Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

OSM Scotland Yard

Jimmy Wagner

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Tobias Rupp, M.Sc.
Beginn am:	8. Oktober 2021
Beendet am:	8. April 2022

Kurzfassung

Spielpläne der mitunter beliebtesten Gamifizierung des *Pursuit-Evasion* Problems, namens *Scotland Yard*, stellen zumeist Abbildungen realer Städte dar. Zur Effizienzoptimierung des Kartengenerierungsprozesses dieser Spielpläne, wird in dieser Arbeit ein voll automatisierter Spielplanerstellungsprozess vorgestellt, der darauf abzielt die Problematiken der sonst üblichen manuellen Erstellungsprozesse zu beheben. Die von der Spielergemeinde gewünschte Realitätsbindung der zu erstellenden Spielpläne wird durch den Bezug von Satellitenbildern realer Städte mittels *Mapbox* erreicht, sowie durch die Verwendung der von *OpenStreetMap* bereitgestellten geographischen Daten. Dabei werden alle Informationen rundum Routen öffentlicher Verkehrsmittel extrahiert und durch Algorithmen einzelner Verarbeitungsschritte generalisiert, sodass die Abbildungen der Routen auf der Satellitenkarte, alle Qualitätskriterien eines übersichtlichen Spielplans erfüllen. Im Anschluss des Generierungsprozesses wird der Spielplan an eine digitale Variante von *Scotland Yard* angebunden, wodurch sich die Spielbarkeit und die Übersicht der generierten Spielpläne in der Praxis beurteilen lassen.

Inhaltsverzeichnis

1 Einleitung **13**
1.1 Motivation 13
1.2 Verwandte Arbeiten 14

2 Grundlagen **15**
2.1 Digitale Variante Scotland Yard 15
2.2 Mapbox 17
2.3 OpenStreetMap 17
2.4 Koordinatenreferenzsysteme 20
2.5 Kartenprojektionen 21
2.6 Internes Datenmodell 23
2.7 Qualitätskriterien des Spielplans 24

3 Kartengenerierungsprozess **27**
3.1 Kompression 28
3.2 Verschmelzung ähnlicher Routen 32
3.3 Positionsidentifikation der Stationen 48
3.4 Lösung von Restkonflikten gemäß Qualitätskriterien 53

4 Experimente **57**
4.1 Abhängigkeit des Zoomlevels 57

5 Zusammenfassung und Ausblick **63**

Literaturverzeichnis **65**

Abbildungsverzeichnis

1.1 Generalisierung einer Kleinstadt mit CartAGen im Maßstab 1:60000 [IGN] . . . 14

2.1 Spielplan Game of Scotland Yard[1] . . . 16
2.2 Parameterisierte Mapbox Static Images API Anfrage [Mapb] . . . 17
2.3 Breiten- und Längengrade [Wikc] . . . 20
2.4 Ellipsoid mit Äquatorradius a und Polradius c [Wikb] . . . 21
2.5 Projektionsfiguren [SI] . . . 22
2.6 Parameterisierung eines *LineStrings* l [Pel] . . . 24
2.7 Visualisierungsbeispiele einiger Qualitätskriterien des Spielplans. . . . 26

3.1 Visualisierung der Routen öffentlicher Verkehrsmittel in London [Mapb] . . . 27
3.2 Visualisierung der Ausführung des Douglas-Peucker Algorithmus [Wika] . . . 29
3.2 Visualisierung der Ausführung des Douglas-Peucker Algorithmus [Wika] . . . 30
3.3 Kleine Hausdorff Distanz δ_H trotz fehlender Ähnlichkeit der *LineStrings* [AG92] 33
3.4 *Free space* zweier *LineSegments* [Pel] . . . 34
3.5 *LineStrings* P, Q mit ihrem *free space* F_ε [Pel] . . . 35
3.6 Grenzen einer *free space* Zelle C_{ij}[AKW01] . . . 35
3.7 *Monotoner free space* R_ε [Pel] . . . 36
3.8 Eröffnung einer neuen vertikalen Passage durch Eintreten von $a_{ij} = b_{kj}$ [Pel] . . 38
3.9 Visualisierungsbeispiele von kritischen Werten für ε [Pel] . . . 38
3.10 Richtungsabhängigkeit der *Fréchet Distanz* δ_F . . . 40
3.11 Gleiche *modifizierte Fréchet Distanz* δ'_F trotz unterschiedlicher Ähnlichkeit . . . 41
3.12 *BoundingBox* B von zwei *LineStrings* P, Q . . . 42
3.13 Lokal ähnliche *LineStrings* P, Q . . . 43
3.14 Identifikation von P^Q und Q^P . . . 44
3.15 Einfluss der *Buffergröße* ε auf die Identifizierung der *lokalen Ähnlichkeiten*. . . . 46
3.16 Beispielhafter Verschmelzungsprozess . . . 48
3.17 Status der Qualitätskriterien . . . 52
3.18 Status der Qualitätskriterien . . . 54
3.19 Generierter Spielplan Stadtzentrum London . . . 56

4.1 England [Geo] . . . 57
4.2 Laufzeit des Verschmelzungsprozesses in Abhängigkeit der Anzahl von *LineStrings*. 58
4.3 Laufzeit des Bentley-Ottmann Algorithmus in Abhängigkeit der Anzahl von *LineStrings*. . . . 58
4.4 Im Spielplan enthaltene Stationen in Abhängigkeit des Zoomlevels des Kartenausschnitts . . . 59
4.5 Generierte Spielpläne für verschiedene Zoomlevel . . . 60
4.5 Generierte Spielpläne für verschiedene Zoomlevel . . . 61

Verzeichnis der Listings

2.1 Repräsentation einer Ampel durch ein Node [Foug] 18
2.2 Repräsentation einer Straße durch einen Way [Foul] 19
2.3 Repräsentation einer Busroute durch eine Relation [Fouh] 20

Verzeichnis der Algorithmen

3.1 Douglas-Peucker Algorithmus 31
3.2 Zur Lösung des Entscheidungsproblems: $\delta_F(P,Q) \leq \varepsilon$? 37
3.3 Berechnung der Fréchet Distanz $\delta_F(P,Q)$ 39
3.4 Modifizierte Fréchet Distanz $\delta'_F(P,Q)$ 41
3.5 Berechnung des Ähnlichkeitsgrads $\gamma(P,Q)$ 43
3.6 Identifikation lokaler Ähnlichkeiten $\mathcal{A}(P,Q)$ 45
3.7 Bentley-Ottmann Algorithmus 51
3.8 Verschmelzung überlappender Stationen p_0,p_1 53
3.9 Eliminierung von Sackgassen 54
3.10 Identifikation der größten Zusammenhangskomponente 55

1 Einleitung

Der Fokus dieser Arbeit liegt auf der automatischen Spielplan- und Graphgenerierung für das Strategiespiel *Scotland Yard*. Als Grundlage der Generierung dienen die von *OpenStreetMap* frei zur Verfügung gestellten geographischen Daten. Die folgenden Kapitel vermitteln ein einführendes Verständnis über den Aufbau des Spiels, sowie über die Notwendigkeit eines automatischen Spielplangenerierungsprozesses.

1.1 Motivation

Scotland Yard, das Spiel des Jahres 1983, ist seit seiner Einführung ein allseits beliebtes Brettspiel, das mittlerweile auch als digitale Variante erhältlich ist. Das Prinzip des Spiels basiert auf dem *Pursuit-Evasion* Problem, welches vereinfacht durch das Aufspüren einer Partei, durch eine andere beschrieben werden kann. Die erste Partei in *Scotland Yard* besteht lediglich aus *Mister X*, dessen Ziel es ist, unentdeckt zu flüchten. Das Ziel der zweiten Partei, bestehend aus mehreren Detektiven, ist das Einkesseln und Fangen des flüchtenden *Mister X*. Alle Detektive, sowie *Mister X* befinden sich zu jedem Zeitpunkt auf einer Station des Spiels. In der ersten und häufigsten Variante des Spiels können die Spieler zwischen verbundenen Stationen im Stadtzentrum von London mit verschiedenen Verkehrsmitteln reisen [Joc08].

Getrieben durch den Wunsch der Spielergemeinde entstanden neben dem Spielplan von London weitere Spielpläne realer Städte wie z.B. von der Stadt Potsdam[1]. Diese Spielplanerstellungen wurden manuell durchgeführt und bringen infolgedessen verschiedene Nachteile mit sich. Zum einen ist ein derartiger manueller Erstellungsprozess sehr zeitintensiv. Zum anderen sind trotz der beabsichtigten Bindung an die Realität, auf den manuell erstellten Plänen des Öfteren starke Abweichungen des Verlaufs der Verkehrsmittelrouten zu den realen Verkehrsmittelrouten zu erkennen. Trotz des Bedarfs an Generalisierung der Routen zur Reduzierung der Komplexität des Spielplans, lassen sich die meisten dieser Abweichungen nur schwer als Abstraktion zur Verbesserung des Spielgeschehens begründen.

Diese Arbeit umfasst die Automatisierung des Spielplanerstellungsprozesses für beliebige Städte dieser Welt. Dabei werden reale Daten von Routen öffentlicher Verkehrsmittel verwendet und nur so weit generalisiert, dass ein für die Spieler übersichtlicher Spielplan entsteht. Dadurch lässt sich ein guter Ausgleich zwischen der Realitätsnähe und der Einfachheit des Spielplans erreichen.

[1] http://www.coralnet.de/potsdam/potsdam.html

1.2 Verwandte Arbeiten

Die Spielplanerstellung lässt sich in großen Teilen auf das *map generalization* Problem zurückführen. Dabei handelt es sich um eine geeignete Abstraktionsfindung von Karten, die einem Betrachter ein Verständnis der abgebildeten realen Welt vermitteln soll [BW88]. Eine geographic information system (GIS) Plattform, die sich auf eine open-source Lösung für Generalisierungsalgorithmen und das damit verbundene *map generalization* Problem spezialisiert, ist CartAGen[2]. Durch die Komplexität einer automatischen Kartengeneralisierung sind mit der Zeit viele verschiedene Ansätze und Algorithmen entstanden, die versuchen dieses Problem zu lösen [SBB+14; TLD19]. Da für dieses Problem keine eindeutige Lösung existiert und diese Ansätze unterschiedliche Generalisierungsziele verfolgen, sind sie jedoch nur schwer miteinander zu vergleichen oder gar zu kombinieren [TLD19]. Die beabsichtigte Generalisierung und Verschmelzung von Netzwerken öffentlicher Verkehrsmittel zur Spielplanerstellung von *Scotland Yard* ist ein sehr spezieller Generalisierungsfall und daher nur beschränkt durch die bereits existierenden Ansätze zu erreichen. Aufgrund dieser Argumente ist die Entscheidung zu Gunsten eines selbst entwickelten Ansatzes gefallen, der sich aus einzelnen Algorithmen zusammensetzt, die sich in mehreren Domänen bewiesen haben und sich gut auf das vorliegende Generalisierungsproblem übertragen lassen.

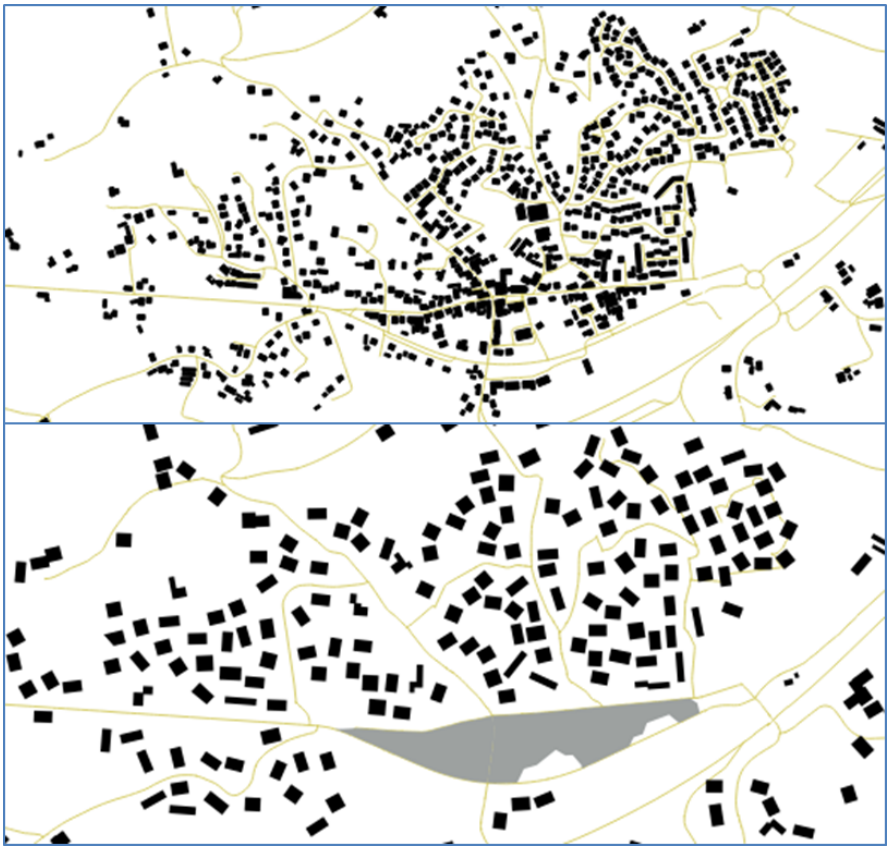


Abbildung 1.1: Generalisierung einer Kleinstadt mit CartAGen im Maßstab 1:60000 [IGN]

[2] https://ignf.github.io/CartAGen/

2 Grundlagen

2.1 Digitale Variante Scotland Yard

Um den generierten Spielplan bewerten und testen zu können, wird dieser im Anschluss des Generierungsprozesses mit seiner unterliegenden Graphstruktur an die Digitale Scotland Yard Variante *Game of Scotland Yard*[1] angebunden. Diese Variante unterstützt in ihrer ursprünglichen Form lediglich ein Spielplan des Stadtzentrums von London. Das Ziel des Spielers ist es, mit der Steuerung von 5 Detektiven den computergesteuerten *Mister X* zu umzingeln und schließlich einzufangen, indem ein Detektiv auf der Station landet, auf welcher *Mister X* steht.

Game of Scotland Yard[1]

Spielregeln

1. Alle Detektive sowie *Mister X* befinden sich zu jeder Zeit des Spiels auf einer nummerierten Station des Spielplans. Jede Spielfigur startet auf einer zuvor festgelegten Station. Stationen sind durch Taxi-, Bus- und U-Bahnrouten miteinander verbunden. In Abbildung 2.1 sind Taxirouten in der Farbe Gelb, Busrouten in Grün und U-Bahnrouten in Rot visualisiert.
2. *Mister X* hat keine Ticketbeschränkung. Jeder Detektiv ist zu Beginn im Besitz von folgenden Tickets:
   a) 10 Taxitickets
   b) 8 Bustickets
   c) 4 U-Bahntickets
3. Die Spielfiguren müssen in jeder Runde eine Station weiter reisen. Diese Reise ist jedoch nur dann möglich, wenn die Figur über das nötige Ticket verfügt und die Zielstation über eine Route mit der aktuellen Station verbunden ist. Jede Reise der Detektive wird mit dem entsprechenden Ticket bezahlt.

[1] https://sourceforge.net/projects/scotland-yard/

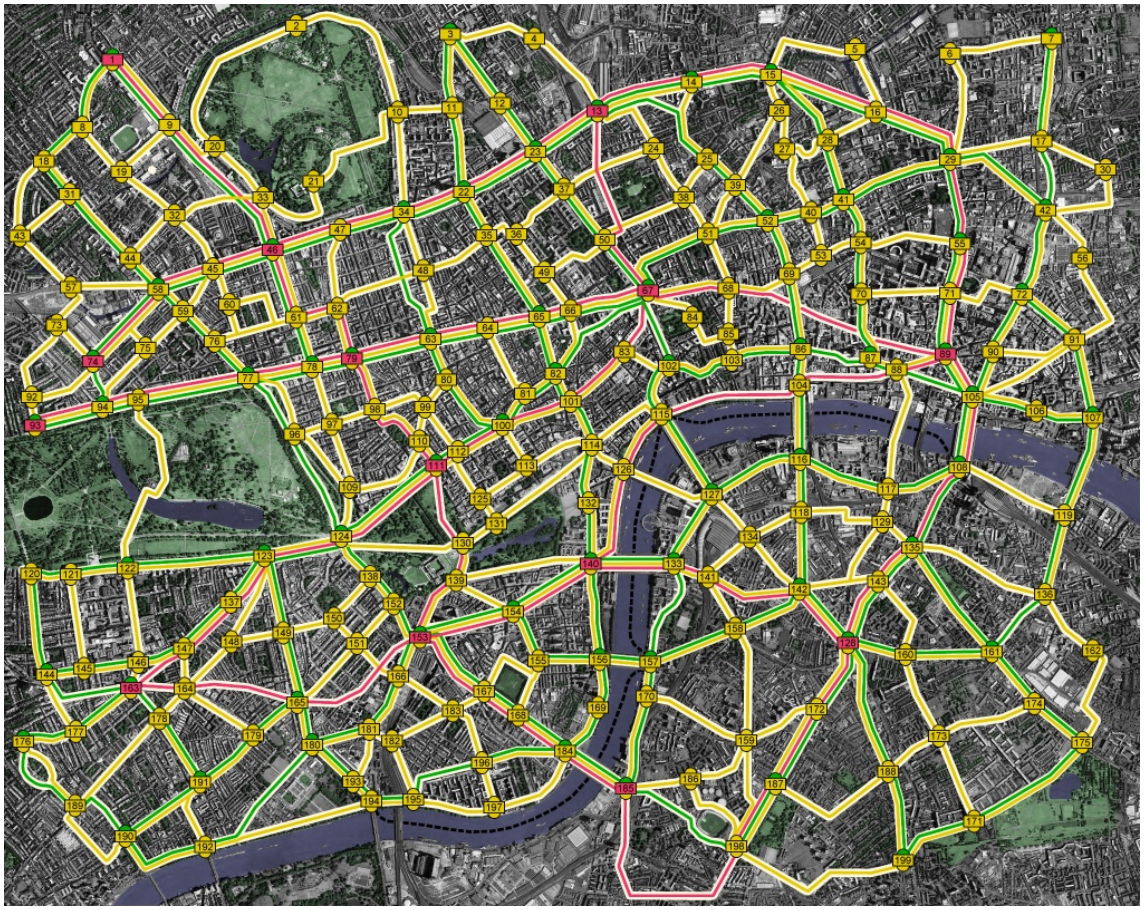


Abbildung 2.1: Spielplan Game of Scotland Yard[1]

Ablauf der Runden

1. *Mister X* startet jede Runde mit seinem Zug und veröffentlicht daraufhin das Transportmittel, das er für seine Reise genutzt hat. In den Runden 3, 8, 13, 18, 23 veröffentlicht er zusätzlich seine Position, um den Detektiven eine Chance zur erneuten Strategiebildung zu geben.
2. Der Spieler bewegt die Detektive 1-5 der Reihenfolge nach auf eine neue Station, sofern sie noch in Besitz der nötigen Tickets sind.

