Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Segmentierung eines Schienennetzes“

Entwicklung eines Algorithmus zum automatisierten Finden von Fahrstrassen für Messzüge

vorgelegt von

Luca Roost
105414, UNIGIS MSc Jahrgang 2019

Betreuer/in:
Prof. Josef Strobl

Zur Erlangung des Grades
„Master of Science – MSc”

Bern, 25.11.2021

Danksagung

An dieser Stelle möchte ich mich bei Prof. Josef Strobel für die Betreuung meiner Masterarbeit bedanken.

Ausserdem bedanke ich mich bei folgenden Personen ganz besonders:
Sarah Zbinden hat mich mit ihren Inputs und der Korrektur der Rechtschreibung tatkräftig unterstützt.

Bei Fragen zu Informatikthemen war Dano Roost stets meine erste Ansprechperson und hat mir mehrmals den Weg aus einer Sackgasse gezeigt.

Beide waren mir in dieser herausfordernden Zeit immer wieder behilflich und ich durfte von ihrem Wissen viel profitieren.

Ebenfalls gebührt meiner Familie und meiner Lebenspartnerin einen grossen Dank, welche mich in der Zeit des Studiums unterstützt und begleitet haben. Ohne ihre Hilfe, Geduld, motivierenden Worte und Unterstützung wäre das Studium und insbesondere diese Arbeit nicht möglich gewesen.

Abstract

The measurement and diagnostic technology department of the Swiss Federal Railways SBB regularly inspects the entire rail network with the aim of ensuring the safety of the transport of goods and people by rail. The planning of these measurement runs is very complex; various bodies are involved, there are multiple dependencies and diverse and changing needs must be addressed. Until now, most of the work involved in planning these measurement runs, has been done by hand. With the presented master thesis, an algorithm was developed for the first time to automate certain work steps and thus save valuable time and costs. The aim was to divide the SBB rail network into individual, drivable segments by means of segmentation. At the beginning, a simplified fictitious network was created with the purpose of keeping the runtime as low as possible and being able to visually control the results. Using a case study with the perimeter of the Val de Travers, the algorithm was then tested on existing, real data. The segmentation of the geodata from the SBB geodatabase was successfully carried out with an unweighted graph and a modified depth-first search in the test perimeter of the Val de Travers and can now be extended to the entire rail network in Switzerland. The segmentation is the basic prerequisite for later network analyses, such as the solution of the postman problem, which aim to find the most cost-effective path in the network. For these, additional factors would still have to be included so that the algorithm can finally be applied to a weighted graph.

Glossar

BAV Bundesamt für Verkehr der Schweizerischen Eidgenossenschaft. 16

BFS Breath First Search/Breitensuche, ein Suchverfahren zum Finden bestimmter Knoten in Graphen (Volbert, 2011). 11

DfA Datenbank feste Anlangen - Geodatenbank der SBB Infrastruktur. 14, 25, 26, 40–42

DFS Depth First Search/Tiefensuche, ein Suchverfahren zum Finden bestimmter Knoten in Graphen (Volbert, 2011; Hoegel, 2021). 11

DFZ Diagnosefahrzeug. 3, 38

EVU Eisenbahnverkehrsunternehmen. 16

FDL Fahrdienstleiter:in - Koordiniert und überwacht der laufende Bahnverkehr.. 39

FDV Fahrdienstvorschriften - Schweizerisches Regelwerk für den Bahnverkehr. 16

gDFZ gezogenes Diagnosefahrzeug. 3

MUD Mess- und Diagnosetechnik. 3

NeTS Netzweites Trassensystem - System zur Planung und Bestellung von Zugstrassen, resp. Fahrplänen (Schweizerische Bundesbahnen, 2021a). 5, 39

Pendelzug Ein Zugsverband, bei dem sich auf der einen Seite eine Lok und auf der anderen Seite ein Steuerwagen befindet. 3

RCS Rail Control System - System zum Einstellen der Fahrwege für Zugfahrten und Rangierbewegungen. 17

SBB Schweizerische Bundesbahnen. 2

SPZ Ultraschall-Schienenprüfzug. 4

Trasse Eine Trasse ist die Berechtigung, eine bestimmte Strecke des Bahnnetzes zu definierten Zeiten mit einem spezifischen Zug (Länge, Gewicht, Profil, Geschwindigkeit) zu befahren.(Trasse Schweiz, 2021). 4, 5

UVEK Eidgenössisches Departement für Umwelt und Verkehr. 16

Inhaltsverzeichnis

1 Einleitung

In naher Zukunft wird die Eisenbahn als Transportmittel für Güter und Personen gemäss verschiedener Prognosen an Bedeutung gewinnen (Bundesamt für Verkehr, 2007; Stölzle et al., 2015). Zum einen können große Lasten schnell und kostengünstig transportiert werden, zum anderen wird ihr bei der Reduzierung der CO2-Emissionen eine grosse Rolle beigemessen (Bundesamt für Statistik, 2021). Das Schweizer Schienennetz umfasst 5'196 km Gleise (Figure 1.0.1) (Bundesamt für Statistik, 2020). Mit mehr als 10'000 Zügen pro Tag wird diese Infrastruktur so stark genutzt wie nirgendwo sonst in Europa (Jorio, 2021). Neben dem dichteren Fahrplan nehmen auch andere Faktoren zu, die zu einem höheren Verschleiß der Gleise führen. Dazu gehören immer höhere Geschwindigkeiten, leistungsfähigere Antriebs- und Bremssysteme und bessere Elektronik, welche das *Gleiten* oder *Schleudern* verhindert. Um die Sicherheit zu gewährleisten, muss diese Infrastruktur in regelmässigen Abständen überprüft werden. Diese Aufgabe übernimmt in der Schweiz die Abteilung *Mess- und Diagnosetechnik* der Schweizerischen Bundesbahnen SBB, welche Messfahrten plant und durchführt. Die Planung und Durchführung solcher Messfahrten ist aufwändig und mit viel Handarbeit verbunden. Da das Schienennetz komplex und stark ausgelastet ist, spielen die Effizienz und die Menge an Leerfahrten eine wichtige Rolle. Um den ständig wechselnden Ansprüchen und dem dynamischen System Rechnung zu tragen, müssen die Fahrten individuell geplant werden. Um die Effizienz zu erhöhen und den Aufwand bei der Planung zu reduzieren, können bestimmte Abläufe mit Werkzeugen der Geoinformatik automatisiert werden. Diese Arbeit soll dabei aufzeigen, wie mithilfe geschickter Gleis-Segmentierung der Zeitaufwand der Planung reduziert und das Fehlerpotenzial minimiert werden kann.

Figure 1.0.1: Gleisnetz der Schweiz

Figure 1.0.2: Selbstfahrendes Diagnosefahrzeug (DFZ) der SBB

1.1 Infrastrukturdiagnose der SBB

Für die Diagnose der gesamten Bahn-Infrastruktur der *SBB* ist die Abteilung *Mess- und Diagnosetechnik* (MUD) verantwortlich. Aufgaben dieser Abteilung sind die Erfassung der Zustandsdaten sowie die Planung und die fahrdienstliche Ausführung der Messfahrten. Für die Durchführung der Messfahrten stehen drei *Messzüge* zur Verfügung, welche speziell für diese Aufgaben entwickelt wurden. Nebst den unten erwähnten Fahrzeugen werden weitere Wagen für spezielle Aufgaben verwendet. In regelmässigen Intervallen unterwegs und somit für die vorliegende Fragestellung von primärem Interesse, sind aber nur die genannten Fahrzeuge.

1.1.1 Diagnosefahrzeug (DFZ)

Beim Diagnosefahrzeug (DFZ) (Figure 1.0.2) handelt es sich um ein mit Diesel betriebenes Spezialfahrzeug, welches sämtliche Fahrbahnparameter, wie zum Beispiel die Oberfläschenbeschaffenheit des Gleises sowie einige Fahrstromparameter, wie die Zick-Zack-Führung des Fahrdrahtes, erfassen kann. Das Fahrzeug wurde im Jahr 2006 in Betrieb genommen und stellt seither das Rückenmark der MUD dar. Es verkehrt mit bis zu 160 $^{km}/_{h}$ auf Überfuhrfahrten und kann Messungen mit bis zu 120 $^{km}/_{h}$ durchführen (Mermec GROUP, 2021). Damit sind auch Messungen am Tag, inmitten des auch sonst dicht befahrenen Schienennetzes möglich, da dank der hohen Geschwindigkeiten die Streckenabschnitte nicht lange belegt werden müssen.

1.1.2 Gezogenes Diagnosefahrzeug (gDFZ)

Das gezogene Diagnosefahrzeug (gDFZ) ist ein sich noch in der Entwicklung befindendes Fahrzeug, welches auf einem Einheitswagen *IV* der SBB basiert. Zweck des Fahrzeuges ist die Erfassung sämtlicher Fahrstromparameter sowie einiger Fahrbahnparameter. Das Fahrzeug muss im Gegensatz zum DFZ stets in einem Zugverbund vekehren. Das bedeutet, dass es entweder in einem Pendelzug integriert sein oder eine Lok vorgespannt werden muss. Der damit verbundene logistische Aufwand, eine Messfahrt durchzuführen, ist somit ungleich grösser als mit dem selbstfahrenden DFZ. Hingegen sind mit dem gDFZ Überfuhrgeschwindigkeiten bis zu 200 $^{km}/_{h}$ und Messfahrten bis zu 160 $^{km}/_{h}$ möglich (Schmid, 2019). Gerade in den Tunnels am Lötschberg und im Gotthard bringt diese Eigenschaft einen gewichtigen Vorteil mit sich, da ein langsamer Zug die Gleisabschnitte zu lange belegt und somit den Alltagsverkehr einschränkt.

1.1.3 Ultraschall-Schienenprüfzug (SPZ)

Beim Ultraschall-Schienenprüfzug (SPZ) handelt es sich um eine Komposition aus einem modifzierten Flachwagen, welcher die Messinstrumente enthält sowie aus einem umgebauten Personenwagen, in welchem sich das Messpersonal aufhält. Da für die Ultraschall-Analyse ein Kontaktmittel zwischen die Messradsätze und die Schienen aufgesprüht werden muss, verkehrt dieses Fahrzeug auf Messfahrten mit lediglich 60 $^{km}/_{h}$. Dies führt dazu, dass die Ultraschall-Messfahrten jeweils bei Nacht stattfinden müssen, da die Streckenabschnitte tagsüber nicht für eine so lange Zeit zur Verfügung stehen.

1.2 Planung von Messfahrten

Nebst den technischen Herausforderungen, welche bei der Durchführung und der Auswertung von Messfahrten entstehen, wird auch die Planung der Fahrten immer anspruchsvoller. Durch das erhöhte Verkehrsaufkommen und die immer höheren Geschwindigkeiten, nehmen die Trassen, welche für die Messfahrten zur Verfügung stehen, jährlich ab. Besonders für kurzfristige Messfahrten, beispielsweise bei Wiederholungsfahrten oder nach Reparaturen an der Gleisanlage, und für Messungen mit den SPZ, stehen nur Schichten in den Nachtstunden oder am Wochenende zur Verfügung, was zu Engpässen beim Personal führen kann. Aus diesem Grund muss der effizienten Planung der Messschichten in Zukunft mehr Aufmerksamkeit gewidmet werden, um den erhöhten Personalkosten, welche in der Nacht und am Wochenende entstehen, entgegen zu wirken. Die Überprüfung der Gleisanlage in der Schweiz wird thematisch getrennt in verschiedene Messkampagnen unterteilt. Grund dafür sind die unterschiedlichen Ansprüche an die Messfahrzeuge. Jede dieser Messkampagnen umschliesst einen Perimeter, der anhand verschiedener Kriterien definiert wird. Für Streckengleise zwischen zwei Bahnhöfen gibt es beispielsweise pro Jahr zwei deckungsgleiche Kampagnen, die sogenannten Regelmessfahrten. Der Perimeter einer Regelmessfahrt-Kampagne erschliesst sich aus allen Streckengleisen ohne Spurwechsel und den Durchfahrtsgleisen in den Bahnhöfen, welche mit einer Geschwindigkeit $\geq 80^{km}/_{h}$ befahren werden können. Da in den Gleisstrang-Geodaten nicht deklariert ist, wie hoch die maximale Geschwindigkeit ist und ob es sich um ein Strecken- oder Bahnhofsgleis handelt, muss die Selektion der entsprechenden Gleisstränge manuell erfolgen, was mit einem hohen Zeitaufwand verbunden ist.

Bis ins Jahr 2019 wurden für die Planung einer Kampagne Pläne der Gleisanlage für jeden Bahnhof ausgedruckt. Die messrelevanten Gleisstränge wurden von Hand farbig markiert, was einen hohen Aufwand mit sich brachte und auch aus ökologischer und ökonomischer Sicht keinen Sinn ergab. Seit dem Jahr 2017 verfügt die SBB über eine Geodaten-Plattform *GSHARP*, welche das Erfassen und Bereitstellen der Gleisanlange als Geodaten ermöglicht (ESRI Schweiz, 2020). Mit Hilfe dieser Daten konnte der Perimeter erstmals so definiert werden, dass jeder Gleisstrang, welcher überprüft werden sollte, aufgelistet war. Die Selektion erfolgte jedoch weiterhin manuell. Trotzdem war am Schluss ein Datensatz vorhanden. Obwohl die Selektion weiterhin von Hand erfolgte, hat dieser neue Prozess den Vorteil, dass daraus ein digitaler Datensatz resultiert. Dieser kann auf seine Vollständigkeit hin überprüft werden. Auch die Bezeichnung eines Gleisstrangs war deklariert. So konnte im Anschluss an die Messung gezielt nach einem Gleisstrang gesucht und erörtert werden, ob er effektiv befahren wurde und ob Mess-

daten vorhanden sind. Die Einteilung des Perimeters in Tages-Touren, welche nur an definierten Standorten starten und enden können, blieb aber Handarbeit. Der Zeitaufwand für die Planung blieb entsprechend hoch. Hinzu kam der Faktor Effizienz. Durch die manuelle Planung kam es zu vielen mehrfach unnötig befahrenen Strecken. Dieser Umstand wird nie ganz behoben werden können, da die Übernachtungsstandorte für das Messfahrzeug in der Schweiz nicht regelmässig verteilt sind. Da jedoch jeder gefahrene Kilometer Kosten verursacht und insbesondere bei thermisch betriebenen Fahrzeugen auch zu mehr $CO_2 - Emissionen$ führt, sollten Leerfahrten möglichst vermieden werden. Die hohe Fehleranfälligkeit der manuellen Planung ergänzt die Liste der Herausforderungen, der sich das Messpersonal zu stellen hat. Bis das Messfahrzeug auf Messfahrt gehen kann, werden die folgenden Schritte benötigt, welche die Komplexität des Bahnsystems aufzeigen: Sobald der gesamte Perimeter in Tages-Einsätze unterteilt ist, müssen die Trassen bestellt werden. Das erfolgt über das Trassenbestellportal *Netzweites Trassensystem* (NeTS). Die Bezeichnungen der Gleise, welche dort hinterlegt werden müssen, stimmen jedoch nicht mit den Gleisstrang-Geodaten überein. Um diese zu erhalten, muss eine erneute *GIS-Überlappungsanalyse* durchgeführt werden. Da NeTS über keine *Application Programming Interface* verfügt, müssen die Gleisdaten auch hier manuell eingetragen werden, was ebenfalls ein hohes Fehlerpotenzial und einen hohen Zeitaufwand mit sich bringt. Die in NeTS erfassten Fahrten werden anschliessend von einem Fahrplanplaner trassiert und mit einem Trassenangebot an den Besteller zurückgesendet. Ist dieser mit den Verkehrszeiten zufrieden und stimmen die trassierten Gleise mit der Bestellung überein, kann die Messfahrt definitiv bestätigt werden.

1.3 Zielsetzung und Fragestellungen

Die Planung und Durchführung einer Messfahrt ist also ein sehr zeit- und arbeitsaufwändiger Prozess. Da sich alle Messfahrten voneinander unterscheiden, ist das Wiederverwenden von vorhandenen Daten nicht möglich oder nur wenig sinnvoll. Trotzdem gibt es bestimmte Teilprozesse, welche mit der heute verfügbaren Geoinformatik automatisiert oder zumindest teilautomatisiert werden können. Insbesondere das Erheben der kritischen Gleisstränge und die Planung der einzelnen Messschichten birgt viel Potenzial, um die Effizienz zu erhöhen und die Kosten zu senken. Bekannte Netzwerk-Algorithmen können ähnliche Probleme sehr gut lösen (Andreev & Racke, 2006). In einem Bahnnetzwerk gelten jedoch bestimmte Spezialregeln, welche berücksichtigt werden müssen. Daher ist die komplette Automatisierung sehr umfangreich. Wäre es jedoch möglich, das Schienennetzwerk in fahrbare Teilstrecken zu segmentieren, könnte der Aufwand entscheidend reduziert werden. Die grundlegende Fragestellung dieser Arbeit lautet also:

- Inwiefern können Bahnnetzwerke automatisch segmentiert werden, um sie für eine allfällige Netzwerk- oder Routenanalyse vorzubereiten?
- Ist es möglich, den Algorithmus so anzupassen, damit vorhandene Geodaten verwendet werden können, ohne dass ein zusätzlicher manueller Aufwand entsteht?
- Wie zuverlässig sind die Resultate des Algorithmus im Vergleich mit manuell segmentierten Daten?

