

# AUTOMATIC GENERATION OF SCHEMATIC DIAGRAMS OF THE DUTCH RAILWAY NETWORK

A thesis submitted to the Radboud University in partial fulfillment  
of the requirements for the degree of

Master of Science in Mathematics

by

Angela Brands

December 2016

Supervisors:  
Dr. Wieb Bosma  
Dr. Rob Udink



Radboud University





# CONTENTS

|       |                                              |    |
|-------|----------------------------------------------|----|
| 1     | INTRODUCTION                                 | 3  |
| 2     | PREVIOUS WORK                                | 9  |
| 2.1   | The metro map layout problem . . . . .       | 9  |
| 3     | RAIL PROPERTIES                              | 11 |
| 4     | IMPLEMENTED ALGORITHM                        | 17 |
| 4.1   | Assumptions . . . . .                        | 17 |
| 4.2   | Aesthetic criteria . . . . .                 | 17 |
| 4.3   | Minor data modification . . . . .            | 18 |
| 4.4   | Straight line algorithm . . . . .            | 18 |
| 4.5   | Linear Referencing . . . . .                 | 24 |
| 4.6   | Drawing algorithm . . . . .                  | 26 |
| 4.6.1 | Parallel tracks . . . . .                    | 26 |
| 4.6.2 | Schema $y$ -values . . . . .                 | 26 |
| 4.6.3 | Representation of diverging tracks . . . . . | 30 |
| 4.6.4 | Schema $x$ -values . . . . .                 | 30 |
| 4.7   | Results and analysis . . . . .               | 35 |
| 4.7.1 | Output examples . . . . .                    | 35 |
| 4.7.2 | Problems and solutions . . . . .             | 44 |
| 5     | CONCEPTUAL ALGORITHM                         | 47 |
| 5.1   | Aesthetic criteria . . . . .                 | 47 |
| 5.2   | Definitions and drawing rules . . . . .      | 47 |
| 5.3   | Concept . . . . .                            | 48 |
| 5.4   | Manually made schematic diagrams . . . . .   | 49 |



# 1 | INTRODUCTION

In the Netherlands, every day one million journeys are made by train. The infrastructure manager of the Dutch railway network, ProRail, is responsible for the network's design, maintenance, management and safety. In order to fulfill this responsibility, among other things, clear overviews of important network areas, mostly around stations, are needed. For example a constructor needs a compact diagram of a marshalling yard, see figure 1.1. Planning construction and maintenance activities can be done by indicating out of service areas in a clear overview, see figure 1.2. While figure 1.3 shows a rather complex presentation needed for evacuation plans, the underlying railway network is thoroughly simplified. And it is precisely the latter what this thesis is about.

A simplified representation of a (partial) transportation network is called a *schematic map*. In a schematic map only significant components of the network are depicted. Unnecessary details are omitted to improve readability. As for the railway network, the exact location and shape of the tracks are not important in diagrams such as mentioned above. For the users of schematic maps only the topology of the network, i.e. the relevant positions of the tracks and interconnections are significant.

The aesthetically pleasing schematic maps shown in figures 1.1, 1.2 and 1.3 are created manually and by professional designers, which is a time consuming and expensive process. Therefore, the automatic generation of schematic maps is of high interest.

This master's thesis has the aim to answer the following question:

- Is it possible to automatically derive a schematic map by using geographical information and network properties, and to what extent?

To answer this, the following sub research questions have to be answered as well:

- What are the railway network properties?
- What software is used to store and manage the geographical information of the network?
- What are the layout criteria of the schematic map?

The railway network consists out of tracks, interconnections, bufferstops and fly-overs. A track is defined to be the element between interconnections or between an interconnection and a bufferstop. The group of interconnections includes crossings and various types of switches. Crossings consist out of two overlaying tracks, where switching from one track to the other track is not possible. When a track has no successor it ends in a bufferstop. A railway switch enables trains to go from one track to another. It divides a

branched track into a *straight track* and a *diverging track*, see figure 1.4. The different types of switches and more network properties will be discussed in Chapter 3.

To store, modify and visualize railway data, the geographic information system ArcGIS is used. The various types of geographic datasets are captured in a common file system folder called a *geodatabase*. This geodatabase can be seen as the physical storage of geographic information.

For the railway network it contains two types of feature classes, lines and points. The feature class *junctions* holds the actual location, i.e. the  $(x, y)$  coordinate of switches, bufferstops and crossings. Junctions are depicted by points. The feature class *tracks* includes the geographical location, i.e. a path of  $(x, y)$  coordinates of the tracks. Tracks are depicted by lines. See figure 3.9 for an example.

ArcGIS has an extension called *Schematics* where geographical networks can be represented as schematic diagrams. CGI and ProRail used Schematics in many different ways to automatically create schematic maps of the railway network. Unfortunately, the generated outputs were not good enough and needed a lot of modifications done manually. The algorithm used in Schematics is called the *Relative Main Line Algorithm*. See figures 1.6 - 1.11 for examples of geographic input networks, the wanted schematized diagrams and the actual generated outputs.

The aesthetic layout criteria of a good schematic map of the railway network are:

- Tracks must be represented as straight line segments.
- If tracks are geographically parallel, then they must be displayed parallel in the schematic diagram.
- The relative positions of the tracks and junctions and their connectivities must remain the same.

These layout criteria are further refined in the process of this thesis.

The main objective of this research is to develop an algorithm that automatically generates a schematic map from geographical input data of a partial railway network. The specific objectives of this study are:

- Studying the concept of schematic maps.
- Understanding the railway network and its properties.
- Exploring the geographical information system ArcGIS and how railway information is stored.
- Examining the metro map problem.
- Review existing algorithms for automatically generating schematic maps.
- Learning the programming language python.

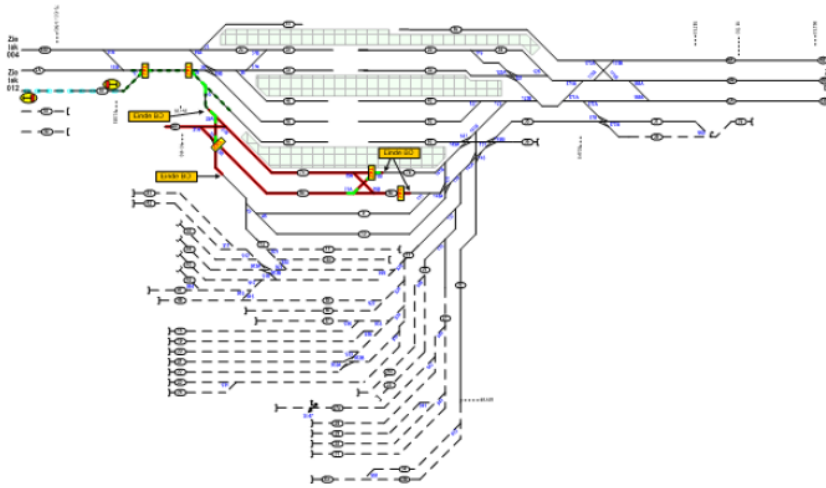


Figure 1.1. Micro level diagram for a constructor.

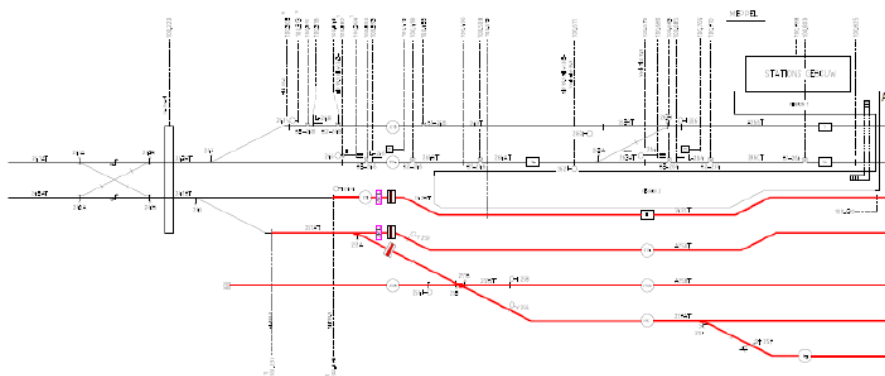


Figure 1.2. Overview indicating out of service areas.

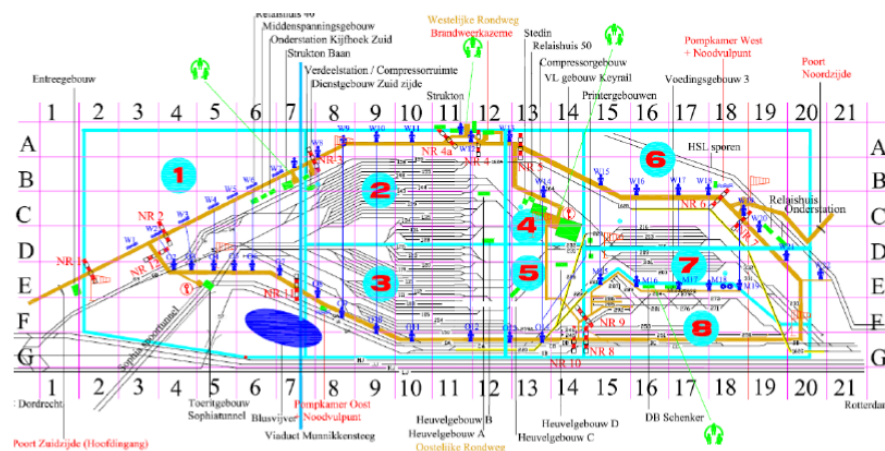


Figure 1.3. Representation of evacuation plans.

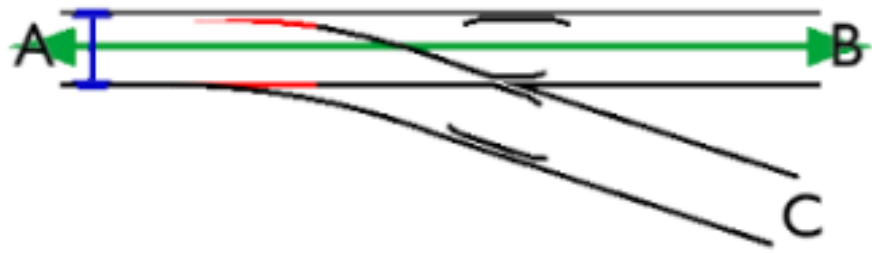


Figure 1.4. Example of a railway switch where AB is the straight track and AC the diverging track.

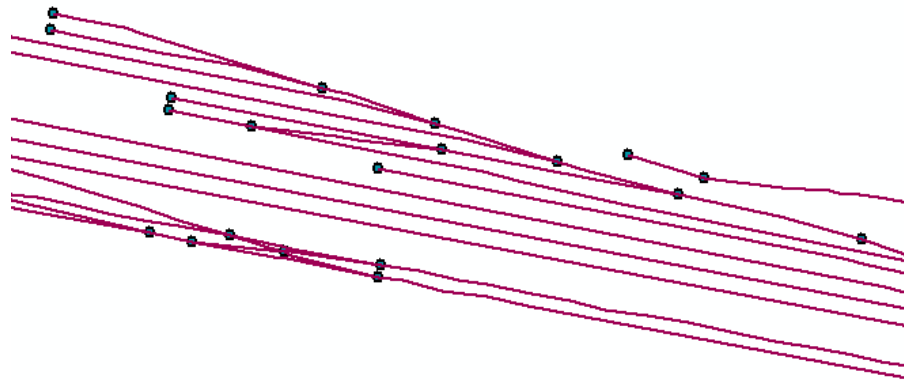


Figure 1.5. Example of a partial network visualization from ArcGIS.



Figure 1.6. Geographical input.



Figure 1.7. Wanted schematic diagram.





Figure 1.8. Actual output using the Relative Main Line Algorithm.



Figure 1.9. Geographical input.