Endbedingungen

a) Einer der Detektive reist zu einer Station, auf der sich aktuell *Mister X* befindet. **(Sieg)**

b) Keiner der Detektive kann sich mehr bewegen, da sie alle ihre Tickets verbraucht haben. **(Niederlage)**

c) Keiner der Detektive kann sich mehr bewegen, da sie sich auf Stationen befinden, die nur mit Tickets eines Typs verlassen werden können, die sie bereits alle ausgegeben haben. **(Niederlage)**

Modifikation

Im Zuge der Implementierung dieser Arbeit wurde die Verwendung von Taxis als Transportmittel verworfen, da in der Realität keine Informationen über festgelegte Taxirouten vorliegen. Im Gegensatz dazu existieren jedoch reichlich Daten über Zugrouten, die als Ausgleich mit in das Spiel aufgenommen wurden.

2.2 Mapbox

Um dem Design des Spielplans in Abbildung 2.1 auf der vorherigen Seite zu entsprechen, sollten als Grundlage der zu erstellenden Spielpläne Satellitenbilder dienen, die Ausschnitte derselben Größe beliebiger Städte visualisieren. Diese Satellitenbilder werden durch die *Mapbox Static Images API*[2] bezogen. *Mapbox* ist ein Provider von open-source mapping Projekten und benutzerdefinierten Karten. Für das Kartenrendering nutzt *Mapbox* die *Web Mercator Projektion* (siehe Abschnitt 2.5.1 auf Seite 22) [Mapa].

```
GET https://api.mapbox.com/styles/v1/{username}/{style_id}/static/{overlay}/{lon},{lat},{zoom},{bearing},{pitch}|{bbox}|{auto}/{width}x{height}{@2x}
```

Abbildung 2.2: Parameterisierte Mapbox Static Images API Anfrage [Mapb]

2.3 OpenStreetMap

OpenStreetMap[3] (OSM) stellt editierbare Karten und die damit verbundenen Geoinformationen unter der *Open Data Commons Open Database Lizenz*[4] frei zur Verfügung [Foub; Fouc]. Seit der Gründung im Jahr 2004 durch Steve Coast erfreuen sich über 8 Millionen Nutzer[5] an OSM, die ihren Teil dazu beitragen das Projekt stetig zu verbessern. Durch ihre Mithilfe werden fehlende Daten ergänzt oder fehlerhafte Daten korrigiert [Foub; Fouk].

OSM-Daten können über mehrere Wege heruntergeladen werden und sind zumeist als XML formatierte .osm Dateien oder komprimierte .osm.pbf Dateien zu finden [Foud]. Da die *Planet.osm*[6] Datei, die alle verfügbaren OSM-Daten der Welt umfasst, im unkomprimierten OSM XML Format über 1552.8 GB[7] groß ist, wird empfohlen kleinere Ausschnitte dieser Datei von Drittanbietern wie GeoFabrik[8] zu beziehen [Fouj].

[2] https://docs.mapbox.com/api/maps/
[3] https://www.openstreetmap.org/
[4] https://opendatacommons.org/licenses/odbl/
[5] Stand 16.03.2022
[6] https://planet.openstreetmap.org/
[7] Stand 01.03.2022
[8] http://download.geofabrik.de/

Das OSM Projekt ist eine häufig genutzte und zuverlässige Alternative zu sonst meist kommerziellen Anbietern von Geodaten [Fouf]. Auch diese Arbeit basiert auf der Verwendung der Geodaten von OSM. Im Speziellen sind alle Informationen rund um öffentliche Verkehrsmittel, wie Bus-, Zug- und U-Bahnrouten, die sich innerhalb des *Mapbox* Kartenausschnitts befinden, von besonderer Relevanz. Um den Aufbau, Informationsgehalt und die Verwendbarkeit der OSM-Daten besser verstehen zu können, gibt die folgende Sektion eine Einführung in das OSM-Datenmodell.

OSM Datenmodell

Die Grundelemente aller OSM-Daten sind *Nodes*, *Ways* und *Relations*. Diese Elemente sind eindeutig definiert und können zusätzliche Attribute in Form von *tags* enthalten. Ein *tag* besteht aus einem Schlüssel und einem Wert [Foue].

Node

Ein *Node* gleicht einem geometrischen Punkt, der ein Objekt der Realwelt repräsentiert, oder als Bestandteil von *Ways* oder *Relations* genutzt wird. Definiert wird ein *Node* durch eine über alle Elemente hinweg eindeutige id und ein Koordinatenpaar k [Foug].

1. id ist eine 64-Bit Integer Zahl ≥ 1
2. k setzt sich aus einer Breitengrad- (lat) und einer Längengradangabe (lon) in der World Geodetic System 84 (WGS84) Standard Projektion (siehe Abschnitt 2.4.1 auf Seite 21) zusammen.
   - $lat \in [-90.0000000°, 90.000000°]$
   - $lon \in [-180.0000000°, 180.0000000°]$

Ein 32-Bit Integer für die id ist nicht mehr ausreichend. Am 10. Juli 2016 überschritt allein die Anzahl der Nodes die maximale Größe eines 32-Bit unsigned Integers von 4,294,967,295 (2^{32}-1) [Foua]. Heute existieren bereits 7.565.416.052 Nodes[9] in der OSM Datenbank [Fouk].

Listing 2.1 Repräsentation einer Ampel durch ein Node [Foug]

```
<node id="25496583" lat="51.5173639" lon="-0.140043">
    <tag k="highway" v="traffic_signals"/>
</node>
```

[9] Stand 16.03.2022

Way

Ein *Way* gleicht einer geometrischen Linie, die linear verlaufende Entitäten wie z.B. Straßen, Flüsse oder Schienen repräsentiert. Definiert wird ein *Way* durch eine eindeutige *id*, sowie einer geordneten Liste von *Nodes*, die den Verlauf des Wegs beschreibt. Referenziert werden die *Nodes* durch ihre eindeutige *id*, anhand welcher alle weiteren Informationen wie die Koordinaten und die Attribute über den Originalnode abgeleitet werden können. Voraussetzung dafür ist, dass alle referenzierten *Nodes* zuvor definiert werden [Foul].

Listing 2.2 Repräsentation einer Straße durch einen Way [Foul]

```
<way id="5090250">
    <nd ref="822403"/>
    ...
    <nd ref="333725776"/>
    <nd ref="823771"/>
    <tag k="highway" v="residential"/>
    <tag k="name" v="Clipstone Street"/>
    <tag k="oneway" v="yes"/>
  </way>
```

Relation

Eine *Relation* wird genutzt, um Beziehungen zwischen Elementen darzustellen. Sie wird mindestens durch ihre eindeutige *id* sowie eine geordnete Liste von Mitgliedern definiert. Diese Liste kann *Nodes*, *Ways* sowie andere *Relations* enthalten, die neben der Referenzierung durch ihre *id* eine Rolle enthalten können. Rollen erlauben es zu beschreiben, welche Funktionalitäten die Mitglieder innerhalb der *Relation* einnehmen [Fouh].

Routen öffentlicher Verkehrsmittel sind als *Relations* beschrieben und lassen sich an den folgenden *tags* erkennen [Foui].

- type=route
- route=$value$, mit $value \in$ {bus, subway, train, ...[10]}

Die Sequenzen der Mitgliederlisten folgen einem strikten, chronologischen Schema. Zunächst sind alle Haltestellen (*Nodes*) des Verkehrsmittels der Route aufgelistet. Nach der letzten Haltestelle folgen alle *Ways*, die die Haltestellen untereinander verbinden. Dabei sind jeweils der letzte *Node* und der erste *Node* von aufeinanderfolgenden *Ways* identisch [Foui]. Zu beachten gilt, dass alle Routen in der Realität gerichtet sind. Um den Spielplan nicht mit zusätzlichen Richtungsangaben zu überladen und dem Design aus Abbildung 2.1 auf Seite 16 konform zu bleiben, werden diese gerichteten Routen als Ungerichtete abstrahiert. Das bedeutet, dass alle Routen sowohl in die eine als auch in die andere Richtung genutzt werden können.

[10] Siehe https://wiki.openstreetmap.org/wiki/Key:route für eine detaillierte Liste aller Möglichkeiten

Listing 2.3 Repräsentation einer Busroute durch eine Relation [Fouh]

```
<relation id="10714086">
   <member type="node" ref="4005786039" role="platform"/>
   ...
   <member type="node" ref="4005786041" role="platform"/>
   <member type="way" ref="399615803" role=""/>
   ...
   <member type="way" ref="837587268" role=""/>
   <tag k="operator" v="Spillmann"/>
   <tag k="ref" v="551A"/>
   <tag k="type" v="route"/>
   <tag k="route" v="bus"/>
 </relation>
```

2.4 Koordinatenreferenzsysteme

Die Modellierung des korrekten Verlaufs von Routen setzt die korrekte Angabe von Positionen der unterliegenden *Nodes* voraus. In diesem Kapitel werden die Modelle der Positionsangaben eingeführt und erklärt, wie sich diese Positionsangaben der 3-dimensionalen Erde auf 2-dimensionale Karten übertragen lassen.

Als Basis der Positionsangaben dienen Koordinatenreferenzsysteme über 3-dimensionalen Formen, die sich der Form der Erde annähern. Die Einfachste dieser Formen beruht auf der Annahme einer Kugelform der Erde mit einheitlichem Radius. Das Koordinatensystem der Erdkugel wird über das Gradnetz, bestehend aus Breitenkreisen (Breitengrad) und Längenkreisen (Längengrad), definiert. Wie in Abbildung 2.3 zu sehen ist, verlaufen die Breitengrade parallel zum Äquator, wohingegen die Längengrade parallel zu einem Nullmeridian verlaufen [Cha08]. Ein Standard des Nullmeridians ist der *Greenwich Meridian* [MSP+15].

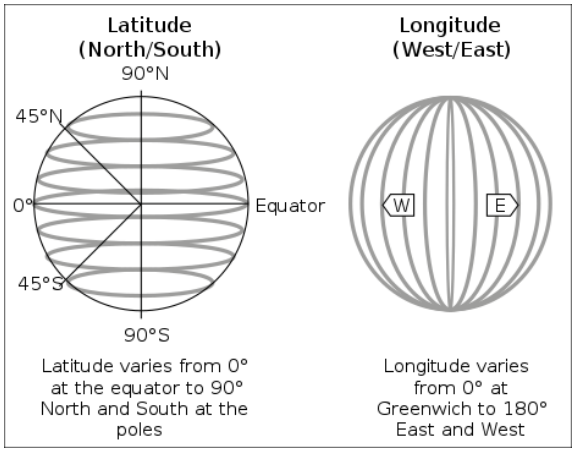


Abbildung 2.3: Breiten- und Längengrade [Wikc]

Die Annahme der Erde als Kugel entspricht jedoch nicht der Realität. Durch neue Möglichkeiten im Bereich der Vermessungstechnik fanden Forscher heraus, dass die Erde vielmehr der Form eines Ellipsoids ähnelt, das durch ein Äquator- und Polradius definiert ist (siehe Abbildung 2.4) [Sny87]. Im Folgenden wird ein Referenzsystem betrachtet, das ein solches Ellipsoid als Referenz verwendet.

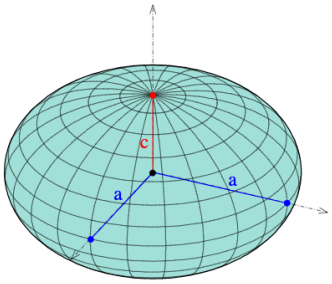


Abbildung 2.4: Ellipsoid mit Äquatorradius a und Polradius c [Wikb]

2.4.1 World Geodetic System 1984

Das World Geodetic System 1984 (WGS84) ist ein Referenzsystem, das auf einem Ellipsoid E mit einem überzogenen Gradnetz basiert. Darüber hinaus besteht es aus einem Geoid, das die Abweichungen der tatsächlichen Form der Erde zu der flachen Oberflächenform des vereinfachten Modells angibt [Age87]. Als Nullmeridian wird in diesem System der *IERS Reference Meridian*, der 102 Meter östlich des *Greenwich Meridians* liegt, verwendet [MSP+15]. Das Ellipsoid E wird mit folgenden Parametern definiert [Age87; BFUY14; Sny87]:

- Äquatorradius $a = 6.378.137$ Meter
- Polradius $b = 6.356.752$ Meter
- Exzentrizität $e = 8,1819190842622 \cdot 10^{-2}$

2.5 Kartenprojektionen

Um die 3-dimensionale Erde auf 2-dimensionalen Karten darstellen zu können, ist eine Abbildung $f : \mathbb{R}^3 \to \mathbb{R}^2$ notwendig. Die runde Erde wird dabei, wie in Abbildung 2.5 auf der nächsten Seite zu sehen ist, auf Figuren projiziert, die ohne Verzerrungen auf eine Ebene ausgerollt werden können. Alle existierenden Kartenprojektionen bewahren unterschiedliche Eigenschaften und erzeugen durch die Projektion auf eine andere Figur verschiedene Arten von unvermeidlichen Verzerrungen. Typischerweise werden die Kartenprojektionen durch die folgenden Arten von erhaltenden Eigenschaften klassifiziert [Sny87].

1. **Flächentreue Projektionen** garantieren, dass alle gleich großen Bereiche der Karte eine gleich große Fläche des Models der Erde abdecken.
2. **Formtreue Projektionen** erhalten die relativen lokalen Winkel für nahezu alle Punkte auf der Karte.

3. **Streckentreue Projektionen** bilden eine oder mehrere Linien korrekt ab.

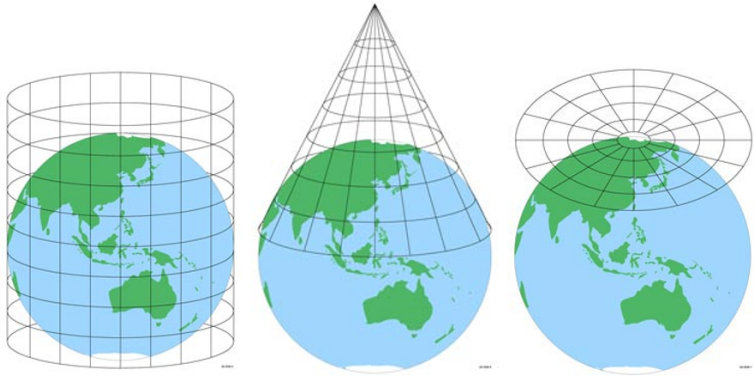


Abbildung 2.5: Projektionsfiguren [SI]

2.5.1 Web Mercator Projektion

Die *Web Mercator Projektion*, auch *Spherical Mercator* oder *Pseudo Mercator* genannt, ist eine Variante der *Mercator Projektion*. Bei dieser Projektion werden die WGS84 Koordinaten projiziert als wären sie auf einer Kugel mit einheitlichem Radius a, statt auf einem Ellipsoid mit Radien a und c definiert worden. Durch diese vereinfachte Annahme lassen sich zwar keine erhaltenden Eigenschaften erreichen, dafür lässt sich die Projektion im Vergleich jedoch schnell berechnen. Die Koordinaten (lat, lon) der OSM-Daten in der WGS84 Standard Projektion lassen sich dabei wie folgt auf die ebenen Koordinaten (x,y) der *Web Mercator Karte* berechnen [BFUY14]:

1. Zunächst sind die Koordinatenangaben von Grad in Radiant zu ändern.

$$f_0 : [-90°, 90°] \times [-180°, 180°] \rightarrow \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [-\pi, \pi],$$

$$(lat, lon) \mapsto \left(lat\frac{\pi}{180°}, lon\frac{\pi}{180°}\right)$$

2. Anschließend lässt sich (x, y) durch folgende Funktion f_1 berechnen.

$$f_1 : \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [-\pi, \pi] \rightarrow \mathbb{R}^2,\ (lat, lon) \mapsto (x, y)\ mit$$

$$x = a \cdot lon$$

$$y = a \ln\left[\tan\left(\frac{\pi}{4} + \frac{lat}{2}\right)\right]$$

Durch eine lineare Transformation $t : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ lassen sich daraufhin alle Positionen (p_x, p_y) der Koordinaten (x, y) auf dem Bildausschnitt der *Mapbox* Karte ermitteln und anzeigen. Koordinaten und damit die OSM-Elemente, die nicht im Bildausschnitt enthalten sind, haben für die weitere Bearbeitung keine Bedeutung und werden verworfen.

2.5.2 Mercator Projektion

Eine genauere Projektion ist die von Gerardus Mercator im Jahr 1569 vorgestellte *Mercator Projektion*. Diese ist eine formtreue Zylinderprojektion, dessen erhaltende Eigenschaft jedoch nur auf Kosten des Berechnungsaufwands und anderer Verzerrungen zu erreichen ist. Besonders die Flächenverzerrung, die sich in Richtung der Pole verstärkt, ist dabei gut zu erkennen. Die Koordinaten (lat, lon) der OSM-Daten in der WGS84 Standard Projektion lassen sich dabei wie folgt auf die ebenen Koordinaten (x,y) der Mercator Karte abbilden:

1. Die Koordinatenangaben lassen sich korrespondierend zu der *Web Mercator Projektion* mit der Funktion f_0 von Grad in Radiant ändern.
2. Anschließend lässt sich (x, y) durch folgende Funktion berechnen.

$$f_2 : \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [-\pi, \pi] \to \mathbb{R}^2,\ (lat, lon) \mapsto (x, y)\ mit$$

$$x = a \cdot lon$$

$$y = a \ln \left[\tan\left(\frac{\Pi}{4} + \frac{lat}{2}\right) \left(\frac{1 - e \sin lat}{1 + e \sin lat}\right)^{\frac{e}{2}} \right]$$

Wobei a den Äquatorradius und e die Exzentrität des Ellipsoids des zugrunde liegenden Referenzsystems definieren. Die x-Achse der projizierten Karte liegt dabei entlang des Äquators und die y-Achse entlang des Nullmeridians [Sny87].

2.6 Internes Datenmodell

Das interne Datenmodell besteht weitestgehend aus den von der *JGraphT*[11] und der *JTS Topology Suite*[12] library bereitgestellten Datenstrukturen. JGraphT ist eine Java library für Graphtheorie-Datenstrukturen und -Algorithmen, mit dessen Hilfe der unterliegende Graph G des Spielplans erzeugt wird.

- **Graph G** = (V, E) mit Knoten V und Kanten E bildet die Konnektivität der Stationen V ab, auf denen sich die Spielfiguren befinden können.

Die JTS Topolgy Suite ist eine Java library, die Strukturen und Funktionalitäten zur Abbildung und Manipulation von Vektorgeometrien bereitstellt. Die OSM-Elemente werden intern durch die folgenden Strukturen der JTS Topolgy Suite abgebildet:

- **Punkt $\mathbf{p_i}$** $\in \mathbb{R}^2$ bildet die Position eines *Nodes i* auf der *Mapbox* Karte ab.
- **LineSegment s** : $(p_i, p_j) \to \mathbb{R}^2$ ist eine lineare Interpolation über zwei Punkten (p_i, p_j) mit $p_i \neq p_j$. Wenn im Folgenden ein *LineSegment* durch $s = (p_i, p_j)$ definiert wird, dann wird stets von der Interpolation über diese beiden Punkte gesprochen.