Ziel dieser Arbeit ist es, exemplarisch aufzuzeigen, wie die Planung von Messfahrten auf einem Eisenbahnnetz mithilfe digitaler Netzwerkanalysen optimiert werden kann. Erreicht werden soll dies durch das Erheben der Topologie-Regeln und der Entwicklung eines Algorithmus, welcher die Segmentierung automatisch ausführt. Der Fokus liegt dabei nicht auf dem Finden von kürzesten Wegen, sondern stattdessen auf einer sinnvollen Segmentierung, welche im Kontext der Infrastrukturdiagnose verwendet werden kann. Die Arbeit soll somit aufzeigen, wie eine solche Segmentierung anhand von Algorithmen, welche auf einen Graphen angewendet werden, effizient berechnet werden können. Dabei wird speziell auf das nötige Preprocessing des Netzwerkes eingegangen, welches die Laufzeit massgeblich verbessern kann. Das Resultat sollte in einem identischen Datenformat wie die Ausgangsdaten vorliegen, damit mit den segmentierten Gleissträngen gearbeitet werden kann.

2 Theoretische Grundlagen

2.1 Graphentheorie

Ein Graph ist eine Menge von Kanten und Knoten $G = (V, E)$. Jede Kante $e = (u, v) \in E$ verbindet dabei zwei Knoten $u, v \in V$. Während ein normaler Graph auch Knoten ohne inzidente Kanten haben kann, tritt diese Situation im Zusammenhang von Schienennetzwerken nicht auf. Auch wird im Folgenden nicht weiter auf Graphen mit mehreren Zusammenhangskomponenten eingegangen, da auch diese Situationen im Allgemeinen nicht für das Eisenbahnnetzwerk der SBB relevant sind. Somit kann im folgenden davon auszugehen werden, dass jede Kante zwei Knoten verbindet und jeder Knoten mindestens über eine Kante mit einem anderen Knoten verbunden ist. Unterschieden werden dabei vier Gruppen (Herrmann, 2020):

- Ein einfacher *Graph* enthält n Knoten und m Kanten. Dabei gibt es zwischen zwei Knoten keine oder exakt eine Kante.
- Der *Multigraph* hingegen erlaubt es auch, dass zwischen zwei Knoten mehrere Kanten existieren.
- Beim *gerichteten Graphen* besitzt jede Kante eine Richtung. Dies wird auch als gerichteter Graph bezeichnet.
- Abschliessend gibt es den *Pseudographen*. Diese zeichnen sich dadurch aus, dass eine Kante am gleichen Knoten starten und enden kann. Diese Kanten werden als Masche bezeichnet (Figure 2.1.3) (Lukácová, 2017).

Eisenbahnnetzwerken liegt dabei immer ein einfacher Graph zugrunde.

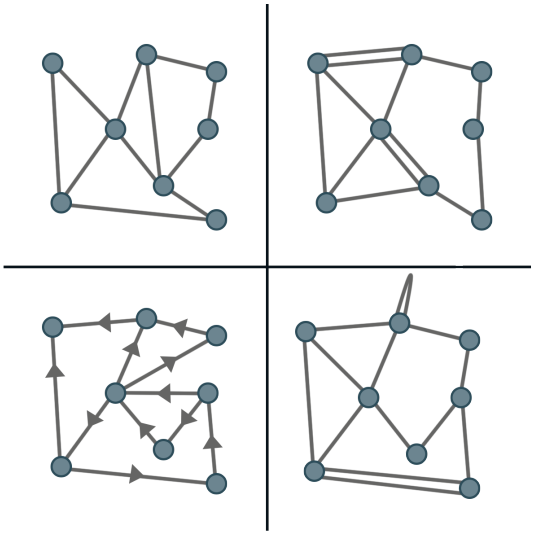


Figure 2.1.3: Typen von Graphen - 1. einfacher Graph, 2. Multigraph, 3. gerichteter Graph, 4, Pseudograph

2.1.1 Geschichte der Graphentheorie

In der Mathematik ist die Graphentheorie seit 1736 ein Begriff (Brandes, 2010; Lukácová, 2017). Der Mathematiker Leonhard Euler formulierte in diesem Jahr das bekannte *Königsberger Brückenproblem*. Durch die Stadt Königsberg (heute Kalinigrad) fliesst der Fluss Pregel, in welchem die zwei Inseln *Kneiphof* und *Wiesen* liegen. Die Inseln waren zu Eulers Zeiten durch insgesamt sieben Brücken miteinander und mit den beiden Uferseiten verbunden. Euler wollte nun wissen, ob es einen Rundweg gibt, bei welchem jede Brücke nur einmal überschritten werden muss. Dazu entwickelte er ein schematisches Modell, welches die Insel und die beiden Uferseiten als Knoten und die Brücken als Kanten darstellte. Die Frage musste Euler negativ beantworten. Er stellte fest, dass für den Fall eines existierenden Rundwegs, für jeden Knoten eine gerade Anzahl Kanten vorhanden sein müsste. Da der Graph aber über vier Knoten (Inseln) und sieben Kanten (Brücken) verfügte, ist ein Rundweg, bei welchem jede Kante nur einmal begangen wird, nicht möglich (Herrmann, 2020).

2.2 Grundbegriffe

Ein Graph $G = (V, E)$ besteht aus einer endlichen Menge Knoten V (Vertices) und einer endlichen Menge Kanten E (Edges). Jede Kante $e = (u, v) \in E$ ist Teilmenge von zwei Knoten $u, v \in V$ (Herrmann, 2020). Ein Graph, welcher ein Viereck darstellt, könnte wie folgt aussehen:

$$V = 1, 2, 3, 4$$

$$E = [1, 2], [2, 3], [3, 4], [4, 1]$$

Im folgenden wird folgende Nomenklatur verwendet:

- Elemente der Knotenmenge werden mit den Kleinbuchstaben u,v,w bezeichnet.
- Elemente der Kantenmenge werden mit den Kleinbuchstaben e oder als Tupel (u, v) bezeichnet.
- Benachbarte Knoten werden *adjazent* genannt.
- Die beiden benachbarten Knoten und die verbindende Kante werden *inzident* genannt.
- Die Anzahl Nachbarsknoten eines jeden Knotens werden als *Grad* bezeichnet. Besitzt jeder Knoten in einem Graphen denselben *Grad* k, wird der Graph als *k-regulär* bezeichnet.

2.2.1 Pfade und Rundwege

Ein zentrales Element jedes Graphen sind Pfade, auch Wege genannt und Rundwege, auch Kreise genannt. Pfade und Rundwege unterscheiden sich in der Eigenschaft, dass sich der Start- und der Endknoten beim Pfad unterscheiden und bei einem Rundweg identisch sind. Pfade, als auch Rundwege, stellen jeweils einen eigenen Graphen dar und sind somit Teilgraph des Ausgangsgraphen (Figure 2.2.4) (Täubig, 2011).

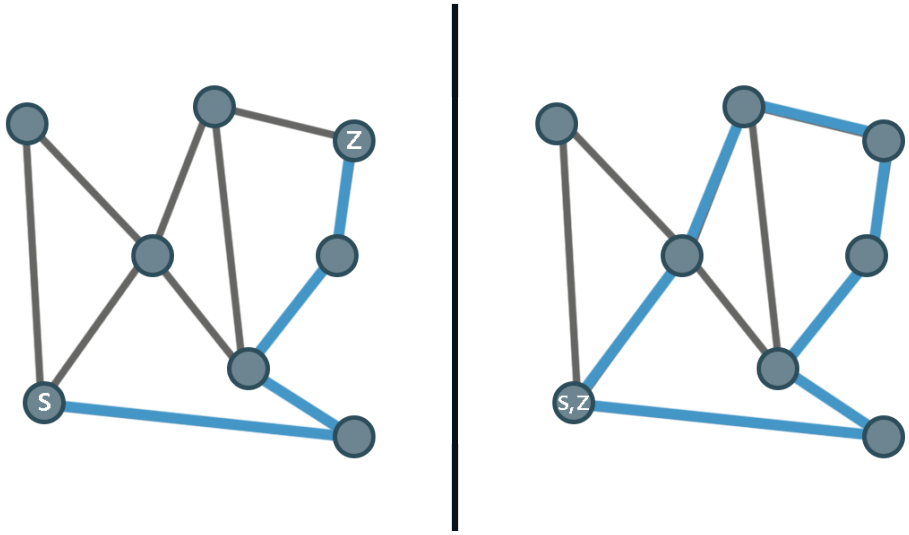


Figure 2.2.4: links: Pfad durch ein Netzwerk, rechts: Rundweg durch ein Netzwerk

2.2.2 Bäume

Ist in einem Graphen kein Rundweg möglich, wird er als kreisfreier Graph oder als Baum bezeichnet. Jeder Knoten mit dem Grad 1, also nur einer Kante, wird als Blatt bezeichnet (Figure 2.2.5). Jedes Blatt stellt ein Teilende des Graphen dar. In vielen Fällen ist in einem Graphen ein Rundweg möglich. Ein Pfad innerhalb dieses Graphen stellt dann jedoch nur einen Baum dar (Herrmann, 2020). In dieser Arbeit kommt dieser Eigenschaft eine wichtige Rolle zu, da sich Gleisstränge identisch verhalten.

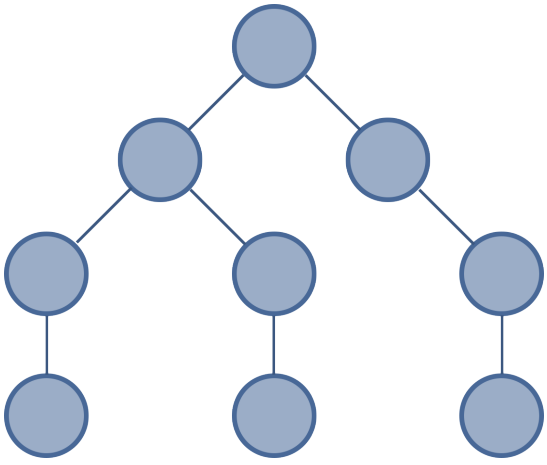


Figure 2.2.5: Schematischer Netzwerkbaum

2.2.3 Gewichteter vs. ungewichteter Graph

Kanten können nebst einer Richtung auch eine Gewichtung besitzen. In diesem Fall wird von einem *gewichteten Graph* gesprochen. Dabei wird jeder Kante ein Kostenfaktor oder eben ein Gewicht zugeteilt (Herrmann, 2020). Das ist dann relevant, wenn der kürzeste Weg gefunden werden soll. Somit ist der Pfad mit der kleinsten Anzahl Kanten nicht zwingend der günstigste Pfad zwischen zwei Knoten im Netzwerk, da unter Umständen ein Knoten mit hohem Gewicht auf diesem Pfad liegt (Figure 2.2.6). Für diese Arbeit werden die Kosten der Kanten nicht berücksichtigt. Die Länge der einzelnen Gleisstränge und weitere Kostenfaktoren werden für die Berechnung ignoriert. Würde im Anschluss an die Segmentierung eine Netzwerkanalyse mit dem Ziel, kürzeste Pfade zu finden, durchgeführt, müssten Kostenfaktoren berücksichtigt werden. Der Algorithmus wird daher für einen ungewichteten Graphen entwickelt.

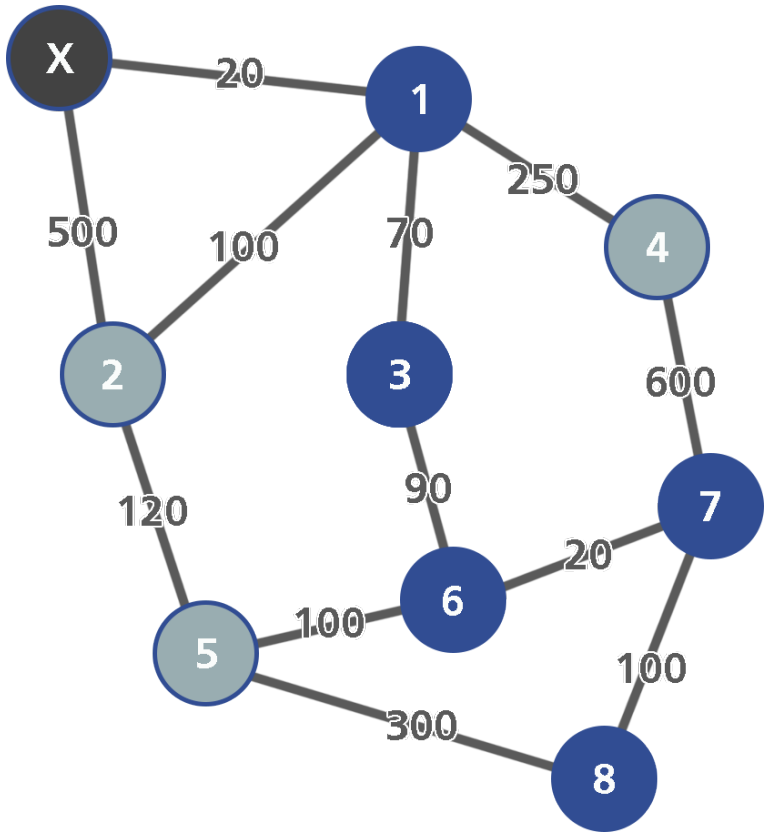


Figure 2.2.6: Gewichteter Graph: Jede Kante verfügt über ein Kostenattribut

2.3 Suchalgorithmen in Netzwerken (Graphtraversierung)

In vielen Fällen besteht Interesse, einen Graphen systematisch zu durchlaufen, um einen Pfad zu extrahieren oder bestimmte Knoten und Kanten zu zählen. Grundlage für die Klärung dieser Fragen stellen die Suchalgorithmen *Depth First Search (DFS)* und *Breadth First Search (BFS)* dar, auch Tiefen- und Breitensuche genannt (Tarjan, 1972; Volbert, 2011). Beide Algorithmen bilden vom Startknoten aus einen Suchbaum, wobei der Startknoten dabei jeweils die Wurzel des Baumes darstellt.

2.3.1 Depth First Search (DFS)

Bei der Tiefensuche (Figure 2.3.7b) beginnt der Algorithmus jeweils bei einem definierten Startknoten und durchläuft den Graphen so weit wie möglich entlang jedes Pfades. Dabei werden zu Beginn alle Knoten als *nicht besucht* markiert. Vom Startknoten wird nun über eine Kante ein beliebiger, benachbarter Knoten ausgewählt. Dort angekommen wird der Knoten als *besucht* markiert bezeichnet und ein nächster, benachbarter Knoten wird besucht. Dieser Vorgang wiederholt sich bis das Ende eines Pfades erreicht ist. Das kann der Fall sein, wenn von einem Knoten kein unmarkierter Knoten mehr erreicht werden kann oder wenn der Knoten ein definiertes Stop-Attribut besitzt. Der Pseudocode einer Tiefensuche könnte wie folgt aussehen (Listing 2.3.2):

2.3.2 Breadth First Search (BFS)

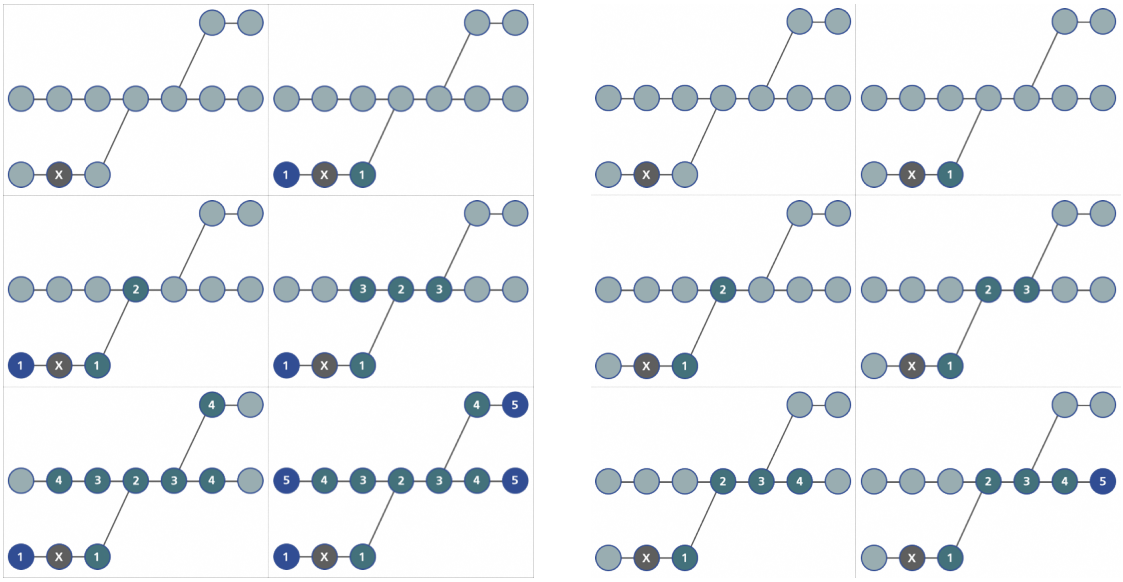
Im Gegensatz zur Tiefensuche wird bei der Breitensuche (Figure 2.3.7a) der Suchbaum jeweils gleichmässig vergrössert, der Suchbaum wächst also Stufe um Stufe (Listing 2.3.2). Die Breitensuche ist besonders geeignet für Situationen, in denen der Abstand in Kanten vom Startknoten aus berechnet werden muss. Auch bei der Breitensuche ist die Wurzel des resultierenden Suchbaumes der Startknoten.

