A valid match according to the EEM. The fork in the road in OSM is included in the match, since the start and end point and sufficiently close to the goal edge in OpenDataDK.

All of these properties lead to what appears to be a good edge matcher in our data-sets. The percentage of edges that the EEM can map in source to some match in target is around 86%, therefore there is still room for a lot of improvement. It should be noted that in order to achieve 100 %, there can be no features in source that are not represented in target. In our case this is not true, therefore 100% is unobtainable in our data-sets.

7.2.4 Matching in General

A large difficulty in the field of matching is to define good heuristics for evaluating matches. Additionally it is difficult to assess results relative to other work, due to the large variability in data-set quality. In many cases we cannot test our matching algorithms on the same data as the other work since they used proprietary data-sets which we do not have access to.

Another problem when strictly evaluating the correctness of matches is to actually define what is correct. Sometimes when manually matching our data-sets, we had to establish what features constituted a correct match. This is not always intuitive and sometimes there is an ambiguity of what features to include. In cases where ambiguities were found, the most similar feature, when inspected visually, was selected.

In general when studying literature we encountered many matching algorithms that approach the problem by optimizing matches locally. Perhaps matching algorithms that consider global heuristics are better suited for matching. The algorithms we present attempt to optimize individual matches based on some metric, such as minimizing Hausdorff distance. A global heuristic on the other hand, would attempt to optimize the sum of all matches based on some metrics, such as minimizing the sum of all Hausdorff distances. One such algorithm is proposed by Walter et al. [19].

Another problem with matching algorithms, which is hinted at a lot in this thesis is that many matchers rely too much on a single quality of the data-set. For example EEM fails in many of the scenarios described in Figure 2.3. NetMatcher performs poorly if the detail of the two data-sets vary, and naive node matching does not work for edges with large differences in length. Perhaps these problems can be remedied by approaching matching from a more balanced perspective by considering spatial, topological and thematic attributes at the same time. The reason why we only considered spatial attributes in the matching for this thesis, is that we are trying to detect and correct errors in the remaining attributes, thus making them unsuitable for performing matches, since errors might be discarded in such an approach.

7.3 Error Detection and Corrections

As seen in section 6.2 and the name dissimilarity histograms for the expansive edge matcher (figure 6.10a and figure 6.10b), the results of the error detection are actually quite good, in the sense that there are not that many of them. The reason why this is good is since we have decided to not automatically correct errors, in favor of forwarding the errors to humans for manual correction. The amount of flags of names can be seen in the histograms for name dissimilarity in the results, and the amount of flags for speeds can be seen in figure 6.11b. We do not feel comfortable with automatically correcting any errors. The reason why it is difficult to automatically correct, is that it is difficult to decide which of the two data-sets has the correct attribute. Even if corrections are done, it is difficult to verify the validity of the corrections, without tedious manual labour verifying corrections by hand. Perhaps there is a need for labelled data-sets to check correction strategies on. Maybe both data-sets are wrong. However thanks to the low amount of flags shown in the result, these flags can be manually checked by a human. This is of course preferable to manually conflating the entire data-set.

The reason for why there are few flags can be many-fold. We did not have time to properly investigate the reasons behind this, but we have some ideas of why. Firstly, this could signify that the Thematic accuracy of the conflated data-sets is high. Since there are few errors in both the input data-sets in regards to names, there will be few flags. Indeed this seems to be the case, since the number of matched edges is large, yet there are only a couple of flags. It is then reasonable to assume that the majority of matched edges are correctly matched since names are exactly similar in most cases.

As seen in Figure 5.2, an incorrect match will not always provide a flag. This will be the case for matchers that do not pay attention to the order of the roads it matches. For example, the naive approach does not do this. Rather it selects the closest road segment according to the Hausdorff distance heuristic. This can lead to the kind of relational errors of Figure 5.2. However, the implementation of the recursion in the EEM specifically deals with the order of edges, therefore such order related errors can be avoided. As stated in Chapter 5, matches which incorrectly match a road segment to an entirely different road will inevitably lead to errors being detected by the heuristics that are proposed. When we looked at the actual names being flagged, an apparent flaw was found in the matching strategy. Many of the errors detected are errors associated with the boundaries of roads. For example, if Ringvejen takes a sharp corner and becomes Hornvejen, that boundary could form a potential relation and be marked as an error. This hypothesis is further strengthened by the result of the errors detected in speeds. All flagged roads in OSM that have the speed 80 km/h have a relation to either a "TrafikVej" or "LokalVej" in OpenDataDK. This could mean that country roads, entering villages/cities, have a boundary which is not represented at the same location in both data-sets, thus matching those road segments become possible, and a village road, which should have a max speed of 70 (however for LokalVej 50) are flagged.