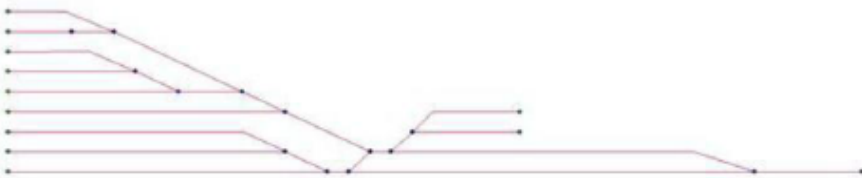


Figure 1.10. Wanted schematic diagram.



Figure 1.11. Actual output using the Relative Main Line Algorithm.



## 2 | PREVIOUS WORK

A schematic map of a transportation network is a simplified presentation of the given network, depicting all significant components. A well known schematic map is the metro map. In 1931, technical draughtsman Harry Beck, while drawing an electrical circuit diagram, came up with the idea of using the same drawing rules to create a map of the London Underground, see figure 2.1. Beck's motivation was based on the needs of the users of a metro map. They only need to know the relative positions of the stations and tracks and are less concerned with the exact geographical locations. His design uses mostly horizontal, vertical and diagonal lines, rather than the geographic polylines. Nowadays schematic maps of transportation networks are still following Beck's basic design cues.

A subset of schematic maps are metro maps. Work done in the field of automatic generation of metro map drawings is generally applicable to schematic maps, [Oke and Siddiqui \[2015\]](#).

### 2.1 THE METRO MAP LAYOUT PROBLEM

A first attempt to automatically create metro maps is from [Hong et al. \[2005\]](#). To automate the process of drawing a metro map, they start with defining the actual problem as follows. Given a graph  $G$  and a set of paths that cover all the vertices and edges of  $G$ . Find a good layout of  $G$  such that each line is drawn as straight as possible, there are no edge crossings and no overlapping labels, lines are mostly drawn horizontally or vertically with some at  $40^\circ$ , and each line is drawn with a unique color.

They demonstrated five different methods using various combinations of forced-directed algorithms. The implementation did not preserve the topology of the input embedding.

[Nollenburg and Wolff \[2011\]](#) translated the metro map layout problem into a mixed integer program (MIP) where hard constraints must be satisfied and soft constraints are globally optimized. The drawback of this mixed integer problem is the potentially long running time.

[Oke and Siddiqui \[2015\]](#) considered the MIP formulation of Nöllenburg and Wolff as a significant improvement compared to previous efforts. They built upon their work by improving the computational efficiency of the model. This is done by relaxing some constraints and reducing the number of objectives.



Figure 2.1. Beck's London Underground Tube Map, published in 1933.

# 3 | RAIL PROPERTIES

The railway network consist out of tracks, different types of switches, crossings and bufferstops. There are three types of switches: the single switch, the full slip and the single slip.

A single switch is a device that enables the train to choose from two possible forward directions and either divides a track into a *straight track* and a *diverging track* or branches off into two diverging tracks. See figure 3.1 for the different types of single switches.

A full slip, as shown in figure 3.2a, is an intersection of two straight tracks with two additional tracks that make it possible for a train to go from one track to the other. A single slip is similar to a full slip but has only one additional track, see figure 3.2b.

A crossing, see figure 3.3, is simply the intersection of two tracks tracks.

A buffer stop, figure 3.4, is a device that prevents trains to go beyond the physical end of a track.

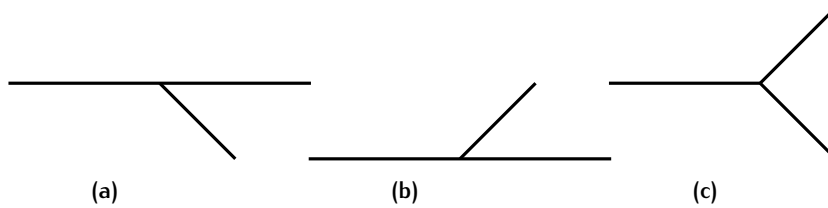


Figure 3.1. Single switches with diverging direction (a) right, (b) left, (c) right and left.

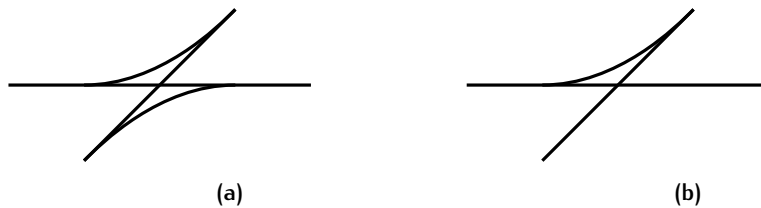


Figure 3.2. (a) Full slip. (b) Single slip.

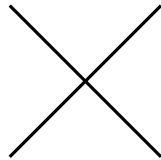


Figure 3.3. Crossing of tracks.



Figure 3.4. Track ending in a buffer stop.

To store, modify and visualize railway data, the geographic information system ArcGIS is used. The various types of geographic datasets are captured in a common file system folder called a *geodatabase*. This geodatabase can be seen as the physical storage of geographic information.

For the railway network bufferstops, crossings and switches are contracted to one point. In the geodatabase these points are called *junctions*.

A *track* is defined to be the element between two junctions. Every track begins in a *source* junction and ends in a *target* junction.

A buffer stop is a junction with only one adjacent straight track. A single switch is a junction with three adjacent tracks, where two opposite tracks are straight and the other one is diverging. Also only one pair of neighboring tracks (one straight and one diverging) form an angle equal or less than  $90^\circ$ . A crossing, full slip and single slip are junctions defining the intersection of two tracks. In this case, every track is divided into two opposite *track segments*, by the intersection. So this kind of junction has four adjacent straight track segments, where every pair of neighboring tracks segments form an angle of less than  $180^\circ$ . From now on we call these tracks segments just tracks. Keep in mind that a track endig or beginning in a junction of type full slip, single slip or crossing needs to have its opposite track to be whole. See figures 3.5 - 3.8 for the various representations of junctions and their associated tracks.



Figure 3.5. Buffer stop as a junction with one adjacent track.

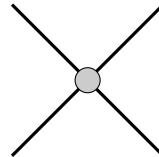


Figure 3.6. Crossing as a junction with four adjacent track segments.

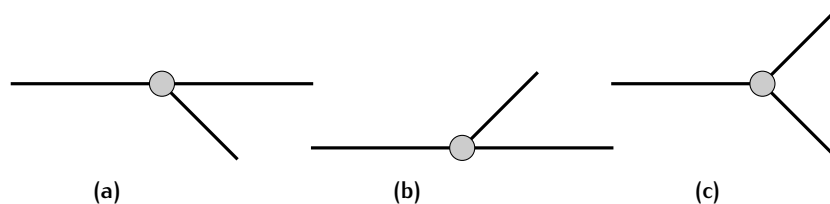


Figure 3.7. Single switches as junctions with three adjacent tracks.

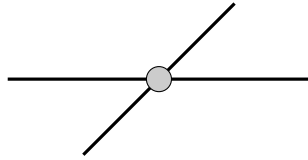


Figure 3.8. Full slip or single slip as a junction with four adjacent track segments.

The geodatabase of the railway network contains two feature classes, junctions and tracks. A junction is a point and is depicted as a circle. A track is a path of points and is depicted as a polyline. See 3.9 for an example of the visualization of a partial railway network.

The feature class junctions contains for every junction the following information:

- unique identification number
- type, i.e. buffer stop, crossing, single switch, full slip or single slip
- diverging direction, i.e. left", right or null
- geographical location, i.e.  $(x, y)$  coordinate.

Additional to the feature classes junctions and tracks, the geodatabase contains a table that stores the topology of the railway network.

By joining this table with the feature class tracks, the topological information of the network will be appended to the information of the tracks. The resulting augmented feature class tracks holds the following information for every track:

- unique identification number
- length
- geographical location, i.e. a path of  $(x, y)$  coordinates of the points that form the track
- identification number of the source junction
- identification number of the target junction
- the positions of the track in the clockwise ordering around the two adjacent junctions. This will be explained below.

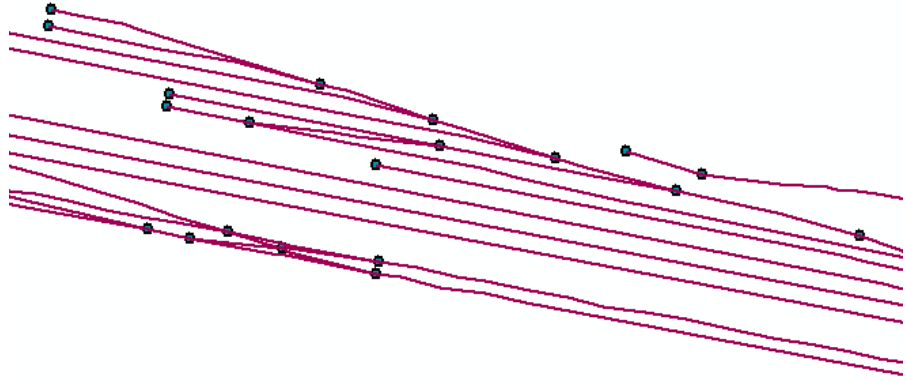


Figure 3.9. Example of a partial network visualization from ArcGIS.



The clockwise ordering of tracks around a junction is denoted by the ordered sequence  $(0, 1, 2, 3)$ . The position in the ordered sequence of a track around its source junction is called a *source index*. Similarly, the position in the ordered sequence of a track around its target junction is called a *target index*. See figures 3.10 and 3.11 for junctions of type buffer stop and single switch, and their adjacent tracks together with their relevant indexes.

By studying junctions of type single switch and the associated indexes of the adjacent tracks, we conclude that the track with index 0 is always a straight track and always opposite to the tracks with indexes 1 and 2. Furthermore, for the latter pair, the track with index 1 is always the left one of the two and the track with index 2 the right one. For junctions of type full slip, single slip or crossing, two track segments with indexes  $i_a$  and  $i_b$ , form a straight track if  $|i_a - i_b| = 2$ . See figures 3.13 and 3.12.

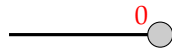


Figure 3.10. Junction of type buffer stop and one adjacent track with the associated index.

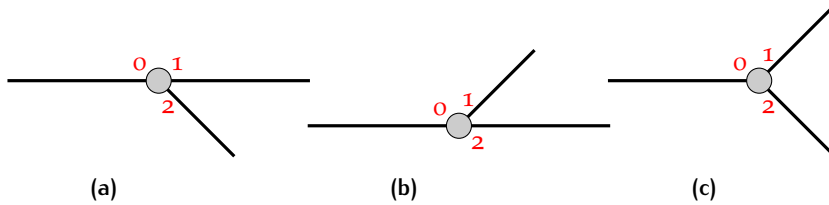


Figure 3.11. Three junctions of type single switch with three adjacent tracks and their associated indexes.

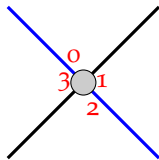


Figure 3.12. Junction of type crossing where blue track segments form a straight track and black track segments form a straight track.

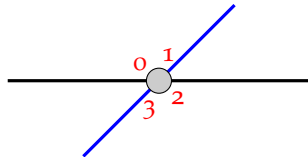


Figure 3.13. Junction of type full slip or single slip where blue track segments form a straight track and black track segments form a straight track..



# 4 | IMPLEMENTED ALGORITHM

We begin this chapter by making some assumptions regarding the input map. We then define the set of aesthetic criteria and make some minor data modifications. Then we will carefully describe the implemented algorithms and give some output diagrams. We conclude by discussing the occurred problems and suggesting their solutions.

## 4.1 ASSUMPTIONS

For this project we make the following assumptions about the geographical input map.

**Assumption 1.** *The majority of the tracks of the input map lie in a common main direction. And tracks perpendicular to the main direction are omitted.*

See for example figure 4.1.

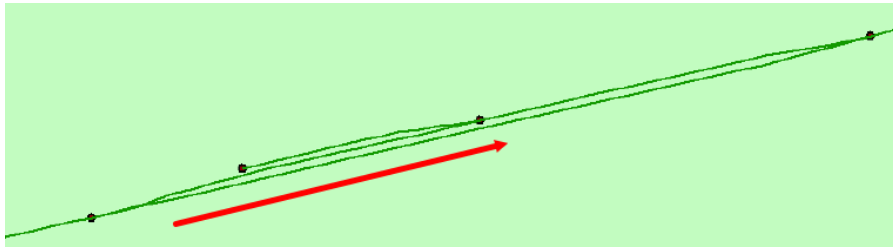


Figure 4.1. The common main direction is indicated in red.