Listing 1: Pseudocode BFS

```
BFS(startnode, goal) {
    push(queue, startnode)
    mark goal
    while (queue is not empty) {
        v := pop(queue)
        if (v == goal)
          return node
        mark v
        for (n in neighbors(v)){
            if (n not marked){
                push(queue, n)
            }
        }
    }
}
```

Listing 2: Pseudocode DFS

```
DFS(node, goal) {
    mark node
    if (node == goal)
      return node
    else {
        for (v in neighbors(node)){
            if (v not marked){
                DFS(v, node)
            }
        }
    }
}
```



(a) Ablauf einer Breitensuche

(b) Ablauf einer Tiefensuche

Figure 2.3.7: Suchalgorithmen in Netzwerken

2.4 Topologie

Die Lagebeziehung zwischen verschiedenen Objekten im Raum wird als *Topologie* bezeichnet. Im Gegensatz zur Topographie, welche sich mit der Darstellung und der Beschreibung der Geodaten befasst, steht die Topologie für die strukturelle Beziehung zwischen den Objekten (Rösch, 1998). Die Eigenschaften einer Topologie können sich derweil in zwei oder in drei Dimensionen manifestieren. Ein Koordinaten-Tupel *(Point-Feature)* kann zwei Koordinaten besitzen oder noch eine dritte Höhenangabe beinhalten. Dreidimensionale Topologien sind sehr viel komplexer als zweidimensionale. Um die Realität möglichst exakt abzubilden, reichen zweidimensionale Topologien jedoch meistens nicht aus. Als Beispiel kann eine Autobahnkreuzung dienen. Die Aus- und Einfahrten führen zum Teil über und unter anderen Strassenabschnitten durch. Auch bei der Eisenbahn ist dieser Faktor zu berücksichtigen. Sogenannte *Unter- und Überwerfungen*, häufig in Bahnhofsnähe, sollen den Bahnverkehr entflechten und ein schnelleres Fahren ermöglichen (Rösch, 1998). Topographische Modelle müssen hingegen der Herausforderung gerecht werden, dass dreidimensionale Daten auf eine zweidimensionale Oberfläche projiizert werden müssen. Eine Verzerrung der Fläche oder der Winkel ist dabei unumgänglich. Die Abbildung der Erdkugel auf einer Karte zählt sicherlich zu den bekanntesten Beispielen dieser Problematik (Snyder, 1997). In der Topologie hingegen, ist dieses Problem nicht bekannt, da das Netzwerk schematisch beschrieben werden kann. Ein Vorteil bei der Modellierung eines Bahnnetzwerkes ist die Tatsache, dass es sich ausschliesslich um Linien-Objekte handelt. Wird beispielsweise ein Platz, welcher Strassen verbindet, in einer Netzwerktopologie dargestellt, birgt das einige Schwierigkeiten. Es muss geklärt werden, ob in der Mitte des Platzes ein Knoten erstellt wird, bei dem alle Kanten zusammenlaufen, oder ob bei den Strassenanfängen jeweils Knoten platziert und neue Kanten zwischen diese Knoten gesetzt werden. Bei einem Schienennetz gibt es diese Probleme nicht, da Weichen die einzige Möglichkeit darstellen, dass zwei Kanten zusammenlaufen oder sich eine Kante aufteilt.

2.4.1 Bahntopologie

Eine Datenbank, welche eine Bahntopologie enthält, muss neben der Position des Objekts weitere Parameter abspeichern können. Dazu gehören zum Beispiel der Typ einer Weiche, ein Wartungsintervall oder die möglichen *Fahrbegriffe*, welche ein Signal anzeigen kann. Ausserdem verändert sich die Datengrundlage täglich, da bei Baustellen die Bahninfrastruktur verändert wird oder Objekte als Ganzes ausgetauscht, entfernt oder hinzugefügt werden (Schweizerische Bundesbahnen, 2021b). Diese Arbeit konzentriert sich indes nur auf die Gleisinfrastruktur, welche die Gleisstränge, inklusive Weichen und Signale enthält.

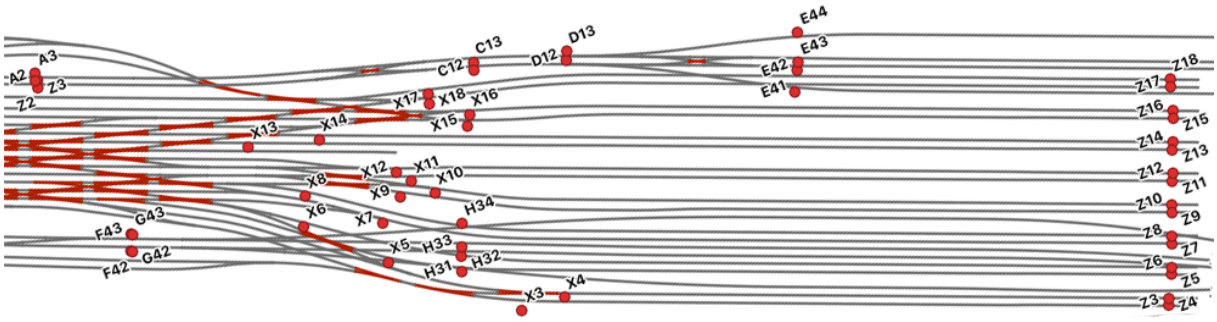


Figure 2.4.8: Auschnitt DfA: Zürich Hauptbahnhof

2.4.2 Gleisstränge

Eine Schiene setzt sich aus mehreren Komponenten zusammen. Zwei Schienenstränge werden mit regelmässig angeordneten Schwellen, welche aus Beton, Holz oder Stahl bestehen, parallel und auf gleicher Höhe angeordnet (Schweizerische Bundesbahnen, 2015). Um den Zentrifugalkräften entgegen zu wirken, welche ein Zug in einer Kurve erfährt, wird einer der Schienenstränge etwas höher angebracht. Fährt ein Zug durch eine Kurve legt er sich dadurch leicht zur Seite, was für mehr Fahrkomfort sorgt und zu einer tieferen Materialbelastung führt. Diese spezielle Befestigung einer Schiene wird *Überhöhung* genannt. Da es im Schienennetz keine Gleisbezeichnungen, analog zu Strassennamen gibt, müssen die einzelnen Teilabschnitte anders unterschieden und bezeichnet werden. Zwei Schienenstränge und die Schwellen bilden gemeinsam einen *Gleisstrang*. Dieser führt jeweils von einer Weiche zur Nächsten oder ist Teil einer Weiche. Die Topologiedatenbank *Datenbank feste Anlagen (DfA)* der SBB enthält diese Gleisstrangdaten. Da die Gleisstränge nicht nur das schematische Bahnnetz abbilden müssen, sondern auch ein topographisches Abbild des Schienennetzes sind, werden die Linien durch Punkte, welche in 10 m Abständen angebracht sind, dargestellt. Dadurch lässt sich jeder Gleisstrang auf einer Karte visualisieren (Figure 2.4.8). Die Gleisstränge entsprechen im Netzwerk immer Kanten. Die Bezeichnung eines Gleisstranges setzt sich immer aus derjenigen der Weichen zusammen, die er verbindet (Schweizerische Bundesbahnen, 2021b).

2.4.3 Weichen

Weichen sind im Schienennetz das, was Verzweigungen und Kreuzungen im Strassennetz sind. Ein Gleisstrang wird bei einer Weiche in zwei Gleisstränge aufgeteilt. Durch eine bewegliche *Weichenzunge* kann ein Fahrweg über einen der beiden Stränge eingestellt werden. Eine Weiche besteht somit im eigentlichen Sinne aus zwei einzelnen Gleissträngen, welche einen gemeinsamen End- respektive Startpunkt haben. Das gemeinsame Ende der Gleisstränge wird *Weichenspitze* genannt. Die sich unterscheidenden Enden werden *Weichenwurzeln* genannt. Fahrten über eine Weiche sind nur von einer Wurzel in Richtung der Spitze oder von der Spitze in Richtung einer der Wurzel möglich. Eine Fahrt von einer Wurzel zur anderen würde einen Fahrtrichtungswechsel bedingen und ist daher zu vermeiden. In der Topologie der SBB entsprechen die Weichen den Knoten des Netzwerks. Jeder Endpunkt ist ein Knoten. Die Weichenstränge entsprechen ebenfalls den Kanten. Um jede Weiche identifizieren zu können, werden eindeutige Namen

zugeordnet. Da Weichen entweder innerhalb eines Bahnhofes liegen oder auf der Strecke innerhalb eines benannten Spurwechsels, kann jeder Weiche ein spezifischer Name zugeteilt werden. Zusätzlich wird der Bezeichnung eine, pro Bahnhof einmalige Nummer hinzugefügt. So wird zum Beispiel die Weiche Nummer 2 im Bahnhof Bern als $BN2$ bezeichnet. BN ist der betriebsinterne *Zweizeichencode* für den Bahnhof Bern. Jeder Bahnhof in der Schweiz besitzt einen einmaligen *Zweizeichencode*. Die Weiche $BN2$ enthält alle drei Knoten und die Kanten zwischen der Spitze und den Wurzeln. Um Verwechslungen zu vermeiden, werden die Endpunkte der Weiche ebenfalls mit einer Nummer gekennzeichnet. Die Spitze wird immer mit der Nummer 1 gekennzeichnet. Von der Spitze gerade aus gesehen liegt die Wurzel mit der Nummer 2. Die Wurzel, welche von der Spitze aus auf dem ablenkenden Gleisstrang liegt, erhält die Nummer 3 (Figure 2.4.9). Fährt ein Zug von der Spitze über den ablenkenden Gleisstrang, befährt er den Gleisstrang $BN2.1-3$. Dadurch ist es möglich, für jede Weiche den Gleisstrang, respektive die Fahrtrichtung zu definieren.

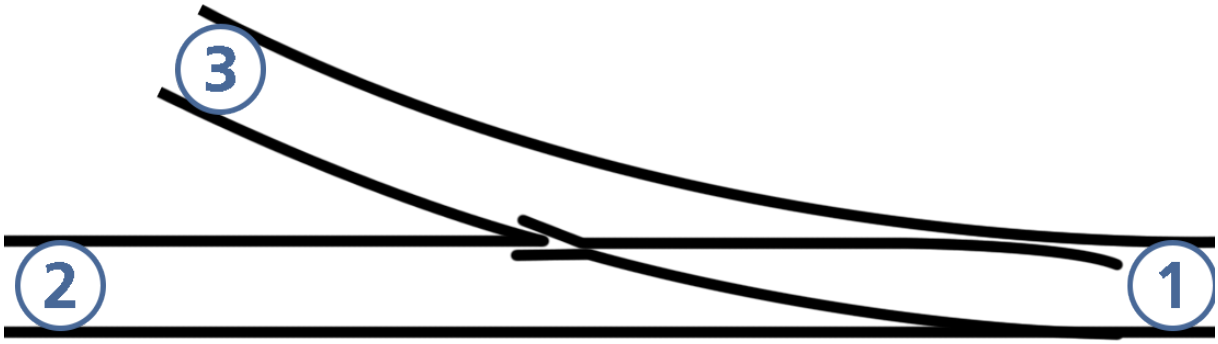


Figure 2.4.9: Schematische Darstellung einer Weiche

2.4.4 Signale

Im Schweizer Schienennetz sind verschiedene Signaltypen vorhanden. Dazu gehören nicht nur die klassischen Lichtsignale *Grün, Orange und Rot*, welche auch aus dem Strassenverkehr bekannt sind. Es gibt eine ganze Palette an Signalen, welche die erlaubte Geschwindigkeit signalisieren, auf Besonderheiten der Strecke aufmerksam machen oder auf Gefahren hinweisen. Für die Segmentierung sind jedoch nur sogenannte *Hauptsignale* relevant. Im Eisenbahnverkehr wird nach Signal und nicht nach Sicht gefahren. Wird im Strassenverkehr eine Geschwindigkeitstafel erkannt, reicht der Bremsweg in der Regel aus, um die Geschwindigkeit bis zur Tafel zu reduzieren. Da Züge viel schwerer sind und zum Teil auch schneller verkehren und somit einen längeren Bremsweg haben, reicht die Distanz vom Moment des Erkennens des Signals bis zur Vorbeifahrt nicht aus. Ausserdem dürfen die Geschwindigkeiten auch in der Nacht und bei schlechter Sicht nicht zu stark reduziert werden, damit der Fahrplan eingehalten werden kann. Aus diesem Grund wird vor jedem Hauptsignal ein Vorsignal aufgestellt. Dieses Signal zeigt dem Lokführer oder der Lokführerin, was das darauffolgende Signal signalisieren wird. Somit bleibt genügend Zeit, um die Geschwindigkeit entsprechend anzupassen. Ein Hauptsignal kann im Gegensatz zum Strassensignal auch Geschwindigkeitsänderungen, sogenannte Fahrbegriffe, anzeigen. Zeigt das Hauptsignal den Fahrbegriff Halt, ist eine Vorbeifahrt am Signal nicht

gestattet (Figure 2.4.10). Auch ist es einem Hauptsignal möglich, mittels bestimmter Kombinationen von Leuchten, eine maximale Geschwindigkeit zu signalisieren. Fährt ein Zug beispielsweise ablenkend über eine Weiche, zeigt das vorangehende Signal einen tieferen Fahrbegriff und entsprechend eine tiefere Geschwindigkeit, als wenn der Zug gerade über die Weiche fährt. Hauptsignale sind auch immer Start- und Endpunkte einer Zugfahrt, womit sie im Graphen, welcher zur Segmentierung verwendet wird, ebenfalls als Knoten betrachtet werden müssen.

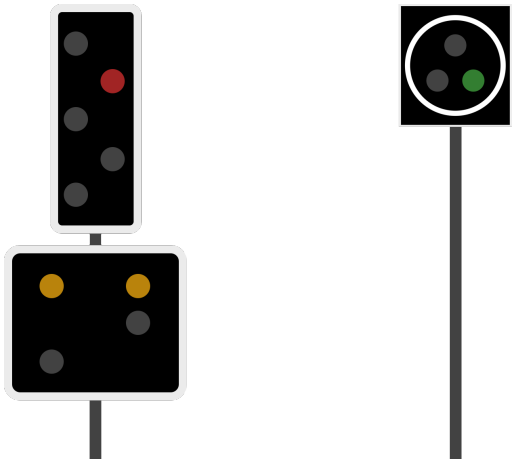


Figure 2.4.10: links: Typ L: Fahrbegriff 0 - Halt, rechts: Typ N: Fahrbegriff 1 - freie Fahrt

2.4.5 Fahrdienstvorschriften FDV

Das Regelwerk für den Eisenbahnverkehr wird in der Schweiz durch die *Fahrdienstvorschriften (FDV)* abgebildet. Dieses Dokument wird von Bundesamt für Verkehr(BAV), welches dem *Eidgenössischen Departement für Umwelt und Verkehr* (UVEK) unterstellt ist, herausgegeben. Das Reglement ist für alle Eisenbahnverkehrsunternehmen (EVU), welche die Schweizer Eisenbahninfrastruktur benützen massgebend. Darin werden sämtliche Prozesse, welche für einen sicheren Bahnverkehr eingehalten werden müssen, geregelt (Schweizerische Bundesbahnen, 2021c). Um die Segmentierung eines Eisenbahnnetzwerkes durchzuführen, müssen diese Vorschriften zwangsläufig berücksichtigt werden, da darin relevante Prozesse, welche zum Beispiel bei einem Fahrtrichtungswechsel angewendet werden müssen, definiert werden. Ausserdem werden unter anderem die Bedingungen für eine Zugfahrt festgelegt, da in der Schweiz nicht jede Zugbewegung als Zugfahrt deklariert ist. Es wird zwischen den zwei Hauptverkehrsarten *Rangierbewegung* und *Zugfahrt* unterschieden.

