Optimizing the Process of Handling Road Network Data

Martin Nordmark
Abstract

Optimizing the Process of Handling Road Network Data

Martin Nordmark

Route optimization for different types of land-based vehicles is the process of finding the most efficient route. To achieve this, all necessary calculations throughout the process require road network data of the relevant surroundings. In order to prevent erroneous calculations and results, it is crucial that the data is correctly prepared and adapted for the algorithms and systems performing the route optimization. The company B&M Systemutveckling is active in this area and their process of handling road network data is troublesome in certain factors. This may harm the possibility to make further progress in some fields. In order to improve the process, this thesis analyses three problematic factors which are time consumption, correctness and the required attention from the user. This was done by analysing all parts of the process and acknowledging if and how these factors could be improved.

With the results of the analyse in mind, a new improved system was designed. The main focus was to improve these three main factors that are considered especially problematic in the process. A prototype was then built from this design which turned out to be a powerful upgrade of the currently used process, making improvements in all of the main problematic factors. The time consumption decreased from at least one week to one hour. The attention required from the user decreased immensely resulting in a decreased risk for human errors. Utilization of more suitable data structures and algorithms combined with the decreased risk for human errors had a beneficial impact on the correctness.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td>3</td>
</tr>
<tr>
<td>1.1 Purpose</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Contribution</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Basic data</td>
<td>5</td>
</tr>
<tr>
<td>2.3 GIS software</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Importfile</td>
<td>8</td>
</tr>
<tr>
<td>2.5 ROA and COO</td>
<td>8</td>
</tr>
<tr>
<td>2.6 Compilation</td>
<td>8</td>
</tr>
<tr>
<td><strong>3 Analysis of the process</strong></td>
<td>10</td>
</tr>
<tr>
<td>3.1 Improvements</td>
<td>11</td>
</tr>
<tr>
<td><strong>4 Designing the Solution</strong></td>
<td>12</td>
</tr>
<tr>
<td>4.1 Method of Choice</td>
<td>12</td>
</tr>
<tr>
<td>4.1.1 Evaluation of Methods</td>
<td>12</td>
</tr>
<tr>
<td>4.2 Creating the Importfile</td>
<td>13</td>
</tr>
<tr>
<td>4.3 Creating ROA and COO</td>
<td>13</td>
</tr>
<tr>
<td>4.4 Final Sketch</td>
<td>14</td>
</tr>
<tr>
<td><strong>5 Implementation</strong></td>
<td>15</td>
</tr>
<tr>
<td>5.1 Data Structures</td>
<td>16</td>
</tr>
<tr>
<td>5.2 Data Modifications</td>
<td>17</td>
</tr>
<tr>
<td>5.2.1 Projection</td>
<td>18</td>
</tr>
<tr>
<td>5.3 Compilation of the Network</td>
<td>18</td>
</tr>
<tr>
<td>5.3.1 Unwanted Links</td>
<td>21</td>
</tr>
<tr>
<td><strong>6 Result and Evaluation</strong></td>
<td>21</td>
</tr>
<tr>
<td>6.1 Time Consumption</td>
<td>21</td>
</tr>
<tr>
<td>6.2 Required Attention</td>
<td>23</td>
</tr>
<tr>
<td>6.3 Correctness</td>
<td>23</td>
</tr>
</tbody>
</table>
7 Conclusion

7.1 Future Work

References
1 Introduction

Route optimization is the process of finding which set of routes that are the most optimal to take while still fulfilling the essential tasks specified by the user [1]. It can be seen as a modified version of the traveling salesman problem [2]. Finding the best possible route is important in terms of both environmental and efficiency aspects, as well as safety aspects. The reduction of fuel consumed is beneficial both economically and environmentally [3]. Less time spent driving will reduce the wear and tear of the vehicle and the risk of accidents. There might also be potential improvements of customer service and productivity.