[11] https://jgrapht.org/
[12] https://github.com/locationtech/jts

- **LineString l** : $(p_0, \ldots, p_n) \rightarrow \mathbb{R}^2$ ist eine Abbildung die durch die lineare Interpolation über alle konsekutiven Punkte der Sequenz $(p_0, \ldots, p_n)$ mit $n > 0$ und $p_i \neq p_{i+1}$, bestimmt ist. Wenn im Folgenden ein *LineString* durch $l = (p_0, \ldots, p_n)$ definiert wird, ist stets die Rede von einer linearen Interpolation über allen, konsekutiven Punkten. Ein *LineString* $l = (p_0, \ldots, p_n)$ besteht dabei aus n *LineSegments* $s_i = (p_i, p_{i+1})$ mit $i \in \{0, \ldots, n-1\}$.

  Ein *LineString* $l = (p_0, \ldots, p_n)$ lässt sich durch ein $x \in \mathbb{R}, 0 \leq x \leq n$ parametrisieren, sodass $l(x)$ eine Position auf dem kontinuierlichen *LineString* angibt. Dabei ist $l(0) = p_0$ und $l(n) = p_n$ (Beispiel in Abbildung 2.6).

  Ein *LineString* $l' = (l'_0, \ldots, l'_k)$ wird *SubLineString* eines *LineStrings* $l = (l_0, \ldots, l_n)$ genannt, wenn $\exists\, x, y \in \mathbb{R}$ mit $0 \leq x < y \leq n : \; l'_0 = l(x), \;\; l'_k = l(y)$ und l' verläuft zwischen l'_0 und l'_k identisch wie l zwischen $l(x)$ und $l(y)$.

  Ein *Way* w der durch eine Sequenz $(n_1, \ldots, n_m)$ von *Nodes* n_i definiert ist, wird durch einen *LineString* $l_w = (p_{n_1}, \ldots, p_{n_m})$ abgebildet.

  Eine Routen *Relation* r, die aus einer Sequenz $(w_1, \ldots, w_h)$ von *Ways* w_k besteht wird durch einen *LineString* $l_r = (l_{w_1}, \ldots, l_{w_h})$ abgebildet, der die einzelnen *LineStrings* l_{w_k} konkateniert. Die Route bzw. der *LineString* der diese abbildet, liegt in gerichteter Form vor. Wenn im Folgenden von einer Route gesprochen wird, dann ist diese stets als ihre Abbildung durch ein *LineString* zu verstehen.

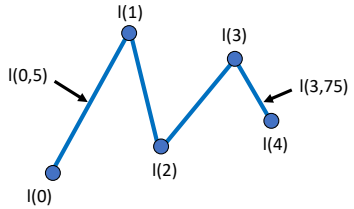


Abbildung 2.6: Parameterisierung eines *LineStrings* l [Pel]

2.7 Qualitätskriterien des Spielplans

Dieser Abschnitt beschreibt welche Kriterien der zu erzeugende Spielplan erfüllen sollte, um dem Spieler ein übersichtliches und intuitives Spielgeschehen zu ermöglichen. Außerdem werden die Auswirkungen dieser Kriterien auf die Eigenschaften des unterliegenden Konnektivitätsgraphen G beleuchtet. Da in der Realität alle Routen gerichtet vorliegen, wird zunächst von einem gerichteten Graphen ($G = (V, E)$ mit $E \subseteq V \times V$) ausgegangen. Der Spielplan hat die folgenden Kriterien zu erfüllen:

1. **Schleifenfrei**: Alle Spielfiguren müssen sich in ihrem Zug zu einer anderen Station bewegen.

   $\Rightarrow G$ ist schleifenfrei $\Leftrightarrow \forall (u, v) \in E : u \neq v$

2. **Richtungslosigkeit**: Wie in Abschnitt 2.3 auf Seite 19 beschrieben sollten alle Routen auf dem Spielplan in beide Richtungen genutzt werden können.

   $\Rightarrow G$ ist ungerichtet $\Leftrightarrow E \subseteq \begin{pmatrix} V \\ 2 \end{pmatrix}$

3. **Keine Sackgassen**: Es sollten keine Sackgassen auf dem Spielplan existieren, da es in diesen keine Auswege für *Mister X* gibt.

   $\Rightarrow \forall u \in V : \exists \{u, v\}, \{u, w\} \in E$ mit $u \neq v \neq w$

4. **Zusammenhängend**: Alle Stationen müssen durch mindestens einen Pfad miteinander verbunden sein, da nur die *Detektive*, die zu Beginn auf derselben Zusammenhangskomponente wie *Mister X* starten, diesen auch fangen können.

   $\Rightarrow G$ ist zusammenhängend $\Leftrightarrow \forall u, v \in V : \exists$ Pfad $u, \ldots, v$

5. **Verschiedene Routen**: Die Stationen werden durch Routen von verschiedenen öffentlichen Verkehrsmitteln (Typ) verbunden.

   $\Rightarrow G = (V, E, \gamma)$ mit $\gamma : E \rightarrow \{\text{Busticket, Zugticket, U-Bahnticket}\}$ ist gewichtet (Gewichtung entspricht den Ticketkosten der Verbindungen).

6. **Keine Mehrfachverbindungen des gleichen Typs**: Es sollte maximal eine Route des gleichen Typs zwischen zwei Stationen visualisiert werden.

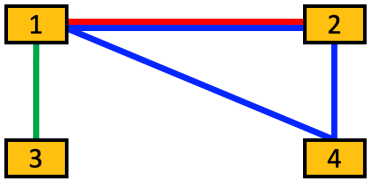
7. **Zusammenfassung von ähnlichen Routen**: Routen die sich sehr ähnlich sind, sollten zusammengefasst werden.

8. **Positionen der Stationen**: Die Stationen sind in einer Weise zu platzieren, die es dem Spieler erlaubt, eindeutig zu erkennen welche Stationen miteinander verbunden sind. Dabei sollten Stationen insbesondere an Kreuzungspunkten der Routen platziert werden, da die Konnektivität an diesen Stellen sonst nur schwer bis gar nicht erkennbar ist. Des Weiteren sind auch an Routenenden Stationen zu platzieren, damit keine Route ins Leere läuft.

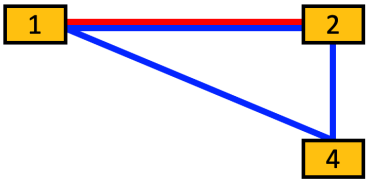
9. **Ausreichender Abstand der Stationen**: Alle Stationen sollten einen gewissen Grundabstand halten. Anderenfalls könnten sich diese auf dem visualisierten Spielplan überlappen.

(a) Verletzung Kriterium 1

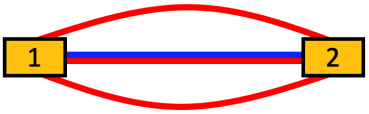
(b) Erfüllung Kriterium 1



(c) Verletzung Kriterium 3



(d) Erfüllung Kriterium 3

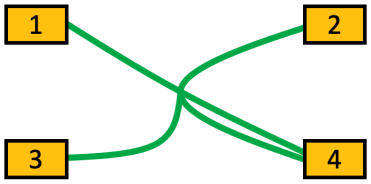


(e) Verletzung Kriterium 6

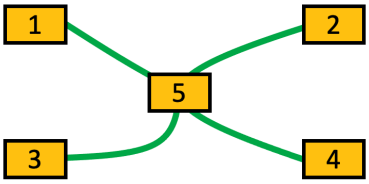
(f) Erfüllung Kriterium 6

(g) Verletzung Kriterium 7

(h) Erfüllung Kriterium 7



(i) Verletzung Kriterium 8



(j) Erfüllung Kriterium 8

(k) Verletzung Kriterium 8

(l) Erfüllung Kriterium 8

(m) Verletzung Kriterium 9

(n) Erfüllung Kriterium 9

Abbildung 2.7: Visualisierungsbeispiele einiger Qualitätskriterien des Spielplans.

3 Kartengenerierungsprozess

Großstädte besitzen ein sehr dichtes Netzwerk öffentlicher Verkehrsmittel. Viele dieser Routen bzw. *LineStrings*, die diese repräsentieren, lassen wie in Abbildung 3.1 zu sehen ist, deutliche Ähnlichkeiten in ihrem Verlauf erkennen. Es ist offensichtlich, dass kein übersichtlicher Spielplan auf Basis von Abbildung 3.1 erzeugt werden kann, der die in Abschnitt 2.7 auf Seite 24 genannten Qualitätskriterien erfüllt, ohne eine Form von Generalisierung vorzunehmen. Dieses Kapitel thematisiert den Kartengenerierungs- und Generalisierungsprozess, der alle Qualitätskriterien der zu erstellenden Karte sicherstellt. Zur Vermittlung eines grundlegenden Verständnisses werden dabei die Ansätze und die verwendeten Algorithmen der einzelnen Schritte erläutert. Die exakten Details der begleitenden Implementierung können dieser entnommen werden.

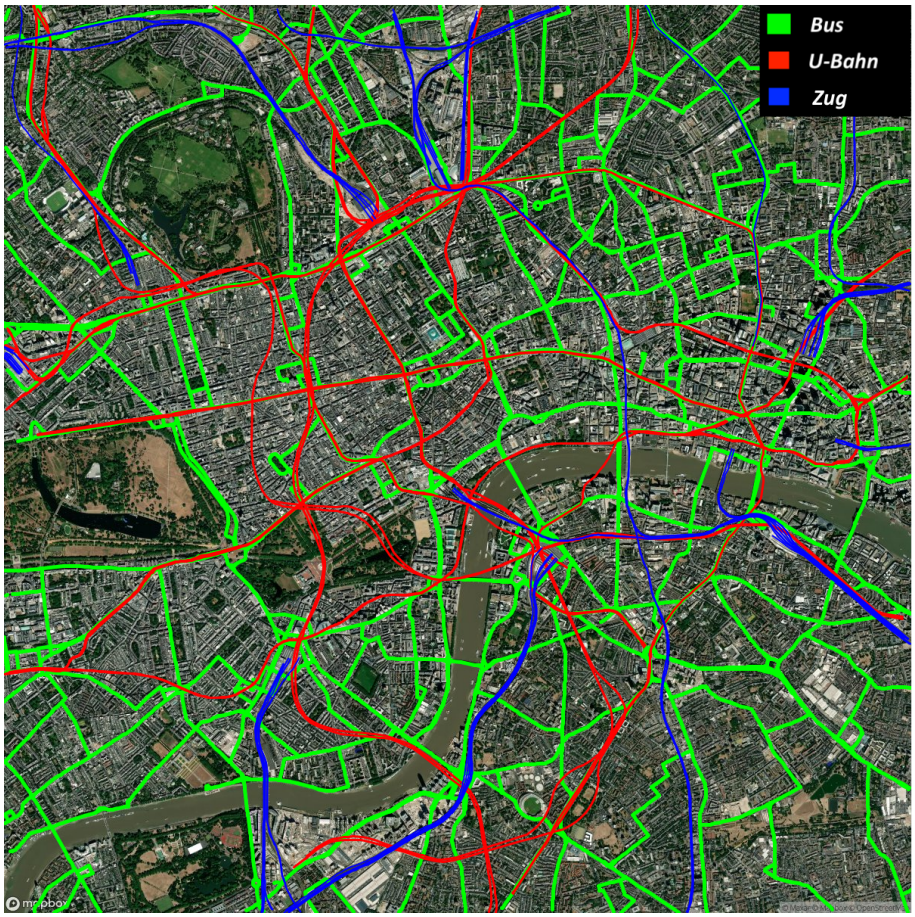


Abbildung 3.1: Visualisierung der Routen öffentlicher Verkehrsmittel in London [Mapb]

3.1 Kompression

Routen mit ihren Wegen sind in den detaillierten OSM-Daten durch sehr viele *Nodes* beschrieben. Diese Beschreibung enthält dabei oft eine Vielzahl an *Nodes*, die zur Visualisierung des generellen Verlaufs dieser Routen auf dem Kartenausschnitt nicht benötigt werden. Da die Laufzeiten der einzelnen Schritte mit ihren Algorithmen vor allem von der Anzahl der *Nodes* in den Routen abhängen, ist es von großer Bedeutung, die Nodedichte konsistent zu reduzieren. Im folgenden Abschnitt wird ein Algorithmus vorgestellt, der dieses Ziel erreicht und die Erhaltung des generellen Verlaufs der Routen sicherstellt.

3.1.1 Douglas-Peucker Algorithmus

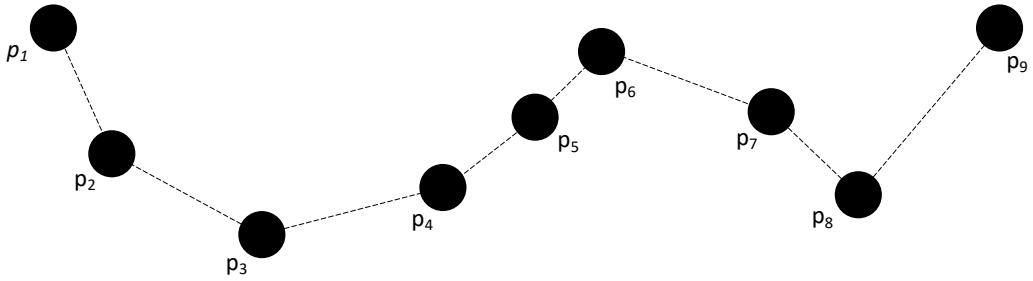
Der Douglas-Peucker Algorithmus ist ein erfolgreicher Algorithmus der *map generalization*, wobei Linien durch Entfernen von Punkten vereinfacht werden und dennoch ihre generelle Form beibehalten. Gemäß dem *Teile-und-herrsche-Ansatz* wird die ursprüngliche Linie aufgeteilt und rekursiv vereinfacht.

Unter Bezugnahme des internen Datenmodells aus Abschnitt 2.6 auf Seite 23 sei der ursprüngliche *LineString* $l = (p_1, \ldots, p_n)$ mit $n > 1$ und eine Toleranz $\varepsilon > 0$ gegeben. Die Toleranz bestimmt dabei den Grad der Vereinfachung des *LineStrings*. Die Funktionsweise des Douglas-Peucker Algorithmus lässt sich generell durch die folgenden Schritte beschreiben:

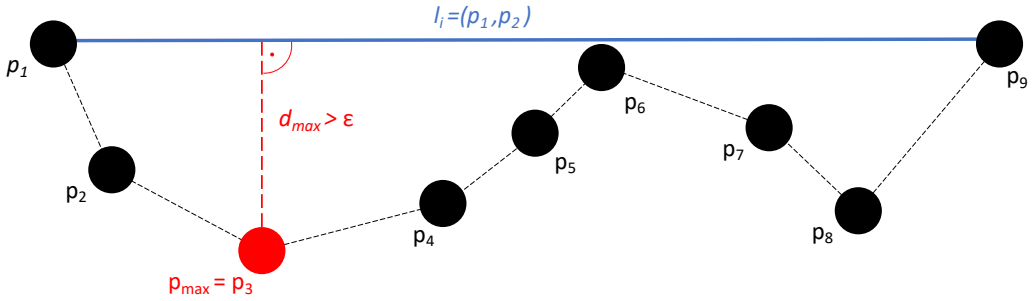
1. Bilde eine lineare interpolation $l_i = (p_1, p_n)$ über den ersten und letzten Punkt des gegebenen *LineStrings*
2. Suche unter allen Punkten $p_2, \ldots, p_{n-1}$ den Punkt p_{max} mit dem größten orthogonalen Abstand d_{max} zu l_i
3. a) $d_{max} \leq \varepsilon \Rightarrow$ entferne $p_2, \ldots, p_{n-1}$ aus l

   b) $d_{max} > \varepsilon \Rightarrow$ starte jeweils an 1. mit den *LineStrings* $l_a = (p_1, \ldots, p_{max})$ und $l_b = (p_{max}, \ldots, p_n)$

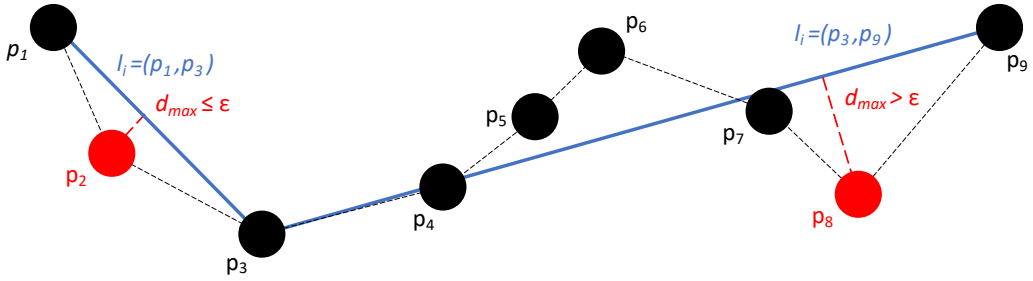
Der *LineString* l liegt nach der Terminierung des Algorithmus in vereinfachter Form vor. Dabei sind die enthaltenen Punkte eine Teilmenge der Punkte des ursprünglich gegebenen *LineStrings*. Alle Punkte, die während dem Algorithmus entfernt wurden, haben zu l einen Abstand größer ε [DP73].