2.4.6 Rangierbewegung

Bewegungen innerhalb eines Bahnhofes, zum Beispiel von einem Abstellgleis ans Perron, werden als Rangierbewegung klassifiziert. Bei dieser Verkehrsart fährt der Lokführer oder die Lokführerin nicht nach den Hauptsignalen, sondern nach sogenannten *Zwergsignalen* (Figure 2.4.11). Dabei handelt es sich um kleine Lichtsignale mit drei weissen Lampen in Bodennähe, welche drei verschiedene Fahrbegriffe signalisieren können. Da auch bei der Rangierfahrt Bremswege länger sind, kann ein Zwergsignal als Vorsignal zu einem nächsten Zwergsignal dienen. Dazu wird der Fahrbegriff *Fahrt mit Vorsicht* angezeigt. Die Fahrstrasse kann folgend auf ein

solches Signal unerwartet enden. Die maximale Verkehrsgeschwindigkeit bei einer Rangierfahrt beträgt 30 km/h. Dies wird einerseits dadurch begründet, dass ein Zug auf Rangierfahrt nicht von einem System überwacht wird. Fährt ein Zug also über ein geschlossenes Signal, wird er durch kein System gestoppt. Andererseits werden bei Rangierfahrten vermehrt Weichen ablenkend befahren. Durch die tiefere Geschwindigkeit wird das Material sowohl am Zug, als auch an der Weiche geschont. Ein Spezialfall stellt die *Rangierbewegung auf die Strecke* dar. Diese wird dann angeordnet, wenn eine reguläre Zugfahrt nicht möglich ist. Ein Anwendungsfall ist zum Beispiel das Abholen eines steckengebliebenen Zuges mit einer Hilfslok. Dabei verlässt die Lok als Rangierbewegung den Bahnhof. Der Lokführer oder die Loführerin muss die Hauptsignale nicht berücksichtigen.

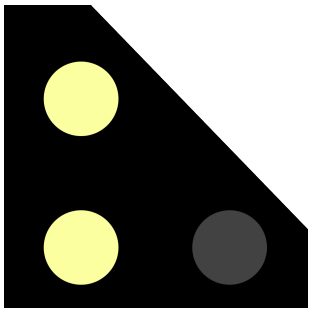


Figure 2.4.11: Zwergsignal - Fahrbegriff: Fahrt

2.4.7 Zugfahrt

Verlässt ein Zug einen Bahnhof, ist er in der Regel gemäss Verkehrsart *Zugfahrt* unterwegs. Um als Zug zwischen zwei Bahnhöfen zu verkehren, müssen verschiedene Bedingungen erfüllt sein. Eine Zugfahrt muss zwingend angeordnet werden. Somit ist eine spontane Fahrt von *Ort A* nach *Ort B* nicht möglich. Personenzüge fahren in der Regel nach Jahresfahrplan. Das heisst, dass sich die Fahrten in bestimmten Intervallen wiederholen. Meist ist dies täglich der Fall. Wird jedoch eine Messfahrt geplant, welche einmalig ist, muss diese separat bestellt werden. Die zuständige Stelle teilt dann dem Zug, gemäss der eingegangenen Bestellung, einen Ausgangspunkt, einen Zielpunkt, eine Verkehrszeit und eine Zugnummer, welche an diesem Tag nur einmal vorkommt, zu. Ausgangspunkt und Zielpunkt müssen jeweils innerhalb eines Bahnhofs liegen. Eine Zugfahrt mit Startpunkt oder Endpunkt ausserhalb eines Bahnhofs ist somit nicht möglich. Neben der Anordnung wird der Zug auch trassiert. Das bedeutet, dass der Fahrweg provisorisch festgelegt wird. Bei einem *Trasse* handelt es sich, anlog zur Aviatik, um ein Zeitfenster, in welchem eine definierte Strecke befahren werden kann. Die Eigenschaften des Zuges, wie Geschwindigkeit, Antriebsart, Länge oder Gewicht müssen deklariert werden. Jede Strecke verfügt über eine bestimmte Anzahl Trassen pro Tag. Für bestimmte Trassen werden Minimalgeschwindigkeiten oder minimale Bremsleistungen vorgeschrieben. Die Trassen werden in der Schweiz durch die nationale Trassenvergabestelle *Trasse Schweiz* vergeben. Trasse Schweiz ist eine öffentlich-rechtliche Anstalt des Bundes und für die diskriminierungsfreie Vergabe und Planung der Trassen in der Schweiz zuständig (Trasse Schweiz, 2021). Wird für einen Zug ein freies Trasse gefunden, wird der Fahrweg ins Zugleitsystem *Rail Control System* (RCS) eingepflegt. Ein Trasse startet jeweils vor einem Hauptsignal in einem Bahnhof und endet in

einem anderen Bahnhof, ebenfalls vor einem Hauptsignal. Dieser Umstand muss bei der Segmentierung beachtet werden. Obwohl eine Zugfahrt nur von Bahnhof zu Bahnhof möglich ist, kann die Fahrt auf der Strecke unterbrochen werden. Dies kann zum Beispiel bei einem Einspurabschnitt vorkommen, um einen entgegenkommenden Zug abzuwarten.

2.5 Segmentierung

Für die Segmentierung sind zusammenfassend folgende Elemente relevant:

- **Gleisstränge:** Kanten
- **Weichen:** Knoten und Kanten
- **Signale:** Knoten

Da ein Knoten nebst dem Namen und der Position noch weitere Attribute besitzen kann, werden diese bei der Erstellung des Netzwerkes mit einbezogen. Dazu gehört insbesondere das Attribut *Typ*. Während alle Knoten, welche die Endposition einer Weiche darstellen, dem Typ *Weiche* entsprechen, gibt es bei den Signalen drei verschiedene Möglichkeiten. Ein Signal kann beispielsweise beim Prellbock eines Kopfbahnhofes stehen. Dieses Signal zeigt immer den Begriff *Halt*. Es dient somit in jeder Situation als *Ankunftssignal*. Auf der gegenüberliegenden Seite des Bahnhofes liegt das zugehörige *Abfahrtssignal*. Da ein Zug niemals von hinten an dieses Signal ranfährt, respektive niemals davor anhalten muss, dient es ausschliesslich als *Abfahrtssignal*. Befindet sich der Zug jedoch in einem Durchfahrtsbahnhof, kann ein Signal sowohl als *Ankunftssignal*, als auch als *Abfahrtssignal* dienen. Alle anderen Signale, inklusive den Hauptsignalen auf der Strecke, sind für die Segmentierung nicht relevant.

2.6 Einfluss der Fahrdienstvorschriften auf die Segmentierung

Zusammenfassend lassen sich folgende Regeln ableiten:
Ein für die Segmentierung kritisches Signal kann entsprechend folgende Eigenschaften besitzen:

- **Nur Abfahrt** - *Bsp: Abfahrtssignal in einem Kopfbahnhof.*
- **Nur Ankunft** - *Bsp: Ankunftssignal in einem Kopfbahnhof.*
- **Abfahrt / Ankunft** - *Bsp: Hauptsignal in einem Durchfahrtsbahnhof.*

Ein fahrbares Segment muss immer folgende Bedingungen erfüllen:

- Ein Segment startet immer an einem Hauptsignal und endet immer an einem Hauptsignal. Diese Signale müssen innerhalb eines Bahnhofes liegen.
- Eine Zugfahrt ausschliesslich innerhalb eines Bahnhofes ist nicht möglich. Dabei würde es sich um eine Rangierfahrt handeln.
- Der Bahnhof des Startpunktes muss sich vom Bahnhof des Endpunktes unterscheiden.
- Eine Weiche kann nur von einer der beiden Spitzen in Richtung des Herzens oder umgekehrt befahren werden. Die Fahrt von einer Spitze zur anderen ist nicht möglich.

3 Methodik

Das Vorgehen wird in eine konzeptionelle Phase und in eine Case Study unterteilt. Ziel ist es, den Algorithmus in einem vereinfachten und überschaubaren Netzwerk zu entwickeln, zu testen und die resultierenden Segmente manuell zu prüfen. So sollen die Grenzen und Limitationen des gewählten Ansatzes bereits frühzeitig erkannt werden und wenn möglich sollen bereits Anpassungen am Algorithmus gemacht werden. Anschliessend wird der Algorithmus an realen Daten in einer Case Study getestet und fertig entwickelt.

3.1 Erstellung eines einfachen Netzwerks

Um das Problem zu veranschaulichen, wurde als erster Schritt ein einfaches Netzwerk erstellt, welches die wichtigsten Spezialfälle beinhaltet (Figure 3.1.12). Einerseits sollten damit die Rechenzeiten möglichst kurz gehalten werden, anderseits sollte eine manuelle Validierung der Resultate dadurch in kurzer Zeit möglich sein. Das Test-Netzwerk enthielt zwei Kopfbahnhöfe sowie einen Durchfahrtsbahnhof mit zwei Gleisen. Jedem Knoten wurde ein Attribut *Typ* zugewiesen. Hier wurden Weichen als *Weichen* erfasst, bei den Signalen wurden die spezifischen Funktionalitäten, also *Abfahrts-, Ankunfts- oder Abfahrt/Ankunftssignal*, als Typ definiert. Um das Netzwerk zu erstellen wurde die Python-Bibliothek NetworkX verwendet (Hagberg et al., 2008).

Als nächster Schritt musste ein Netzwerk initialisiert werden. Dazu wurde die Funktion *nx.Graph()* verwendet, womit ein ungerichteter Graph erstellt wurde. Sobald der Graph *G* vorhanden war, wurden Knoten und Kanten hinzugefügt. Dazu wurden die *nx*-Funktionen *addnodes()* und *addedges()* verwendet. Einem Knoten wurde jeweils sein Name, seine Position und sein Typ zugeordnet. Die Namen von Kanten ergaben sich durch die beiden Knoten, welche sie verbinden. Um die Dateneingabe zu überprüfen, wurde das Netzwerk regelmässig dargestellt. Dazu wurde die Python-Bibliothek *matplotlib* verwendet.

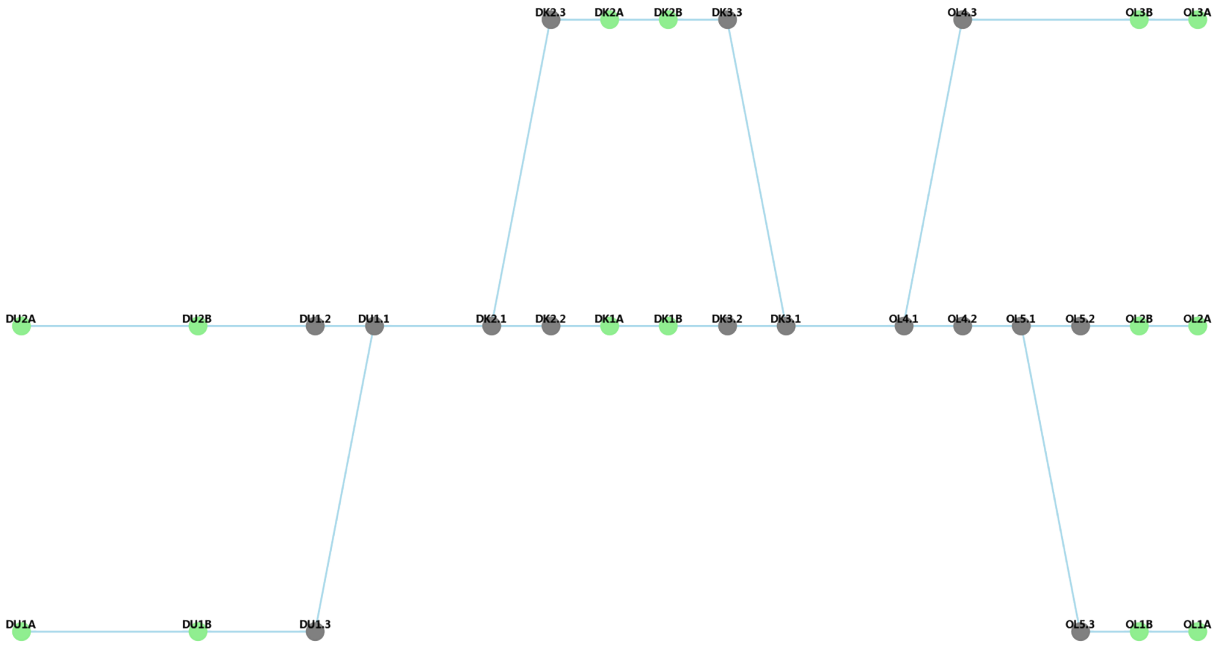


Figure 3.1.12: Plot des vereinfachten Netzwerks

3.1.1 Manuelle Segmentierung

Die Abbildung 3.1.12 zeigt alle Knoten, welche individuell mit ihrem Namen beschriftet sind. Die grünen Knoten entsprechen den Weichen und die braunen Knoten den Signalen. Eine manuelle Segmentierung war dadurch ohne grossen Aufwand möglich. Um diesen Prozess besser zu veranschaulichen, wird im Folgenden das Beispiel des Abfahrtsknoten *DU1B* verwendet:

Von diesem Knoten sind Pfade nach *DU1A, DU2A, DK1B und DK2B* möglich (Figure 3.1.13). Ziel war es nun, allgemein gültige Regeln zu formulieren, welche der Algorithmus befolgen muss, wenn er den Weg eines Zuges beschreibt, welcher verschiedene Kanten und Knoten befährt. Als Resultat wurden alle Pfade von einem spezifischen Startknoten gesucht, welche beim einem Ankunftssignal enden.

- **Pfad 1: DU1B - DU1A:**
  Dieser Pfad ist mit Abstand der Kürzeste. Der Ankunftsknoten liegt direkt neben dem Abfahrtsknoten. Wird diese Situation in die Realität übertragen, so würde dieser Pfad eine Fahrt vom einen Ende das Bahnhofs zum anderen Ende des Bahnhofs bedeuten. Dieser Pfad ist somit für das Schlussresultat nicht relevant und wird verworfen.
  Schlussfolgerung: Pfade, bei welchen der Endknoten die gleiche Bahnhofsbezeichnung besitzt wie der Startknoten, werden verworfen.

- **Pfad 2: DU1B - DU2A:**
  Dieser Pfad ist bereits etwas länger und umfasst mehrere Knoten und Kanten. Auffällig ist der spitze Winkel, welcher der Zug bei der Weiche *DU1* zurücklegen müsste. Werden die Endungen der Bezeichnung betrachtet, fällt auf, dass es sich um die Reihenfolge *DU1.3 - DU1.1 - DU1.2* handelt. Das Befahren einer Weiche in dieser Reihenfolge ist aufgrund des spitzen Winkels nicht möglich. Der Pfad wird daher verworfen. Selbst wenn der Pfad nicht über diese Weiche führen würde, wäre er verworfen worden, da der Endknoten wiederum im gleichen Bahnhof liegt, wie der Startknoten und somit unter die oben beschriebene Regel fällt.
  Schlussfolgerung: Pfade, welche eine Weiche in der Reihenfolge *.3 - .1 - .2 oder .2 - .1 - .3* befahren, werden verworfen.

- **Pfad 3: DU1B - DK1B:**
  Der dritte Pfad führt ebenfalls über die Weiche DU1. Jedoch werden nur die Knoten DU1.3 und DU1.1 befahren. Damit befährt der Zug den ablenkenden Gleisstrang. Anschliessend wird der Bahnhof DU verlassen und über das Streckengleis wird der Bahnhof *DK* erreicht. Die Weiche *DK2* wird von der Spitze gerade befahren, worauf der Pfad auf den Abfahrts-/Endknoten *DK1A* trifft. Wird diese Situation wieder in die Realität projiziert, würde die Spitze des Zuges noch vor dem Perron zu stehen kommen, da sie vor dem Signal steht, welches eigentlich für die Abfahrt in die entgegengesetzte Seite errichtet wurde. Der Zug muss dieses Signal also passieren und bis ans Ende des Perrons fahren, damit der ganze Zug im Bahnhof steht und im Anwendungsfall, die Reisenden auch aus- und einsteigen könnten.
  Schlussfolgerung: Wird bei einem Pfad ein Abfahrts-/Ankunftsknoten erreicht, handelt

es sich noch nicht um das Ende des Pfades. Der Algorithmus muss sich jedoch merken, dass bereits der erste Abfahrts-/Ankunftsknoten erreicht wurde. Sobald der zweite Abfahrts-/Endknoten erreicht wird, ist das Ende des Pfades erreicht.