There are multiple different algorithms and approaches to this problem, requiring several factors to work correctly. One crucial element is data of the environment and surroundings in action. To make the result of the optimization whatsoever usable, gathered road network data must be correct and updated for the essential and necessary calculations. This data can appear as completely different formats with separate approaches.

The company **B&M Systemutveckling** (B&M) see this subject as an important business area in the near future [4]. The road network data used in their system is gathered from different data sources. Their process of handling and converting this data requires several manual steps as well as some idle time in between that can both be really time consuming. The requirement of manual input also introduces the risk of small mistakes which can lead to errors and faulty solutions that are difficult to detect. With all of this in mind, there is a need to improve and optimize this process. This will be done by focusing on three main factors that, when improved, will have the most beneficial impact on the end result. These are correctness, time consumption and the amount of effort required from a user.

1.1 Purpose

The aim of this thesis is to optimize and improve B&M’s current process of handling road network data gathered from the transport administration in Sweden.

1.2 Contribution

The following contributions are covered in this thesis:
• Analyzing the currently used process to discover potential improvements. This include examinations of the relevant data structures and evaluations of the different steps in the process. [3]

• Evaluating the different possible approaches to a solution resulting in a design of a new process. [4]

• Building the system. [5]

• Evaluation of the prototype. [6]

2 Background

The basic road network is gathered mainly from three different data sources with focus on different geographical areas. The procedures of processing and handling data from these three sources are rather similar but they have their differences. This thesis is focusing on data gathered from one of these sources containing geographic information about Sweden. To accomplish the desired task of optimizing the process, a full detailed investigation of all actions is required in order to be able to design a good solution.

This section gives a detailed overview of the whole procedure of developing complete data from this specific data source as well as information about all the important components and data structures. The process consists of up to one hundred different steps where the majority of them require manual input by the user.

2.1 Overview

To get an illustrative overview of how the process works, see figure [1]. The user starts off by identifying the requirements and chooses a suitable data source. Basic data from the source is acquired with the desired geographical information. The data is regularly gathered as shapefiles, although the possibility to choose other formats exists [5]. These shapefiles contain information about roads and environment in the chosen region. Before the process of handling this data can begin, the files need the correct format and projection [6]. The format TAB and the projection WGS84 are used due to being a requirement for the GIS software which is used [7, 1-3][8]. The name of the software is MapInfo Professional and it assists the user to
modify and update the majority of the data \[9\]. Different calculations are made, removal of faulty and undesired data, merging of tables etc. All these tasks result in an **Importfile**. This file is stored in a database for future use before being utilized in the next step of the process.

**Compilation** of all the data is then initialized with the help of the importfile and the unmodified shapefiles. This is essential for the data to be useful for the system and require a lot of different steps and calculations. The main purpose of this part is to connect and assemble each part of the modified data. This compilation results in **ROA and COO** files which can be used by the system for different kinds of route optimization algorithms.

![Diagram of the process](image)

**Figure 1:** Illustration of the whole process.

### 2.2 Basic data

The required data is mainly gathered as **Shapefiles**. This is a geospatial\[1]\ data format for **geographic information system** (GIS) software [1][2]. The possibility to gather data in different formats exists, however that is rarely done in this case. The recent popularity of shapefiles has improved GIS software support of the format, either by directly supporting shapefiles or by

---

\[1\]Data and information having an implicit or explicit association with a location relative to the earth.

---
giving the ability to convert them into other supporting formats. The format is also very simple to work with which make it even more appealing.

Each of these files contains different attributes which all cover different parts in the geographic data. The geographic data contains information of all roads and areas for the specified region which is everything from road name and speed limit to the length of each road. In this work, some available information in the shapefiles is not relevant for the end systems. Although, all the data combined does not always contain all the essential information for every part of the road network. Some data must be modified before being useful for the end systems. Therefore, there are a lot of different steps in order to produce complete data.