Even though many of the flagged errors could be due to potentially incorrect matches, these matches have fulfilled certain heuristics which should make sound claims about the geometry being matched. Therefore it could be the case that there is an error concerning spatial positioning of boundaries, or perhaps connectivity.

7.4 Applications

The obvious application is to conflate data-sources with the OSM data-set in order to improve OSM data. Approaches described in this thesis can be applied to attempt to conflate any two geo-spatial data-sets. Since feature matching in geo-spatial data-sets is very similar to shape matching in image analysis applications, perhaps there are applications in that field that can use these algorithms. For example with small modifications, EEM could be used to match partial coastlines represented as vector data. An important consideration when applying these techniques in other fields or in practice is the running time of the algorithms. Both the EEM (In a non-exponential case) and the naive node matching approach have similar running times, while NetMatcher and the naive edge matcher are relatively more computationally complex. We attribute the computational complexity to the high number of calls to the Hausdorff distance function. However it is noteworthy that the EEM encounters exponential running times when the conditions described in Section 4.4.2 are met. This is seen in Silkeborg when the threshold value is higher than four meters, which results in a significantly worse running time. The addition to the algorithm, a stop at 200 matches, allows the algorithm to complete, but is not a viable or generic solution.

Chapter 8
Conclusions

To conclude this thesis we will try to answer our four research questions stated in 1.1.1.

For what types of errors can correction be automated?

For many types of errors attempts can be made to automatically correct them, but it is not certain whether the correction will improve or degrade the quality of the original data-set using our heuristics. In order to make a good correction, an accurate match is needed, in order to identify errors and compare attribute values. Another necessity is a good heuristic to determine how a correction should be made. Without these tools an automatic match could still be done, but the use of such a correction could be questioned. In these cases it can be satisfactory to only detect conflicts in desired attributes, and let a human correct the error.

How do we detect errors reliably using matching strategies?

Reliable error detection is a complex problem which boils down to having a reference data-set of trustworthy quality, coupled with a complete and accurate matching algorithm. Since the data used in this thesis has questionable quality, no accurate way of validating general errors has been found. Moreover the matching algorithms used in this thesis are not sufficiently accurate to reliably detect true data-set errors, rather a source of errors is the matching itself. About 500 inconsistencies could be detected using our approach.

How well do matching algorithms perform on open source data-sets?

The matching algorithms presented in this thesis had varying success when matching the appended data-set to the reference data-set. EEM was the best performing algorithm presented in this thesis. EEM performed well on our data but is believed to need significant improvements in order to provide a more robust matching. The challenge is making an accurate match considering all the different error types that can occur.

How generic is an algorithmic approach to solution suggestion?

There was not enough time in the thesis work to investigate this question. When producing suggestions for error correction using an algorithmic approach, more information about the problem domain makes it easier to provide a good suggestion. However we believe that this might reduce the generic properties of such an algorithm, since it may become tailored to a specific domain.

Remarks

The questions posed in this thesis are quite broad, and we believe that in order to provide more detailed and specific answers, much more research must be done, which is discussed in the next chapter.

Chapter 9
Future Work

This section will present ideas for improvements that have not yet been tested.

9.1 Data-sets

The data-sets from the Danish municipalities could be significantly improved by investigating if some of the errors in figure 2.3 could be resolved by some pre-processing step. This would significantly improve the chances of correctly detecting errors, also corner-cases in the algorithms can potentially be reduced. The errors in figure 2.3 which we think can potentially be fixed are: the double end error (Figure 2.3a), the unclosed loop error (Figure 2.3b), the additional node error (Figure 2.3d) and the disconnected intersection error (Figure 2.3f). Sometimes these cases do not constitute errors, therefore caution must be taken in order to minimize the risk of degrading data-set quality.

9.2 Matching

When studying literature, little information was found on the potential of matching using a global heuristic instead of using the local heuristics that characterize the matchers presented in this thesis. One such article we found is "Matching spatial data sets: a statistical approach" [19]. Hence we think that an important area of future work is to evaluate methods and performance of global matching heuristics, such as minimizing the sum of Hausdorff distances over all matches.

NetMatcher

There is a possibility in the future to implement the entire NetMatcher algorithm, including the last edge matching step. This is detailed in the NetMatcher article [9]. The Net-

Matcher article also suggests some improvements that can be made to improve the matching heuristic. Including the suggested improvements and implementing the edge matching step, would allow us to compare NetMatcher to the EEM.