(a) Gegeben $l = (p_1, p_2, p_3, p_4, p_5, p_5, p_6, p_7, p_8, p_9)$

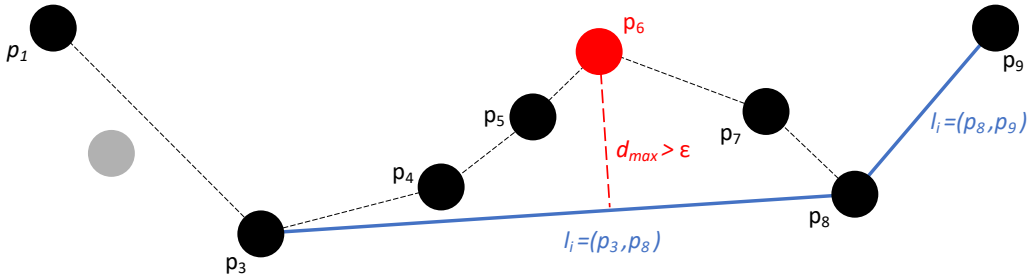


(b) Es folgt ein Rekursiver Aufruf mit (p_1, p_2, p_3) und $(p_3, p_4, p_5, p_5, p_6, p_7, p_8, p_9)$

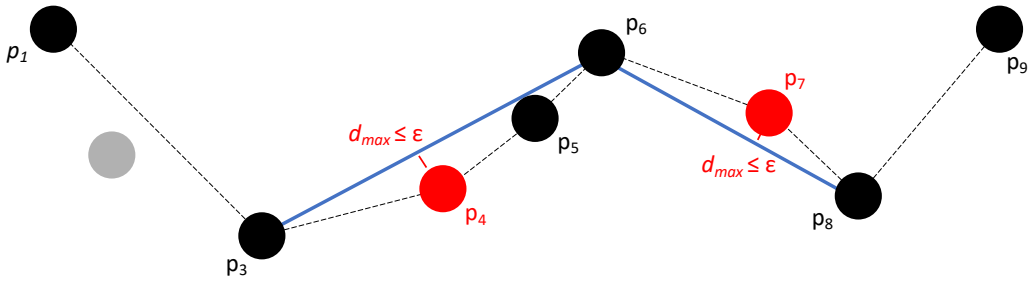


(c) Entferne p_2 und rekursiver Aufruf mit $(p_3, p_4, p_5, p_5, p_6, p_7, p_8)$ und (p_8, p_9)

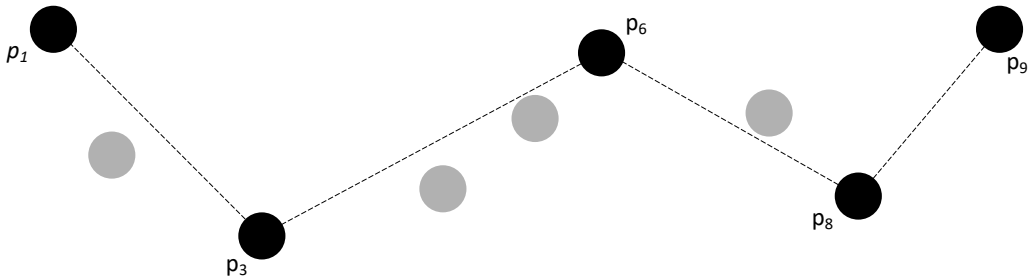
Abbildung 3.2: Visualisierung der Ausführung des Douglas-Peucker Algorithmus [Wika]



(d) Rekursiver Aufruf mit (p_3, p_4, p_5, p_6) und (p_6, p_7, p_8)



(e) Entferne p_4, p_5, p_7



(f) Approximierter *LineString* $l = (p_1, p_3, p_6, p_8, p_9)$

Abbildung 3.2: Visualisierung der Ausführung des Douglas-Peucker Algorithmus [Wika]

Algorithmus 3.1 Douglas-Peucker Algorithmus

procedure DOUGLAS-PEUCKER($l = (p_1, \ldots, p_n)$,$\varepsilon > 0$)
  $l_i \leftarrow (p_1, p_n)$
  $d_{max} \leftarrow 0$
  $p_{max} \leftarrow$ null
  for all $p_i \in \{p_2, \ldots, p_{n-1}\}$ **do**
    $d \leftarrow$ORTHOGONALDISTANZ(l_i, p_i)
    if $d > d_{max}$ **then**
      $d_{max} \leftarrow d$
      $p_{max} \leftarrow p_i$
    end if
  end for
  if $d_{max} > \varepsilon$ **then**
    $l_{links} \leftarrow$ DOUGLAS-PEUCKER($(p_1, \ldots, p_{max})$
    $l_{rechts} \leftarrow$ DOUGLAS-PEUCKER($(p_{max}, \ldots, p_n)$
    return KONKATENIERE(l_{links}, l_{rechts})
  else
    return l_i
  end if
end procedure

Komplexitätsanalyse

Zunächst wird das worst-case Szenario des Algorithmus untersucht. Hierbei werden die *LineStrings* $l = (p_1, \ldots, p_n)$ der einzelnen Funktionsaufrufe so geteilt, dass ein *LineString* maximaler Größe ($|l_{max}| = n - 1$) und ein *LineString* minimaler Größe ($|l_{min}| = 2$) resultieren. Der entscheidende Punkt der Komplexität ist die Anzahl der Aufrufe der Berechnung der orthogonalen Distanz. Die folgende Rekursionsgleichung gibt die Anzahl dieser Aufrufe in Abhängigkeit der Anzahl der Punkte n des übergegebenen *LineStrings* im worst-case an:

$$T(n) = \begin{cases} 0 & , n = 2 \\ n - 2 + T(2) + T(n-1) & , n > 2 \end{cases}$$

Daraus lässt sich die worst-case Laufzeit in $O(n^2)$ ableiten. Im Optimalfall liegt p_{max} in jedem Funktionsaufruf in der Mitte der *LineStrings*. Dadurch sind die Parameter der beiden neuen Rekursionsaufrufe in etwa gleich groß. Die Anzahl der Distanzberechnungen wird in diesem Fall durch die folgende Rekursionsgleichung definiert:

$$T(n) = \begin{cases} 0 & , n = 2 \\ n - 2 + T(\lfloor \frac{n+1}{2} \rfloor + T(\lceil \frac{n+1}{2} \rceil)) & , n > 2 \end{cases}$$

Durch diese Rekursionsgleichung ergibt sich eine best-case Laufzeit in $O(n \log_2 n)$ [HS92].

3.2 Verschmelzung ähnlicher Routen

Dieser Abschnitt erläutert den Verschmelzungsprozess von ähnlichen Routen und zielt damit auf die Erfüllung des Qualitätskriteriums *"7. Zusammenfassung ähnlicher Routen"* des zu erstellenden Spielplans ab. Um entscheiden zu können, welche Routen miteinander verschmolzen werden sollten, gilt es zunächst, den abstrakten Begriff der *Ähnlichkeit* zu konkretisieren.

Das Problem der Bestimmung der Ähnlichkeit von Routen lässt sich auf ein zentrales Problem im Bereich der Computer Vision zurückführen. Dieses Problem liegt in der Erkennung von Formen in Bildern. Die Formen sind dabei durch ihre Grenzen definiert, die als eine Menge von *LineStrings* im 2-dimensionalen Raum verstanden werden können. Alle Vergleichsfunktionen zur Erkennung der Formen basieren dabei auf einer Metrik, die die Distanz der *LineStrings* zur weiteren Berechnung definiert.

Im Folgenden werden zunächst zwei bekannte Metriken vorgestellt, die von Vergleichsfunktionen zur Bestimmung der Ähnlichkeit von Formen genutzt werden können [HKR93]. Diese Metriken gehen jedoch mit gewissen Problemen in Bezug auf die gewählte Abstraktion der Routen einher, die es daraufhin ab Abschnitt 3.2.3 auf Seite 39 durch Adaptionen zu lösen gilt.

3.2.1 Hausdorff Metrik

Die *Hausdorff Metrik* wurde erstmals 1914 von Felix Hausdorff in seinem Buch *Grundzüge der Mengenlehre* vorgestellt [Hau20]. Seien gemäß dem internen Datenmodell zwei *LineStrings* $l = (p_1, \ldots, p_n)$ und $l' = (p'_1, \ldots, p'_m)$ gegeben. Die folgende Berechnung nach [HKR93] der Hausdorff Distanz δ_H der beiden *LineStrings* basiert auf ihren Punktmengen $L = \{p_1, \ldots, p_n\}$ und $L' = \{p'_1, \ldots, p'_m\}$:

$$\delta_H(L, L') = \max(\tilde{\delta_H}(L, L'), \tilde{\delta_H}(L', L))$$

mit

$$\tilde{\delta_H}(L, L') = \max_{p \in L} \min_{p' \in L'} \|p - p'\|$$

wobei $\|\|$ eine Norm der Punkteebene ist. Häufig wird dafür die *Euklidische Norm* $\|\|_2$ verwendet. Vereinfacht ausgedrückt, lässt sich die *Hausdorff Distanz* bestimmen, indem man für jeden Punkt der beiden Mengen den nähesten Punkt der jeweils andere Menge bestimmt und anschließend die maximale Distanz dieser Punktpaare ermittelt. Der Trivialfall der Laufzeit zur Berechnung der *Hausdorff Distanz* in $O(nm)$ kann wie in [ABB95] gezeigt wurde zu einer Laufzeit in $O((n+m)\log(n+m))$ verbessert werden.

Wobei die *Hausdorff Metrik* in den meisten Fällen ein passendes Maß zur Grundlage der Vergleichsberechnung von *LineStrings* ist, gilt es zu beachten, dass Fälle existieren in welchen die Metrik versagt. Die *LineStrings* in Abbildung 3.3 auf der nächsten Seite haben beispielsweise trotz eines sehr geringen Grades an Ähnlichkeit eine kleine *Hausdorff Distanz* δ_H. Das ist auf die fehlende Beachtung des Verlaufs der *LineStrings* bei der Berechnung zurückzuführen [AG92]. Da der Verlauf der zu verschmelzenden Routen jedoch eine wichtige Rolle spielt, ist eine Metrik als Grundlage für

die Vergleichsberechnung der Linien zu wählen, die den Verlauf der Routen betrachtet. Die von Maurice Fréchet definierte *Fréchet Metrik* erfüllt dieses Kriterium und erweist sich damit als eine geeignete Grundlage für die weiteren Vergleichsoperationen [Fré06].

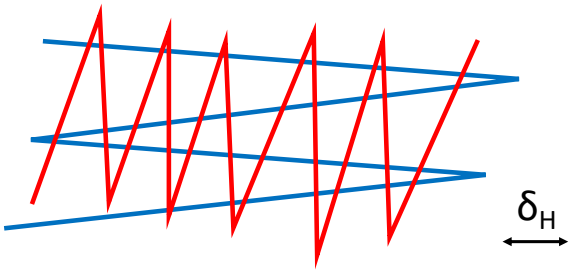


Abbildung 3.3: Kleine Hausdorff Distanz δ_H trotz fehlender Ähnlichkeit der *LineStrings* [AG92]

3.2.2 Fréchet Metrik

Dieses Kapitel gibt zunächst eine Einführung in die *Fréchet Metrik*, wie sie in [ABB95; AG95] vorgestellt wurde. Anschließend wird ein bestehender Konflikt der *Fréchet Metrik* mit der gewählten Abstraktion der Routen dargestellt, der durch die in Abschnitt 3.2.3 auf Seite 39 eingeführte Adaption der *Fréchet Metrik* gelöst wird.

Für eine Illustration der *Fréchet Metrik* sorgt das folgende Beispiel: Seien zu Beginn ein Hund und sein Besitzer auf dem Start zweier Routen platziert. Beide können ihre Routen unabhängig voneinander ablaufen, bis sie das Routenende erreicht haben. Dabei dürfen sie sich jedoch nur in die Richtung der Route bewegen und nicht zurück gehen. Die *Fréchet Distanz* beschreibt bei diesem Beispiel die kleinste mögliche Leine, die benötigt wird, um den Besitzer mit seinem Hund während des gesamten Prozesses zu verbinden [ABB95].

Zur Erinnerung: Ein *LineString* $l = (p_0, \ldots, p_n)$ lässt sich durch ein $x \in \mathbb{R}, 0 \leq x \leq n$ parametrisieren, sodass $l(x)$ eine Position auf dem kontinuierlichen *LineString* angibt. Dabei ist $l(0) = p_0$ und $l(n) = p_n$ (Beispiel in Abbildung 2.6 auf Seite 24).

Formeller seien nun zwei *LineStrings* $P = (p_0, \ldots, p_n)$ und $Q = (q_0, \ldots, q_m)$ für die Route des Mannes (P) und des Hundes (Q) gegeben. Das Ablaufen der jeweiligen Routen des Hundes und des Besitzers lassen sich in Abhängigkeit der Zeit $t \in \mathbb{R}$, $0 \leq t \leq 1$ durch kontinuierliche und monoton steigende Funktionen $\alpha(t)$ und $\beta(t)$ beschreiben, wobei $\alpha(0) = 0$, $\alpha(1) = n$, $\beta(0) = 0$ und $\beta(1) = m$ sind. Die Position des Mannes auf der Route P in Abhängigkeit der Zeit wird durch $P(\alpha(t))$ bestimmt, während die Position des Hundes auf Route Q durch $Q(\beta(t))$ bestimmt wird. Daraus lässt sich schließen, dass die Positionen des Mannes und des Hundes von der Definition der Funktionen α und β abhängen. Die Berechnung der *Frechet Distanz* lässt sich dadurch auf das Finden der Funktionen α und β zurückführen, die die maximale Distanz zwischen den Positionen des Mannes und des Hundes minimieren. Die genaue Definition der *Fréchet Distanz* zweier *LineStrings* P und Q lautet [AG95]:

$$\delta_F(P, Q) = \min_{\substack{\alpha:[0,1]\to[0,n] \\ \beta:[0,1]\to[0,m]}} \{ \max_{t\in[0,1]} \|P(\alpha(t)) - Q(\beta(t))\| \}$$

wobei $|||$ eine Norm der Punktebene ist, für die in dieser Arbeit die *Euklidische Norm* $|||_2$ gewählt wurde. Bevor die tatsächliche Berechnung der *Fréchet Distanz* δ_F beschrieben wird, gilt es als Vorbereitung zunächst die Lösung des *Entscheidungsproblems* zu betrachten, das eine vereinfachte Variante des Berechnungsproblems darstellt [AG95].

Das Entscheidungsproblem

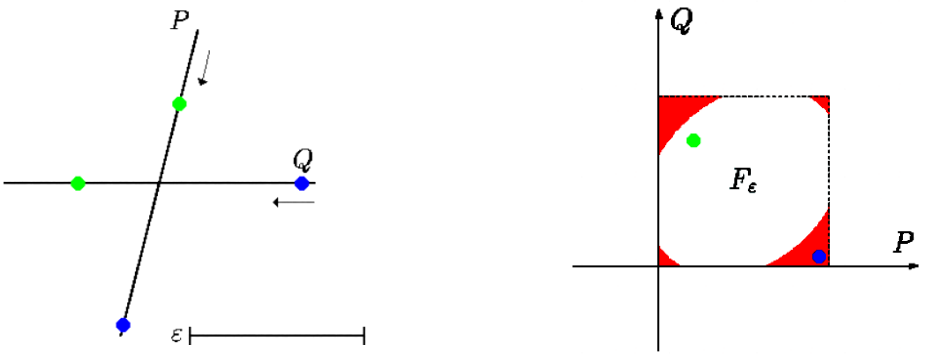
Gegeben: *LineStrings* P, Q und eine Toleranz $\varepsilon \geq 0$

Entscheide: ob $\delta_F(P, Q) \leq \varepsilon$

Zur Einführung des *Entscheidungsproblem* wurde zunächst der einfachste mögliche Fall für die gegebenen *LineStrings* P und Q betrachtet. Nach der Definition in Abschnitt 2.6 auf Seite 23 sind das $P = (p_0, p_1)$ und $Q = (q_0, q_1)$, die jeweils einem *LineSegment* entsprechen. Definiert wurde:

$$F_\varepsilon = \{(s,t) \in [0,1]^2 \mid |||P(s) - Q(t)||_2 \leq \varepsilon\}$$

F_ε beschreibt demnach alle Punktpaare aus P und Q, dessen Abstand höchstens ε beträgt. Zur Veranschaulichung von F_ε dient Abbildung 3.4. Wie in Abbildung 3.4a zu erkennen, ist die euklidische Distanz zwischen dem grünen Punktpaar kleiner als ε. Demnach befindet sich der korrespondierende Punkt in Abbildung 3.4b innerhalb des sogenannten *free spaces* F_ε. Das blaue Punktpaar hingegen besitzt im euklidischen Raum eine Distanz größer ε. Daher befindet sich der korrespondierende Punkt nicht im *free space*. Wie in [AG95] gezeigt wird, ist der *free space* von zwei *LineSegments* stets ein Schnitt einer Ellipse mit dem Einheitsquadrat und damit konvex.



(a) *LineSegments* P, Q und Toleranz ε

(b) *Free space Diagramm* von P und Q

Abbildung 3.4: *Free space* zweier *LineSegments* [Pel]

Um das Entscheidungsproblem für willkürliche *LineStrings* $P = (p_0, \ldots, p_n)$ und $Q = (q_0, \ldots, q_m)$ lösen zu können, wurde F_ε wie folgt erweitert:

$$F_\varepsilon = \{(s,t) \in [0,n] \times [0,m] \mid |||P(s) - Q(t)||_2 \leq \varepsilon\}$$

Das *free space Diagramm* $[0,n] \times [0,m]$ kann als ein System bestehend aus $(n \cdot m)$ Zellen $C_{ij} = [i-1, i] \times [j-1, j]$ mit $1 \leq i \leq n, 1 \leq j \leq m$ betrachtet werden. Dabei gleicht $F_\varepsilon \cap C_{ij}$ dem *free space* der *LineSegments* $p_i = (P(i-1), P(i))$ und $q_j = (Q(j-1), Q(j))$ nach der vereinfachten

Definiton von F_ε. Abbildung 3.5 zeigt zwei *LineStrings*, eine Toleranz ε und wie sich der weiße *free space* über die einzelnen Zellen C_{ij} erstreckt. Durch die vorangegangenen Definitionen wurden alle Grundbausteine zur folgenden Definition der Äquivalenz des *Entscheidungsproblems* gelegt:

$$\text{Zwei } \textit{LineStrings } P = (p_0, \ldots, p_n), Q(q_0, \ldots, q_m) \text{ haben } \delta_F(P, Q) \leq \varepsilon$$
$$\Longleftrightarrow$$
$$\exists \textit{ LineString } M = (m_0, \ldots, m_k) \text{ mit } m_h \in F_\varepsilon\text{: } m_0 = (0,0), m_k = (n, m) \text{ und } M \text{ verläuft monoton in beiden Koordinatenrichtungen von } [0, n] \times [0, m]$$

In Abbildung 3.5 ist der schwarze *LineString*, der durch das *free space* verläuft ein Beispiel für M. Demnach gilt für die *LineStrings* P, Q aus Abbildung 3.5, dass $\delta_F(P, Q) \leq \varepsilon$ gilt.

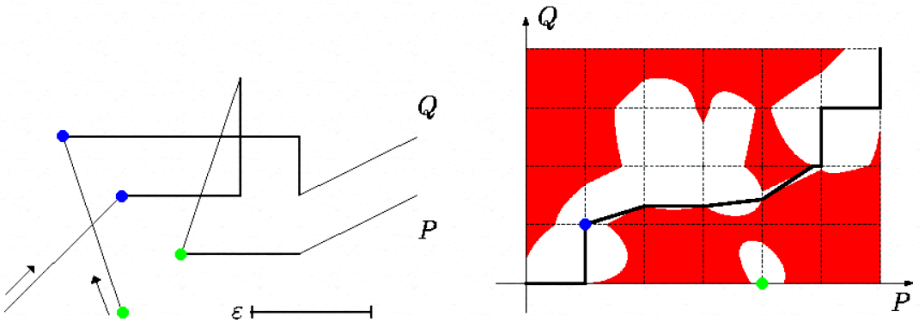


Abbildung 3.5: *LineStrings* P, Q mit ihrem *free space* F_ε [Pel]

Für $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$ seien die in Abbildung 3.6 veranschaulichten *LineSegments* $L^F_{ij} = ((i-1, a_{ij}), (i-1, b_{ij}))$ und $B^F_{ij} = ((c_{ij}, j-1), (d_{ij}, j-1))$ die linke bzw. die untere Grenze von $C_{ij} \cap F_\varepsilon$. Durch diese Grenzen können die Zellen betreten werden. Da die *free spaces* $F_\varepsilon \cap C_{ij}$ der einzelnen Zellen konvex sind, kann nun leicht entschieden werden ob ein monotoner *LineString* M von $(0, 0)$ nach (n, m) existiert.