2.3 GIS software

To update and modify the data, the tools and features of a GIS software is utilized. Such a system can capture, store, manipulate and present geographical data [8]. The GIS software used is MapInfo Professional (MI). The shapefiles are first converted into TAB which is a format intended for MI [9]. This is done only for the use of MI as it does not give any advantages over shapefiles when it comes to the process itself. When all the essential files are in TAB format, the user can then start to execute all the required tasks. The main goal with all these tasks is to create an Importfile.

When opening a file in MI, there are several ways of representing the data. An example can be seen in figure 2. The user can modify every cell in the table, add new attributes, merge tables and much more. All these modifications and changes can be done completely manually, doing that would however take far too much time, especially for larger areas. Additionally, the risk of errors is rather significant which will delay the process even further. To save time and simplify the work for the user, several scripts have been created for the majority of the tasks. These scripts are written in MapBasic which is a programming language used to create custom applications on top of MI [10].
Figure 2: Example of an opened file in MI that contains the base network of Jönköping in Sweden. Every attribute and all the data can be seen in the left window. A part of the data is visually presented in the right window.

Each of these tasks updates the importfile in one way or another. The first step is to create the importfile and give it the base network of the region. As seen in figure 2, the base network contains RLID, Start Distance, End Distance, From Date, To Date and Length for each road link. The RLID is a unique id for each road link, see figure 3. After this, each task keeps on adding new attributes and data. Most of the tasks are rather straightforward as they are about collecting and adding desired data to the importfile. Nevertheless, some parts are a bit more complex. One important task is to update the id of each road link correctly. One road link can potentially be divided into smaller parts, as seen as “LinkID” in figure 3. This is due to changes of different kinds in the road link. It can be new speed limits, crossings, one-way parts, etc. The problem is that an specific id does not exist for these smaller links in the basic data. It is essential to be able to distinguish these links to get the best possible results from the algorithms. Therefore, an internal id for each link is specified and inserted into the importfile. When creating an importfile, all of these internal ids must be correctly obtained and maintained from the importfile made with the previous version of the road network.
2.4 Importfile

The end result of all the tasks done in the GIS software is an **Importfile**. The file is in TAB format since it is created in MapInfo Professional. A finished importfile contains all relevant information about the specified region. This importfile is stored into a local database for future use before being used in the compilation. The information in the database is essential when a new importfile is to be created due to the internal ids of each link. It is also a useful source for the end system when doing different calculations.

2.5 ROA and COO

The outcome of the whole process are a few files in B&M's internal formats. These are called **ROA** and **COO** files. The ROA file contains all necessary information about each road link and the COO file contains all coordinates for each road link. The ROA and COO files are both in binary format in order to make the files as fast and small as possible. Aside from those files, some text files containing additional information about the road network can also be generated if desired. All of the produced files are used by the end system when different kinds of route optimizations are made.

2.6 Compilation

The compilation is divided into two steps. Both steps are done with internal programs. The first program is built in the programming language **C++** and with the software **MapX**\[^2\]**. MapX is a mapping ActiveX control\[^2\] that can be used to create and execute applications over the Internet through web browsers.

\[^2\]Can be used to create and execute applications over the Internet through web browsers.
allows for the possibility to add powerful mapping capabilities to applications in Mapinfo [13]. This gives the user a visual presentation of the data which is helpful in order to find potential errors. The main purpose with this step is to, with the help of the importfile, connect every road link together correctly. This results in an internal format where the road network is seen as links and nodes, see figure 4. Every link has a start node and an end node. It is crucial for the route optimization that every node knows all adjacent roads and nodes it is connected to. It is also important to keep the coordinates of all points on the links as the drawing of the map would not be correct otherwise, after the compilation. The system could then misinterpret what road the user is located at which could lead to faulty and bad route choices. This program results in ROA and COO files. These are the final files that will be used in the end system. However, they still need some modifications before being completely finished.

Figure 4: Example of three road links. The compiling process make sure that each node knows all roads and nodes it is connected to.