Since fly-overs are not yet objects in the data we assume the following:

**Assumption 2.** *The input map does not contain fly-overs.*

## 4.2 AESTHETIC CRITERIA

The wanted schematic diagrams are those that visualize the railway network around a station. Before the development of the algorithm, the criteria of the schematic diagram were somewhat vague. But since we want to automate the process we need to have some conditions on the schematic output map. We decided to use the following set of aesthetic criteria:

1. the topology of the underlying graph of the railway network is preserved
2. the order of the junctions in the main direction remains the same
3. every track is represented as straight line segments with at most one bend

4. straight tracks are drawn horizontally as much as possible
5. with the exception of crossings, all angles are  $60^\circ$  or  $-60^\circ$

After studying the railway network and its properties, we make the following assumption regarding the representation of crossings in the schematic diagram.

**Assumption 3.** *Tracks that form a crossing are never straight and therefore are never drawn horizontally.*

### 4.3 MINOR DATA MODIFICATION

As mentioned, every track begins in a source junction and ends in a target junction. For the railway network this means that a junction can be the source of two opposite tracks, see figure 4.2.

To make the network easier to work with, we modify the relevant data such that for all tracks with source junction at  $(x_0, y_0)$ , target junction at  $(x_1, y_1)$  and  $x_0 \neq x_1$  holds:  $x_0 < x_1$ . See figure 4.3.

Note that tracks with  $x_0 = x_1$  are no longer defined. For now, we leave out geographical input maps where the main direction is vertical.

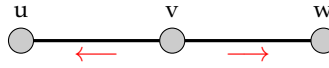


Figure 4.2. Two adjacent tracks where junction  $v$  is the source of track  $vu$  and the source of track  $vw$ .

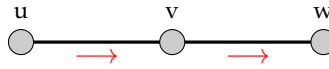


Figure 4.3. After modification:  $v$  is the target of track  $uv$  and the source of track  $vw$ .

After the modification we can create two useful python dictionaries. The dictionary *incoming tracks* contains for every junction  $u$ , all tracks with target junction  $u$ . The dictionary *outgoing tracks* includes for every junction  $v$ , all tracks with source junction  $v$ .

### 4.4 STRAIGHT LINE ALGORITHM

One of the aesthetic criteria of the schematic diagram is that straight tracks need to be drawn horizontally as much as possible.

Since a track is defined to have a source junction and a target junction, it can be straight and diverging at the same time. For example think of a track  $uv$  with single switch type source junction  $u$  and single switch type target junction  $v$ . Suppose that junction  $u$  has an incoming straight track, an outgoing straight track and assume that track  $uv$  is the diverging track out of  $u$ . Also assume that track  $uv$  is the straight track into  $v$ . This makes track  $uv$  simultaneously diverging and straight. So, do we draw it horizontal or not? No, since junction  $u$  already has an outgoing straight track that needs to be drawn horizontally. Hence, if a track is in any way a diverging one, it cannot be drawn horizontally. We need a new definition for tracks that

need to be drawn horizontally.

**Definition 1.** A track is called completely straight if it is the straight track out of its source junction and the straight track into its target junction.

Since a full slip or single slip need two opposite tracks to form a whole track, it makes sense to call a track ending or beginning in such junction completely straight only if its other half is completely straight.

**Definition 2.** A straight line is a maximal sequence of connected tracks that are completely straight. Where maximal means that the sequence cannot be augmented.

After observing straight lines in the network we make the following assumption.

**Assumption 4.** Every straight line contains at least one junction of type single switch.

So starting at a random single switch type junction  $u$ , how can we find the straight line containing  $u$ ?

We mark junction  $u$  as placed. Then we first walk to the right by looking at the outgoing tracks of junction  $u$  and appending, the completely straight track out of  $u$ , if it exists, to the line. Say  $uv$  is the completely straight track out of  $u$ . We then repeat this by looking at the outgoing tracks of junction  $v$ , appending the possible completely straight track out of  $v$  to the line. And so on. The line ends in a junction if it has no outgoing track or if the outgoing track is not completely straight.

Similarly we walk to the left by looking at the incoming tracks of junction  $u$  and inserting, the completely straight track into  $u$ , if it exists, at first position of the line.

Lets explain the part of "walking to the right" more carefully, see algorithm 1.

Given a single switch type junction  $u$ . In the algorithm we call  $u$  the *current junction*. Obviously  $u$  has to have an outgoing track to walk to the right. If it does we have to know its type. Of course we started with knowledge of  $u$  being of type single switch, but as we will see later on, we can and will arrive at this part of the algorithm with a junction of type full slip or single slip. So, we know junction  $u$  is of type single switch. We know that a junction of type single switch has either one or two outgoing tracks.

For an outgoing track of a single switch junction to be straight, one of the following possibilities must be true.

- (i) its source index = 0, this is by definition a straight track out of a single switch type junction
- (ii) its source index = 1 and the diverging direction of  $u = right$
- (iii) its source index = 2 and the diverging direction of  $u = left$ .

If junction  $u$  has no outgoing straight track, the right side of the straight line is finished and  $u$  is the end of the line.

Suppose now, that there is a straight track out of  $u$ , say  $uv$ . In the algorithm

$uv$  is called the *current\_track*. First we need to check if its target junction  $v$  is not already placed in a straight line. If it is already placed then the right side is finished with the line ending in  $u$ . If target  $v$  is not placed yet, we need to decide if track  $uv$  is the straight track into target  $v$ . Therefore, we need to know the type of junction  $v$ . If target  $v$  is of type single switch, one of the following possibilities must be true for  $uv$  to be a straight track into target  $v$ .

- (i) its target index = 0, this is by definition a straight track into a single switch type junction
- (ii) its target index = 1 and the diverging direction of  $v = right$
- (iii) its target index = 2 and the diverging direction of  $v = left$ .

If none of the possibilities holds for track  $uv$ , it is not the straight track into its target, and thus track  $uv$  is not completely straight. Hence, track  $uv$  is not appended to the straight line, finishing the line in junction  $u$ . Furthermore, if one of the possibilities holds for track  $uv$ , it is the straight track into its target, and thus track  $uv$  is a completely straight track. So now we append  $uv$  to the line, and mark target junction  $v$  as placed. Also we remove track  $uv$  from the outgoing tracks of  $u$  and the incoming tracks of  $v$ . We then define target  $v$  to be the *current\_junction* and we walk further to the right.

Lets go back to the point were we knew that track  $uv$  was the straight track out of junction  $v$ . But now, assume that  $v$  is of type crossing. Assumption 3 forces the line to stop in junction  $u$ .

There are still two possible types for target  $v$ . Suppose target  $v$  is of type buffer stop. A buffer stop has exactly one adjacent track and this track is by definition straight. So  $uv$  is the straight track into target  $v$  and therefore a completely straight track. We append  $uv$  to the line, remove it from the outgoing tracks of  $v$  and mark junction  $v$  as placed. Since a junction of type buffer stop has only one adjacent track, we cannot go any further to the right. Thus the line ends in junction  $v$ .

The last option for  $v$  is to be of type full slip or single slip. We already know that track  $uv$  is actually one of two track segments that form a whole track through  $v$ . As mentioned above, we call track  $uv$  completely straight if its other half is also completely straight. So we will save track  $uv$  and target junction  $v$  and need to go further to the right, defining target  $v$  to be the *current\_junction*. This brings us to the part of the algorithm where the junction from which we want to walk to the right, is of type full slip or single slip.

So *current\_junction*  $v$  is of type full slip or single slip. To go any further to the right, the other half of track  $uv$  must exist. If it does not exist, saved track  $uv$  cannot be appended to the line, so the line end in junction  $u$ .

Now suppose the other half of track  $uv$  exists, say  $vw$ . In the algorithm  $vw$  is called *current\_track*. Since by definition all adjacent tracks to a full slip or single slip type of junction are straight, we need to decide if track  $vw$  is the straight track into its target  $w$ . Again we have to check the type of target  $w$ .

Suppose junction  $w$  is of type single switch. We already know how to check

if a track is the straight track into a single switch type junction. If track  $vw$  is not the straight track into target  $w$ , saved track  $uv$  cannot be appended to the line, finishing it at junction  $u$ . Assume now, that  $vw$  is the straight track into target  $w$ . We first need to append every saved track (in this case only track  $uv$ ) to the line. Then append track  $vw$  to the line. Also we mark every saved junction (in this case only junction  $v$ ) and target junction  $w$  as placed. Now we walk further to right by defining target to be the *current\_junction*. If target  $w$  is of type crossing. Again by assumption 3  $vw$  is not a straight track into  $w$ , so  $vw$  is not completely straight. Thus, the line ends at junction  $u$ , since the other half of  $vw$ , saved track  $uv$ , cannot be appended.

Suppose target  $w$  is of type buffer stop. This makes  $vw$  a completely straight track. Hence,  $uv$  is a completely straight track. We append every saved track (in this case only  $uv$ ) and *current\_track*  $vw$  to the line. Also we mark every saved junction (in this case only  $v$ ) and target junction  $w$  as placed. We cannot walk any further to the right, since a buffer stop type of junction has only one adjacent track. This means that the line ends in junction  $w$ .

The last option for target  $w$  is to be of type full slip or single slip. As before we have to save *current\_track*  $vw$  and target junction  $w$ . By defining target  $w$  to be the *current\_junction* we try to walk further to right, by checking its outgoing tracks.

Note that the line never ends in a junction of type full slip or single slip.

At any point in the algorithm, where the right side of the line is finished, we do the same thing to determine the left side of the line by walking to the left of the starting junction  $u$ .

**Algorithm 1:** Straight line algorithm

**Input** : Geographical data of the input map and a random single switch type junction  $u$   
**Output** : The straight line containing junction  $u$

"to the right"

```

done ← []
line ← []
temp_tracks ← []
temp_junctions ← []
current_junction ←  $u$ 
stop ← false
done.append(current_junction)
while (current_junction has outgoing tracks) and (stop == false) do
    if type(current_junction) == Single Switch then
        if no straight track out of current_junction exists then
            stop ← true
        else
            current_track ← straight track out of current_junction
            target ← target of current_track
            if target in done then
                stop ← true
            else
                if type(target) == Single Switch then
                    if current_track is not the straight track into target then
                        stop ← true
                    else
                        line.append(current_track)
                        remove track from associated outgoing and incoming tracks
                        done.append(current_junction)
                        current_junction ← target
                else if type(target) == Crossing then
                    stop ← true
                else if type(target) == Buffer Stop then
                    line.append(current_track)
                    remove track from associated outgoing and incoming tracks
                    done.append(current_junction)
                else
                    temp_tracks.append(current_track)
                    temp_junctions.append(current_junction)
                    current_junction ← target
    else
        if other half of current_track not exists then
            stop ← true
        else
            current_track ← other half
            target ← target of current_track
            if target in done then
                stop ← true
            else
                if type(target) == Single Switch then
                    if current_track is not the straight track into target then
                        stop ← true
                    else
                        for track in temp_tracks do
                            line.append(track)
                        line.append(current_track)
                        remove all appended tracks from associated outgoing and incoming tracks
                        for junction in temp_junctions do
                            done.append(junction)
                        done.append(current_junction)
                        current_junction ← target
                else if type(target) == Crossing then
                    stop ← true
                else if type(target) == Buffer Stop then
                    for track in temp_tracks do
                        line.append(track)
                    line.append(current_track)
                    remove all appended tracks from associated outgoing and incoming tracks
                    for junction in temp_junctions do
                        done.append(junction)
                    done.append(current_junction)
                else
                    temp_tracks.append(current_track)
                    temp_junctions.append(current_junction)
                    current_junction ← target

```

"to the left"

```

temp_tracks ← []
temp_junctions ← []
current_junction ←  $u$ 
stop ← false
while (current_junction has incoming tracks) and (stop == false) do
    Similar to the above

```



Now we know how to find the straight line containing at least one single switch type of junction, it is not difficult to find all the straight lines of the input map, see algorithm 2.