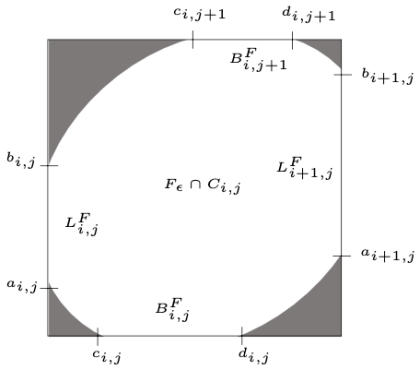


Abbildung 3.6: Grenzen einer *free space* Zelle C_{ij}[AKW01]

Weiter definiert worden ist:

$$R_\varepsilon = \{(s,t) | \exists \text{ monotoner } \textit{LineString } l \text{ in } F_\varepsilon \text{ mit } l = ((0,0), \ldots, (s,t))\}$$

Dabei ist der *monotone free space* R_ε eine Teilmenge von F_ε, bei der alle Stellen des *free spaces* entfernt wurden, die nicht durch einen monotonen *LineString* erreicht werden können. In Abbildung 3.7 ist beispielhaft R_ε des *free spaces* F_ε aus Abbildung 3.5 auf der vorherigen Seite zu sehen.

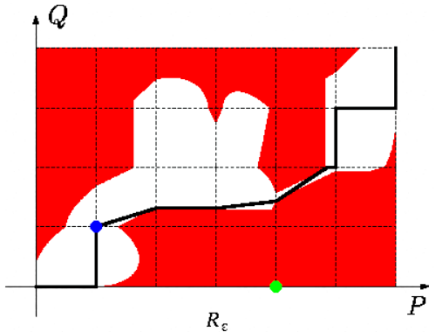


Abbildung 3.7: *Monotoner free space* R_ε [Pel]

Korrespondierend zu L^F_{ij} und B^F_{ij} seien $L^R_{ij} = ((i-1, a_{ij}), (i-1, b_{ij}))$ und $B^R_{ij} = ((c_{ij}, j-1), (d_{ij}, j-1))$ die linke bzw. die untere Grenze von $C_{ij} \cap R_\varepsilon$. Der Algorithmus 3.2 auf der nächsten Seite des *Entscheidungsproblems* nutzt die folgende offensichtliche Äquivalenz zur Berechnung der Entscheidung:

$$\text{Zwei } \textit{LineStrings } P = (p_0, \ldots, p_n), Q(q_0, \ldots, q_m) \text{ haben } \delta_F(P, Q) \leq \varepsilon$$
$$\Longleftrightarrow$$
$$(n, m) \in L^R_{n+1,m}$$

Algorithmus 3.2 Zur Lösung des Entscheidungsproblems: $\delta_F(P, Q) \leq \varepsilon$?

procedure ENTSCHEIDUNGSPROBLEM($P = (p_0, \ldots, p_n)$,$Q = (q_0, \ldots, q_m)$,$\varepsilon \geq 0$)
  for all $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$ **do**
    Berechne L_{ij}^F und B_{ij}^F
  end for
  for all $i \in \{1, \ldots, n\}$ **do**
    bestimme B_{i1}^R
  end for
  for all $j \in \{1, \ldots, m\}$ **do**
    bestimme L_{1j}^R
  end for
  for all $i \in \{1, \ldots, n\}$ **do**
    for all $j \in \{1, \ldots, m\}$ **do**
      Berechne $L_{i+1,j}^R$ und $B_{i,j+1}^R$ aus $L_{ij}^R, B_{ij}^R, L_{i+1,j}^F$ und $B_{i,j+1}^F$
    end for
  end for
  if $(n, m) \in L_{n+1,m}^R$ **then**
    return true
  else
    return false
  end if
end procedure

Aus gegebenen $L_{ij}^R, B_{ij}^R, L_{i+1,j}^F$ und $B_{i,j+1}^F$ können $L_{i+1,j}^R$ und $B_{i,j+1}^R$ in $O(1)$ berechnet werden [AG95]. Demnach entscheidet Algorithmus 3.2 in $O(nm)$ ob $\delta_F(P, Q) \leq \varepsilon$ gilt.

Berechnung der Fréchet Distanz

Durch das gewonnene Verständnis zur Lösung des einfacheren *Entscheidungsproblems*, lässt sich das ursprüngliche Problem der Berechnung der *Fréchet Distanz* δ_F lösen. Um $\delta_F(P, Q)$ für $P = (p_0, \ldots, p_n)$ und $Q = (q_0, \ldots, q_m)$ zu berechnen, wird in [AG95] folgendermaßen vorgegangen: Es wird mit einer Toleranz $\varepsilon = 0$ des *Entscheidungsproblems* gestartet, welche kontinuierlich vergrößert wird. Daraus resultiert auch eine kontinuierliche Vergrößerung des *free spaces* F_ε. Definiert wurde:

$$\delta_F(P, Q) = \text{ das kleinste } \varepsilon\text{:}$$
$$\exists \textit{ LineString } M = (m_0, \ldots, m_k) \text{ mit } m_h \in F_\varepsilon\text{: } m_0 = (0, 0), m_k = (n, m) \text{ und } M \text{ verläuft monoton}$$
$$\text{in beiden Koordinatenrichtungen von } [0, n] \times [0, m]$$

Die Existenz eines solchen *LineStrings* M kann nur dann eintreten, wenn einer der folgenden kritischen Fälle bei der kontinuierlichen Vergößerung von ε eintritt:

a) $(0, 0), (n, m) \in F_\varepsilon$

   Das Eintreffen dieses Falls hängt von $\|p_0 - q_0\|_2$ und $\|p_n - q_m\|_2$ ab.

b) Ein L_{ij}^F oder B_{ij}^F wird nichtleer (d.h. dass sich eine Passage zu einer Nachbarzelle geöffnet hat, die es zuvor nicht gab).

   Das Eintreffen dieses Falls hängt von der Distanz zwischen Punkten von einem *LineString* zu den *LineSegments* des anderen *LineStrings* ab.

c) $a_{ij} = b_{kj}$ oder $c_{ij} = d_{ik}$ (d.h. dass sich wie in Abbildung 3.8 zu sehen, eine neue horizontale oder vertikale Passage im *free space* F_ε bildet).

   Das Eintreten dieses Falls hängt von der gemeinsamen Distanz von zwei Punkten k_1, k_2 des gleichen *LineStrings* zu einem Schnittpunkt s ab. Sei $k = (k_1, k_2)$ das *LineSegment*, das die zwei Punkte verbindet und k' ein *LineSegment*, das k in der Mitte im 90 Grad Winkel schneidet. Dann ist s der Schnitt zwischen k' und einem *LineSegment* des *LineStrings* in dem k_1, k_2 nicht enthalten sind.

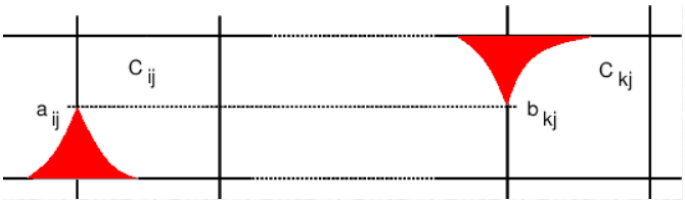


Abbildung 3.8: Eröffnung einer neuen vertikalen Passage durch Eintreten von $a_{ij} = b_{kj}$ [Pel]

Alle genannten Werte von denen das Eintreten der Fälle a),b) und c) abhängen, werden auch *kritische Werte* von ε genannt (Beispiele in Abbildung 3.9). Um δ_F zu berechnen, wird der kleinste *kritische Wert* von ε gesucht für den das *Entscheidungsproblem* positiv ist. Die Funktionsweise des Algorithmus 3.3 auf der nächsten Seite zur Berechnung der *Fréchet Distanz* lässt sich generell durch die folgenden Schritte beschreiben [AG95]:

1. Bestimme alle *kritischen Werte* von ε
2. Sortiere die *kritischen Werte*
3. Führe eine *binäre Suche* auf den sortierten *kritischen Werten* durch. Während jedem Suchschritt wird das *Entscheidungsproblem* für den aktuellen *kritischen Wert* gelöst. Ist das *Entscheidungsproblem* positiv, so setze die *binäre Suche* mit der Hälfte der kleineren *kritischen Werte* fort. Ist das *Entscheidungsproblem* negativ, so setze die *binäre Suche* mit der Hälfte der größeren *kritischen Werte* fort. Der letzte kritische Wert der bei der *binären Suche* übrig bleibt ist $\varepsilon = \delta_F$.

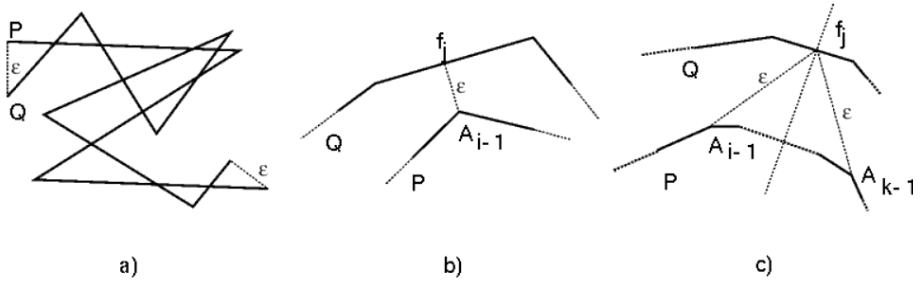


Abbildung 3.9: Visualisierungsbeispiele von kritischen Werten für ε [Pel]

Algorithmus 3.3 Berechnung der Fréchet Distanz $\delta_F(P, Q)$

```
procedure FréchetDistanz(P = (p_0, ..., p_n), Q = (q_0, ..., q_m))
    E ← getAllCriticalValuesOfε(P, Q)                    // E = {v_0, ..., v_u}
    sortedE ← mergeSort(E)                               // sortedE = [v_0, ..., v_u]
    ε ← binarySearch(E, P, Q)
end procedure

procedure binarySearch(E = [v_0, ..., v_u], P, Q)
    if |E| ≤ 1 then
        return v_0
    end if
    mid ← ⌊u/2⌋
    decisionPositive ← Entscheidungsproblem(P,Q,v_mid)
    if decisionPositive then
        E = [v_0, ..., v_mid]
    else
        E = [v_mid+1, ..., v_u]
    end if
    return binarySearch(E,P,Q)
end procedure
```

Komplexitätsanalyse

Es existieren $O(n^2m + nm^2)$ viele *kritische Werte* für ε, die sich jeweils in $O(1)$ berechnen lassen [AG95]. Demnach erfordert die Berechnung aller *kritischen Werte* eine Laufzeit in $O(n^2m+nm^2)$. Der effiziente Sortieralgorithmus *merge sort* sortiert eine gegebene Menge von n Elementen in $O(n \log n)$, was zur Folge hat, dass die Menge aller *kritischen Werte* in $O((n^2m+nm^2)\log(n^2m+nm^2))$ sortiert wird. Das *Entscheidungsproblem* erfordert Laufzeit $O(nm)$ und wird $O(log(n^2m + nm^2))$ mal gelöst. Demnach liegt die Laufzeit des dritten Schritts in $O(nm \log(n^2m + nm^2))$.

Aus den eben genannten Laufzeiten der einzelnen Schritte lässt sich erkennen, dass die Laufzeit der Berechnung der *Fréchet Distanz* durch die Laufzeit der Sortierung der *kritischen Werte* bestimmt wird. Demnach liegt die worst-case Laufzeit in $O((n^2m + nm^2)\log(n^2m + nm^2))$.

3.2.3 Modifizierte Fréchet Metrik

Die *Fréchet Metrik* liefert wie bereits beschrieben eine geeignete Grundlage zur Berechnung der Ähnlichkeit von gerichteten Routen. Da die *Fréchet Distanz* jedoch ausschließlich mit richtungserhaltenden Approximationen von Routen kompatibel ist [AG95], ergibt sich bei der bereits vorgestellten Approximation der gerichteten Routen durch ungerichtete Routen das in Abbildung 3.10 auf der nächsten Seite dargestellte Problem. Abgebildet sind zur Linken die *LineStrings* $P = (p_0, \ldots, p_n)$, $Q = (q_0, \ldots, q_m)$ und ihre *Fréchet Distanz* $\delta_F(P, Q)$. Auf der rechten Seite sind die *LineStrings* $P = (p_0, \ldots, p_n)$, $Q' = (q_m, \ldots, q_0)$ und ihre *Fréchet Distanz* $\delta_F(P, Q')$ dargestellt. Trotz der

anfänglich scheinenden jeweiligen Äquivalenz der *LineStrings* der linken und rechten Seite des Schaubilds, unterscheiden sich die *Fréchet Distanzen* enorm. Das ist darauf zurückzuführen, dass P und Q in entgegengesetzte Richtungen laufen während P und Q' gleich gerichtet sind.

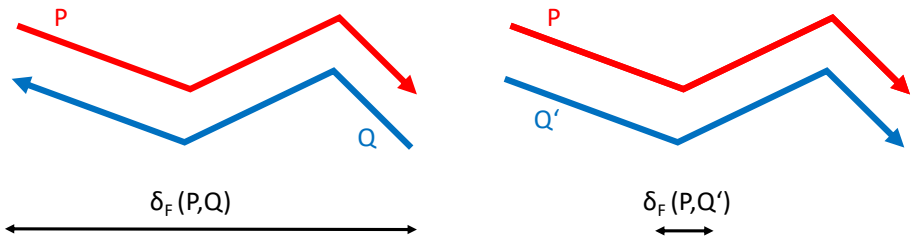


Abbildung 3.10: Richtungsabhängigkeit der *Fréchet Distanz* δ_F

Seien $P = (p_0, \ldots, p_n)$ und $Q = (q_0, \ldots, q_m)$ zwei beliebige *LineStrings*. Sei $Q' = (q_m, \ldots, q_0)$ der entgegen gerichtete *LineString* zu Q. Da Q und Q' ungerichtet betrachtet äquivalent sind, ist derjenige $Q_P = (q_{p0}, \ldots, q_{pm}) \in \{Q, Q'\}$ zur Berechnung der *Fréchet Distanz* $\delta_F(P, Q_P)$ zu wählen, der $\delta_F(P, Q_P)$ minimiert und damit gleichgerichtet zu P ist. Um $\delta_F(P, Q_P) = \varepsilon$ zu minimieren sind nach Definition die *kritischen Werte* für ε zu minimieren. Da die *kritischen Werte* die sich aus b) und c) in Abschnitt 3.2.2 auf Seite 37 ergeben, identisch für P, Q und P, Q' sind, gilt es die kritischen Werte aus a) zu minimieren. Für $\delta_F(P, Q_P) = \varepsilon$ gilt nach Definition von a) aus Abschnitt 3.2.2 auf Seite 37:

$$\begin{gathered}(p_0, q_{p0}), (p_n, q_{pm}) \in F_\varepsilon \\ \Longleftrightarrow \\ \|p_0 - q_{p0}\|_2 \leq \varepsilon \wedge \|p_n - q_{pm}\|_2 \leq \varepsilon\end{gathered}$$

Das bedeutet, dass die *Fréchet Distanz* mindestens so groß ist wie der Abstand zwischen den Start- und Endpunkten der jeweiligen *LineStrings* P, Q_P. Um den zu P gleichgerichteten *LineString* $Q_P \in \{Q, Q'\}$ zu finden, gilt es also zu überprüfen, welches Q_P die Funktion $\max(\|p_0 - q_{p0}\|_2, \|p_n - q_{pm}\|_2)$ minimiert. Der Algorithmus 3.4 auf der nächsten Seite berechnet die *modifizierte Fréchet Distanz* δ'_F zweier *LineStrings* gemäß der Abstraktion der Routen.

Algorithmus 3.4 Modifizierte Fréchet Distanz $\delta'_F(P, Q)$

```
procedure FréchetDistanzModifiziert(P = (p_0, ..., p_n),Q = (q_0, ..., q_m))
    distStartPoints ← ||p_0 − q_0||_2
    distEndPoints ← ||p_n − q_m||_2
    distStartPointsInverse ← ||p_0 − q_m||_2
    distEndPointsInverse ← ||p_n − q_0||_2
    max ← max(distStartPoints, distEndPoints)
    maxInverse ← max(distStartPointsInverse, distanceEndPointsInverse)
    if max ≤ maxInverse then
        return FréchetDistanz(P,Q)
    else
        Q' = (q_m, ..., q_0)
        return FréchetDistanz(P,Q')
    end if
end procedure
```

Komplexitätsanalyse

Alle Berechnungen der Distanzen zwischen zwei Punkten liegen in $O(1)$. Die Erzeugung des invertierten *LineStrings* Q' liegt in $O(m)$. Demnach liegt die Laufzeit der Berechnung der *modifizierten Fréchet Distanz* in derselben Komplexitätsklasse wie die Berechnung der *Fréchet Distanz*.

3.2.4 Berechnung der Ähnlichkeit

Dieser Abschnitt beschreibt die Vergleichsberechnung auf Basis der *modifizierten Fréchet Distanz* δ'_F zur Messung der Ähnlichkeit zweier *LineStrings*. Trotz einem offensichtlichen, höheren Grad an Ähnlichkeit haben die linken *LineStrings* in Abbildung 3.11 dieselbe *modifizierte Fréchet Distanz* wie die rechten *LineStrings*. Das ist den unterschiedlichen Größen der Paare von *LineStrings* geschuldet. Demnach liefert die *modifizierte Fréchet Distanz* zwar eine gute Grundlage, aber reicht allein nicht zur Bestimmung der Ähnlichkeit von *LineStrings* aus. Um den tatsächlichen Grad an Ähnlichkeit bestimmen zu können, wird eine konsistente Normierung der *modifizierten Fréchet Distanzen* benötigt.

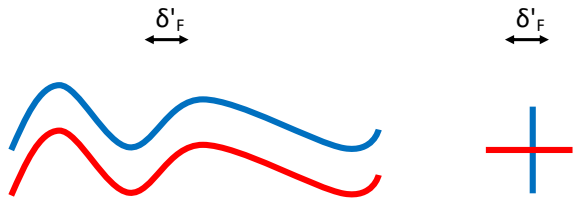


Abbildung 3.11: Gleiche *modifizierte Fréchet Distanz* δ'_F trotz unterschiedlicher Ähnlichkeit

Normierung

Zur Normierung bzw. Skalierung auf den Wertebereich $[0, 1]$ der *modifizierten Fréchet Distanzen* $\delta'_F(P, Q) \in \mathbb{R}$ für beliebige *LineStrings* $P = (p_0, \ldots, p_n), Q = (q_0, \ldots, q_m)$ mit $p_i = (x_i, y_i)$, $q_j = (u_j, v_j) \in \mathbb{R}^2$ ist zunächst eine achsenorientierte *BoundingBox* B für P, Q zu definieren:

$$B = \{(s,t) \in \mathbb{R}^2 | \ (g_l \leq s \leq g_r) \wedge (g_u \leq t \leq g_o)\}, \text{ mit}$$

$$g_l = \min(\min_{i \in \{0,\ldots,n\}}(x_i), \min_{j \in \{0,\ldots,m\}}(u_j))$$

$$g_r = \max(\max_{i \in \{0,\ldots,n\}}(x_i), \max_{j \in \{0,\ldots,m\}}(u_j))$$

$$g_u = \min(\min_{i \in \{0,\ldots,n\}}(y_i), \min_{j \in \{0,\ldots,m\}}(v_j))$$

$$g_o = \max(\max_{i \in \{0,\ldots,n\}}(y_i), \max_{j \in \{0,\ldots,m\}}(v_j))$$

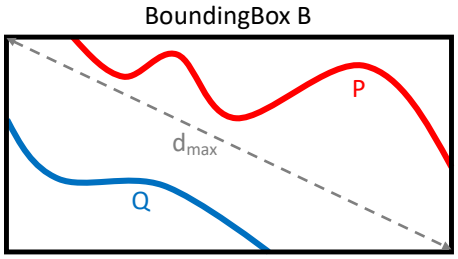


Abbildung 3.12: *BoundingBox B* von zwei *LineStrings* P, Q

Demnach stellt $\|(g_l, g_o) - (g_r, g_u)\|_2$ die maximale Distanz d_{max} in $\|\|_2$ zwischen zwei Punkten in B dar.