The second step is done with another program built in the .NET framework using the programming language C# [14]. The purpose of this program is to update and add important information about each link. Examples of important fields are road class, allowed directions, road type, allowed vehicles, etc. The original compiling program cannot do this due to lack of functionality and data. The majority of this information is not needed in the database containing information from the importfile, resulting in it being
3 Analysis of the process

To find potential improvements and optimizations in the process, all problems must be identified and investigated. There are three main factors that will be focused on when analyzing and evaluating the process. These are correctness, time consumption and the amount of effort required from a user.

The correctness of every single step in the process is crucial. Any mistakes or errors might lead to broken solutions and results. These mistakes are not easy to identify and unfortunately there is no good way to test everything until the compiling part. Therefore, the majority of the mistakes are found in the latter parts of the process. The whole process must then be repeated from the beginning which is a huge waste of time. The most common mistakes are caused by human error. Even though the user has access to a lot of scripts, it does exist parts that the user is forced to do manually due to lack of appropriate scripts. Thus, to improve the correctness of the process, there is a need to automate the steps that does not require input from the user.

The time it takes from gathering the basic data to having complete compiled data ready to be used is different each time. It depends on several factors that are hard to pass by in the current process. Firstly, each of the scripts focuses only on one task. This means that each time one script is done, the next one has to be executed by the user. This causes problems since, depending on the difficulty of the functionality and the amount of data, a script can take anything from a couple of minutes up to a whole day to execute. The user cannot sit and wait between each script for it to finish. Other assignments and work might come between. Therefore, there is a lot of wasted time where the whole process is idle.

Another problem is the scripts themselves. Some of them are not optimally written and could possibly be improved. A lot of functionality involves finding matches between links in different files in order to update or modify them. Checking every link is really time consuming when the number of links reach a couple of millions.

The process does not require a lot of active attention from the user. It
does, however, require the user to constantly go back to the process when input from the user is needed. This is something the user is required to do during the whole process which is inconvenient. Most of these situations in the process does not actually require input from the user to work functionally, it is only necessary in order to keep everything running. Nevertheless, some parts require manual changes to the data. The basic data received from the data source might have some faulty road links. This forces the user to fix them manually and is done right before the compilation with the help of MapInfo.

3.1 Improvements

Overall, it would be much more convenient for the user to only have to make an effort on the essential parts. The first step of the compiling process is done by an internal program that is starting to get outdated. It is a 32 bit software that cannot handle large files. In cases where the targeted areas are huge, the ROA files must be divided into smaller parts for it to work. Thus, the compilation part is in need of a new replacement program, targeting 64-bit machines. If possible, the implementation should be written in C#. It would be very convenient since the majority of the applications produced by the company are written in C#.
4 Designing the Solution

In order to build a good prototype and improve the process, it is essential to carefully think through each step and produce a well-planned design. There are a large number of different ways to go when optimizing this process. All choices must be made after thoughtful consideration of all identified problems and requirements.

4.1 Method of Choice

There exist two different major methods of doing the whole process. One is to proceed with the same applications and tools that are being used in the current process. This would mean that all the data goes through MapInfo and other existing software.

The second would be to do the complete opposite and skip all the tools that are not considered important for the process. This would require new functionality for all the parts that were handled by these unwanted tools.

4.1.1 Evaluation of Methods

Using the same tools would be very convenient. It would allow the use of already working methods. All the scripts made for MapInfo could be utilized. Compiling the data could be done with the internal compilation program.

One huge advantage with MapInfo is the smooth ability for the user to interact with the geographical data. It is possible to build a program that can create instances of MapInfo using C# in the .NET development platform [10]. This opens up the possibility to completely automate the first part of the process by consecutively calling every script on these instances. The user would only have to start the program and manually fix the broken links when it is necessary. The idle time of the process would be removed, and the program can compile the data by executing the compiling program when the first part is done.