By assumption 4 we can find all the straight lines of the input map by randomly start at a junction of type single switch, untill there are no more single switch type of junctions.

---

**Algorithm 2:** Find all straight lines algorithm
 

---

```

Input      :Geographical data of the input map
Output     :All the straight lines of the input map

single_switch_junctions  $\leftarrow$  []
all_straight_lines  $\leftarrow$  []

for junction in junctions do
    if type(junction) == Single Switch then
        single_switch_junctions.append(junction)

while single_switch_junction  $\neq \emptyset$  do
    u  $\leftarrow$  random.choice(single_switch_junctions)
    straight_line  $\leftarrow$  the straight line containing junction u by algorithm 1
    all_straight_lines.append(straight_line)
    single_switch_junctions.remove(u)

return all_straight_lines
  
```

---

## 4.5 LINEAR REFERENCING

In the previous section we explained that straight lines need to be drawn horizontally in the schematic diagram. In order to do that we must give every straight line a schema  $y$ -value. Therefore we need to know the vertical order of the straight lines. This means that by looking at a straight line, we must decide if its above or below another straight line. Since the network of the input map has a main direction that can be anything but totally vertical, we cannot simply compare the two straight lines by looking at the geographical  $y$  positions of the included junctions. If only we could rotate the network in a way that the main direction becomes horizontal. Luckily this can be done by *Linear Referencing*. In this case, Linear Referencing is used to store the geographic locations of the tracks and junctions by using relative positions along a linear line.

After observing various input maps and their straight lines, we conclude that for most of the input maps, the longest straight line covers the network, i.e. there are no tracks or junctions before the beginning and after the end of the longest straight line. So by using this line as the line for linear referencing, we can rotate the network. It is easy to find the longest straight line, since we know the length of every track, see algorithm 3

---

### Algorithm 3: Longest straight line algorithm

---

**Input** : Geographical data of the input map and all straight\_lines  
**Output** : The longest straight\_line

Function: length\_of\_straight\_line(straight\_line, length)

$L \leftarrow 0$   
**for** track in straight\_line **do**  
     $L \leftarrow L + \text{length}(\text{track})$

**return**  $L$

Algorithm:

$T \leftarrow 0$   
**for** straight\_line in all\_straight\_lines **do**  
     $L \leftarrow \text{length\_of\_straight\_line}(\text{straight\_line}, \text{length})$   
    **if**  $L > T$  **then**  
         $T \leftarrow L$   
        longest\_straight\_line  $\leftarrow$  straight\_line

**return** longest\_straight\_line

---

Suppose the input map is the network around a very small station in a town called Ommen, see figures 4.21 and 4.22.

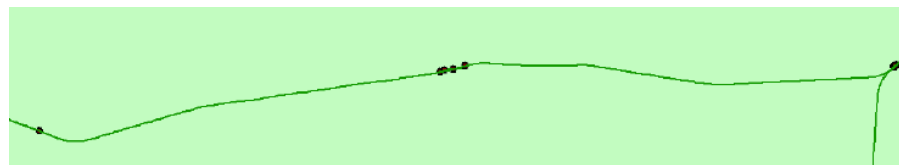


Figure 4.4. Input map of Ommen.

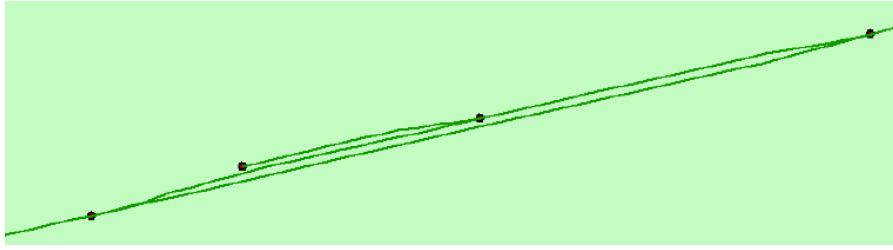


Figure 4.5. Input map of Ommen, zoomed in.

So once we know the longest straight line, we have to select this line manually in ArcMap. And since the longest straight line consists of multiple tracks, we merge these tracks into one, figure 4.6.



Figure 4.6. Track defining the longest straight line of Ommen are merged into one.

Recall that every track in the original network data was assigned a direction. When merging the tracks that make up the longest straight line, we must give it the properties of one of the included tracks. Suppose we gave it the properties of the first track. The merged line will then possess the same direction as the first track. In this case, the direction of the first track was from left to right. Thus the merged line has now a direction from left to right. Next, the merged line is scaled, straightened and rotated until it is horizontal.

To explain how the linear referencing algorithm in ArcGIS works, we give the following example. Suppose we walk along the unstraightened merged line, starting on the left. While walking we look 100m to the right and 100m to the left. Every time we see a track or junction it is placed at its relative position with reference to the scaled straightened merged line. See figures 4.24 and 4.25 for the Ommen input map after linear referencing.



Figure 4.7. The input map of Ommen after linear referencing.

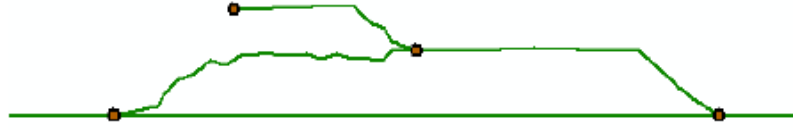


Figure 4.8. The input map of Ommen after linear referencing, zoomed in.

## 4.6 DRAWING ALGORITHM

After linear referencing, the junctions and tracks have new coordinates. From now on we use these new coordinates in stead of the initial coordinates that represent their geographical locations.

### 4.6.1 Parallel tracks

In the input map we often find two single switch type of junctions connected by two different tracks, see figure 4.9 for an example.

**Definition 3.** Two tracks are called *parallel* if they share the same source junction and the same target junction.

By observing parallel tracks in the railway network, we make the following assumptions.

**Assumption 5.** The common source junction and target junction of parallel tracks are of type single switch.

**Assumption 6.** Parallel tracks consist of a completely straight track and a diverging track.

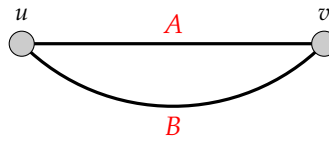


Figure 4.9. Parallel tracks between two junctions  $u$  and  $v$  of type single switch. Track A is a completely straight track. Track B is the diverging track.

### 4.6.2 Schema $y$ -values

By linear referencing we obtain an input map with a horizontal main direction. From this we can derive the vertical order of the straight lines.

After finding all the straight lines, there are still some features left as they are not included in a straight line, i.e. the rest of the junctions and diverging tracks. We need to assign a schema  $y$ -value to all the junctions in the

straight lines (leaving out the longest straight line), the diverging parallel tracks and the rest of the junctions except those of type crossing. By joining two track segments for each track through the crossing, a junction of type crossing does not need a schema  $y$ -value because the intersection of the two associated tracks will appear naturally in the schematic diagram.

So, after putting all the straight lines and the rest of the junctions together in a python list, we can determine the vertical order as follows. Every element in the list is either a list of junctions and tracks (straight line) or a list of a single junction.. After the linear referencing algorithm, the longest straight line has  $y$ -value 0. We start by dividing the elements into positive elements, i.e. elements that lie above  $y = 0$  and negative elements, i.e. elements that lie below  $y = 0$ . Next, we order the positive elements and the negative elements. Let's explain algorithm 4 more precisely.

In the schematic diagram we use the longest straight line as a base line. All the included junctions get the schema  $y$ -value 0.

The list of all elements that need a schema  $y$ -value is called *lines*.

In the first part of the algorithm we check for every line in *lines* if it lies above base line, by testing if the  $y$ -values of the first and last junctions of the line are greater than 0. If so, the line is appended to a list called *lines\_above\_base*. If not, the line is appended to a list called *lines\_below\_base*.

From now we will explain the algorithm as if the input map consist of only positive lines, since the ordering of negative lines will go similarly.

So we now have a list of lines that lie above the base line. We define a new list called *positive lines*. We start by selecting a random line from *lines\_above\_base*, put it in *positive lines* and remove from *lines\_above\_base*. Next, we select another random line, called *current\_line* and remove it. Then we check if this line lies below the line already placed in *positive lines*. We call this placed line the *test\_line*. Thus, we want to know if *current\_line* lies above *test\_line*. To do this we first define *start\_current\_line* and *end\_current\_line* to be the first and last junctions of *current\_line*.

Similarly we define *start\_test\_line* and *end\_test\_line*. Let *current\_interval* be the interval defined by the  $x$ -value of *start\_current\_line* and the  $x$ -value of *end\_current\_line*. Similarly we define *test\_interval*. There are two possibilities, the two lines overlap or not. We check if the two lines overlap by testing if

- *start\_current\_line* lies in *test\_interval* or
- *end\_current\_line* lies in *test\_interval* or
- *start\_test\_line* lies in *current\_interval* or
- *end\_test\_line* lies in *current\_interval*.

If one of the above options holds, we can determine their vertical order.

Suppose *start\_current\_line* lies in *test\_interval*. We know the  $x$ -value of *start\_current\_line* and since we have for each track (and thus for each line) its path of coordinates, we can search for the nearest  $x$ -value in that line. Next we compare the associated  $y$ -value to the  $y$ -value of *start\_current\_line*. Now, if *current\_line* lies below *test\_line* we put it before *test\_line* in *positive lines*. If not, we append it to *positive lines*. Note that if a line consists out of one junction, its path is just one coordinate.

Now suppose the two lines do not overlap. We then determine if *current\_line* lies before *test\_line* or after. If *current\_line* lies before *test\_line*, we compare the *y*-values of *end\_current\_line* and *start\_test\_line* and define *current\_line* to lie below *test\_line* if *y*-value of *end\_current\_line* is less than the *y*-value of *start\_test\_line*. Similarly for the other way around. Again if *current\_line* lies below *test\_line* we put it before *test\_line* in *positive\_lines*. If not, we append it.

At this point we have two lines in *positive\_lines*. We repeat the above until there are no more lines in *lines\_above\_base*. This means that for the next random line there are two test lines. If the line lies below the first test line it is inserted immediately. If not, the next test line will be used to determine if the random line lies below it.

Now we have all the straight lines and the rest of the junctions ordered in *positive\_lines*. To make this vertical order complete, we need to insert the diverging parallel tracks. We do this by finding the straight line that contains the two junctions of the diverging parallel track and determine if it is above or below the completely straight track out of its source junction. Since we know the source index of the diverging parallel track and the diverging direction of its single switch type source junction, we can decide if it is above or below the completely straight track. We already assigned a schema *y*-value to the source junction, so the diverging parallel track will have schema *y*-value of one less or one more.

**Algorithm 4:** Schema  $y$ -values algorithm

**Input** : Geographical data of the input map, all straight.lines and  $y$ -value of the longest straight line

**Output** : Schema  $y$ -values for all junctions

lines  $\leftarrow$  all straight lines and the rest of the junctions except of type crossing  
 diverging.parallel.tracks  $\leftarrow$  all diverging parallel tracks  
 copy.diverging.parallel.tracks  $\leftarrow$  a copy of diverging.parallel.tracks  
 lines.above.base  $\leftarrow []$   
 lines.below.base  $\leftarrow []$   
 schema.y.junctions  $\leftarrow \{\}$   
 schema.y.diverging.parallel.tracks  $\leftarrow \{\}$

"part one"

**while** lines  $\neq \emptyset$  **do**  
   current.line  $\leftarrow$  random.choice(lines)  
   **if** current.line lies above base line **then**  
     lines.above.base.append(current.line)  
   **else**  
     lines.below.base.append(current.line)

"part two for positive lines"

**while** lines.above.base  $\neq \emptyset$  **do**  
   current.line  $\leftarrow$  random.choice(lines.above.base)  
   lines.above.base.remove(current.line)  
  
   positive.lines  $\leftarrow []$   
    $N \leftarrow$  #lines in positive.lines  
   done  $\leftarrow$  false  
    $i \leftarrow 0$   
  
   **while** done == false **do**  
     **if**  $i \neq N$  **then**  
       test.line  $\leftarrow$  positive.lines[ $i$ ]  
       **if** current.line lies below test.line **then**  
         positive.lines.insert( $i$ , current.line)  
         done  $\leftarrow$  true  
       **else**  
          $i \leftarrow i + 1$   
     **else**  
       positive.lines.append(current.line)  
       done  $\leftarrow$  true

**if** copy.diverging.parallel.tracks  $\neq \emptyset$  **then**  
   **for** track in copy.diverging.parallel.tracks **do**  
     source  $\leftarrow$  source junction of track  
     target  $\leftarrow$  target junction of track  
     **for** line in positive.lines **do**  
       **if** source in line and target in line **then**  
         insert track at the right position in positive.lines  
         copy.diverging.parallel.tracks.remove(track)

**for**  $i$ , line in positive.lines **do**  
   **if** line in diverging.parallel.tracks **then**  
     schema.y.diverging.parallel.tracks[line]  $\leftarrow i + 1$   
   **else**  
     **for** junction in line **do**  
       schema.y.junctions[junction]  $\leftarrow i + 1$

"part two for negative lines"

**while** lines.below.base  $\neq \emptyset$  **do**  
   Similar to the above

**return** schema.y.junctions, schema.y.diverging.parallel.tracks

### 4.6.3 Representation of diverging tracks

From now on we use  $u_y$  to denote the schema  $y$ -value for junction  $u$ .