$\Rightarrow \forall a, b \in B : \|a - b\|_2 \leq d_{max}$

$\Rightarrow \forall s \in [0, n], t \in [0, m] : \|P(s) - Q(t)\|_2 \leq d_{max}$

$\Rightarrow F_{d_{max}} = \{(s,t) \in [0,n] \times [0,m] \mid \|P(s) - Q(t)\|_2 \leq d_{max}\} = [0,n] \times [0,m]$

$\Rightarrow \exists$ *LineString* $M = (m_0, \ldots, m_k)$ mit $m_h \in F_{d_{max}}$: $m_0 = (0,0), m_k = (n,m)$ und M verläuft monoton in beide Koordinatenrichtungen von $[0,n] \times [0,m]$

$\Rightarrow \forall P, Q$ mit *BoundingBox B*: $\delta'_F(P,Q) \leq d_{max}$

Aus der gewonnen Kenntnis $0 \leq \delta'_F(P,Q) \leq d_{max}$ für beliebige P, Q wird die folgende genormte Funktion γ abgeleitet, die zur Berechnung des *Ähnlichkeitsgrads* zwischen zwei beliebigen *LineStrings* dient:

$$\gamma : \mathbb{R}^2 \times \mathbb{R}^2 \to [0,1], \text{ mit } (P,Q) \mapsto \frac{\delta'_F(P,Q)}{d_{max}}$$

Algorithmus 3.5 Berechnung des Ähnlichkeitsgrads $\gamma(P, Q)$

procedure ÄHNLICHKEITSGRAD($P = (p_0, \ldots, p_n)$,$Q = (q_0, \ldots, q_m)$)
  $B \leftarrow$ GETBOUNDINGBOX(P,Q)
  $d_{max} \leftarrow$ GETDIAGONAL(B)
  return FRÉCHETDISTANZMODIFIZIERT(P,Q) / d_{max}
end procedure

Komplexitätsanalyse

Die Berechnung der *Boundingbox* und der Diagonalen liegt in $O(1)$. Demnach wird deutlich, dass die Laufzeit des Algorithmus in derselben Komplexitätsklasse wie die Berechnung der *modifizierten Fréchet Distanz* liegt.

3.2.5 Globale und lokale Ähnlichkeit

Unter Verwendung des Ähnlichkeitsgrads γ ist die folgende Äquivalenz zu definieren, in der $\gamma_{min} \in [0, 1]$ ein gegebenen *Mindestähnlichkeitsgrad* darstellt:

$$\text{Zwei } \textit{LineStrings } P \text{ und } Q \text{ sind ähnlich} \iff \gamma(P, Q) \geq \gamma_{min}$$

Mit Hilfe dieser Äquivalenz lassen sich Paare von *LineStrings* ermitteln, die sich in ihrer globalen Form ähneln. Das sind jedoch nicht die einzigen *LineStrings* die miteinander verschmolzen werden sollten. Häufig liegt, wie in Abbildung 3.13 zu sehen, keine globale Ähnlichkeit, sondern lediglich eine lokale Ähnlichkeit zwischen Routen und damit den *LineStrings* vor, die diese abbilden. Lokal ähnliche Routen sind diejenigen, die in Teilen ihres Verlaufs einen hohen Grad an Ähnlichkeit aufweisen. Dieses Kapitel thematisiert die Identifikation der Bereiche der *LineStrings*, die einen hohen Grad an lokaler Ähnlichkeit besitzen und demnach anschließend lokal verschmolzen werden sollten.

Abbildung 3.13: Lokal ähnliche *LineStrings* P, Q

Zur Erinnerung: Ein *LineString* $l = (p_0, \ldots, p_n)$ lässt sich durch ein $x \in \mathbb{R}, 0 \leq x \leq n$ parameterisieren, sodass $l(x)$ eine Position auf dem kontinuierlichen *LineString* angibt. Dabei ist $l(0) = p_0$ und $l(n) = p_n$ (Beispiel in Abbildung 2.6 auf Seite 24).

Es sind beliebige *LineStrings* $P = (p_0, \ldots, p_n)$ und $Q = (q_0, \ldots, q_m)$ auf *lokale Ähnlichkeiten* $\mathcal{A}(P, Q) \subset \{(P', Q') \mid \text{mit } P' \text{ bzw. } Q' \text{ ist } \textit{SubLineString} \text{ von } P \text{ bzw. } Q\}$ zu untersuchen. Dafür definieren wir zunächst:

$$P_\varepsilon = \{(s,t) \in \mathbb{R}^2 \mid \exists x \in \mathbb{R}, 0 \leq x \leq n : \ \|(s,t) - P(x)\|_2 \leq \varepsilon\}$$

$$Q_\varepsilon = \{(s,t) \in \mathbb{R}^2 \mid \exists x \in \mathbb{R}, 0 \leq x \leq n : \ \|(s,t) - Q(x)\|_2 \leq \varepsilon\}$$

Der sogenannte *Buffer* P_ε bzw. Q_ε enthält alle Punkte, die einen kleineren Abstand als ε zu P bzw. Q haben. Die *SubLineStrings* aus P und Q, die sich für die Berechnung des *Ähnlichkeitsgrads* qualifizieren sind folgendermaßen zu ermitteln:

$$P^Q = P \cap Q_\varepsilon \text{ bzw. } Q^P = Q \cap P_\varepsilon$$

Ein Beispiel für $P^Q \subset \{P_i^Q \mid \text{mit } P_i^Q \text{ ist } \textit{SubLineString} \text{ von } P\}$ bzw. $Q^P \subset \{Q_j^P \mid \text{mit } Q_j^P \text{ ist } \textit{SubLineString} \text{ von } Q\}$ ist in Abbildung 3.14 dargestellt. Zur Identifikation aller *lokalen Ähnlichkeiten* $\mathcal{A} \subseteq P^Q \times Q^P$ wird der Ähnlichkeitsgrad $\gamma(P_i^Q, Q_j^P)$ zwischen den *LineStrings* aus P^Q und Q^P berechnet und anschließend überprüft, ob γ größer gleich ein gegebener *Mindestähnlichkeitsgrad* γ_{min} ist, der definiert, ab welchem *Ähnlichkeitsgrad* die *LineStrings* als ähnlich gelten. Der Algorithmus 3.6 auf der nächsten Seite implementiert die eben genannte Vorgehensweise.

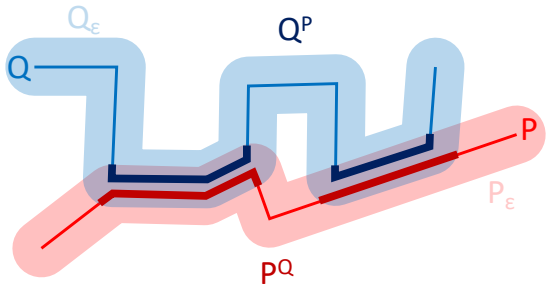


Abbildung 3.14: Identifikation von P^Q und Q^P

Algorithmus 3.6 Identifikation lokaler Ähnlichkeiten $\mathcal{A}(P, Q)$

procedure IdentifiziereLokaleÄhnlichkeiten($P = (p_0, \ldots, p_n)$,$Q = (q_0, \ldots, q_m)$,ε,γ_{min})
  $\mathcal{A} \leftarrow \{\}$
  $P_\varepsilon \leftarrow$ getBuffer(P,ε)
  $Q_\varepsilon \leftarrow$ getBuffer(Q,ε)
  $P^Q \leftarrow P \cap Q_\varepsilon$
  $Q^P \leftarrow Q \cap P_\varepsilon$
  for all $P_i^Q \in P^Q$ **do**
    $Q_{best} \leftarrow$ IdentifiziereLokaleÄhnlichkeit(P_i^Q,Q^P,γ_{min})
    if $Q_{best} \neq$ null **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup (P_i^Q, Q_{best})$
    end if
  end for
  return $\mathcal{A}$
end procedure

procedure IdentifiziereLokaleÄhnlichkeit(P_i^Q, Q^P, γ_{min})
  $\gamma_{max} \leftarrow 0$
  $Q_{best} \leftarrow$ null // Zu P_i^Q ähnlichster *LineString* aus Q^P
  for all $Q_j^P \in Q^P$ **do**
    $\gamma \leftarrow$ Ähnlichkeitsgrad(P_i^Q,Q_j^P)
    if $\gamma > \gamma_{max}$ **then**
      $\gamma_{max} \leftarrow \gamma$
      $Q_{best} \leftarrow Q_j^P$
    end if
  end for
  if $\gamma_{max} \geq \gamma_{min}$ **then**
    return Q_{best}
  else
    return null
  end if
end procedure

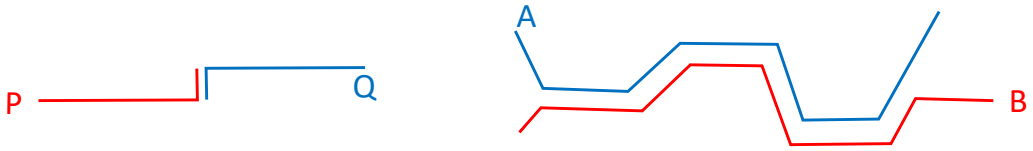
Komplexitätsanalyse

Es wird für alle $P_i^Q \in P^Q$ mit $|P^Q| = n$ der ähnlichste *LineString* $Q_j^P \in Q^P$ mit $|Q^P| = m$ ermittelt. Jeder Vergleich zwischen P_i^Q und Q_j^P erfordert die Berechnung des *Ähnlichkeitsgrads (siehe Abschnitt 3.2.4 auf Seite 41)*, die hier k genannt wird. Demnach liegt die Laufzeit in $O(nmk)$.

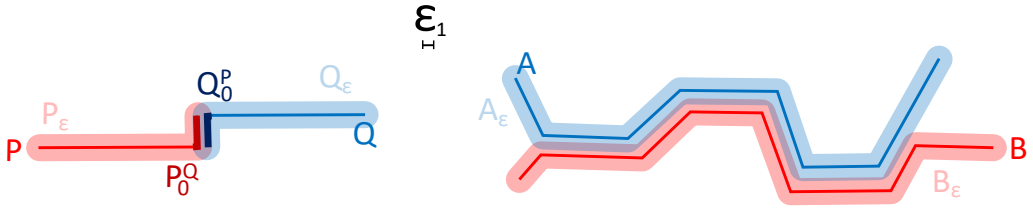
Wahl der Buffergröße

Wie aus Algorithmus 3.6 entnommen werden kann, spielt die Wahl der *Buffergröße* ε für die Identifikation der *lokalen Ähnlichkeiten* $\mathcal{A}$ eine entscheidende Rolle. Dabei geht sie mit der in Abbildung 3.15 auf der nächsten Seite dargestellten Herausforderung einher.

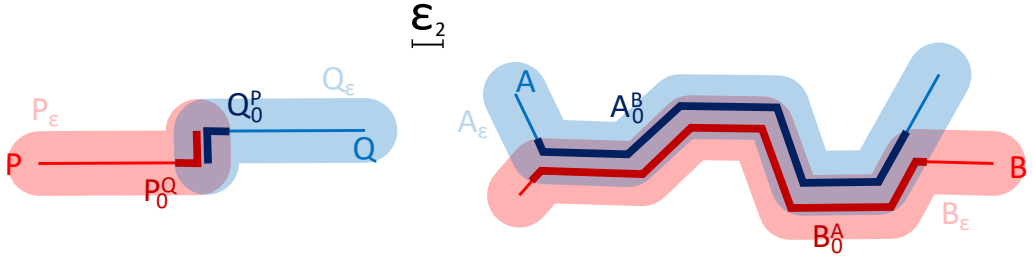
Zu sehen sind die Paare von *LineStrings* P,Q und A,B die jeweils *lokale Ähnlichkeiten* besitzen und als solche identifiziert werden sollten. Um den Einfluss der *Buffergröße* auf die Identifikation der *lokalen Ähnlichkeiten* zu illustrieren, wird angenommen, dass ein fester *Mindestähnlichkeitsgrad* γ_{min} gegeben ist, sowie in Abbildung 3.15b und Abbildung 3.15c zwei unterschiedliche *Buffergrößen* ε_1 und ε_2 zur Identifikation verwendet. Unter Verwendung der kleineren *Buffergröße* ε_1 in Abbildung 3.15b lässt sich die *lokale Ähnlichkeit* (P_0^Q, Q_0^P) identifizieren, da für den gegebenen *Mindestähnlichkeitsgrad* $\gamma(P_0^Q, Q_0^P) \geq \gamma_{min}$ gilt. Die *lokale Ähnlichkeit* zwischen A und B lässt sich unter der Verwendung von ε_1 jedoch nicht identifizieren, da $A^B = \emptyset$ und $B^A = \emptyset$ sind. Um auch die *lokale Ähnlichkeit* zwischen A und B erkennen zu können, wird in Abbildung 3.15c die minimale *Buffergröße* ε_2 gewählt, sodass A^B und B^A die zugehörigen *SubLineStrings* der *lokalen Ähnlichkeit* enthalten. Unter der Verwendung von ε_2 wird die *lokale Ähnlichkeit* (A_0^B, B_0^A) erkannt und als solche identifiziert, da $\gamma(A_0^B, B_0^A) \geq \gamma_{min}$ gilt. Die unter der Verwendung von ε_1 identifizierte *lokale Ähnlichkeit* (P_0^Q, Q_0^P) wird unter der Verwendung von ε_2 jedoch nichtmehr als solche erkannt, da P_0^Q und Q_0^P nun größere *SubLineStrings* von P und Q sind, die sich nicht mehr genug ähneln und damit $\gamma(P_0^Q, Q_0^P) < \gamma_{min}$ gilt.



(a) Ziel ist die Identifikation der *lokalen Ähnlichkeiten* zwischen P und Q sowie zwischen A und B.



(b) Identifikation der *lokalen Ähnlichkeit* zwischen P und Q.



(c) Identifikation der *lokalen Ähnlichkeit* zwischen A und B.

Abbildung 3.15: Einfluss der *Buffergröße* ε auf die Identifizierung der *lokalen Ähnlichkeiten*.

Aus dem Beispiel in Abbildung 3.15 lässt sich ableiten, dass keine einheitliche *Buffergröße* ε existiert, die es ermöglicht alle Arten von *lokalen Ähnlichkeiten* zu identifizieren. Demnach muss für unterschiedliche Größen von *lokalen Ähnlichkeiten* auch mit unterschiedlichen *Buffergrößen*

gearbeitet werden. Da es für den folgenden Verschmelzungsprozess wünschenswert ist, zuerst die großen *lokalen Ähnlichkeiten* zu identifizieren, wurde ein Ansatz gewählt, der die *Buffergröße* iterativ verringert und damit jegliche *lokale Ähnlichkeiten* $\mathcal{A}$ identifiziert.

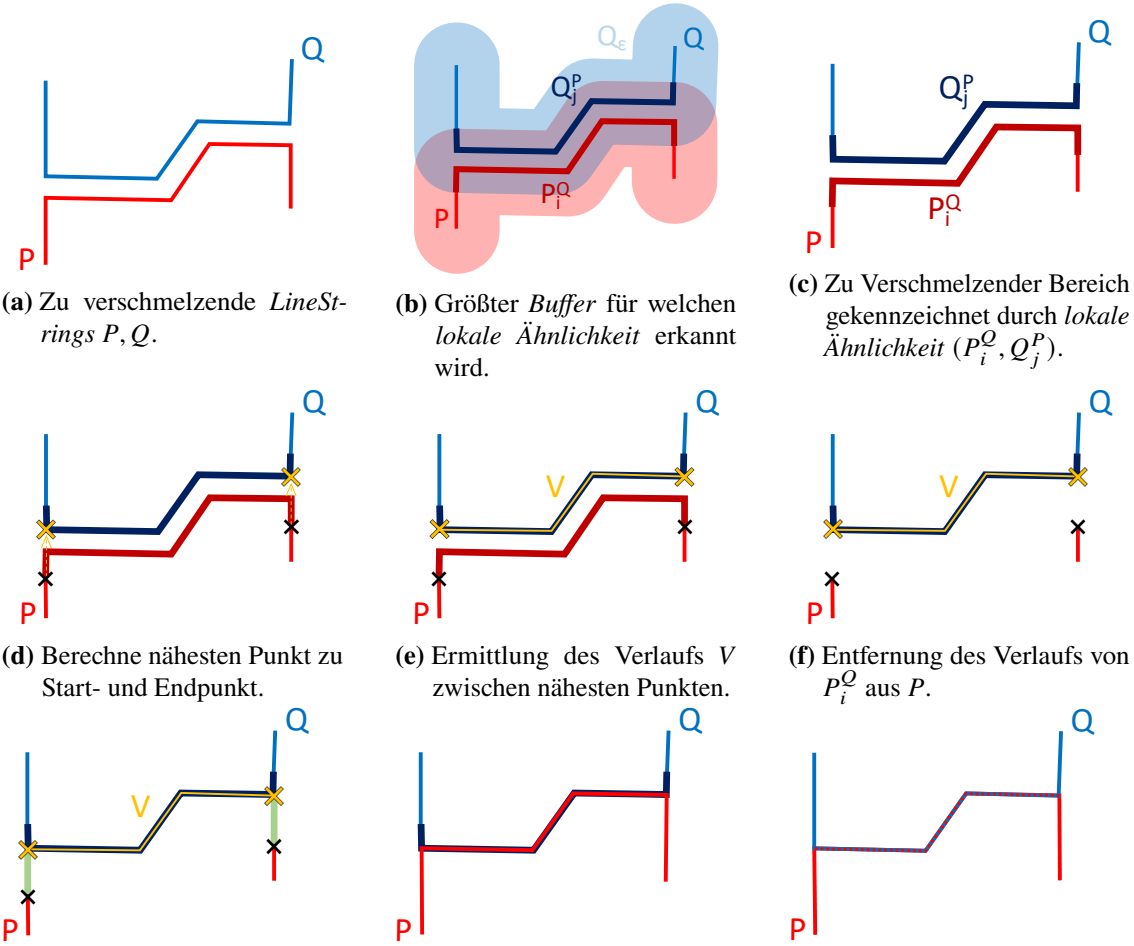
3.2.6 Verschmelzungsprozess

Zur Erstellung des Spielplans gemäß des Qualitätskriteriums *"7. Zusammenfassung ähnlicher Routen"* werden alle identifizierten *lokalen Ähnlichkeiten* durch den in diesem Abschnitt erläuterten Prozess verschmolzen.