- **Pfad 4: DU1B - DK2B:**
  Bei diesem Pfad handelt es sich um die identische Situation wie bei Pfad 3, mit dem Unterschied, dass die Weiche *DK2* über den ablenkenden Strang befahren wird.
  Schlussfolgerung: Auch hier muss der Algorithmus das erste Signal vom Typ Abfahrt/Ankunft erkennen und einen Knoten weiterfahren.

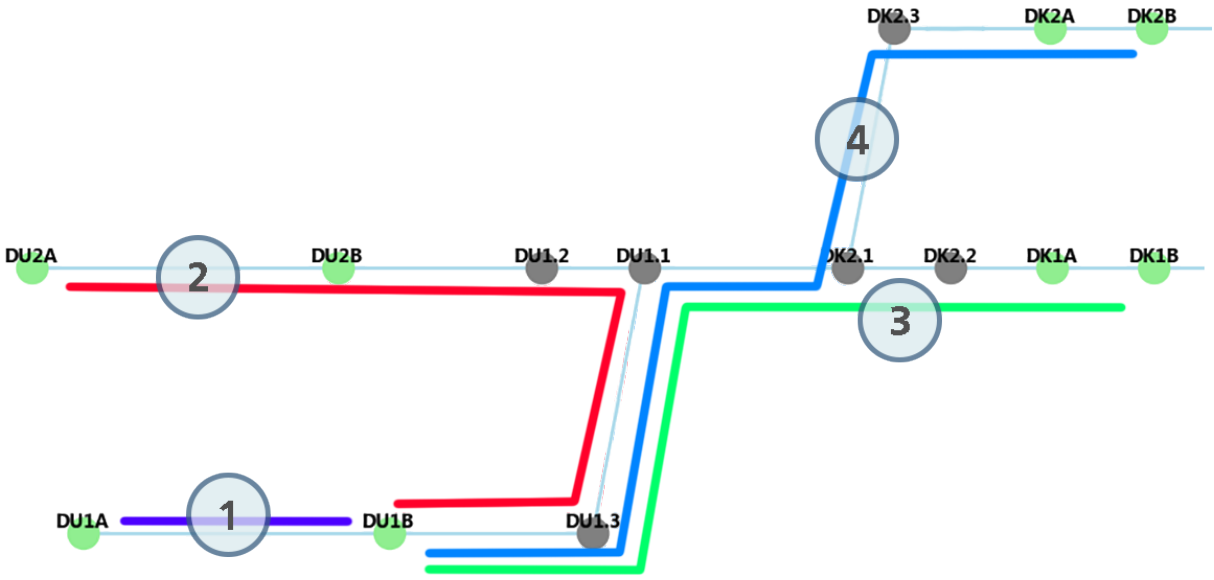


Figure 3.1.13: Mögliche Pfade im vereinfachten Netzwerk.

Von den vier möglichen Pfaden konnten im erstellten vereinfachten Netzwerk also zwei verworfen und zwei behalten werden. Dieser Vorgang wurde nun für alle *Abfahrts-* und *Abfahrts-/Ankunftsknoten* im Netzwerk wiederholt, um sämtliche Pfade, respektive Segmente zu prüfen. Als Schlussfolgerung muss der Algorithmus folgende Regeln umsetzen können:

- **Regel Nº 1:** Alle Pfade von jedem Startknoten zu den benachbarten Endknoten müssen gesammelt werden.
- **Regel Nº 2:** Alle Pfade, bei denen Weichen in der Reihenfolge *3 - 1 - 2* oder *2 - 1 - 3* werden, müssen verworfen werden.
- **Regel Nº 3:** Ankunftsknoten dürfen nicht im gleichen Bahnhof liegen wie der Startknoten.
- **Regel Nº 4:** Der zweite Knoten im Pfad darf kein Endknoten sein.
- **Regel Nº 5:** Wenn ein erster Abfahrts-/Ankunftsknoten erreicht wird, muss der Pfad bis zum zweiten Knoten dieser Art weitergeführt werden.

3.2 Modifizierte Tiefensuche

Für das Sammeln aller Pfade von jedem Startknoten wurde ein Algorithmus verwendet, der auf dem in den Grundlagen beschriebene *Depth First Search Algorithmus* beruht (Listing 3). Der Algorithmus durchläuft den Graphen bei der *Depth First Search* vom Startknoten her, bis er einen potentiellen Endknoten erreicht und markiert alle durchlaufenen Punkte. Sobald er einen bereits markierten Pfad aufnehmen möchte, geht er zurück und prüft andere Optionen, damit ein noch unmarkierter Pfad abgelaufen werden kann. Die Standard-Tiefensuche musste für diese Aufgabe modifiziert werden, da beim Erreichen eines ersten Abfahrts-/Ankunftsknoten ansonsten der Pfad als beendet betrachtet werden müsste, dies darf gemäss der oben beschriebenen Regel NR.5 nicht geschehen, da sonst ein Zug noch vor dem Perron zu stehen kommen würde. Bei der Modifizierung wurde für jeden Pfad die Variable *canSkipStartStop* mit *True* initialisiert. Sobald ein Knoten vom Typ Abfahrt-/Ankunft erreicht wird, wird damit diese Variable auf *False* gesetzt. Wird der nächste Knoten vom Typ Abfahrt/Ankunft erreicht, ist das Ende des Pfades erreicht und der Algorithmus springt zum nächsten Startknoten.

Listing 3: Modifizierte Tiefensuche

```
def dfs(G, node, can_skip_startstop, source):
  G.nodes[node]['Marked'] = True
  if node != source:
    if G.nodes[node]['typ'] == 'abfahrtAnkunft':
      if can_skip_startstop:
        # We've encountered the first start-stop node,
        # Can't skip anymore of them
        can_skip_startstop = False
      else:
        # This is the second startstop-node encountered.
        # This is therefore the end of the path.
        G.nodes[node]['Marked'] = False
        return [[node]]

    if G.nodes[node]['typ'] == 'ankunft':
      # If we reach an ankunfts-node that is not the source-node
      # it's always the end of the path
      G.nodes[node]['Marked'] = False
      return [[node]]

  found_paths = []
  for _, u in G.edges(node):
    if 'Marked' not in G.nodes[u]:
      cont_paths = dfs(G, u, can_skip_startstop, source)
      for continuting_path in cont_paths:
        found_paths.append([node] + continuting_path)

  G.nodes[node]['Marked'] = False
  return found_paths
```

Während die Standard-Tiefensuche lediglich einen bestimmten Knoten im Netzwerk sucht und - sobald dieser gefunden wird - die Suche terminiert, liefert diese modifizierte Version einen Baum als Resultat, wobei jeder Pfad im Baum ein potentiell befahrbares Schienenseg-

ment darstellt. Somit sammelt der Code alle möglichen Pfade zwischen Start- und Endknoten. Um nur die tatsächlich fahrbaren Pfade zu erhalten, mussten jedoch für jeden der Pfade mehrere Prüfungen durchgeführt werden:

3.3 Problemfall Weichen

Um die Pfade auszuschliessen, welche von einer Weichenspitze zur anderen Weichenspitze führen, wurde für jedes Knoten-Tripel eine Prüfung durchgeführt. Dabei wurde über jeden Pfad iteriert. Gestartet wurde beim zweiten Knoten jedes Pfades und die Prüfung lief jeweils bis zum zweitletzten Knoten. Dieses Vorgehen wurde gewählt, da sich Start- und Endknoten nie auf einer Weiche befinden. Diese Prüfung wurde mit einen *while-Loop* vorgenommen. Für die Prüfung eines Knotens v_i werden jeweils der Knoten vor dem kritischen Knoten v_{i-1} und der Knoten danach v_{i+1} betrachtet. Für alle drei Knoten werden die hintersten Zeichen des Namens kontrolliert. Eine Weiche kann, wie in der oben beschriebenen Regel (Kapitel 3.1.1), weder über die Knotenfolge *2 - 1 - 3*, noch über die Knotenfolge *3 - 1 - 2* befahren werden. Erkennt der Algorithmus eine dieser Folgen, wird der ganze Pfad aus der Liste gelöscht (Listing 4). Er handelt sich somit um einen nicht fahrbaren Pfad.

Figure 3.3.14: Nicht fahrbarer Pfad über Weiche

Listing 4: Check Weichen

```
#SWITCHES 3-1-2 or 2-1-3
switchesChecked = []
for path in paths:
    keep_path = True
    for p in range(1, len(path) - 1):
        if path[p-1].split('.')[0] == path[p].split('.')[0] == path[p+1].split('
    .')[0]:
            if (path[p-1].split('.')[1] == '3' and path[p].split('.')[1] == '1'
    and (path[p+1].split('.')[1] == '2')):
                keep_path = False
            elif (path[p-1].split('.')[1] == '2' and path[p].split('.')[1] == '1
    ' and (path[p+1].split('.')[1] == '3')):
                keep_path = False
    if keep_path is True:
    switchesChecked.append(path)
```

3.4 Problemfall Abfahrt rückwärts

Angenommen, ein Zug steht in einem Kopfbahnhof. In diesem Fall macht eine Abfahrt nur in eine Richtung Sinn, nämlich zum Ausgang des Bahnhofs. Um Pfade auszuschliessen, welche eine Abfahrt in Richtung des Prellbocks vorsehen, wurde eine weitere Prüfung durchgeführt (Listing 5). Dabei wurden von jedem Pfad die ersten drei Knoten betrachtet. In den meisten Fällen würde eine Prüfung der ersten zwei Knoten ausreichen. Da in Ausnahmefällen jedoch zwischen dem Abfahrtssignal und dem Prellbock eine Weiche eingebaut ist, wurde zusätzlich auch der zweite Knoten nach der Abfahrt kontrolliert. Wenn der Abfahrtsknoten sich an Stelle 0 befindet, prüft der Algorithmus die Knoten an der Stelle 1 und 2 auf ihren Attribut Typen hin. Handelt es sich dabei um den Typ *Ankunft*, wird der Pfad verworfen.

Listing 5: Check Abfahrt Rückwärts

```
#Abfahrt retour
ankunftChecked = list()
for path in switchesChecked:
  keep_path = True
  if (G.nodes[path[0]]['typ'] == 'abfahrtAnkunft' and
    (G.nodes[path[1]]['typ'] == 'abfahrtAnkunft' or
    G.nodes[path[2]]['typ'] == 'abfahrtAnkunft')):
    keep_path = False
  if (G.nodes[path[0]]['typ'] == 'abfahrt' and (G.nodes[path[1]]['typ'] ==
    'ankunft' or G.nodes[path[2]]['typ'] == 'ankunft')):
    keep_path = False
  if keep_path is True:
    ankunftChecked.append(path)
```

3.5 Resultate und Erkenntnisse

Der Algorithmus funktionierte im vereinfachten Netzwerk einwandfrei. Es wurden alle Pfade gesammelt und die nicht fahrbaren Pfade wurden erfolgreich erkannt und sogleich aussortiert (Table 1). Trotzdem wurden einige Herausforderungen deutlich, wie beispielsweise der exakte Standort der Signale, welche bei der Anwendung mit realen Daten zu einem Problem werden könnten. Diese Probleme werden im folgenden Kapitel 4 mit Daten aus der Geodatenbank DfA anhand ener Case Study Schritt für Schritt besprochen und gelöst.

Table 1: Resultat - Vereinfachtes Netzwerk

	K1	K2	K3	K4	K5	K6	K7	K8	K9
Pfad 1	DU1B	DU1.3	DU1.1	DK2.1	DK2.2	DK1A	DK1B		
Pfad 2	DU1B	DU1.3	DU1.1	DK2.1	DK2.3	DK2A	DK2B		
Pfad 3	DU2B	DU1.2	DU1.1	DK2.1	DK2.2	DK1A	DK1B		
Pfad 4	DU2B	DU1.2	DU1.1	DK2.1	DK2.3	DK2A	DK2B		
Pfad 5	DK1A	DK2.2	DK2.1	DU1.1	DU1.3	DU1B	DU1A		
Pfad 6	DK1A	DK2.2	DK2.1	DU1.1	DU1.2	DU2B	DU2A		
Pfad 7	DK2A	DK2.3	DK2.1	DU1.1	DU1.3	DU1B	DU1A		
Pfad 8	DK2A	DK2.3	DK2.1	DU1.1	DU1.2	DU2B	DU2A		
Pfad 9	DK1B	DK3.2	DK3.1	OL4.1	OL4.2	OL5.1	OL5.2	OL2B	OL2A
Pfad 10	DK1B	DK3.2	DK3.1	OL4.1	OL4.2	OL5.1	OL5.3	OL1B	OL1A
Pfad 11	DK1B	DK3.2	DK3.1	OL4.1	OL4.3	OL3B	OL3A		
Pfad 12	DK2B	DK3.3	DK3.1	OL4.1	OL4.2	OL5.1	OL5.2	OL2B	OL2A
Pfad 13	DK2B	DK3.3	DK3.1	OL4.1	OL4.2	OL5.1	OL5.3	OL1B	OL1A
Pfad 14	DK2B	DK3.3	DK3.1	OL4.1	OL4.3	OL3B	OL3A		
Pfad 15	OL1B	OL5.3	OL5.1	OL4.2	OL4.1	DK3.1	DK3.2	DK1B	DK1A
Pfad 16	OL1B	OL5.3	OL5.1	OL4.2	OL4.1	DK3.1	DK3.3	DK2B	DK2A
Pfad 17	OL2B	OL5.2	OL5.1	OL4.2	OL4.1	DK3.1	DK3.2	DK1B	DK1A
Pfad 18	OL2B	OL5.2	OL5.1	OL4.2	OL4.1	DK3.1	DK3.3	DK2B	DK2A
Pfad 19	OL3B	OL4.3	OL4.1	DK3.1	DK3.2	DK1B	DK1A		
Pfad 20	OL3B	OL4.3	OL4.1	DK3.1	DK3.3	DK2B	DK2A		

4 Case Study - Val de Travers

Das vorherige Kapitel 3 zeigt, dass der Algorithmus in einem vereinfachten Netzwerk funktioniert. Um den Algorithmus an existierenden Daten zu testen, wurde ein Perimeter ausgewählt, welcher sowohl Durchfahrtsbahnhöfe, als auch Kopfbahnhöfe umfasst. Dabei handelt es sich um das südwestlich von Neuenburg gelegene *Val de Travers*. Das Tal ist für seine markante Felsarena *Creux-du-Van* über die Schweizer Grenzen hinaus bekannt und bietet auch einige sehenswerte Zugstrecken.

4.1 Shapefile zu Netzwerk

Die Geodatenbank der SBB (DfA) stellt sämtliche Daten im Dateiformat Shapefile *.shp*, welches von der Firma *ESRI* im Jahr 1998 vorgestellt wurde, bereit (Figure 4.1.15) (Environmental Systems Research Institut, Inc., 1998). Da der Algorithmus in der Programmiersprache Python entwickelt wurde, müssen die Daten in ein Format gebracht werden, welches sowohl die Attribute, als auch die dazugehörenden Geometrien enthalten kann. Dazu eignet sich die OpenSource-Bibliothek *Geopandas* (Jordahl, 2014). Diese kann eine Shapefile-Datei lesen und sie als Tabelle darstellen. Eine Spalte beinhaltet dabei die Geometrien. Dazu nutzt sie eine weitere Bibliothek namens *Shapely*, welche zur Manipulation von Geodaten entwickelt wurde (Gillies, 2013). Für die Anwendung im gewünschten Netzwerk mussten die Kanten und Knoten als einzelne Dateien eingelesen werden.

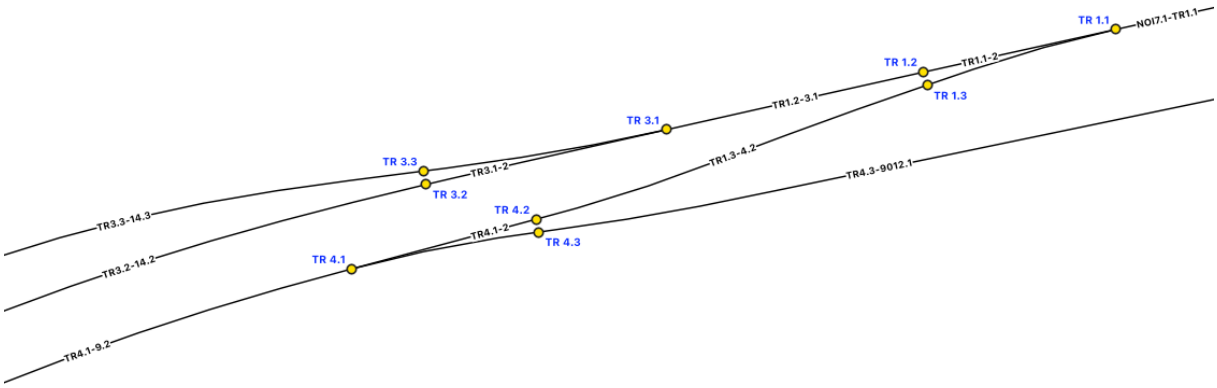


Figure 4.1.15: Daten der DfA in QGIS dargestellt.