Before we can determine the schema  $x$ -values we first need know how tracks should be represented in the schematic diagram. Of course, the straight lines are drawn horizontally. This means that every track in a straight line, connects its source junction to its target junction by a horizontal line. But how do we draw the rest of the tracks (the diverging tracks)? Well, there are a few possibilities.

First recall that one of the criteria of the schematic diagram is that, except for tracks through crossings, diverging tracks are drawn using the same angle. Now, suppose track  $uv$  is the outgoing diverging track and suppose that  $y_u < y_v$ . The possible representations of track  $uv$  are shown in figure 4.10. Where 4.10a shows the representation of track  $uv$  as a linear line making an angle of  $60^\circ$  with the grid line  $y = y_u$ . We call the intersection of this linear line with a grid line, a *track corner* of junction  $u$ . In 4.10b track  $uv$  is represented as a bended line making that same angle of  $60^\circ$  with the grid line  $y = y_u$ . Figure 4.10c shows the representation of track  $uv$  as a bended line making an angle of  $60^\circ$  with the grid line  $y = y_v$ .

From now on we call the line in subfigure 4.10a an *outgoing up line*, the bended line in subfigure 4.10b an *outgoing up-right line* since the line goes up first and then to the right. The bended line in subfigure 4.10c is called an *outgoing right-up line*.

For  $uv$  an outgoing track and  $y_u > y_v$  we define an *outgoing down line*, an *outgoing down-right line* and *outgoing right-down line*.

Let's define the possible *track corners* around a junction, in clockwise order starting left above the junction, by  $c_1, c_2, c_3$  and  $c_4$ . To be precise, let  $u(c_1)_i$  define the upper corner left of junction  $u$  where  $|y(u) - y(u(c_1))| = i$ , see figure 4.11.

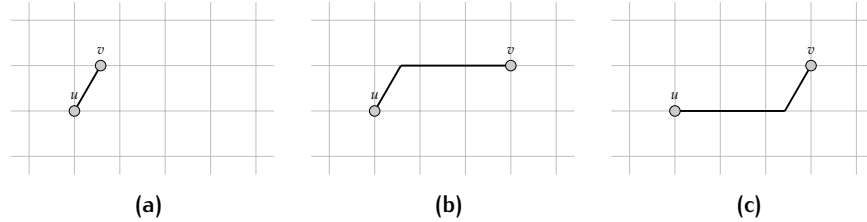


Figure 4.10. Representations of a diverging track in the schematic diagram.

### 4.6.4 Schema $x$ -values

We start by sorting the junctions in ascending order by their new  $x$ -values obtained after linear referencing. Next, we give all the junctions a first schema  $x$ -value such that each pair of neighboring junctions has equal distance. These first assigned schema  $x$ -values are not necessarily the final values, because a few of them need to change in order to satisfy the criteria of the lines.

Note that the first and last junctions in the horizontal order are the first and last junctions of the longest straight line. From now on we use  $x_u$  to denote the current schema  $x$ -value of junction  $u$ .



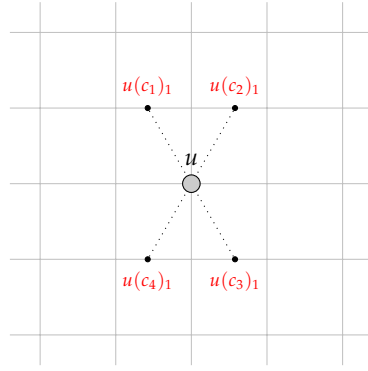


Figure 4.11. Possible track corners of  $u$  at  $y = y_u - 1$  and  $y = y_u + 1$ .

The following algorithm determines for each junction its final schema  $x$ -value. By checking how each junction is connected and how that connection, i.e. the diverging track, can be drawn.

To get a grasp of how this algorithm works, we will explain it by example.

Suppose we are looking at the first line from *positive\_lines*, the longest straight line. We call this line the *current\_line* in the algorithm. Say junction  $v$  is in *current\_line*. We then check if  $v$  has an incoming track such that  $y_{source} = 1$ . Suppose it does, and let the source be  $u$ . Now we need to determine how to draw this diverging track  $uv$  using one of the representation discussed in the previous section. If we know how to draw track  $uv$ , we can derive the associated schema  $x$ -value for  $u$ . This  $x$ -value can only be the current first assigned  $x$ -value or it is changed to a corner position around a certain junction.

So, to decide how to draw track  $uv$ , we check if  $u$  has another outgoing track. If so, we want to know if this other track is completely straight or diverging. Suppose that the other outgoing track is completely straight, and let its target be  $w$ , figure 4.12. Since a completely straight track is drawn horizontally, we have only two options for  $uv$  left, an outgoing down-right line or an outgoing down line. We do not like to shift junctions to a new  $x$ -position since that includes the shifting of all other junctions in the associated interval. So let's check when we can use an outgoing down-right line to represent  $uv$ . It means that  $v$  will have an incoming line at  $y_v$ . We can do this only if  $v$  does not have an incoming track that is completely straight, since that track will be drawn incoming at  $y_v$ . Assume that  $v$  has not an incoming completely straight track. We can draw  $uv$  using an outgoing down-right line, see 4.13. Now suppose that  $v$  has an incoming completely straight track, see figure 4.14. Now we have to shift  $u$  to the position of  $v(c_1)1$  so we can use the outgoing down line, see figure 4.15. In order to keep the correct horizontal order of junctions, the junctions included in the interval (old  $u,v$ ) need to be shifted to the interval  $(u,v)$ , see figure 4.16. If the interval (old  $u,v$ ) includes a junction that was shifted before, say junction  $a$ , we know that  $a$  is connected, by a down line or up line, to a junction, say  $b$ . We need to check if  $b$  is also in the interval (old  $u,v$ ). If not, then conflict?!

Now consider the option that the other outgoing track,  $uw$  of  $u$  is a diverging track. If  $w$  does not have an incoming track that is completely straight

en thus  $uw$  can be represented by an outgoing up-right line, we can draw track  $uv$  using an outgoing right-down line, see figure 4.17. Obviously, if  $w$  has an incoming completely straight track,  $uv$  cannot be represented by an outgoing right-down line, since  $uw$  must be represented by an outgoing right-up line. Assume that  $v$  has not a completely straight incoming track. We can draw  $uv$  using an outgoing down-right line, see figure 4.18. Now, assume that  $v$  has an incoming completely straight track. This means that we have to shift  $u$ , the new position depends on the horizontal order of  $v$  and  $w$ . Suppose  $x_v < x_w$ . Then  $u$  is shifted to the position of  $v(c_1)_1$ , see 4.19. If  $x_v > x_w$ ,  $u$  is shifted to the position of  $w(c_4)_1$ , figure 4.20.

At the end of the algorithm we know all the schema  $x$ -values and how every track needs to be represented. Hence, we can draw the schematic diagram.

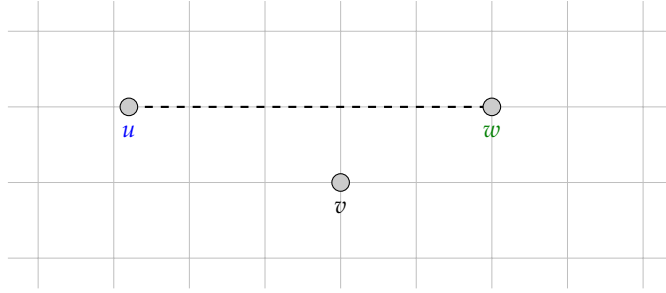


Figure 4.12.  $u$  has a completely straight outgoing track  $uw$ .

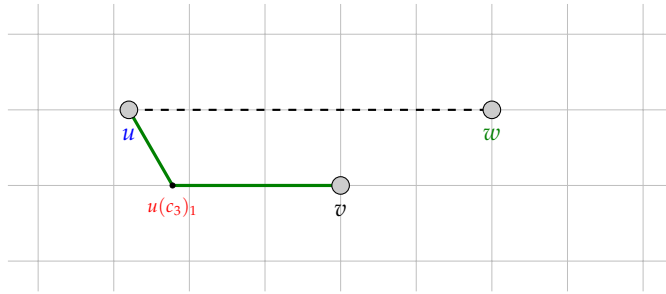


Figure 4.13. Track  $uv$  represented by an outgoing down-right line.

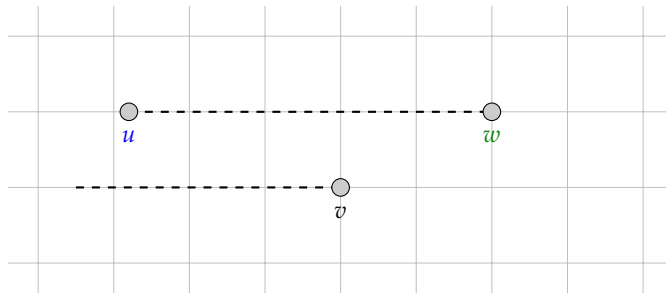


Figure 4.14.  $u$  has a completely straight outgoing track  $uw$  and  $v$  has a completely straight incoming track.

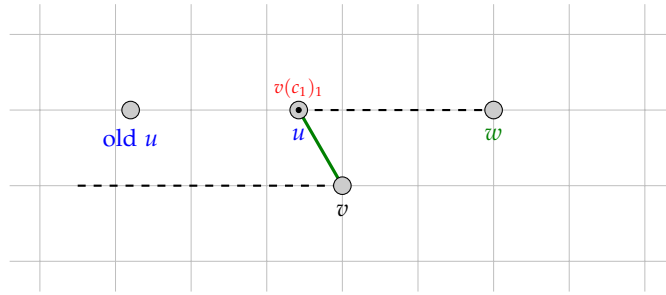


Figure 4.15. Track  $uv$  represented by an outgoing down line from  $u = v(c_1)_1$ .

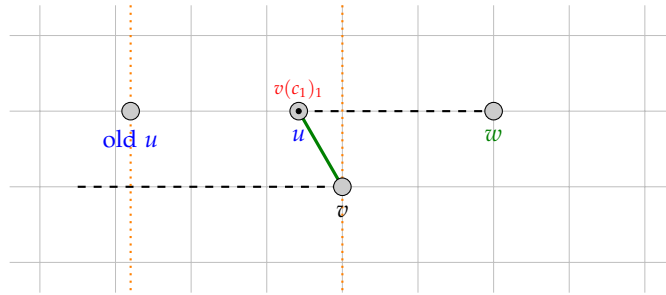


Figure 4.16. The orange indicated interval needs to be moved to the small interval between  $u$  and  $v$ .