Es gilt beliebige *LineStrings* P,Q in ähnlich verlaufenden Bereichen zu verschmelzen. Voraussetzung dafür ist, die Identifikation der ähnlich verlaufenden Bereiche, die durch die *lokalen Ähnlichkeiten* $\mathcal{A}(P,Q)$ definiert sind. Für jede *lokale Ähnlichkeit* $(P_i^Q, Q_j^P) \in \mathcal{A}$ mit $P_i^Q = (p_0^Q, \ldots, p_k^Q)$ und $Q_j^P = (q_0^P, \ldots, q_t^P)$ wird die Verschmelzung von P und Q nach folgendem vereinfachten Schema durchgeführt, welches in Abbildung 3.16 auf der nächsten Seite beispielhaft veranschaulicht wird:

1. Berechne zu Start- und Endpunkt p_0^Q,p_k^Q aus P_i^Q jeweils den Punkt $Q_j^P(x)$ und $Q_j^P(y)$ mit $x, y \in [0,t]$ aus Q_j^P mit der geringsten euklidischen Distanz
2. Ermittle den Verlauf V von Q_j^P zwischen $Q_j^P(x)$ und $Q_j^P(y)$
3. Lösche den Verlauf des *SubLineStrings* P_i^Q aus P
4. Bilde *LineSegements* über Punkte, die den unterbrochenen Verlauf von P markieren, zu dem nähesten Punkt $Q_j^P(x)$ bzw. $Q_j^P(y)$
5. Verbinde den Verlauf von P über die erstellten *LineSegments* mit dem Verlauf V von Q_j^P

Nach der Verschmelzung an der *lokalen Ähnlichkeit* zweier *LineStrings*, weisen diese im verschmolzenen Bereich einen identischen Verlauf vor, welcher durch eine identische Sequenz von *Punkten* definiert ist. Die Start- und Endpunkte der identischen Sequenzen spielen später in Abschnitt 3.3 auf der nächsten Seite zur Positionierung der Stationen des Spielplans eine entscheidende Rolle. Zu beachten gilt, dass durch den beschriebenen Verschmelzungsprozess auch *globale Ähnlichkeiten* verschmolzen werden, da diese nach Definition eine Teilmenge der *lokalen Ähnlichkeiten* darstellen. In diesem Fall stellt der Verlauf eines *LineStrings* nach der Terminierung, den identischen *LineString* oder einen *SubLineString* des anderen *LineStrings* dar.



(a) Zu verschmelzende *LineStrings* P, Q.

(b) Größter *Buffer* für welchen *lokale Ähnlichkeit* erkannt wird.

(c) Zu Verschmelzender Bereich gekennzeichnet durch *lokale Ähnlichkeit* (P_i^Q, Q_j^P).

(d) Berechne nähesten Punkt zu Start- und Endpunkt.

(e) Ermittlung des Verlaufs V zwischen nähesten Punkten.

(f) Entfernung des Verlaufs von P_i^Q aus P.

(g) Bildung der *LineSegments* zur Verbindung.

(h) Verbindung von P und V durch erstellte *LineSegments*.

(i) Verschmolzene *LineStrings* P, Q.

Abbildung 3.16: Beispielhafter Verschmelzungsprozess

3.3 Positionsidentifikation der Stationen

Dieses Kapitel definiert den Prozess der Positionsidentifikation der Stationen des Spielplans, die durch die generalisierten Routen verbunden sein müssen. Demnach stellt die Positionierung einer Station auf dem Verlauf einer Route die simpelste aller Anforderungen innerhalb der Platzierung dar. Um jedoch dem Qualitätskriterium *"8. Positionen der Stationen"* aus Abschnitt 2.7 auf Seite 24 gerecht zu werden, sind forderndere Bedingungen an die Positionierung der Stationen zu erfüllen. Diese sind zum einen durch die Platzierung der Stationen an Routenenden und zum anderen durch die Platzierung an Kreuzungspunkten der Routen definiert.

Die Identifkation der generalisierten Routenenden zur Erfüllung der ersten Teilbedingung des Qualitätskriteriums ist durch die Definition der *LineStrings*, welche die Routen abbilden, trivial und wird nicht weiter betrachtet.

Zur Erfüllung der zweiten Teilbedingung des Qualitätskriteriums ist die Identifikation der Kreuzungspunkte zu betrachten, die sich in zwei Kategorien unterteilen lassen. Die erste Kategorie der Kreuzungspunkte wird durch Punkte repräsentiert, die das Ende einer verschmolzenen *lokalen Ähnlichkeit* markieren und kein Routenende darstellen. An diesen Punkten trennen sich die Verläufe der lokal verschmolzenen Routen und entsprechen demnach einem Kreuzungspunkt. Da diese Punkte durch den Verschmelzungsprozess in Abschnitt 3.2.6 auf Seite 47 bereits definiert sind, gilt es die Identifikation der Kreuzungspunkte der zweiten Kategorie genauer zu untersuchen.

Die zweite Kategorie der Kreuzungspunkte umfasst diejenigen Punkte, die eine Überkreuzung des Verlaufs von zwei oder mehr Routen beschreiben. Diese Punkte wurden bisher durch keinen Prozess identifiziert und sind über die Schnittpunkte der *LineSegments* der *LineStrings* zu ermitteln, welche die Routen abbilden. Sei die Anzahl aller *LineSegments* aller *LineStrings* durch n definiert. Der *brute-force* Ansatz der Schnittpunktberechnung aller *LineSegments* erfordert den Vergleich aller $\binom{n}{2}$ Paare von *LineSegments* und liegt damit in $O(n^2)$. Dabei geht dieser Ansatz mit einem überflüssigen Aufwand einher, da die meisten der *LineSegments* sich auf Grund ihrer Lage nicht gegenseitig schneiden können und damit auch nicht miteinander verglichen werden müssen. Eine bessere Alternative stellen die *plane sweep* Algorithmen dar. Den vereinfachten Ablauf dieser Algorithmen kann man sich wie eine durch die Ebene monoton fortschreitende Linie vorstellen, die *sweep line* genannt wird. Dabei wird durch die Positionen der *sweep line* bestimmt, welche Objekte zu welchem Zeitpunkt miteinander verglichen werden [NP82].

3.3.1 Bentley-Ottmann Algorithmus

Der *Bentley-Ottmann Algorithmus* ist ein *sweep line* Algorithmus zur Identifikation aller Schnittpunkte von einer Menge S von *LineSegments*, der von Jon Bentley und Thomas Ottmann zum ersten Mal in [BO79] veröffentlicht wurde. Dieser Algorithmus erweitert die von Shamos und Hoey in [SH76] vorgestellte Arbeit, die den Test auf Existenz eines Schnitts in einer Menge von *LineSegments* beschreibt. Die Struktur und der Ablauf des *Bentley-Ottmann Algorithmus* nach [BO79] wird beschrieben durch:

Es wird eine vertikale *sweep line* (SL) strikt von links nach rechts zwischen endlichen Positionen Q, die im Folgenden definiert werden, durch die Ebene der *LineSegments* bewegt. Dabei sei R die Menge aller *LineSegments*, die mit SL zur jeweiligen Position einen Schnittpunkt $p_i(x_i, y_i)$ besitzen. Die Elemente von R werden dabei in einem *balancierten binären Suchbaum* verwaltet, der nach y_i sortiert ist. Die Positionen Q sind die Start-, End- und Schnittpunkte aller *LineSegments* und definieren damit die Stellen, an denen sich die Struktur von R ändert. Die Elemente aus Q sind nach ihren x-Werten zu sortieren und stellen demnach eine *priority queue* für SL dar. Q lässt sich effizient durch einen *binären heap* implementieren. Für jede Position q der SL sind in den jeweiligen Fällen die folgenden Schritte auszuführen:

a) q ist linker Endpunkt eines *LineSegments* s

   1. Füge s in R ein
   2. Überprüfe ob s seine Nachbarn v, n in R (*LineSegments*, die sich in R vor und nach s befinden) schneidet, und füge in dem Fall die Schnittpunkte in Q ein.

b) q ist rechter Endpunkt eines *LineSegments* s

1. Lösche s aus R
2. Füge im Falle der Existenz eines Schnittpunkts zwischen den ehemaligen Nachbarn von s in R, den Schnittpunkt in Q ein, sofern er noch nicht enthalten ist.

b) q ist Schnittpunkt von *LineSegments* s und t

1. Speicher q
2. Tausche Positionen von s und t in R
3. Füge im Falle der Existenz von Schnittpunkten zwischen s und t mit ihren jeweiligen neuen Nachbarn die Schnittpunkte in Q ein

Der Algorithmus terminiert nachdem die SL den rechten Endpunkt des am weitesten rechts liegenden *LineSegments* erreicht hat, da rechts von dieser Position keine weiteren Schnittpunkte zwischen *LineSegments* existieren können.

Algorithmus 3.7 Bentley-Ottmann Algorithmus

```
procedure BENTLEY-OTTMANN(S = {s_1, ..., s_n})
    AllPoints ← GETALLSTARTANDENDPOINTS(S)
    Q.ADDALL(AllPoints)                    // Punkte nach x-Wert sortiert in binären heap Q
    R ← {}        // LineSegments nach y-Wert sortiert in balancierterten binären Suchbaum R
    for all q ∈ Q do                                  // in aufsteigender x-Wert Reihenfolge
        s ← GETSEGMENT(q)
        if s.ISLEFTENDPOINT(q) then
            R.ADD(s)
            v ← R.GETPREDECESSOR(s)
            n ← R.GETSUCCESSOR(s)
            if s.INTERSECTS(v) then
                p_vs ← INTERSECTIONPOINT(v,s)
                Q.ADD(p_vs)
            end if
            if s.INTERSECTS(n) then
                p_ns ← INTERSECTIONPOINT(n,s)
                Q.ADD(p_ns)
            end if
        else if s.ISRIGHTENDPOINT(q) then
            v ← R.GETPREDECESSOR(s)
            n ← R.GETSUCCESSOR(s)
            R. REMOVE(s)
            if v.INTERSECTS(n) then
                p_vn ← INTERSECTIONPOINT(v,n)
                if ! Q.CONTAINS(p_vn) then
                    Q.ADD(p_vn)
                end if
            end if
        else                                  // q ist Schnittpunkt von LineSegments s und n
            v ← R.GETPREDECESSOR(s)
            n ← R.GETSUCCESSOR(s)
            SAVE(q,s,n)
            l ← R.GETSUCCESSOR(n)
            R.SWAP(s,n)
            if v. INTERSECTS(n) then
                p_vn ← INTERSECTIONPOINT(v,n)
                Q.ADD(p_vn)
            end if
            if s.INTERSECTS(l) then
                p_sl ← INTERSECTIONPOINT(s,l)
                Q.ADD(p_sl)
            end if
        end if
    end for
end procedure
```

Komplexitätsanalyse

Es wird angenommen, dass insgesamt $k \leq n^2$ Schnittpunkte zwischen den n *LineSegments* vorliegen. Die *For-Schleife* wird demnach $2n + k$ mal durchlaufen. Da in R zu jedem Zeitpunkt maximal n Elemente enthalten sein können, ist die Ausführung aller Operationen auf R durch $O(\log n)$ beschränkt. Die Kosten der Operationen auf Q liegen in $O(\log(2n + k) = O(\log n)$, da zu jedem Zeitpunkt maximal $2n + k$ Elemente enthalten sein können. Somit liegen die Kosten jedes Schleifendurchlaufs in $O(\log n)$ und demnach die Gesamtlaufzeit in $O((2n + k) \log n) = O((n + k) \log n)$.

3.3.2 Grapherstellung

Mit Hilfe der gewonnenen Erkenntnis bezüglich der Positionen der Stationen und ihren Verbindungen durch die *LineStrings*, die die Routen abbilden, lässt sich eine vorläufige Variante des Konnektivitätsgraphen $G = (V, E, \gamma)$, gemäß der Abstraktion des Spielplans erstellen. Dabei ist V die Menge der identifizierten Stationen, die in G genau dann verbunden sind, wenn sie durch mindestens einen *LineString* direkt verbunden sind. Das heißt, dass keine andere Station auf diesem *LineString* zwischen den verbundenen Stationen existieren darf. Die Gewichte der Kanten entsprechen den Ticketkosten, die bei der Reise über die Kanten während des Spiels zu bezahlen sind.

Formaler sei $G = (V, E, \gamma)$ mit

$V = \{p_i |\ p_i$ sind Positionen der Stationen$\}$

$E = \{\{p_0, p_1\} \in \binom{V}{2} \mid \exists$ *LineString* $l = (l_0, \ldots, l_n) \wedge x, y \in [0, n]$: $l(x) = p_0 \wedge l(y) = p_1 \wedge \nexists\, p \in V$ mit $p = l(z)$ für $x < z < y\}$

$\gamma : \{p_0, p_1\} \rightarrow \{$Routenticket $\subseteq \{$Busticket, Zugticket, U-Bahnticket$\}|\ p_0$ ist über *LineStrings* dieser Routentypen mit p_1 verbunden $\}$

Es gilt zu beachten, dass der eben definierte Graph nicht allen in Abschnitt 2.7 auf Seite 24 beschriebenen Qualititätskriterien entspricht und demnach lediglich als Basis für weitere Verarbeitungsschritte dient. In Abbildung 3.17 wird der Status der jeweiligen Qualitätskriterien zum aktuellen Verarbeitungsstand veranschaulicht. Durch den *Verschmelzungsprozess* aus Abschnitt 3.2.6 auf Seite 47 wird die Einhaltung des Qualitätskriteriums *"7. Zusammenfassung von ähnlichen Routen"* sichergestellt. Mit Hilfe der in diesen Kapitel beschriebenen Identifikation der Positionen von Stationen lässt sich das Qualitätskriterium *"8. Positionen der Stationen"* erfüllen. Die Qualitätskriterien *"1. Schleifenfrei"*, *"2. Richtungslosigkeit"* und *"5. Verschiedene Routen"* werden durch die Definition des Graphen G impliziert. Die Sicherstellung der übrigen Qualitätskriterien wird im folgenden Kapitel thematisiert.

Abbildung 3.17: Status der Qualitätskriterien

3.4 Lösung von Restkonflikten gemäß Qualitätskriterien

In diesem Kapitel werden die übrigen Konflikte des Graphen G und des Spielplans mit den Qualitätskriterien beleuchtet und durch simple Verarbeitungsschritte eliminiert.

3.4.1 Vermeidung von Mehrfachverbindungen des gleichen Typs

Zur Einhaltung des Qualitätskriteriums *"9. Keine Mehrfachverbindungen des gleichen Typs"* wird jeder Kante $\{p_0, p_1\} \in E$ mit $\gamma(\{p_0, p_1\}) \subseteq \{\text{Busticket, Zugticket, U-Bahnticket}\}$ für jedes Ticket $\in \gamma(\{p_0, p_1\})$ genau eine Route bzw. *LineString*, dieses Typs zugeordnet.

3.4.2 Abstand zwischen Stationen

Die Stationen V des Graphen können nach bisheriger Definition sehr nah beieinander liegen, sich demnach auf dem Spielplan überlappen und damit das Qualitätskriterium *"9. Ausreichender Abstand der Stationen"* verletzen. Die Suche nach Stationen innerhalb einer gewissen Distanz im euklidischen Raum lässt sich effizient durch die Nutzung eines *k-d trees* gestalten. Sobald zwei zu nah gelegene Stationen p_0 und p_1 erkannt werden, sollten sie in G durch den Algorithmus 3.8 verschmolzen werden.

Algorithmus 3.8 Verschmelzung überlappender Stationen p_0,p_1

```
procedure VerschmelzeStationen(G = (V, E, γ),p0,p1)
    oldEdges ← getEdges(p0)
    for all {p0, pk} ∈ oldEdges do
        if pk ≠ p1 then                          // Verhindert Erzeugung von Schleifen
            γ({p1, pk}) ← γ({p1, pk}) ∪ γ({p0, pk})
            E ← E ∪ {p1, pk}
        end if
        E ← E \ {p0, pk}
    end for
    V ← V \ {p0}
    return G
end procedure
```

Auf dem Spielplan ist die Position der verschmolzenen Station entsprechend anzupassen. Aus den beiden Stationen p_0, p_1 mit Positionen $(x_0, y_0), (x_1, y_1)$ ergibt sich nach der Verschmelzung eine angepasste Station p_1 mit Position (x, y). Die Berechnung der Position (x, y) der angepassten Station erfolgt durch:

$$(x, y) = \left(\frac{x_0 + x_1}{2}, \frac{y_0 + y_1}{2}\right)$$

Es ist zu beachten, dass die Verläufe der *LineStrings*, auf denen die zu verschmelzenden Stationen liegen, korrespondierend zu der Verschiebung der Stationen geändert werden müssen. Dabei werden die *Punkte* der *LineStrings*, die die Positionen der zu verschmelzenden Stationen markieren, durch die neue Position der verschmolzenen Station ersetzt.

3.4.3 Eliminierung von Sackgassen

Um dem Qualitätskriterium *"3. Keine Sackgassen"* gerecht zu werden sind alle potentiellen Sackgassen aus dem Graphen G und korrespondierend aus dem Spielplan zu entfernen. Eine Sackgasse ist eine Station p_0, die maximal zu einer anderen Station adjazent ist. Um jegliche Sackgassen zu eliminieren, ist für jede gefundene Sackgasse der rekursive Algorithmus 3.9 ausführen.

Algorithmus 3.9 Eliminierung von Sackgassen

```
procedure EliminiereSackgasse(G = (V, E, γ), p_0)
    oldEdges ← getEdges(p_0)
    if |oldEdges| ≤ 1 then
        for all {p_0, p_k} ∈ oldEdges do
            E ← E \ {p_0, p_k}
            G ← EliminiereSackgasse(G, p_k)
        end for
        V ← V \ {p_0}
    end if
    return G
end procedure
```

3.4.4 Identifikation der größten Zusammenhangskomponente

Zur Erfüllung des Qualitätskriteriums *"4. Zusammenhängend"* wird die größte Zusammenhangskomponente des Graphen G gesucht und alle anderen Zusammenhangskomponenten verworfen. Durch eine Breitensuche oder die in [HT73] verwendete Tiefensuche zur Identifikation aller Zusammenhangskomponenten, auf welcher der Algorithmus 3.10 auf der nächsten Seite basiert, lässt sich die größte Zusammenhangskomponente in $O(\max(|V|, |E|))$ identifizieren.

Nach Ausführung aller in diesem Kapitel vorgestellten Verarbeitungsschritte, garantieren der Spielplan sowie der unterliegende Graph G, wie in Abbildung 3.18 dargestellt, jegliche Qualitätskriterien und ermöglichen damit ein übersichtliches Spielgeschehen.