Expansive Edge Matcher

Our algorithm called Expansive Edge Matcher has several flaws that could be fixed to improve the results.

The first problem is the worst case time complexity of this algorithm. In worst case it is exponential, which is a big problem when there are many nodes close together. One approach to fix this is to come up with a good heuristic so that a match can be evaluated locally in the recursion, instead of the current solution (evaluating the match after the recursion), using dynamic programming. This would mean that, in the recursion, the best result would be known if that particular node have been visited before. One possibility for a local heuristic could be to use the one-sided Hausdorff distance.

The other problem of the recursive step is that it cannot evaluate any edges past the match found in the **Match Found** base case. This means that potentially better matches are discarded and the first encountered match is returned. This can be remedied by evaluating further than a match case, e.g. checking if any edges incident on the node causing the **Match Found** intersects the goal edge with its threshold buffer. Doing so could allow recursion to continue past a match case in order to find potentially better matches.

9.3 Error Detection and Correction

A clear area of future work is to attempt Error Detection on two data-sets with larger overlap of attribution and of better quality. Since overlap was small in our data-sets, a full evaluation of methods concerning Error Detection and Corrections was difficult to accomplish. Moreover errors that were detected might be attributed to errors in the matching process.

In the case of corrections, it would be interesting to evaluate methods for deciding which data-set is correct in case of a conflict, or suggest a separate correction based on the matched pair. Perhaps this can be accomplished with the use of Machine Learning techniques. A machine learning classifier could possibly classify a match as being either a false match or a correct match. It may also be able to solve the trust issue in error correction described in Chapter 5, by classifying which of the data-sets to trust on a per-match basis.

Bibliography

[1] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, and Tim Schaub. The geojson format, August 2016. https://tools.ietf.org/html/rfc7946.

[2] Barron Christopher, Neis Pascal, and Zipf Alexander. A comprehensive framework for intrinsic openstreetmap quality analysis. *Transactions in GIS*, 18(6):877–895.

[3] Heshan Du, Natasha Alechina, Michael Jackson, and Glen Hart. A method for matching crowd-sourced and authoritative geospatial data. *Transactions in GIS*, 21(2):406 – 427, 2017.

[4] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals, 1965. https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf.

[5] Linna Li and Michael F. Goodchild. An optimisation model for linear feature matching in geographical data conflation. http://www.geog.ucsb.edu/~good/papers/510.pdf.

[6] Paul Longley. *Geographic information systems and science*. John Wiley & Sons, 2005.

[7] matthieun. Atlas 5.0.7, 2018. https://github.com/osmlab/atlas/releases/tag/5.0.7.

[8] matthieun. Josm atlas, 2018. https://github.com/osmlab/josm-atlas.

[9] Sébastien Mustière and Thomas Devogele. Matching networks with different levels of detail. *GeoInformatica*, 12(4):435–453, Dec 2008.

[10] Open Street Map. About openstreetmap. https://wiki.openstreetmap.org/wiki/About_OpenStreetMap.

[11] Spatial Reference. Epsg:25832. http://spatialreference.org/ref/epsg/etrs89-utm-zone-32n/.

[12] Günther Rote. Computing the minimum Hausdorff++ distance between two point sets on a line under translation, 1991. https://ac.els-cdn.com/0020019091902338/1-s2.0-0020019091902338-main.pdf?_tid=7d27d705-9023-40e6-976d-c70ffc1bc5bb&acdnat=1527944040_52ff0b9acf0f83a65769002862aa2e34.

[13] Alan Saalfeld. Conflation: Automated map compilation, 1987. https://www.census.gov/srd/papers/pdf/rr87-24.pdf.

[14] Ashok Samal, Sharad Seth, and Kevin Cueto. A feature-based approach to conflation of geospatial sources. *International Journal of Geographical Information Science*, 18(5):459 – 489, 2004.

[15] Wenzhong Shi, Michael F Goodchild, Brian Lees, and Yee Leung. *Advances in Geo-Spatial Information Science*. CRC Press, 2012.

[16] Vejdirektoratet. Ny klassificering af vejnettet, 2008. http://www.vejdirektoratet.dk/DA/vejsektor/samarbejde/kommuner/samkom/Documents/vejklassificering_faser%20og%20trin.pdf.

[17] Vejdirektoratet. Planlægning af veje og stier i åbent land, 2012. http://vejdirektoratet.dk/DA/vejsektor/vejregler-og-tilladelser/vejregler/h%C3%B8ringer/Documents/H%C3%B8ring%20-%20%C3%85bent%20land/Planl%C3%A6gning_veje%20og%20stier.pdf.