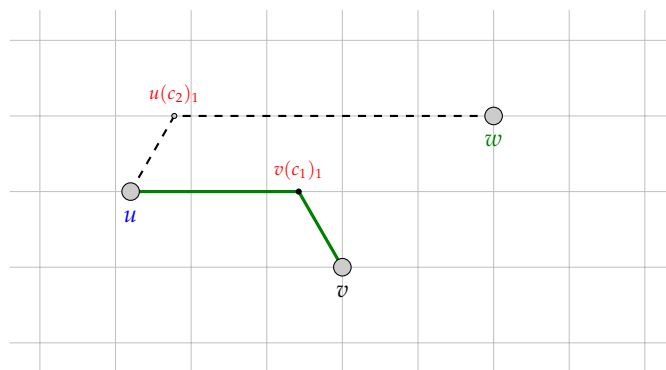


Figure 4.17. Track  $uv$  represented by an outgoing right-down line.

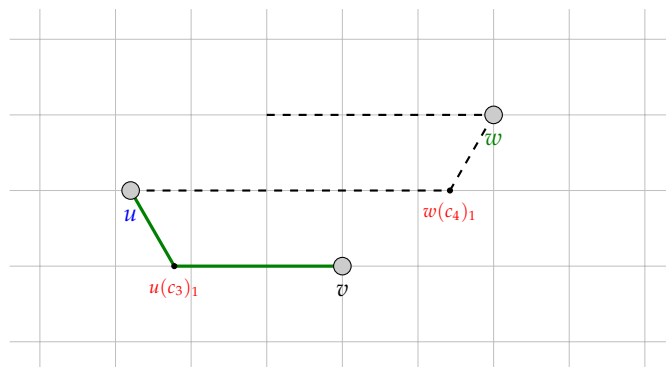


Figure 4.18. Track  $uv$  represented by an outgoing down-right line.

**Algorithm 5:** Assigning schema  $x$ -values

---

**Input** : LR data of the input map and all straight lines, schemay-values, diverging parallel tracks, ordered list of junctions, positive\_lines, negative\_lines, first assigned schema  $x$ -values

**Output** : Final schema  $x$ -values

```

done ← []
shifted ← []
representations ←
N ← #positive_lines
schema_x ← first assigned x-values

for i in range(0, N) do
  if positive_lines[i] is not a diverging parallel track then
    current_line ← positive_lines[i]
    for junction in current_line do
      v ← junction
      k ← i
      while v not in done do
        for j in range(k + 1, N) do
          if positive_lines[j] is not a diverging parallel track then
            if v is the target of a track with  $y_{source} == j$  and source not in done then
              u ← source
              uv ← track

              if u has another outgoing track then
                w ← target of other track
                uw ← other track
                if uw is completely straight then
                  if v has an incoming straight track then
                    schema_x[u] ←  $v(c_1)_{|j-i|}$ 
                    shifted.append(u)
                    representations[uv] ← outgoing down line
                  else
                    representations[uv] ← outgoing down-right line
                else
                  /* uw is also diverging */
                  if w has not a completely straight incoming track then
                    representations[uv] ← outgoing right-down line
                  else if w has a completely straight incoming track and v not then
                    representation[uv] ← outgoing down-right line
                  else
                    /* both have a completely straight incoming track */
                    if  $x_v < x_w$  then
                      schema_x[u] ←  $v(c_1)_{|j-i|}$ 
                      shifted.append(u)
                      representations[uv] ← outgoing down line
                    else
                      schema_x[u] ←  $w(c_4)_{|j-i|}$ 
                      shifted.append(u)
                      representations[uv] ← outgoing right-down line
              else
                /* u has no other outgoing track */
                done.append(v)
                break
            else
              if v is the source of a track with  $y_{target} == j$  and target not in done then
                Similar

```

---

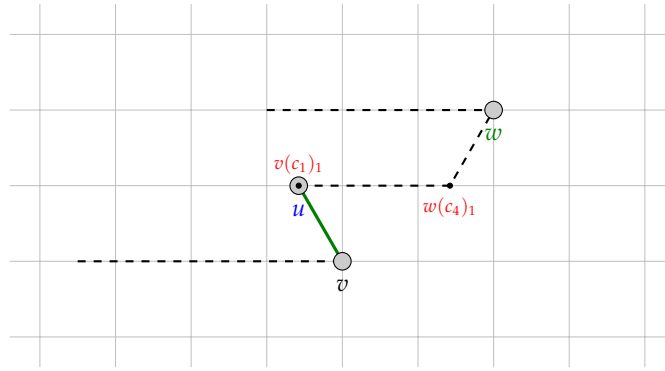


Figure 4.19. Track  $uv$  represented by an outgoing down line.

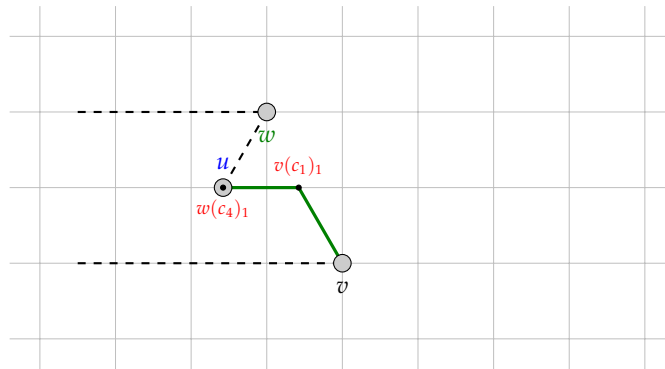


Figure 4.20. Track  $uv$  represented by an outgoing right-down line.

## 4.7 RESULTS AND ANALYSIS

### 4.7.1 Output examples

We start with some small input maps where junctions are only of type single switch. Figures 4.21 and 4.22 show the geographical input map of Ommen. Next, we find all straight lines and determine the longest straight line. The latter consists out of merged tracks, depicted in figure 4.6. The longest straight line is used as a base line for the linear referencing algorithm, see figures 4.24 and 4.25. The schematic representation of the Ommen network is shown in figure 4.26.

Figures 4.27 - 4.32 show the same process for the network of Rijen, and figures 4.33 - 4.38 depict the steps to schematization of the network of Steenwijk.

While the created schematic diagrams satisfy the conditions, there occurred a problem in the output of Steenwijk. See figures 4.39 - 4.41. According to the original  $x$ -values of the junctions, junction  $B$  lies between junction  $A$  and junction  $C$ . And since track  $AC$  must be drawn using an angle of  $60^\circ$ , junction  $B$  is placed in the very small interval between  $A$  and  $C$ , causing junctions  $B$  and  $C$  to partially overlap in the visualization. Technically  $B$  still precedes  $C$ , so the order is correct, but according to the most important reason of schematization, i.e. improving the visualization of the network, this is a problem.

Next, we have created a schematic representation of a small part of the

network of Utrecht, including junctions of type full slip additional to single switch type junctions, see figures 4.42 and 4.43. We see that the left side of the network after linear referencing is sort of aligned, figure 4.44, while this is lost in the schematic diagram figure 4.45. Again all the conditions are met, but it appears more chaotic.

While the left side of the network appears to be aligned, the associated junctions have slightly different  $x$ -values. And since the junctions are ordered, their difference will be greater in the schematic diagram.

Figures 4.46 and 4.57 show the input map of Zuthpen after linear referencing and the associated schematic output map. Now the junctions are of type single switch and crossing. The crossings in the schematic diagram are no longer aligned, see figures 4.48 and 4.49. Also there is a problem, see figure 4.50. Junction  $E$  appears before junction  $D$  while in the original input map it is the other way around. Actually, this part of the network leads to a conflict regarding the drawing conditions. Since track  $AE$  needs to be drawn using an angle of  $60^\circ$ , all junctions between  $A$  and  $E$  must fit in the small area created by the schema  $x$ -values of  $A$  and  $E$ . So junctions  $B$ ,  $C$ , and  $D$  have a schema  $x$ -value between  $A$  and  $E$ . The problem occurs when track  $BD$  needs to be drawn using an angle of  $60^\circ$  as well. This forces junction  $D$  to lie beyond junction  $E$ . Adjusting the schema position of junction  $E$  will lead to the wrong order of  $A$  and  $B$ . This configuration is therefore not solvable using the given drawing conditions.

To get the crossings more aligned we set a margin, see figure 4.57.

The last example includes a network with all junction types. See figures 4.51 and 4.52 regarding the network of Weert. Junctions in the red indicated areas, see figures 4.53 and 4.54, are too close together due to the  $60^\circ$  angle condition and the order condition.



Figure 4.21. Input map of Ommen.

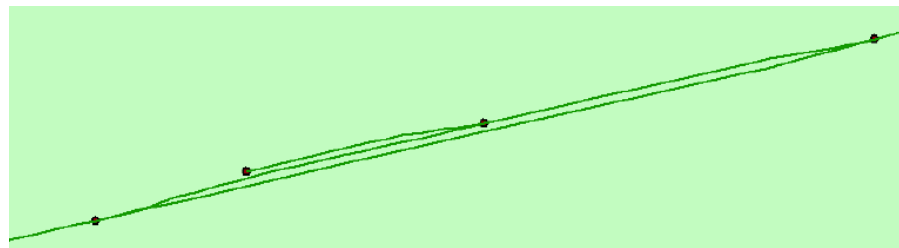


Figure 4.22. Input map of Ommen, zoomed in.



Figure 4.23. Merged tracks that form the longest straight line.



Figure 4.24. Network of Ommen after linear referencing.



Figure 4.25. Network of Ommen after linear referencing, zoomed in.



Figure 4.26. Schematic representation of Ommen.

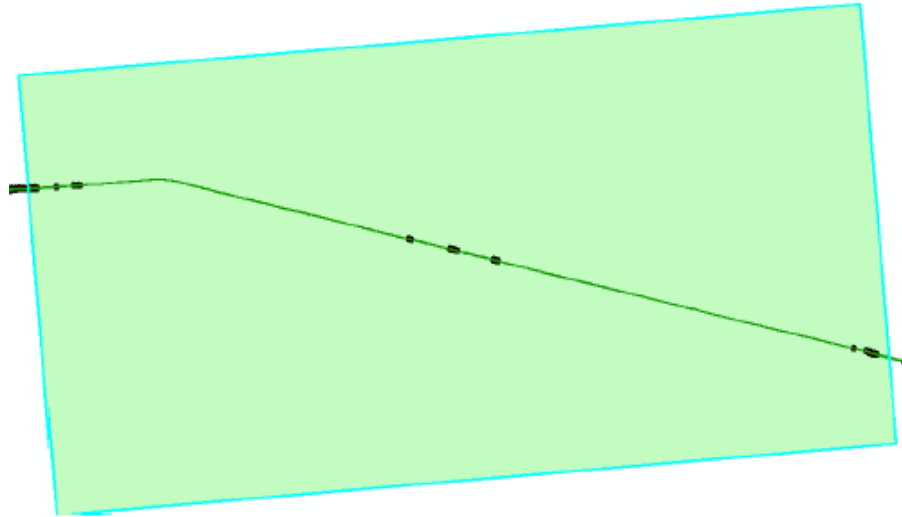


Figure 4.27. Input map of Rijen.

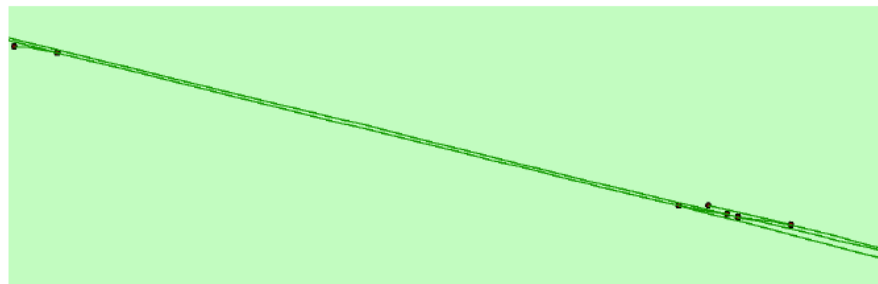


Figure 4.28. Input map of Rijen, zoomed in.



Figure 4.29. Merged tracks that form the longest straight line.



Figure 4.30. Network of rijen after linear referencing.





Figure 4.31. Network of Rijen after linear referencing, zoomed in.