4.2 Netzwerk vorbereiten

Im Gegensatz zum vereinfachten Netzwerk, konnte aus den Daten der Datenbank für den Perimeter der Case Study nicht direkt ein Netzwerk erstellt werden. Die Daten mussten zuerst bereinigt und angepasst werden, damit aus ihnen ein Netzwerk erstellt werden kann. Die Daten sind nicht in einer Datei abgespeichert. Attribute und Positionen der Objekte sind zum Teil für den Algorithmus nicht interpretierbar. Daher werden sie mit den folgenden Schritten für die Erstellung des Netzwerks vorbereitet:

4.2.1 Punkte an Linien einrasten

Bei der Betrachtung der Daten in einer GIS-Umgebung, fällt auf, dass die Signalpunkte nicht deckungsgleich mit den Gleissträngen sind (Figure 4.2.16). Das rührt daher, dass Signale wenn möglich auf der linken Seite des Gleises aufgestellt werden. Ist dies nicht der Fall, müssen sie mit einem Hinweispfeil gekennzeichnet sein, für welches Gleis sie gelten. Um die Signale als Knoten verwenden zu können, müssen sie jedoch mit den Gleissträngen deckungsgleich sein, damit die Stränge an den entsprechenden Stellen geteilt werden können. Um dies zu erreichen, wurde die Funktion *snap()* von Shapely verwendet.

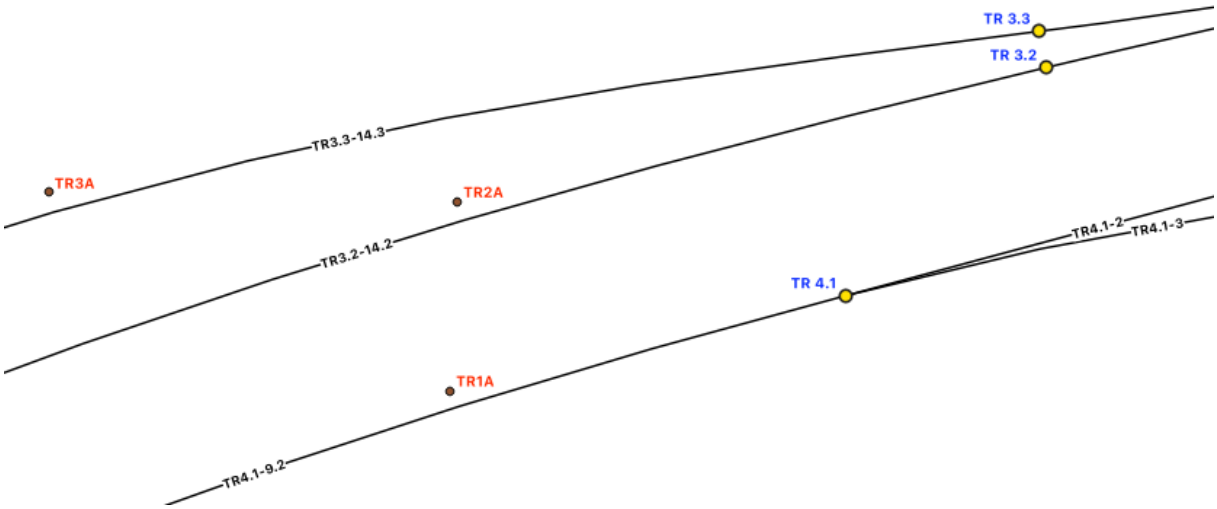


Figure 4.2.16: Noch nicht eingerastete Signalpunkte

Die Funktion 6 braucht drei Argumente. An erster Stelle steht die Variable, welche jene Punkte enthält, die eingerastet werden sollen. An zweiter Stelle stehen die Gleisstränge. Diese müssen vorausgehend vereinigt werden, damit die Datei nur noch eine Linie enthält. An der dritten Stelle steht die Distanz, welche die Punkte zur Linie haben dürfen, damit sie noch eingerastet werden. Die Distanz, welche Punkte zur Linie haben können, ist beim Einrasten der einzelnen Knoten in den realen Daten relevant. Die Knoten der Weichen haben jeweils einen Versatz von wenigen Millimetern zum Gleisstrang. Die Signale hingegen stehen zum Teil mehrere Meter von der Linie entfernt. Zwei Weichenspitzen können jedoch ebenfalls nur wenige Meter voneinander entfernt sein. Daher muss, um das Resultat nicht zu verfälschen, bei den Weichen-Knoten eine kurze Distanz gewählt werden und bei den Signalen eine grössere.

Listing 6: Punkte an Linien einrasten

```
def snapPoints(signals, switches, trackLines):
    #LOAD DATA
    signalList = list()
    for row in signals.itertuples():
        signalList.append({'NAME': row.NAME, 'TYP': row.TYP, 'geometry': row.
    geometry})

    switchList = list()
    for row in switches.itertuples():
```

```
        switchList.append({'NAME': row.BEZEICHNUN, 'TYP': 'WEICHE', 'geometry':
    Point(row.geometry.coords[0][0], row.geometry.coords[0][1])})

    #SNAP POINT TO LINE
    unionLine = trackLines.unary_union
    for signal in signalList:
        signal['geometry'] = snap(signal['geometry'], unionLine, 2)

    for switch in switchList:
        switch['geometry'] = snap(switch['geometry'], unionLine, 0.5)

    #REMOVE DUPLICATES
    removeList = {}
    for signal in signalList:
        signalbuffer = pnt3 = signal['geometry'].buffer(0.1)
        for switch in switchList:
            if signalbuffer.contains(switch['geometry']):
                switchList.remove(switch)
                removeList[switch['NAME'].replace(' ', '')] = signal['NAME'].
    replace(' ', '')
```

4.2.2 Weichen- und Signalpunkte zusammenfügen

Die Weichen-Punkte und Signalpunkte müssen zusammengeführt werden, damit ein Datensatz entsteht. Nur so können die Gleisstränge korrekt geteilt werden. Dazu wurden die beiden Datensätze *Weichen* und *Signale* mit der Funktion *gpd.merge()* vereint (Listing 7).

Listing 7: Knoten zusammenführen

```
#MERGE POINTS
trackPoints = signalList + switchList
for tracks in trackPoints:
  tracks['NAME'] = tracks['NAME'].replace(' ', '')
```

4.2.3 Daten für Netzwerk vorbereiten

Da die Gleisstränge nur über ein Namensfeld verfügen, musste daraus ein *From*- und ein *To*-Feld erstellt werden, welche sich aus den bestehenden Namen zusammensetzen. Dazu wurde der Name des Gleisstrangs beim Bindestrich geteilt und die Resultate den beiden Feldern zugewiesen. Der Gleisstrang der Weiche BN2.1-3 wurde so modifiziert, dass das *From*-Feld BN2.1 und das *To*-Feld BN2.3 enthalten. Diese Modifikation wurde für jeden Gleisstrang vorgenommen (Listing: 8).

Listing 8: Extrahiere From und To Bezeichnungen

```
def findeFromAndTo(trackLines, trackpoints, removeList):
    # Get From and To Points
    trackList = []
    for row in trackLines.itertuples():
        trackList.append({'NAME': row.BEZEICHNUN, 'FROM': np.nan, 'TO': np.nan,
    'geometry': row.geometry})
    tracks = []
```

```
for track in trackList:
    bhf = ''
    #Trackname gets splitted by '-'
    trackName1 = track['NAME'].split('-')[0].replace(' ', '')
    trackName2 = track['NAME'].split('-')[1].replace(' ', '')
    #Define fromPoint
    fromPoint = trackName1

    #Get fromPoint
    #Check if trainstation has a two- or threeletter-code
    if (trackName1[:3])[-1] in ['1', '2', '3', '4', '5', '6', '7', '8', '9',
'0']:
        bhf = trackName1[:2]
    else:
        bhf = trackName1[:3]

    #Get toPoint
    #Check if toPoint is a standalone track
    if trackName2[0] not in ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0
']:
        toPoint = trackName2
    elif len(trackName2) == 1:
        toPoint = trackName1[:-1] + trackName2
    elif len(trackName2) == 1:
        toPoint = trackName1[:-1] + trackName2
    else:
        toPoint = bhf + trackName2
    #Generate tracklist
    tracks.append({'FROM': fromPoint, 'TO': toPoint, 'geometry': track['
geometry']})

##REPLACE DUPLICATES
for track in tracks:
    if track['TO'] in removeList.keys():
        track['TO'] = removeList[track['TO']]
    if track['FROM'] in removeList.keys():
        track['FROM'] = removeList[track['FROM']]
```

Im Anschluss wird aus dem Geodataframe eine Shapefile-Datei *.shp* erstellt, damit die Daten in einer GIS-Umgebung auf ihre *From-* und *To*-Felder überprüft werden können (Table 2).

Table 2: Resultat - Extrahiere From und To

Bezeichnung	From	To
TR15.1-2	TR15.1	TR15.2
TR15.1-3	TR15.1	TR15.3
TR15.2-VER1.1	TR15.2	VER1.1
VER1.1-2	VER1.1	VER1.2
VER1.1-3	VER1.1	VER1.3
VER1.2-3A	VER1.2	VER3A

4.2.4 Kanten bei Knoten teilen

Sind die Daten soweit vorbereitet, müssen die Kanten an den neuen Knoten, welche von den Signalen dargestellt werden, geteilt und gegebenenfalls umbenannt werden (Listing 9).

Listing 9: Kanten bei Knoten teilen

```
#SPLIT TRACKS BY POINTS
  splitPoints = []
  for trackPoint in trackPoints:
    if not '.' in trackPoint['NAME']:
      splitPoints.append(trackPoint)

  while len(splitPoints) > 0:
    pnt = splitPoints.pop(0)
    pointName = pnt['NAME']
    pointGeometry = pnt['geometry']
    for row in list(trackList):
      fromPoint = row['FROM']
      toPoint = row['TO']
      ln = row['geometry']
      if ln.distance(pointGeometry) < 0.001:
        pnt2 = ln.interpolate(ln.project(pointGeometry))
        pnt3 = pnt2.buffer(0.0000001)
        splitter = split(ln, pnt3)
        if len(splitter.geoms) == 3:
          for pint in trackPoints:
            if pint['NAME'] == fromPoint:
              fromPointGeom = pint['geometry']
          elmList = [splitter.geoms[0], splitter.geoms[1]]
          line1 = linemerge(elmList)
          line2 = splitter.geoms[2]
          trackList.remove(row)
          if fromPointGeom.distance(line1) > 0.1:
            tmp = line1
            line1 = line2
            line2 = tmp
          trackList.append(
            {'FROM': fromPoint, 'TO': pointName, 'geometry': line1})
          trackList.append(
            {'FROM': pointName, 'TO': toPoint, 'geometry': line2})
          break
```

Auch im Anschluss an diese Funktion wird eine Shapefile-Datei *.shp* erstellt, welche benutzt wird, um die Segmente darauf zu referenzieren. Dieser Datei werden schlussendlich die einzelnen Segmente zugeordnet.

4.2.5 Daten überprüfen

Mit den geteilten Gleissträngen wären die Daten eigentlich bereit, um aus ihnen ein Netzwerk zu erstellen. Um das Fehlerpotenzial zu minimieren, werden die Datensätze der Knoten und der Kanten gegeneinander validiert (Listing 10). Das bedeutet, dass kontrolliert wird, ob jede Bezeichnung in den *From*- und *To*-Spalten auch im Knoten-Datensatz enthalten ist. Wäre eine Kante vorhanden, welche auf einen Knoten verweist, welcher nicht vorhanden ist, könnte der Graph nicht korrekt erstellt werden. Anschliessend werden auch alle Knoten überprüft, ob sie mit mindestens einer Kante verbunden sind. Wäre dies nicht der Fall, wäre der Knoten für den Graphen nutzlos.

Listing 10: Überprüfung der Daten vor der Netzwerkerstellung

```
def checkData(edgeList, nodeList):
  ##CHECK EDGES
  nodes_ = []
  for node in nodeList:
    nodes_.append(node['NAME'])

  edges_ = []
  for edge in edgeList:
    edges_.append(edge['FROM'])
    edges_.append(edge['TO'])

  ##CLEAN EDGES
  edges = []
  for edge in edgeList:
    if edge['FROM'] in nodes_ and edge['TO'] in nodes_:
      edges.append(edge)

  ##CLEAN NODES
  nodes = []
  for node in nodeList:
    if node['NAME'] in edges_:
      nodes.append(node)

  return edges, nodes
```

Zurückgegeben werden die überprüften Listen mit den Kanten und Knoten

4.3 Graph erstellen

Nun ist es möglich, mit den Daten einen Graphen zu erstellen (Listing 11). Dazu wird die OpenSource-Bibliothek *NetworkX* verwendet (Hagberg et al., 2008). Beim Erstellen des NetworkX-Netzwerkes werden gleichzeitig Informationen zum Typ des Knotens übermittelt. Diese Informationen definieren, ob es sich beim Knoten um eine Weiche oder um ein Signal, bei welchem nur abgefahren, angehalten oder beides gemacht werden kann, handelt.

Listing 11: Graph wird erstellt

```
def createGraph(edges, nodes):
    #Initiate graph
    G = nx.Graph()
    for track in nodes:
        # Add Nodes
        G.add_node(track['NAME'], pos=(track['geometry'].coords[0]
                   [0], track['geometry'].coords[0][1]), typ=track['TYP'])

    for edge in edges:
        # Add Edges
        G.add_edge(edge['FROM'], edge['TO'])

    #Get position attributes
    pos = nx.get_node_attributes(G, 'pos')

    #Draw Network
    nx.draw_networkx_nodes(G, pos, node_color='Grey')
    nx.draw_networkx_labels(G, pos, font_family='segoe ui', font_weight='bold',
                            font_color='black', verticalalignment='bottom',
    font_size=11)
    nx.draw_networkx_edges(G, pos, edge_color='#AFDCEC', width=2)
    plt.show()

    return G
```

Um abschliessend zu überprüfen, ob der Graph korrekt und vollständig erstellt wurde, kann er geplottet werden (Figure 4.3.17). Da bei den Gleissträngen jedoch die Geometrie nicht berücksichtigt wird, werden nur gerade Linien dargestellt. Dies erschwert die Kontrolle.

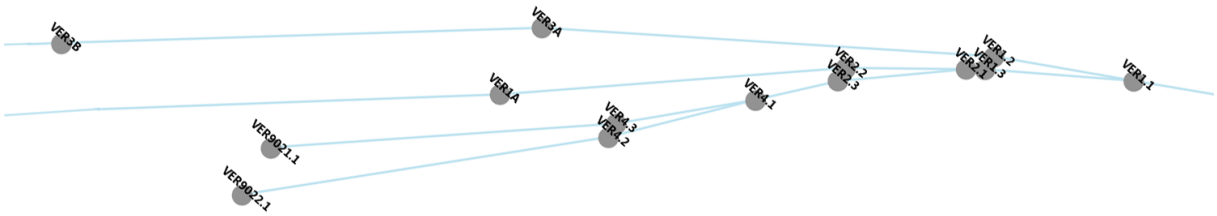


Figure 4.3.17: Ausschnitt des Plots vom erstellten Netzwerks

4.4 Befahrbare Pfade sammeln

Im erstellten Netzwerk können nun die für die Planung der Messfahrten relevanten Pfade gesammelt werden. Dazu kann der identische Algorithmus verwendet werden, welcher auch auf das vereinfachte Netzwerk angewendet wurde. Dazu werden mit der Funktion *findAllPaths()* die Knoten vorbereitet und anschliessend nur die Startknoten gewählt, um die Tiefensuche von diesen Punkten ausgehend auszuführen (Listing 12).