[18] Vocabulary. Conflate. https://www.vocabulary.com/dictionary/conflate.

[19] Volker Walter and Dieter Fritsch. Matching spatial data sets: a statistical approach. *International Journal of Geographical Information Science*, 13(5), 1999.

[20] xni06 and nickw1. Jcoord. https://github.com/xni06/JCoord.

[21] Bisheng Yang, Yunfei Zhang, and Xuechen Luan. A probabilistic relaxation approach for matching road networks. *International Journal of Geographical Information Science*, 27(2):319 – 338, 2013.

[22] Meng Zhang, Wei Yao, and Liqiu Meng. Automatic and accurate conflation of different road-network vector data towards multi-modal navigation. *ISPRS International Journal of Geo-Information*, 5(5), 2016.

[23] Alexander Zipf. Quantitative studies on the data quality of openstreetmap in Germany. https://www.researchgate.net/profile/Alexander_Zipf/publication/267989860_Quantitative_Studies_on_the_Data_Quality_of_OpenStreetMap_in_Germany/links/54d99a590cf25013d0426ba0/Quantitative-Studies-on-the-Data-Quality-of-OpenStreetMap-in-Germany.pdf.

EXAMENSARBETE Algorithmic Approach to Error Correction in Map Datasets using Conflation Techniques
STUDENT Linus Röman och Simon Finnman
HANDLEDARE Krzysztof Kuchcinski (LTH)
EXAMINATOR Flavius Gruian (LTH)

Automatisk felsökning i kartor via fusion av flera data-källor

POPULÄRVETENSKAPLIG SAMMANFATTNING **Linus Röman och Simon Finnman**

Digitala kartor är ett verktyg som används vardagligen i moderna samhällen. Dessa kartor måste ständigt uppdateras och rättas när nya vägar byggs, hastigheter ändras och fel upptäcks. Vi har undersökt möjligheten att utföra automatisk felsökning på digital kartdata, genom en teknik som heter sammanslagning.

Sammanslagning av kartdata är en mycket viktig process för att underhålla och uppdatera kartdata. En sammanslagning innebär att två kartor över samma område analyseras och en ny, bättre, karta kan framställas. Detta görs till väldigt stor del manuellt och är mycket tidskrävande. Vi har implementerat automatiska tillvägagångssätt för att slå samman två kartor, med målet att peka ut konflikter emellan de två kartorna.

Kartor beskriver världsliga objekt med hjälp av en slags digitala objekt. Dessa digitala representationer, kräver beskrivande attribut för att vara användbara. Bland annat kan dessa objekt ha namn, en geometri eller koordinater. Det är inte nödvändigtvis sant att två digitala kartor beskriver objekt på liknande sätt, eller att det finns motsvarigheter av alla objekt mellan två kartorna.

Eftersom vi försöker hitta fel i kartor, är vi särskilt intresserade av de fall där beskrivningarna inte överensstämer. Ett fel kan hittas i två steg. Det första steget går ut på att bestämma vilka objekt från de två kartorna som korrelerar. Efter detta steget kan man jämföra beskrivningar, och den omkringliggande geometrin, för att hitta om ett fel har förekommit. Markerade fel kan sedan manuellt rättas.

I vårt arbete tittade vi på kartor i danska kommuner. Vi använder två källor, OpenStreetMap och OpenDataDK.

Korrelationer etableras sedan genom s.k. matchningsalgoritmer. I arbetet testas tre olika tillvägagångsätt. Den första metoden är simpel och bygger på att hitta minsta avstånd mellan de digitala objekten. Den andra metoden matchar objekt baserat på topologi. Den tredje metoden grupperar objekt och försöker sedan korrelera grupper av objekt.

Efter detta studeras korrelationerna för att hitta konflikter. I en metod jämförs textuella beskrivningar av korrelerade objekt. Exempelvis kan konflikter i namn hittas på detta sätt. Utöver detta verifierar vi hastighetsbegränsningar i en av kommunerna.

Våra resultat visar att gruppering av objekt är det bästa sättet att skapa korrelationer mellan digitala objekt, av de metoder som vi har testat. Vi lyckas även hitta ett hundratal konflikter mellan attributen av de korrelerade objekten. Totalt markeras 3% av namn, och omkring 3% av hastighetsbegränsingar som felaktiga.

I framtiden tror vi att mer kartdata kommer samlas in automatiskt, därmed kommer behovet av automatisk felsökning och rättning att öka.