The problems with this method is the fact that the internal compilation program and MapInfo would be used. Even though the compilation program is able to do its job with some struggles today, it is not sustainable in the long term. Using MapInfo could work but the lack of user activity really disputes the need of it. A GIS software allows the user to produce graphic displays of the geographic data for analysis and presentations. Making this
part fully automatic with no interaction from a user would almost make the
use of a GIS software useless. It would, however, be needed for the manual
modifications, but not for the rest of the process.

Proceeding with the method of doing everything from scratch would elim-
inate these problems. The importfile would not be created through MapInfo.
Instead there are other possible ways of doing it directly from the shapefiles.
A GIS software could still be used when manual modifications is absolutely
necessary. Implementing the functionality from all the scripts from scratch
would be easier to optimize than trying to improve already written scripts.
A new way of compiling the data would be needed, thus opening up the
possibility to fix the problems with the current compilation process.

Therefore, everything will be done from scratch with a few exceptions.

4.2 Creating the Importfile

As the new system will mainly not be using MapInfo when creating the
importfile, the procedure will look rather different. The best way of doing
this would be to go directly from the basic data to the importfile. Since
the system should be written using the .NET framework, the ability to fully
access the data from this platform is essential. This support exists for the
shapefile format which is very appropriate. Thus, as before, the basic data
will be gathered as shapefiles. The functionality of all scripts will then be
used on the data to finally produce a finished importfile.

The problem here is potential broken basic data. Making the system able
to fix this data, if even possible, will not be implemented in this system.
Consequently, the user must manually fix the faulty data somewhere in the
process, presumably with MapInfo. The most suitable place for this to hap-
pen would be before the system is started. Doing it after the whole process
would have no impact on the end result at all, and doing it in the middle of
the process would split the system into two parts which is neither practical
nor user-friendly. Therefore, it is up to the user to make sure that the basic
data is as complete and correct as possible when starting the system.

4.3 Creating ROA and COO

The new system will try to merge the two parts mentioned in 2.6 that are
currently being used to produce the final ROA and COO files. The first part,
the internal compiling program, offered a lot of visual presentation and tools
outside of the compiling itself. Unlike that program, the new application will only compile the data and not offer other tools as they are not required anymore. If done correctly, the data will not require any error searching or modifications when finished. The second part that involves updating the roads with new information will be done simultaneously with the compilation.

4.4 Final Sketch

Merging everything together results in a different design than the current one. The new system will generate both the importfile and the ROA and COO files, as seen in figure 5. As mentioned in 4.2, it is up to the user to send in correct data into the program. Therefore, optional modifications of the basic data might be necessary before initiating the process.

![Diagram](image)

Figure 5: Design of the new process.
5 Implementation

The program is written in C# on the .NET platform, which is preferable by the company, and is targeting 64-bit machines. As mentioned in section 3, the amount of memory available on a 32-bit application is not enough for the data that is processed.

When the program is initialized, an interface appears that allows the user to specify where the basic data is located, the desired projection and the destinations and names for the resulting files, see figure 6. The basic data should not be modified between being downloaded and handled by the program, with the exception of fixing faulty road links. When the process is started, a backgroundworker is initialized and assigned all the work that needs to be done [18]. A backgroundworker is a type of thread that offers simple progression updates. The thread reports back to the parent about how much of each task that is done. The information is used to notify the user about the whole progress by using a window that pops up as soon as the process is started, see figure 7.

Figure 6: The main menu for the application. Allows the user to specify information about data and files.
5.1 Data Structures

All the data is gathered from the shapefiles with the help of an API called SharpMap\[3\]. The desired data is stored in different classes and data structures [14]. The essential properties for a road in the importfile are stored in a class called ImportData. An object of this type is created for each unique road link that is read from the basic data.

Every internal ID is fetched from the old importfile and assigned to the new links. To constantly find and update properties for the correct road link, every object of this type is stored in a dictionary called importDict. It is structured as \(<\text{string, list<ImportData>>}\) where the key is the RLID of the link. Since the internal ID for each link does not exist in the shapefiles, having the internal ID for each link as the key would not contribute to speeding up the process. The value assigned to each key is a list containing all links with the key as their RLID, see figure 8.