Listing 12: Modifzierte Tiefensuche und Funktion zum Finden der gültigen Pfade

```
def dfs(G, node, can_skip_startstop, source):
    G.nodes[node]['Marked'] = True
    if node != source:
        if G.nodes[node]['typ'] == 'abfahrtAnkunft':
            if can_skip_startstop:
                # We've encountered the first start-stop node,
                # Can't skip anymore of them
                can_skip_startstop = False
            else:
                # This is the second startstop-node encountered.
                # This is therefore the end of the path.f
                #del G.nodes[node]['Marked']
                G.nodes[node]['Marked'] = False
                return [[node]]

        if G.nodes[node]['typ'] == 'ankunft':
            # If we reach an ankunfts-node that is not the source-node
            # it's always the end of the path
            G.nodes[node]['Marked'] = False
            return [[node]]

    found_paths = []
    for _, u in G.edges(node):
        if 'Marked' not in G.nodes[u]:
            cont_paths = dfs(G, u, can_skip_startstop, source)
            for continuting_path in cont_paths:
                found_paths.append([node] + continuting_path)

    G.nodes[node]['Marked'] = False
    return found_paths

def find_all_paths(G):
    paths = []
    for v in G.nodes():
        # Unmark all nodes
        for u in G.nodes():
            if 'Marked' in G.nodes[u]:
                del G.nodes[u]['Marked']

        if G.nodes[v]['typ'] == 'abfahrt':
            paths += dfs(G, v, True, v)
        if G.nodes[v]['typ'] == 'abfahrtAnkunft':
            paths += dfs(G, v, True, v)
```

```
    #SWITCHES 3-1-2 or 2-1-3
    switchesChecked = []
    for path in paths:
        keep_path = True
        for p in range(1, len(path) - 1):
            if path[p-1].split('.')[0] == path[p].split('.')[0] == path[p+1].
split('.')[0]:
                if (path[p-1].split('.')[1] == '3' and path[p].split('.')[1] ==
                        '1' and (path[p+1].split('.')[1] == '2')):
                    keep_path = False
                elif (path[p-1].split('.')[1] == '2' and path[p].split('.')[1]
==
                      '1' and (path[p+1].split('.')[1] == '3')):
                    keep_path = False
        if keep_path is True:
            switchesChecked.append(path)

    #CHECK DEPARTURE BACKWARD
    ankunftChecked = []
    for path in switchesChecked:
        keep_path = True

        if (G.nodes[path[0]]['typ'] == 'abfahrtAnkunft' and (G.nodes[path[1]]['
typ'] == 'abfahrtAnkunft' or G.nodes[path[2]]['typ'] == 'abfahrtAnkunft')):
            keep_path = False
        if (G.nodes[path[0]]['typ'] == 'abfahrt' and (G.nodes[path[1]]['typ'] ==
 'ankunft' or G.nodes[path[2]]['typ'] == 'ankunft')):
            keep_path = False
        if keep_path is True:
            ankunftChecked.append(path)

    #Ende an Ankunft / Abfahrt/Ankunft
    paths_to_keep = []
    for path in ankunftChecked:
        keep_path = True
        if G.nodes[path[-1]]['typ'] == 'abfahrtAnkunft' or G.nodes[path[-1]]['
typ'] == 'ankunft':
            paths_to_keep.append(path)
    return paths_to_keep

G = createGraph(edges, nodes)

paths = find_all_paths(G)
```

Zurückgegeben wird ebenfalls eine Liste mit allen Pfaden von den jeweiligen Startknoten zu sämtlichen passenden Endknoten gemäss Fahrdienstvorschriften.

Table 3: Resultat - gefundene und validierte Pfade

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
VER3A	VER1.2	VER1.1	TR15.2	TR15.1	TR14.1	TR14.2	TR2B	TR2A		
VER3A	VER1.2	VER1.1	TR15.2	TR15.1	TR14.1	TR14.3	TR3B	TR3A		
VER1A	VER2.2	VER2.1	VER1.3	VER1.1	TR15.2	TR15.1	TR14.1	TR14.2	TR2B	TR2A
VER1A	VER2.2	VER2.1	VER1.3	VER1.1	TR15.2	TR15.1	TR14.1	TR14.3	TR3B	TR3A
TR4B	TR8.2	TR8.1	TR9.1	TR9.2	TR1B	TR1A				
TR4B	TR8.2	TR8.1	TR9.1	TR16.2	TR16.1	COU1A	COU1B			
TR1B	TR9.2	TR9.1	TR8.1	TR8.2	TR4B	TR4A				
TR1B	TR9.2	TR9.1	TR16.2	TR16.1	COU1A	COU1B				
TR1A	TR4.1	TR4.2	TR1.3	TR1.1	NOI7.1	NOI7.2	NOI3B	NOI3A		
TR1A	TR4.1	TR4.2	TR1.3	TR1.1	NOI7.1	NOI7.3	NOI2B	NOI2A		
TR2A	TR3.2	TR3.1	TR1.2	TR1.1	NOI7.1	NOI7.2	NOI3B	NOI3A		
TR2A	TR3.2	TR3.1	TR1.2	TR1.1	NOI7.1	NOI7.3	NOI2B	NOI2A		
TR3A	TR3.3	TR3.1	TR1.2	TR1.1	NOI7.1	NOI7.2	NOI3B	NOI3A		
TR3A	TR3.3	TR3.1	TR1.2	TR1.1	NOI7.1	NOI7.3	NOI2B	NOI2A		
TR2B	TR14.2	TR14.1	TR15.1	TR15.2	VER1.1	VER1.2	VER3A	VER3B		
TR2B	TR14.2	TR14.1	TR15.1	TR15.2	VER1.1	VER1.3	VER2.1	VER2.2	VER1A	VER1B
TR2B	TR14.2	TR14.1	TR15.1	TR15.3	TR16.3	TR16.1	COU1A	COU1B		
TR3B	TR14.3	TR14.1	TR15.1	TR15.2	VER1.1	VER1.2	VER3A	VER3B		
TR3B	TR14.3	TR14.1	TR15.1	TR15.2	VER1.1	VER1.3	VER2.1	VER2.2	VER1A	VER1B
TR3B	TR14.3	TR14.1	TR15.1	TR15.3	TR16.3	TR16.1	COU1A	COU1B		
NOI2A	NOI3.2	NOI3.1	NOI2.3	NOI2.1	CDM3.1	CDM3.2	CDM2A	CDM2B		
NOI2A	NOI3.2	NOI3.1	NOI2.3	NOI2.1	CDM3.1	CDM3.3	CDM1A	CDM1B		
NOI3A	NOI2.2	NOI2.1	CDM3.1	CDM3.2	CDM2A	CDM2B				
NOI3A	NOI2.2	NOI2.1	CDM3.1	CDM3.3	CDM1A	CDM1B				
NOI3B	NOI7.2	NOI7.1	TR1.1	TR1.2	TR3.1	TR3.2	TR2A	TR2B		
NOI3B	NOI7.2	NOI7.1	TR1.1	TR1.2	TR3.1	TR3.3	TR3A	TR3B		
NOI3B	NOI7.2	NOI7.1	TR1.1	TR1.3	TR4.2	TR4.1	TR1A	TR1B		
NOI2B	NOI7.3	NOI7.1	TR1.1	TR1.2	TR3.1	TR3.2	TR2A	TR2B		
NOI2B	NOI7.3	NOI7.1	TR1.1	TR1.2	TR3.1	TR3.3	TR3A	TR3B		
NOI2B	NOI7.3	NOI7.1	TR1.1	TR1.3	TR4.2	TR4.1	TR1A	TR1B		
CDM2A	CDM3.2	CDM3.1	NOI2.1	NOI2.2	NOI3A	NOI3B				
CDM2A	CDM3.2	CDM3.1	NOI2.1	NOI2.3	NOI3.1	NOI3.2	NOI2A	NOI2B		
CDM1A	CDM3.3	CDM3.1	NOI2.1	NOI2.2	NOI3A	NOI3B				
CDM1A	CDM3.3	CDM3.1	NOI2.1	NOI2.3	NOI3.1	NOI3.2	NOI2A	NOI2B		
COU1A	TR16.1	TR16.3	TR15.3	TR15.1	TR14.1	TR14.2	TR2B	TR2A		
COU1A	TR16.1	TR16.3	TR15.3	TR15.1	TR14.1	TR14.3	TR3B	TR3A		
COU1A	TR16.1	TR16.2	TR9.1	TR9.2	TR1B	TR1A				
COU1A	TR16.1	TR16.2	TR9.1	TR8.1	TR8.2	TR4B	TR4A			

4.5 GIS-Daten erstellen

Die ermittelten Pfade entsprechen den Knoten, welche für den jeweiligen Pfad begangen werden müssen. Da sich der Name des Gleisstranges aus den Namen der Knoten, die er verbindet, ergeben, kann aus diesen Listen auch die Liste der Gleisstränge abgeleitet werden. Die Knoten *VER3A* und *VER1.2* entsprechen dabei dem Gleisstrang *VER3A - 1.2*. Diese Art der Namensgebung kann genutzt werden, um für jeden Pfad die Stränge zu gruppieren. Dazu wird eine weitere Funktion verwendet (Listing 13).

Listing 13: Erstelle aus Liste ein Shapefile

```
def makeTracks(paths):
  pathList = []
  #Merge nodenames to edgenames
  for path in paths:
    print(path)
    edgeList = []
    counter = 1
    for i in range(0, len(path)-1):
      edgeList.append([path[i], path[counter]])
      counter += 1
    pathList.append(edgeList)

  masterList = []
  for path in pathList:
    fromPoint = path[0][0]
    toPoint = path[-1][-1]
    geometryList = []
    for i in range(0, len(path)):
      for track in trackList:
        if (path[i][0] == track['FROM'] or path[i][0] == track['TO']) and (path[
   i][1] == track['FROM'] or path[i][1] == track['TO']):
          geometryList.append(track['geometry'])

    trackLine = MultiLineString(geometryList)
    masterList.append(
      {'FROM': fromPoint, 'TO': toPoint, 'geometry': trackLine})

  lineNew = gpd.GeoDataFrame(masterList)
  lineNew.set_geometry('geometry')
  lineNew.set_crs('EPSG:2056')
  lineNew.to_file('./Data/TEMP/MASTER.shp', driver='ESRI Shapefile')
  prj = open('./Data/TEMP/MASTER.prj', "w")
  epsg = 'PROJCS["CH1903+_LV95",'
  epsg += 'GEOGCS["GCS_CH1903+",'
  epsg += 'DATUM["D_CH1903+",'
  epsg += 'SPHEROID["Bessel_1841",6377397.155,299.1528128]],'
  epsg += 'PRIMEM["Greenwich",0.0],'
  epsg += 'UNIT["Degree",0.0174532925199433]],'
  epsg += 'PROJECTION["Hotine_Oblique_Mercator_Azimuth_Center"],'
  epsg += 'PARAMETER["False_Easting",2600000.0],'
  epsg += 'PARAMETER["False_Northing",1200000.0],'
```

```
epsg += 'PARAMETER["False_Northing",1200000.0],'
epsg += 'PARAMETER["Scale_Factor",1.0],'
epsg += 'PARAMETER["Azimuth",90.0],'
epsg += 'PARAMETER["Longitude_Of_Center",7.43958333333333],'
epsg += 'PARAMETER["Latitude_Of_Center",46.9524055555556],'
epsg += 'UNIT["Meter",1.0]]'
prj.write(epsg)
prj.close()
```

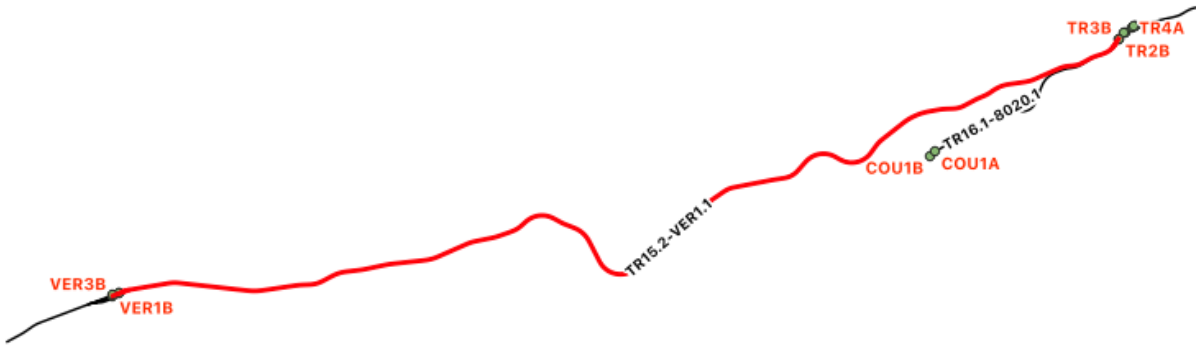


Figure 4.5.18: Markierter Pfad zwischen Les Verriers und Travers

Aus der Liste wird ein Geodataframe erstellt. Aus dem Geodataframe kann anschliessend mit der Funktion gpd.tofile() eine Shapefile-Datei erzeugt werden. Das Resultat lässt sich nun in einer GIS Umgebung wie *ArcGIS Pro* oder *QGIS* darstellen, validieren und weiterverwenden.

5 Diskussion

Zum präsentierten Ansatz der Segmentierung stellen sich nun folgende, seit Beginn der Arbeit leicht modifizierte und ergänzte Fragen:

- Inwiefern kann der Ansatz dazu verwendet werden, um Messfahrten effizienter zu planen?
- Wie ist es möglich, den Algorithmus so anzupassen, damit vorhandene Geodaten verwendet werden können, ohne dass ein zusätzlicher manueller Aufwand entsteht?
- Welche Limitationen beschränken den Einsatz des präsentierten Ansatzes aktuell?
- Kann das Ergebnis des Algorithmus an vorhandenen Daten manuell validiert werden?

5.1 Verwendung der Segmentierung zur Planung von Messfahrten

Nachdem die befahrbaren Streckensegmente extrahiert wurden, hat sich die Planung von Messfahrten erheblich vereinfacht. Im folgenden wird skizziert, wie die Segmentierung für die vereinfachte Planung verwendet werden kann: Da das Ziel einer Messfahrt darin besteht, alle Streckensegmente eines Perimeters zu besuchen, muss lediglich mit einem geeigneten Algorithmus eine Reihenfolge ermittelt werden, mit welcher alle definierten Segmente abgefahren werden können. Dies kann beispielsweise erreicht werden, durch den Einsatz eines Algorithmus zur Lösung des Briefträgerproblems (Eiselt et al., 1995; Thimbleby, 2003). Das Briefträgerproblem stellt die Frage, in welcher Reihenfolge die Knoten abgefahren werden sollen, so dass alle Kanten mindestens einmal besucht wurden und dabei das Gesamtgewicht der Kanten möglichst klein wird. Einem solchen Algorithmus kann nun ein neues (gerichtetes) Netzwerk übergeben werden, welches wie folgt definiert ist:

- **Knoten**: Jeder Abfahrts- bzw. Ankunftsknoten im ursprünglichen Netzwerk bleibt auch Abfahrts- bzw. Ankunftsknoten im neuen Netzwerk. Knoten, welche sowohl Ankunft wie auch Abfahrt zulassen, werden aufgeteilt zu einem Abfahrts- und einem Ankunftsknoten.
- **Kanten**: Jedes extrahierte Segment stellt eine Kante im Netzwerk dar. Diese startet jeweils an einem Abfahrtsknoten und endet an einem Ankunftsknoten. Das Gewicht der Kante ist dabei die Gesamtlänge des Segments. Zusätzlich werden Kanten zwischen allen zusammengehörigen Ankunfts- und Abfahrtsknoten eingefügt, welche somit den Übergang von der Ankunft zur Abfahrt symbolisieren. An Kopfbahnhöfen stehen diese Kanten für das Wenden des DFZ.

Das Ergebnis eines Algorithmus für das Briefträgerproblem wäre somit eine befahrbare Strecke für die gesamte Messfahrt in einem Perimeter. Letztlich muss noch die Frage geklärt werden, wie diese Funktionalität den Endbenutzerinnen und Endbenutzern zur Verfügung gestellt wird. Dazu würde sich eine besonders eine Applikation mit grafischer Benutzeroberfläche eignen, damit keine technische Expertise nötig ist, um die Planung durchzuführen. Der Entwurf einer solchen Applikation ist jedoch jenseits des Rahmens dieser Arbeit.