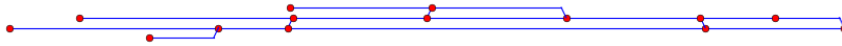


Figure 4.32. Schematic representation of Rijen.

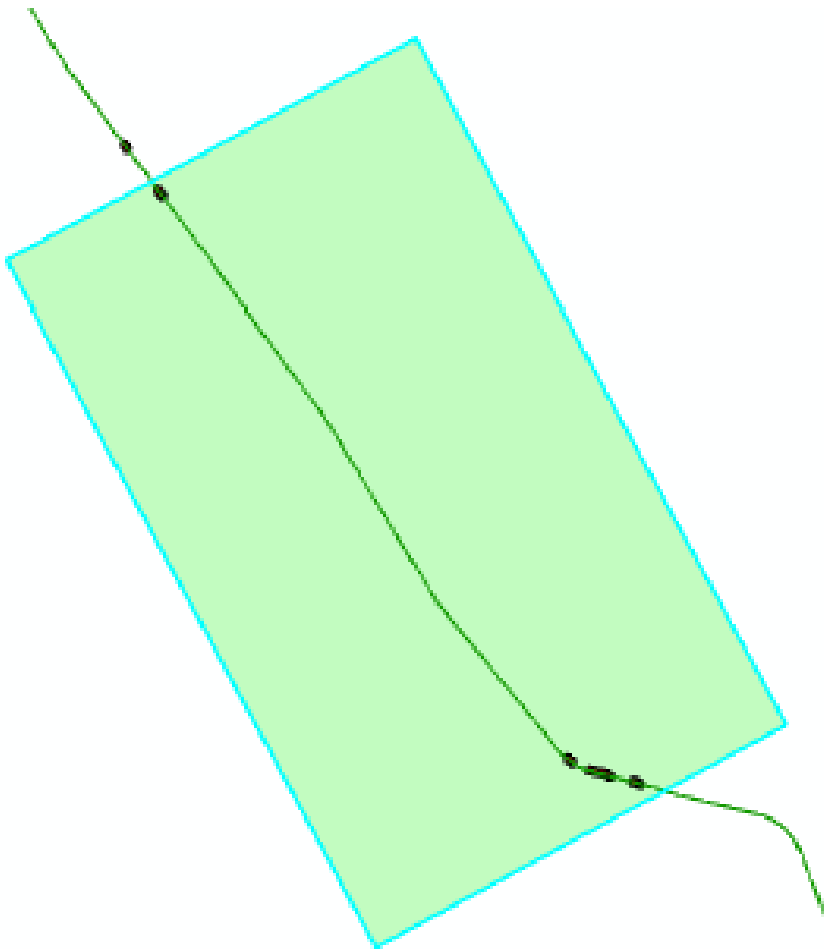


Figure 4.33. Input map of Steenwijk.

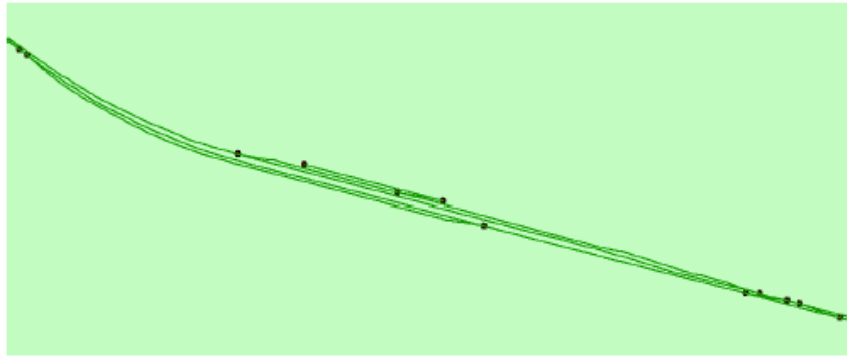


Figure 4.34. Input map of Steenwijk, zoomed in.

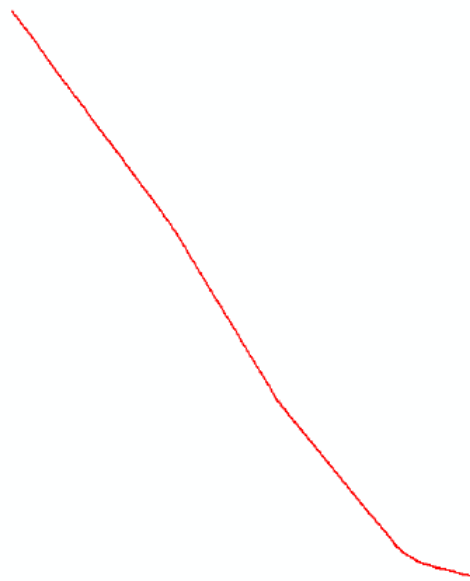


Figure 4.35. Merged tracks that form the longest straight line.



Figure 4.36. Network of Steenwijk after linear referencing.

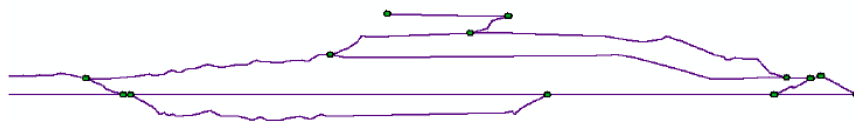


Figure 4.37. Network of steenwijk after linear referencing, zoomed in.



Figure 4.38. Schematic representation of Steenwijk.

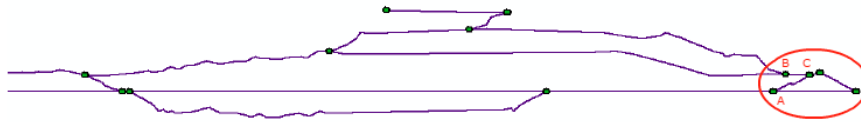


Figure 4.39. Network of Steenwijk after linear referencing, zoomed in. Indicating a problem in red.

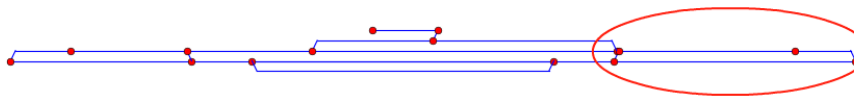


Figure 4.40. Schematic representation of Steenwijk. Indicating a problem in red.



Figure 4.41. Junctions *B* and *C* overlap.

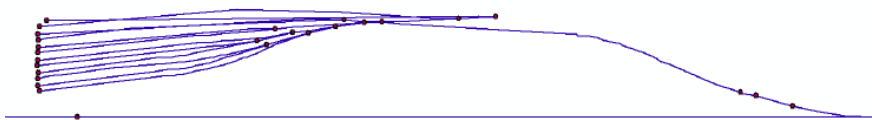


Figure 4.42. Part of the network of Utrecht, after linear referencing.

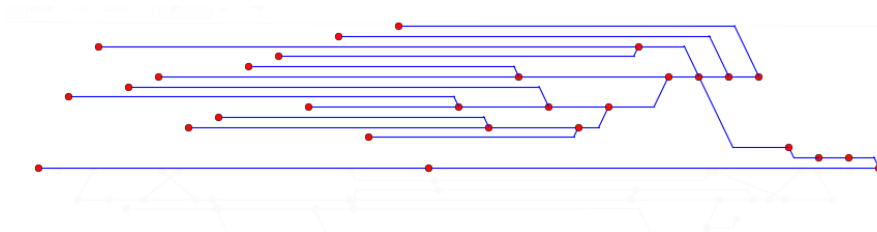


Figure 4.43. Schematic representation of a part of the network of Utrecht.

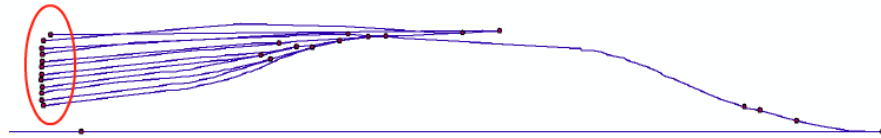


Figure 4.44. Part of the network of Utrecht, after linear referencing.

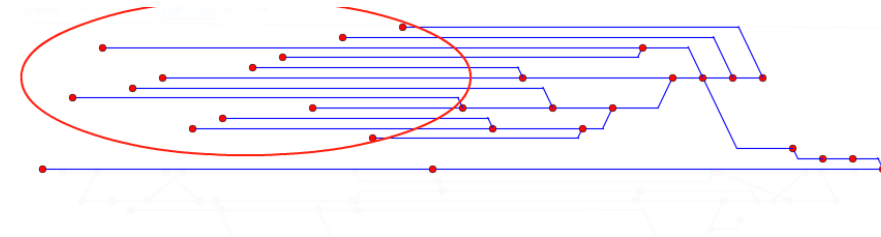


Figure 4.45. Schematic representation of a part of the network of Utrecht.



Figure 4.46. The network of Zutphen, after linear referencing.

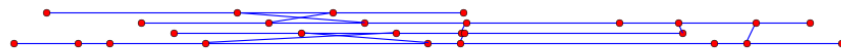


Figure 4.47. Schematic representation of the network of Zutphen.



Figure 4.48. The network of Zutphen, after linear referencing.

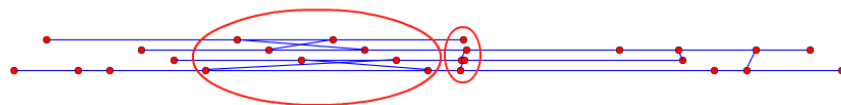


Figure 4.49. Schematic representation of the network of Zutphen.

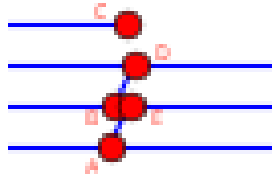


Figure 4.50. Schematic representation of the network of Zutphen.

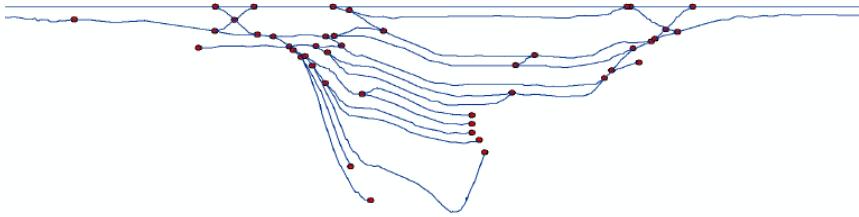


Figure 4.51. The network of Weert, after linear referencing.

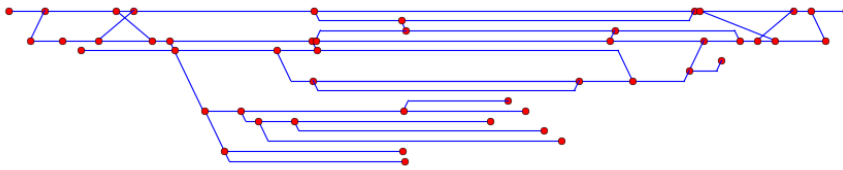


Figure 4.52. Schematic representation of the network of Weert.

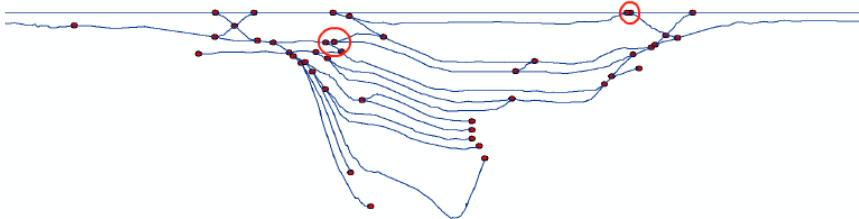


Figure 4.53. The network of Weert, after linear referencing.

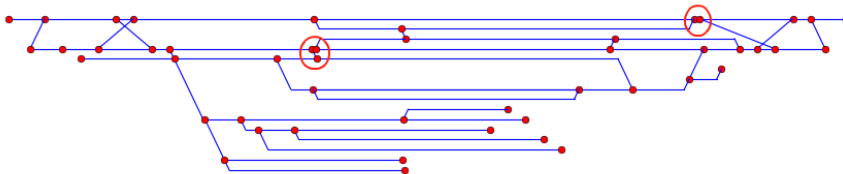


Figure 4.54. Schematic representation of the network of Weert.