To find the correct link in that list, the start and end points for a link is used to match with the desired link. Therefore, instead of having the search space for a link be the whole network, it is limited down to the number of links with the same RLID. Each property in the importfile is directly gathered from the basic data without any calculations being necessary.

\[3\]https://github.com/SharpMap/SharpMap
Figure 8: Example of an entry in the dictionary containing information for the importfile. Each RLID has a list with smaller links within the road. Each link is then found in the list by comparing their start and end positions on the whole road.

The information for the ROA file is stored in another class called \textit{GraphRoad}. This class contains all required properties for a road in the ROA file. As with the other class, each GraphRoad is also stored in a dictionary, \texttt{<int, GraphRoad>}. However, instead of using RLID as a key, the internal ID of each link is being used. The majority of the data in the ROA and COO files is updated simultaneously with the importfile, making it useful to have the ID as the key. When the correct link has been found, the ID can be gathered from the importDict with the same approach as before.

5.2 Data Modifications

In an optimal situation, every link contains one value for each property. That is unfortunately not the case which forces some different strategies and rules in order to handle links with multiple values in a good way. These set of rules were already decided and used in the previous process.

For most properties, the data is chosen based on priority. A link that has different information about maximum vehicle weight will have the lowest value prioritized. If a link has multiple road classes, the highest one is prioritized. Each field has different priorities depending on the property itself.

Properties like speed limit and road width are special cases where an average is chosen. The values are calculated based on how much each value covers the link.
5.2.1 Projection

The standard projection for the incoming data is called \textit{SWEREF 99 TM}. The user can decide, from a list with supported projections, in which one the system should compile with. All the supported projections are specified before. It is possible to convert the projection of a coordinate by using a library made by B&M. Therefore, the system can convert all the coordinates from SWEREF 99 TM to the specified projection during run time.

5.3 Compilation of the Network

The basic data gathered from the data source unfortunately does not contain information about how road links are connected to each other. Therefore, when connecting the whole road network, the coordinates of the end and start node for each link are used. Each node must then be correctly connected to other nodes depending on their respective position. The nodes that either have the same position or are really close to each other are seen to be connected, see figure 9. Each node receives a \textit{Node ID} depending on the adjacent nodes they are connected to. The nodes that are supposed to be connected get the same ID and are treated as one node by the end system.

![Figure 9: Visual presentation of how road links are seen before and after compilation.](https://github.com/viceroypenguin/RBush)

The size of the targeted region can be huge and include millions of road links. Connecting the links to each other can therefore be really time consuming. Instead of matching every node with all the other, a much more effective way of connecting them would be very beneficial. The implementation of the compilation of data uses an optimized version\footnote{https://github.com/viceroypenguin/RBush} of the data structure \textit{R-tree}.
An R-tree is a height-balanced tree commonly used for storing and accessing information about objects related to a space such as geographical coordinates, rectangles and polygons. The idea of this data structure is to group nearby objects and represent them with their bounding box in the level above of the tree, see figure 10. Since it is balanced, all leaves appear on the same level. At this level, each bounding box describes and refers to a single object. At higher levels, the bounding boxes contains multiple objects. The search algorithm descends the tree from the root by using the bounding boxes to decide whether or not to search inside a subtree.

Figure 10: Simple example of an R-tree for 2D rectangles. Retrieved from: https://en.wikipedia.org/wiki/R-tree#/media/File:R-tree.svg
Every node of the specified region is inserted into the tree with a minimum bounding box. The bounding box is created given a specified length and consists of four points, one for each corner of the box, see figure 11. The length depends on the correctness of the coordinates for each node. A really small length lower the amount of nodes to check, but it might result in missing nodes due to not being in the bounding box. The length is by default set to 1 meter as it provided the best result in both accuracy and time taken. It can however, be set to something else.