5.2 Einsatz des Ansatzes mit realen Geodaten

Die Case Study im Val de Travers hat erfolgreich aufgezeigt, dass sich mit den Segmenten die Planung einer Messkampagne in einem Bruchteil der Zeit im Vergleich zu vorher erledigen lässt, da nicht jeder Gleisstrang von Hand ausgewählt werden muss. Die Gleisstränge wurden bisher mit der Maus angeklickt und so markiert, was ein hohes Fehlerpotenzial mit sich brachte. Insbesondere bei Weichen kam es häufig vor, dass versehentlich beide Weichenstränge ausgewählt wurden, da die Zoomstufe nicht optimal gewählt war. Dieses Risiko kann mit den Segmenten stark reduziert werden, da einerseits weniger Klicks getätigt werden müssen und andererseits die Segmente nie bei Weichen starten oder enden. Wird der Perimeter auf das vollständige Schienennetz der Schweiz erweitert und im Falle einer Weiterverarbeitung der Daten, müssen die folgenden Herausforderungen berücksichtigt werden:

5.3 Limitationen des präsentierten Segmentierungsansatzes

5.3.1 Gleisnummern - NeTS

Da die Bezeichnungen der Gleise, welche ins Trassenbestellprogramm NeTS implementiert werden müssen, sich nicht mit den Bezeichnungen der Topologie decken, wäre eine zusätzliche *GIS-Überlappungsanalyse* notwendig, um die Gleisnummern zu erhalten. Ein Problem hierbei stellt die Tatsache dar, dass die Gleisnummern nicht einmalig sind. Das heisst, dass sie im Gegensatz zu den Gleissträngen keine Bahnhofsbezeichung besitzen, sondern lediglich die Nummer. Das Gleis *120* kann so beispielsweise mehrfach vorkommen. Dies führt dazu, dass die Bahnhofsbezeichnung von den Gleissträngen übernommen werden müsste, um eine eindeutige Kennzeichnung der Gleisnummern zu generieren.

5.3.2 Nicht einstellbare Fahrstrassen

Die Fahrstrassen für jeden Zug werden vom Fahrdienstleiter (FDL) in der Betriebszentrale eingestellt. Auf einer schematischen Darstellung sieht der FDL den Standort des Zuges mit der Zugnummer. Für diesen Zug kann nun eine Fahrstrasse eingestellt werden. Die wählbaren Fahrstrassen sind im System abgespeichert. Es können also nicht alle existierenden und fahrbaren Wege zwischen zwei Hauptsignalen eingestellt werden. Ein Beispiel dafür sind Spurwechsel. Ein Spurwechsel ist eine Weichengruppe auf der Strecke. Diese werden hauptsächlich im Störungsfall verwendet, um das Gleis zu wechseln. Je nach Fahrtrichtung ist es möglich, über zwei Weichen das Gleis zu wechseln und wenige Meter später über zwei weitere Weichen auf das ursprüngliche Gleis zurück zu fahren. Dieser Vorgang wird umgangssprachlich *Badewannen-Fahrt* genannt. Dieses Manöver lässt sich nicht für jeden Spurwechsel einstellen. Somit ergibt der Segmentierungs-Algorithmus einen Pfad, welcher theoretisch gefahren werden kann, in der Realität aber nicht umsetzbar ist.

5.3.3 Standort der Signale

Da eine Zugfahrt bei einem Signal beginnt und endet, muss der Gleisstrang an dieser Stelle auch unterteilt sein. Da in der Topologie der Eisenbahn diese Bedingung nicht erfüllt sein muss, muss diese Unterteilung für die Segmentierung zusätzlich erfolgen. Ein Hauptsignal befindet

sich jedoch nie über dem Bahntrasse. Im Schweizer Eisenbahnnetz sind Signale, sofern möglich und sinnvoll, auf der linken Seite angebracht. Im Ausnahmefall kann ein Signal auch auf der rechten Seite angebracht werden. In diesem Fall muss der Lokführer oder die Lokführerin auf den Umstand hingewiesen werden. Dies geschieht in aller Regel mit einem Hinweispfeil, welcher vom Signal auf das Gleis zeigt, für welches es gilt. Der Algorithmus versucht dieses Problem zu lösen, indem er Punkt-Daten, welche sich in einem definierten Abstand zum Gleis befinden, an die Gleisstränge einrastet. Dieses Verfahren führt in den meisten Fällen zu einer korrekten und vollständigen Lösung. Es kann jedoch nicht automatisch überprüft werden, ob das eingerastete Signal tatsächlich zum gewählten Gleis gehört.

5.3.4 Laufzeit

Ein weiteres Problem stellt die Laufzeit des Algorithmus dar. Die Laufzeit einer Tiefensuche nimmt mit der zunehmenden Anzahl Startknoten stark zu. Die Laufzeit für den gewählten Testperimeter Val de Travers beträgt lediglich ca. 1.5 Sekunden. Da jedoch nicht nur die Tiefensuche erfolgen muss, sondern auch die Vor- und Nachbereitung der Daten und diese in ein Shapefile geschrieben werden müssen, wird die Laufzeit zu einem relevanten Faktor bei der Anwendung des Algorithmus. Um diesem Problem entgegen zu wirken, könnte der Algorithmus an einigen Stellen verbessert werden, um Laufzeit einzusparen.

5.3.5 Dynamik der Daten

Die Daten der *Datenbank feste Anlagen* (DfA) werden täglich aktualisiert. Um immer mit den aktuellsten Daten arbeiten zu können, müsste der Algorithmus jeden Tag ausgeführt werden. Weiter kommt hinzu, dass die Planung einer Messkampagne bisher mehrere Wochen dauerte; bis die definitiven Anordnungen eingetroffen sind, jeweils mehrere Monate. Die Messfahrten finden dann etwa ein halbes Jahr nach der Planung statt. Somit besteht ein hohes Risiko, dass die Segmente am Tag der Messung nicht mehr mit den Segmenten vom Tag der Planung übereinstimmen. Die Auswertung und Überprüfung, ob der geplante Perimeter vollständig befahren und gemessen wurde, ist dadurch erschwert. Eine Möglichkeit, dieses Problem zu lösen, wäre das Definieren eines Standarddatensatzes. Dieser müsste zu Beginn der Kampagnenplanung erstellt werden. Die gesamte Planung sowie die Messfahrt selbst, würden diese Daten als Grundlage benutzen.

5.3.6 Qualität der Signal-Daten

Entscheidend für die Qualität der Resultate, ist die Qualität des Signal-Datensatzes. Die bereits beschriebenen Herausforderungen betreffend Platzierung sind mitunter einer der Gründe. Weiter kommt hinzu, dass es im Schweizer Schienennetz unzählige Ausnahme-Regelungen gibt, welche das Automatisieren erschweren. So kann es vorkommen, dass in bestimmten Bahnhöfen Blechtafeln die Signale ersetzen. Weit verbreitet sind ausserdem sogenannte Gruppensignale. Dabei werden mit einem Signal die Fahrstrassen von mehr als einem Gleis eingestellt. Ein Zug erhält in solchen Situationen die Abfahrtserlaubnis anhand des gezeigten Fahrbegriffs, welcher nur für ein einzelnes Gleis gilt. Um solche Ausnahmen im Algorithmus zu berücksichtigen, wäre weitere Forschung angezeigt.

5.4 Manuelle Valdierung

Da das Resultat des Algorithmus in Form einer Shapefile-Datei *.shp* vorliegt, ist es möglich, die Daten in einer GIS-Umgebung zu betrachten. Werden dazu die Daten aus der DfA hinzugefügt, lässt sich die Validierung von Hand ausüben. Da es sich bei dieser Tätigkeit jedoch um eine zeitaufwändige Aufgabe handelt, können auch Stichproben überprüft werden. Gerade komplexe Bahnhöfe, wie der Hauptbahnhof Zürich oder Lausanne, können Spezialfälle enthalten, welche der Algorithmus nicht verarbeiten kann. Eine vollständige manuelle Validierung würde hingegen keinen Mehrwert bringen, da der Zeitaufwand zu gross ist und davon ausgegangen werden kann, dass durchschnittliche Streckenabschnitte korrekt segmentiert wurden. Die manuelle Prüfung des Testperimeters hat ergeben, dass der Algorithmus den Datensatz vollständig und korrekt segmentiert hat.

6 Fazit

Um die Planung von Messfahrten zur Überprüfung der Bahninfrastruktur zu vereinfachen und effizienter zu gestalten, sollte im Rahmen dieser Masterarbeit ein Algorithmus entwickelt werden, welcher das Schienennetz in einzelne, fahrbare Segmente unterteilt. Anhand einer Case Study mit dem Perimeter des Val de Travers wurde der Algorithmus an vorhandenen, real existierenden Daten getestet. Die Segmentierung der Geodaten aus der SBB Geodatenbank (DfA) konnte erfolgreich durchgeführt werden. Dieser Schritt ist eine Grundvoraussetzung für spätere Netzwerkanalysen, welche das Ziel verfolgen, den günstigsten Pfad zu finden.

Um den Algorithmus zu entwickeln und zu testen, wurde zu Beginn ein vereinfachtes fiktives Netzwerk erstellt, mit dem Zweck, die Laufzeit möglichst gering zu halten und die Resultate visuell kontrollieren zu können. Dabei wurde berücksichtigt, dass in einem Bahnnetz - unter Anwendung der Graphentheorie - die Weichen als Knoten und die Gleisstränge als Kanten betrachtet werden können. Züge verkehren jeweils von Bahnhof zu Bahnhof, wobei die Fahrt jeweils vor einem Signal startet und auch wieder endet. Aus diesem Grund wurde nebst den Weichen auch ein Signal-Datensatz erstellt, welcher ebenfalls Knoten im Netzwerk darstellt. Um in einem Netzwerk alle Pfade zwischen allen Startknoten und allen Endknoten zu finden, eignet sich die Tiefensuche, auch *Depth First Search* genannt, am besten. Mit diesem Ansatz wurden sämtliche Pfade von allen Abfahrtssignalen zu den benachbarten Ankunfts- /Abfahrtssignalen gesammelt. Von den gesammelten Pfaden lassen sich jedoch nicht alle befahren. So ist es beispielsweise aufgrund der physikalischen Eigenschaften von Bahnanlagen nicht möglich, mit einem Zug von einer Weichenwurzel direkt zur nebenanliegenden Weichenwurzel zu fahren. Identifiziert der Algorithmus in einem Pfad eine solche Abfolge, wird dieser Pfad verworfen. Ein weiteres Problem stellen Durchfahrtsbahnhöfe dar. Würde der Zug beim ersten Abfahrts-/Ankunftssignal anhalten, wäre lediglich die Lok am Perron. Der Algorithmus erkennt, wenn ein erstes Abfahrts-/Ankunftssignal passiert wird und führt die Fahrt dann bis zum nächsten Abfahrts-/Ankunftssignal fort. Weiter prüft der Algorithmus auch Pfade, welche einer Fahrt von der einen Seite des Perrons zur anderen Seite entsprechen dahingehend, ob der zweite Knoten in der Abfolge ebenfalls ein Signal ist. Eine solche Fahrt ist nicht möglich und wird als Konsequenz verworfen. Alle verbleibenden Pfade werden anschliessend als Shapefile abgespeichert, damit eine Weiterverarbeitung möglich ist. Der Algorithmus vereinfacht die Planung einer Messfahrt und vermeidet Fehler, die bei der manuellen Planung gezwungenermassen entstehen. Ausserdem werden damit die Grundlagen geschaffen, weitere Prozesse zu automatisieren und die Segmentierung auf einen grösseren Perimeter auszudehnen. Dazu müssen jedoch noch weitere Daten mit einbezogen werden, da der Prozess mit einem ungewichteten Graphen arbeitet. Die Distanz, respektive Kosten, welche für das Befahren einer Kante entstehen, wurden in dieser Arbeit nicht berücksichtigt. Dies wäre Gegenstand einer nun anstehenden Netzwerkanalyse, welche den günstigsten Weg durchs Netzwerk sucht.

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Literaturverzeichnis

Tarjan, R. E. (1972). Depth-first search and linear graph algorithms. *SIAM J. Comput.*, *1*, 146–160.

Eiselt, H. A., Gendreau, M., & Laporte, G. (1995). Arc routing problems, part i: The chinese postman problem. *Operations Research*, *43*(2), 231–242.

Snyder, J. P. (1997). *Flattening the earth: Two thousand years of map projections*. University of Chicago Press.

Environmental Systems Research Institut, Inc. (1998). *ESRI shapefile technical description* (White Paper) [White Paper].

Rösch, N. (1998). *Topologische beziehung in geo-informationssystemen* (Doctoral dissertation). Universität Fridericiana zu Karlsruhe.

Thimbleby, H. (2003). The directed chinese postman problem. *Software: Practice and Experience*, *33*(11), 1081–1096.

Andreev, K., & Racke, H. (2006). Balanced graph partitioning. *Theory of Computing Systems*, *39*(6), 929–939.

Bundesamt für Verkehr. (2007). *Die zukunft der bahn in der schweiz*. Retrieved November 19, 2021, from https://www.admin.ch/gov/de/start/dokumentation/medienmitteilungen.msg-id-11941.html

Hagberg, A., Swart, P., & S Chult, D. (2008). *Exploring network structure, dynamics, and function using networkx* (tech. rep.). Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Brandes, U. (2010). Graphentheorie. In C. Stegbauer & R. Häußling (Eds.), *Handbuch netzwerkforschung* (pp. 345–353). VS Verlag für Sozialwissenschaften. https://doi.org/10.1007/978-3-531-92575-2_31

Täubig, D. H. (2011). *Fortgeschrittene netzwerk- und graph-algorithmen* (Vorlesung) [Vorlesung].

Volbert, P. D. K. (2011). *Algorithmen und datenstrukturen*.

Gillies, S. (2013). The shapely user manual. *URL https://pypi. org/project/Shapely*.

Jordahl, K. (2014). Geopandas: Python tools for geographic data. *URL: https://github. com/-geopandas/geopandas*.

Schweizerische Bundesbahnen. (2015). Die fahrbahn kurz erklärt.

Stölzle, W., Weidmann, U., Klaas-Wissing, T., Kupferschmid, J., & Riegel, B. (2015). *Vision mobilität schweiz 2050* (tech. rep.). ETH Zurich.

Lukácová, P. D. M. (2017). *Eine einführung in die mathematik*.

Schmid, P. (2019, February 9). *Künstliche Intelligenz auf Schienen*. Retrieved November 23, 2021, from https://www.bulletin.ch/de/news-detail/kuenstliche-intelligenz-auf-schienen.html

Bundesamt für Statistik. (2020, March 11). *Infrastruktur und streckenlänge* [Infrastruktur und streckenlänge]. Retrieved April 17, 2021, from https://www.bfs.admin.ch/bfs/de/home/statistiken/mobilitaet-verkehr/verkehrsinfrastruktur-fahrzeuge/streckenlaenge.html

ESRI Schweiz. (2020). *Case Study: SBB - VISO gibt Störungen und Alarmen ein Gesicht*. Retrieved November 19, 2021, from https://www.esri.ch/de-ch/case-studies/sbb-viso-gibt-stoerungen-und-alarmen-ein-gesicht

Herrmann, P. D. M. (2020). *Graphentheorie* (Skript der Vorlesung) [Skript der Vorlesung].

Bundesamt für Statistik. (2021). *Umweltauswirkungen*. Retrieved November 21, 2021, from https://www.bfs.admin.ch/bfs/de/home/statistiken/mobilitaet-verkehr/unfaelle-umweltauswirkungen/umweltauswirkungen.html

Hoegel, B. (2021, June 11). *Tiefensuche* [Tiefensuche]. Retrieved November 23, 2021, from https://www.biancahoegel.de/computer/inform/tiefensuche.html

Jorio, L. (2021, September 20). *Der Schweizer Eisenbahn-Gigant in Zahlen* [SWI swissinfo.ch]. Retrieved November 21, 2021, from https://www.swissinfo.ch/ger/wirtschaft/oeffentlicher-verkehr_der-schweizer-eisenbahn-gigant-in-zahlen/45226108

Mermec GROUP. (2021). *ROGER 1000 Messzug*. Retrieved November 23, 2021, from http://www.mermecgroup.com/de/pageview2.php?i=104&sl=1

Schweizerische Bundesbahnen. (2021a). *NeTS Plan: Trassenplanung für Europas dichtestes Bahnnetz — SBB* [NeTS Plan: Trassenplanung für Europas dichtestes Bahnnetz — SBB]. Retrieved October 18, 2021, from https://bahninfrastruktur.sbb.ch/de/produkte-dienstleistungen/bahninformatiksysteme/fahrplanplanung/nets-plan.html

Schweizerische Bundesbahnen. (2021b). *TOPO: Eine Plattform für alle Infrastruktur-Topologiedaten* [SBB News]. Retrieved November 19, 2021, from https://news.sbb.ch/artikel/89558/topo-eine-plattform-fuer-alle-infrastruktur-topologiedaten

Schweizerische Bundesbahnen. (2021c). *Vorgaben - inhalte*. Retrieved November 19, 2021, from https://vorgaben.sbb.ch/inhalte/17871?regulation-redirect=0

Trasse Schweiz. (2021). *Schweizerische Trassenvergabestelle - Glossar* [Glossar]. Retrieved October 18, 2021, from https://www.tvs.ch/erlauterungen/glossar

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren und Autorinnen beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bern, 25.November.2021