## 4.7.2 Problems and solutions

We found out that there is a configuration such that no solution exists using the given drawing conditions. The  $60^\circ$  angle rule and the order condition can create a conflict. Also we saw that junctions can overlap in the schematic diagram. These problems can be solved by choosing a smaller angle say of  $30^\circ$  and setting the maximum number of bends per line at two.

A possible drawing of the conflict area in the network of Zutphen is shown in figure 4.55.

We saw that if an input map contains visually aligned junctions, while their  $x$ -values are slightly different, this visual alignment is lost in the schematic map. To solve this we could set a margin for the distance between junctions. If the distance between two junctions is within that margin, we'll treat them as if they have the same position in the order and they are assigned the same schema  $x$ -value. See for example figures 4.56 and 4.57.

Also crossings appear a bit messy since they have different angles in the output map. This is easily solved by using the same drawing rule for all angles in the schematic diagram.

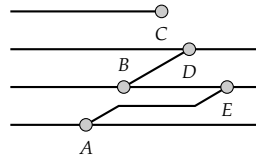


Figure 4.55. Possible drawing solution of the conflict area in the network of Zutphen.

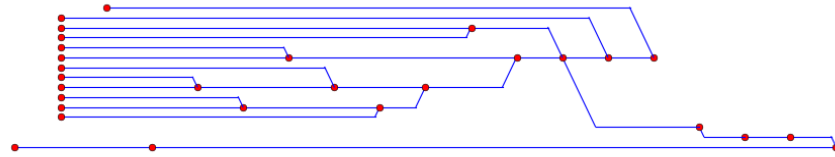


Figure 4.56. Aligned schematic representation of a part of the network of Utrecht.



Figure 4.57. Aligned schematic representation of the network of Zutphen.

While testing the algorithm on bigger input maps, we found out that there can be a consistency problem, i.e. the same input map yields different schematic output maps. This is due to the randomness in the schema  $y$ -value algorithm together with the way we order two non-overlapping straight lines. See for example figure 4.59. When starting with  $C$ ,  $B$  lies above  $C$  and  $A$  lies below  $C$ . The vertical order from above is then  $BCA$ . Again, startig with  $C$  but now the algorithm first tests  $A$ . Then  $A$  is below  $C$ , and  $B$  is below  $A$ , giving the vertical order from above  $CAB$ .

This consistency problem can be solved by only defining a vertical order of overlapping straight lines. This means that in the above given example, we cannot say that  $C$  lies above  $A$ . We only know that  $A$  is above  $B$  and  $B$  is above  $C$ . This will always give the vertical order of  $ABC$ .

Recall that we use the longest straight line to rotate, scale and straighten the network. As mentioned a longest straight line covering the whole network exists most of the time but not always. To overcome this problem we can create an additional linear line below the network. See for example figure 4.58. This additional linear line is then used as a base line for linear referencing.

Furthermore, assumption 3 seems incorrect. So a crossing can lie on a straight line. This can be solved by adjusting the straight line algorithm.

The proposed solutions give rise to a new set of aesthetic criteria and drawing rules. And in the process of developing the algorithm described in this chapter, we gained some new insights to develop a new algorithm. In the next chapter we will define the new set of criteria and outline the idea of our new algorithm.



Figure 4.58. Input map of Ommen with an additional linear line.

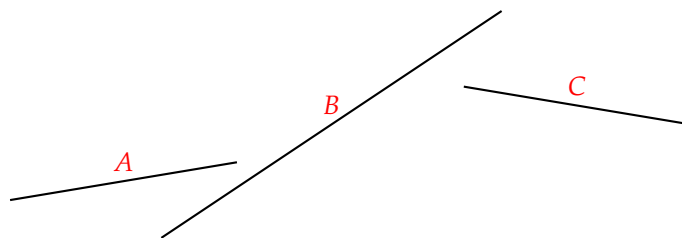


Figure 4.59. Three straight lines.





# 5 | CONCEPTUAL ALGORITHM

In this chapter we will describe the concept of our new algorithm. First we will give a new set of aesthetic criteria, some straight forward definitions and drawing rules. Then we globally describe the steps of the algorithm leaving out the technical details. We conclude with a few schematic diagrams, manually made following the steps of the algorithm.

## 5.1 AESTHETIC CRITERIA

We define the following new set of aesthetic criteria:

1. the topology of the underlying graph of the railway network is preserved
2. after including a margin, the order of the junctions in the main direction remains the same
3. every track is represented as straight line segments with at most two bends
4. straight tracks are drawn horizontally as much as possible
5. all angles are  $30^\circ$  or  $-30^\circ$

Furthermore, some additional wishes include:

- the schematic diagram must be compact
- the output is consistent

## 5.2 DEFINITIONS AND DRAWING RULES

Before we outline the approach of our new algorithm we need to make some new definitions.

As explained before, a track is straight or diverging out of its source junction and straight or diverging into its target junction. Recall the definition of a completely straight:

**Definition 4.** *A track is called completely straight if it is the straight track out of its source junction and the straight track into its target junction.*

Similarly we define the following:

**Definition 5.** *A track is called completely diverging if it is the diverging track out of its source junction and the diverging track into its target junction.*

**Definition 6.** *A track is called straight-diverging if it is the straight track out of its source junction and the diverging track into its target junction.*

**Definition 7.** A track is called *diverging-straight* if it is the diverging track out of its source junction and the straight track into its target junction.

We introduce the following drawing rules. A completely straight track is always drawn horizontally, see figure 5.1. A straight-diverging track can be drawn using at most one bend as shown in figure 5.2. In a similar manner we draw a diverging-straight track. We prefer to draw a completely diverging track as shown in figures 5.3a and 5.3b. If this is not possible then the line that represents the track includes two bends as in figures 5.3c and 5.3d.

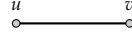


Figure 5.1. Representation of the completely straight track  $uv$ .

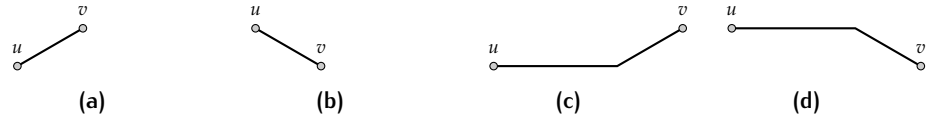


Figure 5.2. Possible representations of the straight-diverging track  $uv$ .

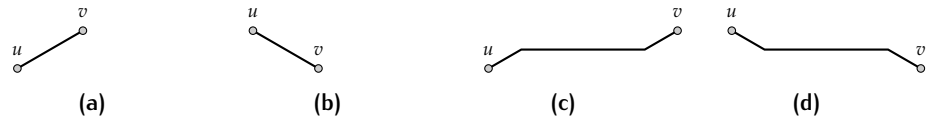


Figure 5.3. Possible representations of the completely diverging track  $uv$ .

### 5.3 CONCEPT

1. Start with an input map and create an additional linear line, a *base line* below the network of the input map. See for example figure 4.58. Rotate the network such that the added line becomes horizontal. This can be done by linear referencing.
2. To make the network easier to work with, modify the relevant data such that for all tracks with source junction at  $(x_0, y_0)$ , target junction at  $(x_1, y_1)$  holds:  $x_0 \leq x_1$ . See figure 4.3.
3. Determine for each track its "type", i.e. if it is completely straight, completely diverging, straight-diverging or diverging-straight.
4. Determine all the straight lines. \*Aanpassingen kruisingen en slips\*.
5. Determine the vertical order of the straight lines. This is done by using lines, through each junction, perpendicular to the base line. (nog meer uitleggen, parallel tracks)
6. Use the vertical order of the straight lines to assign a schema  $y$ -value to every junction and track.
7. Including a margin, create a horizontal order of the junctions.
8. Place the first junction at  $(0, 0)$ . Then, since we know for each junction its schema  $y$ -value and its type, we can calculate its schema  $x$ -position.

## 5.4 MANUALLY MADE SCHEMATIC DIAGRAMS

Using the above mentioned approach, and by carefully calculating the schema  $x$ -positions, we made a schematic map of the network of Ommen, see figure 5.4, the network of Steenwijk, figure 5.5, and the network of Weert, see figures 5.6 and 5.7.

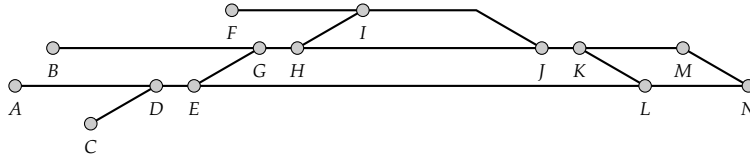


Figure 5.4. Manually made schematic map of the network of Ommen.

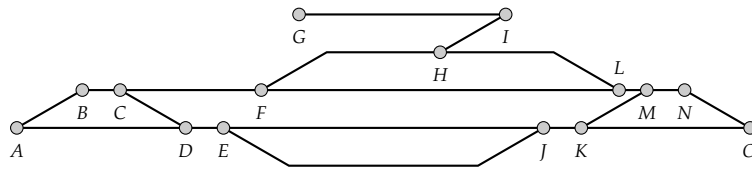


Figure 5.5. Manually made schematic map of the network of Steenwijk.

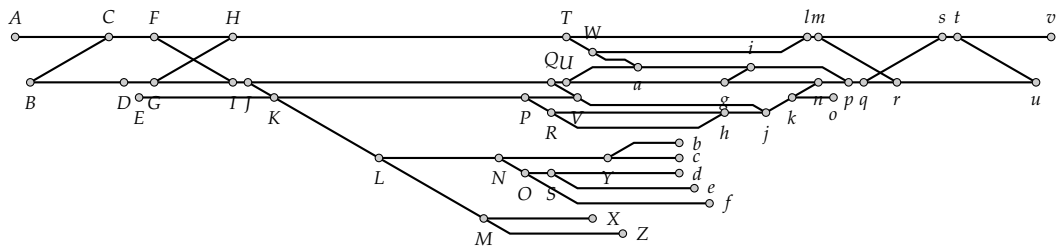


Figure 5.6. Manually made schematic diagram of the network of Weert.

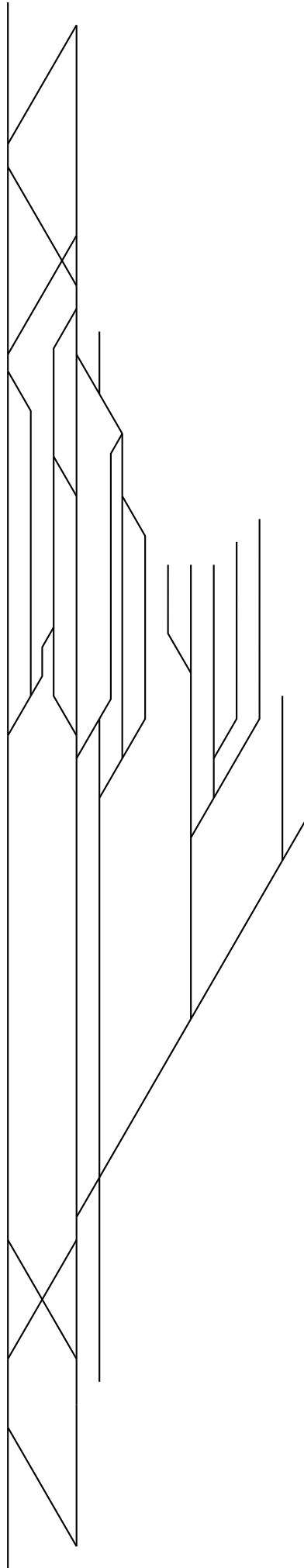


Figure 5.7

## BIBLIOGRAPHY

- Hong, S.-H., Merrick, D., and do Nascimento, H. A. D. (2005). *The Metro Map Layout Problem*, pages 482–491. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Nollenburg, M. and Wolff, A. (2011). Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641.
- Oke, O. and Siddiqui, S. (2015). Efficient automated schematic map drawing using multiobjective mixed integer programming. *Computers & Operations Research*, 61:1 – 17.