![Figure 11: Example of a bounding box for a given node.](https://github.com/scottschluer/geolocation)

When every node in the whole network is in the tree, the process of connecting the nodes starts with a traditional recursion. The tree is iterated, making sure that every node has been visited when done. When one node has been visited and has received an ID, the adjacent nodes inside the bounding box are recursively visited and given the same ID, see figure 12. When all the adjacent nodes have been visited the next node in the tree that has not yet been visited do the same procedure with a new ID.

![Figure 12: The adjacent nodes inside the bounding box will be visited and given the same ID as the current node.](https://github.com/scottschluer/geolocation)
5.3.1 Unwanted Links

Certain links are not desired and are removed from the network completely. Small links exist in the basic data that do not contribute to any functionality. These links can be troublesome since both their start and end node can be inside a bounding box. A chain of smaller links that are connected can potentially result in gaps between roads, if the distance exceeds the bounding box. Removal of them is therefore done after the compilation when each of these smaller links start and end node are seen as the same node, making it safe to remove them. Any link that is below half a meter will be removed. The limit was decided together with the people at B&M and can be changed if desired.

In the current prototype, the user can also choose to remove walking and cycling paths.

6 Result and Evaluation

To evaluate the prototype, different tests and comparisons with the previous system has been made. The main focus in evaluating the prototype are correctness, required effort by the user and time consumption. All these factors were analyzed on the original process in section 3.

6.1 Time Consumption

All experiments and test when measuring time were done on basic data covering every part of Sweden. It is hard to get an exact number of the time taken for the original process because of idle time and potential errors. Since those factors has been two of the main problems with the original process, it will be taken into account in the evaluations. Going through the original process again would take a lot of time and, due to lack of experience, could potentially result in an inaccurate representation of the process. Therefore, the numbers for the original process are estimations from older procedures done by employees of B&M. From the estimations, completing the process could take anything from one week up to two weeks, excluding weekends.

To measure the time taken for the prototype to finish, five runs were made, with each run having the same settings. The time taken for the user to gather the basic data and make potential fixes was not included in the tests. The focus was on the parts handled by the prototype. The results can
be seen in figure 13. The difference in time between each run most likely depends on the amount of other work the CPU had to do. Other factors could be how the Rtree is built and in which order the links are handled. In any case, a powerful improvement can be acknowledged when comparing those numbers with the original process, which can be seen in figure 14. What can be estimated takes at least 120 hours doing it manually, including potential errors, delays and idle time, takes under an hour for the prototype. The estimated time includes all hours of the day since one of the problems with the original process is the idle time after working hours.

Since almost everything are processed simultaneously, it is hard to compare each part separately. When the importfile is finished, a large part of the ROA and COO files are also finished. Therefore, it would not give a fair comparison between each part of the original process and the prototype.

Figure 13: Diagram showing the time taken for the prototype to finish, not including the procedure of gathering and fixing the basic data.
6.2 Required Attention

The amount of time the user have to spend on the prototype is minimal. The main responsibility for the user is the basic data that needs to be downloaded and modified. Aside from that, the user just needs to start the prototype with the correct settings. When it is started, it handles everything on its own without any needed input.

It is hard to do an exact comparison with the original process since there is a lot of idle time between each step. However, apart from preparing the basic data, the user needs to constantly check the progress of each task in order to know when to start the next one. Extra steps before and after the compilation are also done by the user. Thus, instead of having to be somewhat active throughout the whole process, the user can start the prototype and receive the finished files when the process is done.

6.3 Correctness

Testing if the produced data is correct turned out to be a bit tricky. One of the largest problems in the original process, human errors, has almost been
completely removed. The only way for human errors to affect the result is if the gathered basic data is not correct when initiating the system. However, the produced files can still contain faulty data due to different implementation issues which must be tested.

Since the importfile only contains values that are stored in a database, the best way to test it is to compare it with the existing importfile, made with the original process, and assume that it is correct. Since the importfile contains millions of different values an effort to make automated tests were made. Unfortunately without any progress due to some properties in the previously made importfiles having preset values. These preset values would often not match with the gathered data which would lead to faulty results. Thus, multiple random samples were tested, giving good results with the majority of the tested data being identical. The few differences were due to the original file having these preset values for some properties instead of gathering them from the basic data which the prototype does.

In order to check the correctness of the produced ROA and COO files, the latest basic data of the region Uppsala was gathered. The same data was later processed with both the original system and the prototype resulting in two versions of ROA and COO files. This was done to get the most accurate comparisons possible. Several random start and end destinations were then inserted into an application using the files. The produced routes for the different files could then be compared and evaluated, see figure 15 for a few examples.

The small difference in distance is due to the start and end points not being exactly at the same position. These positions are specified by a mouse click and not by inserting exact coordinates which may result in some differences, especially for larger scaled areas.
Figure 15: A few examples of optimal routes produced from files of both versions. All images to the left is from the prototype and every image to the right is from the original process. The distance and expected time can be seen in the small window in the upper left corner of each image.
7 Conclusion

This report describes an attempt to improve and optimize a procedure of handling and processing road network data. This included learning how the whole system works and identifying problems within it, designing a new improved system and finally, from the design, build a prototype.

Analyzing the process of the old system revealed problems in multiple areas. It takes at least one week for the system to finish, including idle time and potential errors. The user has to be active throughout the process, constantly checking whether a task is done in order to initiate the next one. The process requires a lot of input from the user, increasing the risk for mistakes and hard detected errors. A new design of the system was made with the aim to improve the problematic areas. The main idea for the new system was to completely remove all the manual tasks and improve the data structures. This could save the user time and effort while also decrease the risk for errors. Building the system based on the design turned out to be successful, resulting in a working prototype.

An evaluation of the prototype showed promising results. With the combination of making the process automatic and improving the data structures and search algorithms, improvements were made in terms of time consumption as well as required effort by the user and correctness. For the targeted region (Sweden), the prototype managed to produce road network files within an hour. Comparing that with the original process, which takes at least one week to finish, a powerful advancement can be seen. After the prototype has been initialized any sort of input from the user is not required. This contributes to decreasing both the risk for human errors and amount of time the user have to spend on the process.

The files produced by the prototype gave identical optimized routes when compared to files produced by the original process. This does not confirm that everything is correct since every possible route was not tested, however the results are promising.

The main problem with the prototype is the lack of testing, confirming that everything is correctly done. To make the system trustworthy enough to being put into operation, more reliable tests than checking random samples are preferred. There are also many other smaller things that can be improved. Some areas require better error handling. Not much time was assigned to the interface, therefore making it somewhat mediocre. Aside from all that, the prototype improved the process of handling road network data in multiple
aspects and actually exceeded set expectations.

7.1 Future Work

There are multiple areas and features that would improve the system in different ways. As mentioned, a good way of automatically checking the correctness of each produced file would increase the reliability of the system. It is however a difficult area to make tests for. Each file is huge and does not guarantee the same order between updates. It is also hard to know when a produced route is correct without any sort of comparison. One suggestion would be to have a set of core routes with approximate known traveling times and distances. This could open the possibility to discover potential deviant behaviour when something is wrong.

The prototype only supports geographic data that is gathered from a data source covering Sweden. The use of the system might expand to other regions, making it essential for it to support areas outside of Sweden. The majority of the process would most likely be similar, however some modifications and extensions of the system would be needed.

Improving the performance could be done with multithreading and other optimizations of the current algorithms and data structures. As of now, all of the work is made by merely one thread. Increasing the number of threads, fully exploiting all CPU cores, and dividing the work between them would most likely improve the performance immensely. To completely remove the risk for human errors, expanding the system to gather and fix the basic data automatically could be interesting to look into. Some way to constantly gather data from the sources database would also open the possibility to run the system more frequently.

A field that was not included in this thesis is the creation of the map where all calculated routes are visually presented. It is now done separately but could be merged into the system to improve the whole process even further. Other desired features could also be added.
References