Abbildung 3.18: Status der Qualitätskriterien

Algorithmus 3.10 Identifikation der größten Zusammenhangskomponente

```
procedure IdentifiziereHauptKomponente(G = (V, E, γ))
    biggestComponent ← (V_0 = {}, E_0 = {}, γ)
    unvisited ← V
    stack ← {}
    while unvisited ≠ ∅ do
        V_1 ← {}
        E_1 ← {}
        first ← unvisited.pop
        stack.push(first)
        while stack ≠ ∅ do
            p ←stack.pop
            V_1 ← V_1 ∪ {p}
            unvisited ← unvisited \ {p}
            edges ←getEdges(p, E)
            for all {p, p_k} ∈ edges do
                if p_k ∈ unvisited then
                    stack.push(p_k)
                    E_1 ← E_1 ∪ {p, p_k}
                end if
            end for
        end while
        if |V_1| > |V_0| then
            V_0 ← V_1
            E_0 ← V_1
        end if
    end while
    return biggestComponent
end procedure
```

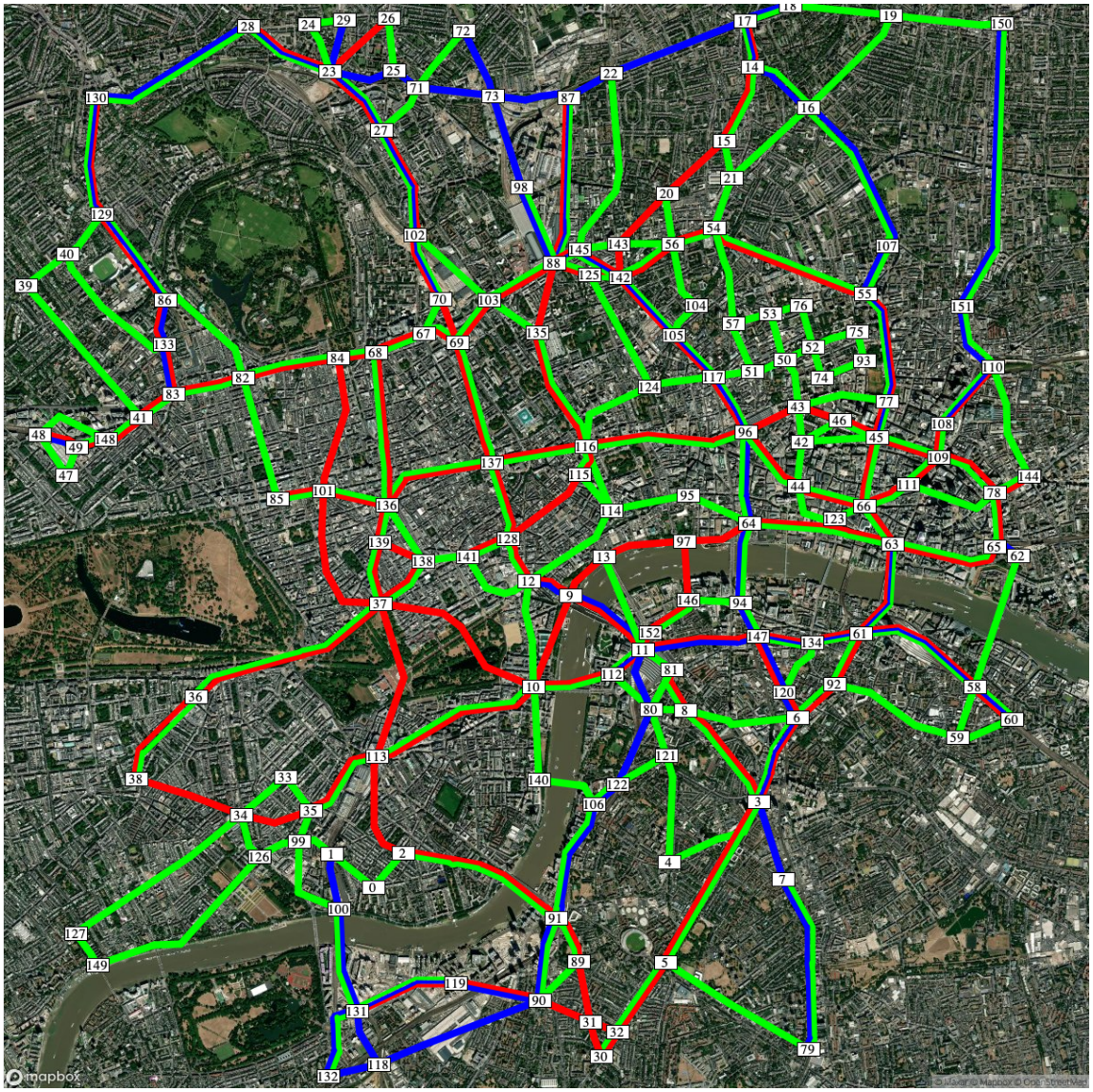


Abbildung 3.19: Generierter Spielplan Stadtzentrum London

4 Experimente

Dieses Kapitel thematisiert Eigenschaften des Generalisierungsprozesses, die mangels ihrer Intuität genauer beleuchtet werden. Um diese zu illustrieren, wurden die Laufzeiten der einzelnen Algorithmen unter Verwendung der OSM-Daten von *England* gemessen. Die Daten umfassen 9.441 Routen öffentlicher Verkehrsmittel, die aus insgesamt 2.149.105 *Nodes* bestehen.

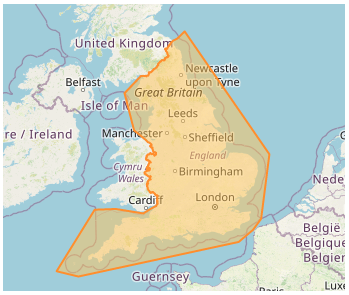


Abbildung 4.1: England [Geo]

4.1 Abhängigkeit des Zoomlevels

Die erste Eigenschaft lässt sich basierend auf der Ursache der scheinbaren Anomalie beschreiben, die sich aus den gemessenen Laufzeiten des Verschmelzungsprozesses in Abbildung 4.2 auf der nächsten Seite ableiten lässt. Als Folge der steigenden Anzahl von *LineStrings*, die es zu verschmelzen gilt, ergibt sich die natürliche Annahme einer Vergrößerung der Laufzeit des Algorithmus. Diese Annahme lässt sich in dem dargestellten Diagramm jedoch nur teilweise beobachten. In dem Bereich $x \geq 1.655$, wobei x die Anzahl der zu verschmelzenden *LineStrings* definiert, ist trotz steigender Anzahl von *LineStrings*, eine Abnahme der Laufzeit des Verschmelzungsprozesses zu beobachten. Diese Abnahme ist auf die Reduzierung des Zoomlevels der *Mapbox* Karte zurückzuführen, welches in dem Diagramm in Rot dargestellt ist. Eine Verkleinerung des Zoomlevels bedeutet eine Vergrößerung des Kartenausschnitts und ist demnach für die Zunahme der Anzahl an *LineStrings* zwingend notwendig. Neben einer steigenden Anzahl an *LineStrings*, bewirkt eine Verkleinerung des Zoomlevels jedoch auch eine Komprimierung der Größe und der Abstände, der *LineStrings* auf der Karte. Durch die Komprimierung der Größe der *LineStrings*, ergeben sich sehr viele nah beieinander liegende Schnittpunkte der *LineStrings*, die dementsprechend verschmolzen werden müssen, damit sie sich auf dem Spielplan nicht überlappen. Durch die Verschmelzung aller Stationen, die in Verbindung mit einem sehr kleinen *LineString* stehen, ergibt sich durch die Integration dieser *LineStrings* keinerlei Vorteil auf die Konnektivität des Spielplans. Aus diesem Grund werden sehr kleine *LineStrings* verworfen und nicht weiter verarbeitet. Die Komprimierung der Abstände bewirkt ein Zusammenschieben der *LineStrings* auf dem Kartenausschnitt und erhöht damit ihre

Ähnlichkeit. Das hat zufolge, dass viele der *LineStrings* global verschmolzen werden und demnach weniger Vergleiche und Berechnungen der Ähnlichkeiten zwischen den *LineStrings* notwendig sind.

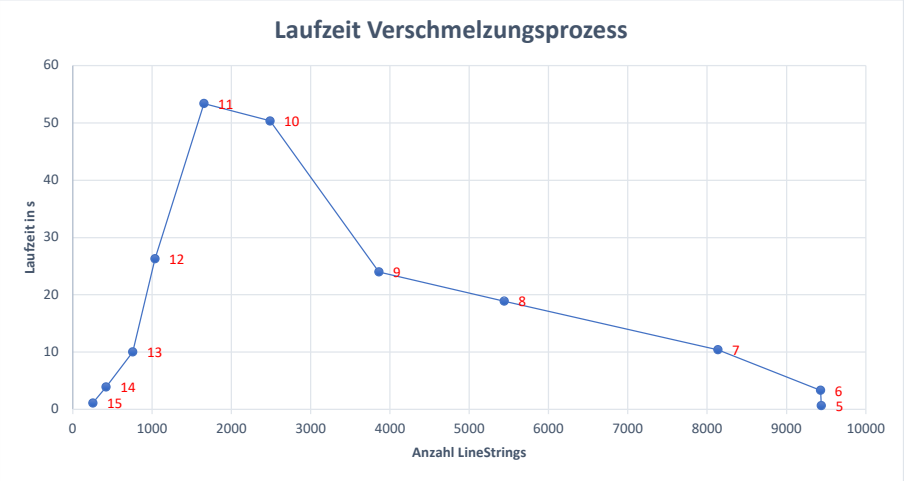


Abbildung 4.2: Laufzeit des Verschmelzungsprozesses in Abhängigkeit der Anzahl von *LineStrings*.

Neben der Abnahme der Laufzeit des Verschmelzungsprozesses, ist für korrespondierende Zoomlevel auch eine Abnahme der Laufzeit des *Bentley-Ottmann Algorithmus* zu beobachten (siehe Abbildung 4.3). Diese Abnahme lässt sich auf die steigende Anzahl der verworfenen *LineStrings* und die häufigeren Verschmelzungen zurückführen. Mit jeder Verschmelzung sinkt die globale Anzahl der verschiedenen *LineSegments*, die als Grundlage der Schnittberechnung des *Bentley-Ottmann* Algorithmus dienen.

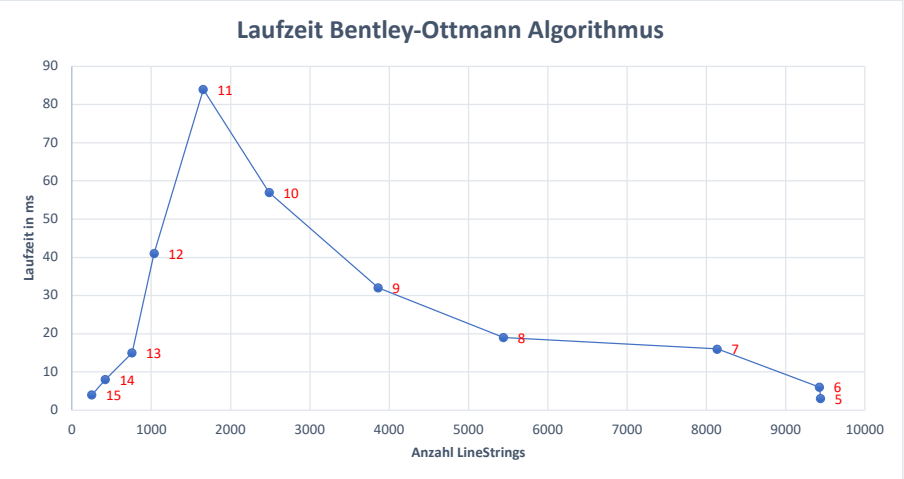


Abbildung 4.3: Laufzeit des Bentley-Ottmann Algorithmus in Abhängigkeit der Anzahl von *LineStrings*.

Auch wenn die Reduzierung des Zoomlevels des Kartenausschnitts sich vorteilhaft auf die Laufzeit einiger verwendeter Algorithmen auswirkt, ist ohne eine Erweiterung der vorgestellten Ansätze dennoch davon abzuraten. Ab einem bestimmten Punkt werden die Größen der *LineStrings* auf dem visualisierten Kartenausschnitt so klein, dass sie nicht weiter verwertbar sind und demnach nicht im Spielplan enthalten sein können. Man stelle sich dazu die Abbildung einer einfachen Busroute auf einem Kartenausschnitt der gesamten Welt vor. Diese Abbildung würde lediglich einem Punkt entsprechen und damit ohne eine erweiterte Zusammenhangsanalyse der Routen, keinerlei Konnektivität ergeben. Die eben beschriebenen Auswirkungen, der sinkenden Größen der *LineStrings* auf den Spielplan, werden in Abbildung 4.4 veranschaulicht. Korrespondierend zu der Abnahme der zuvor thematisierten Laufzeiten, lässt sich ab einem gewissen Punkt, eine Abnahme der im Spielplan enthaltenen Stationen, für sinkende Zoomlevel beobachten.

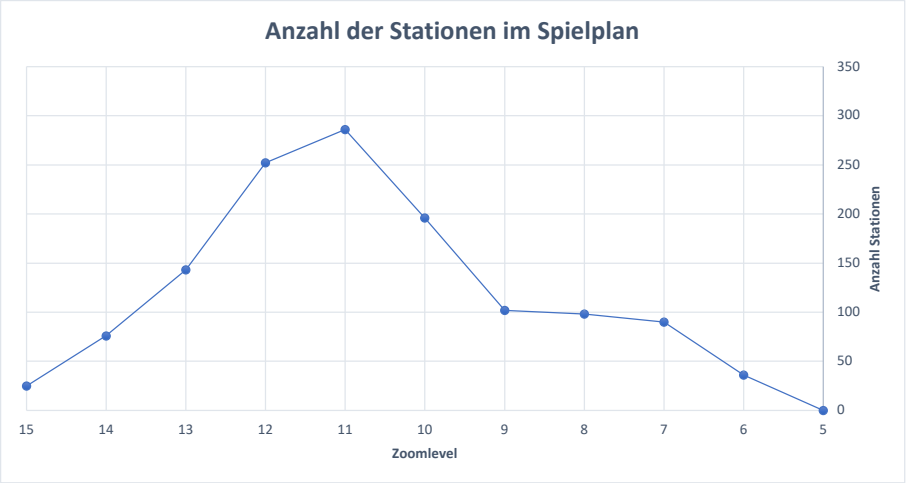
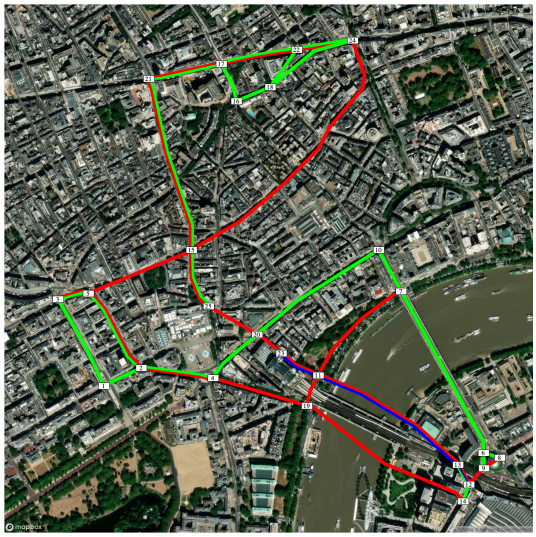
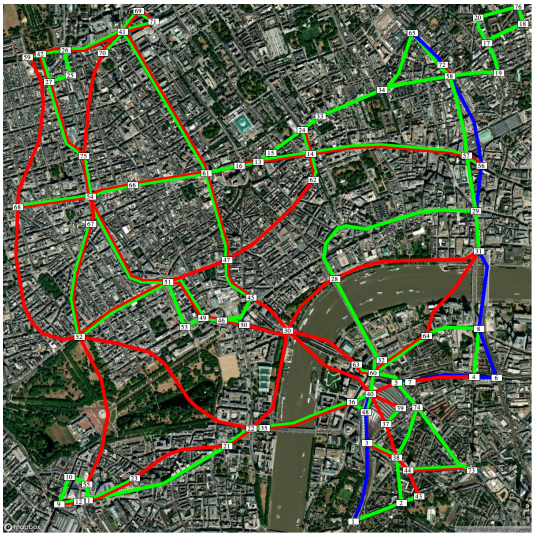


Abbildung 4.4: Im Spielplan enthaltene Stationen in Abhängigkeit des Zoomlevels des Kartenausschnitts

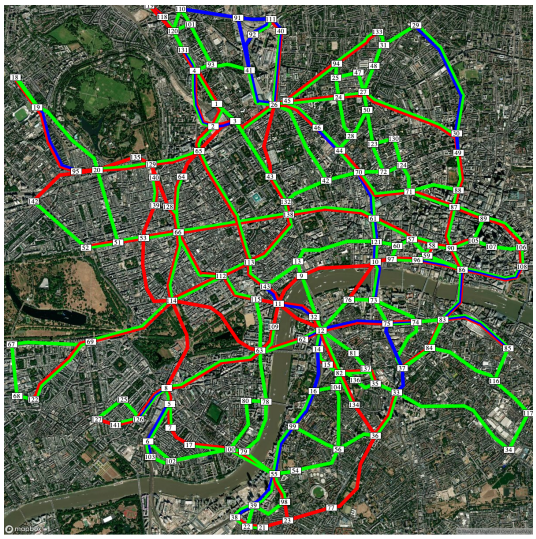
Die zur Laufzeitmessung erzeugten Spielpläne für verschiedene Zoomlevel der Kartenausschnitte, sind in Abbildung 4.5 auf der nächsten Seite veranschaulicht. Die Anzahl der in den Spielplänen enthaltenen Stationen, kann über das Zoomlevel aus Abbildung 4.4 entnommen werden. Es fällt auf, dass in Abbildung 4.5k auf Seite 61 kein Spielplan erkennbar ist. Das ist auf den zuvor beschriebenen Fall zurückzuführen, für den jegliche Routen zu nah beieinander liegen oder zu klein sind. Demnach lässt sich für den Kartenausschnitt dieses Zoomlevels, kein sinnvoller Spielplan erzeugen.



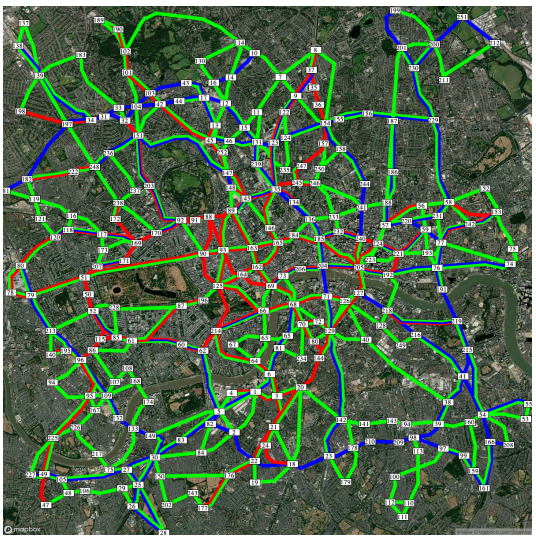
(a) Zoomlevel 15



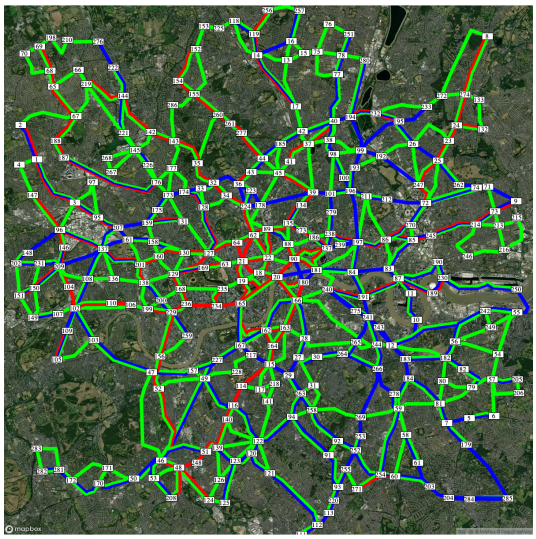
(b) Zoomlevel 14



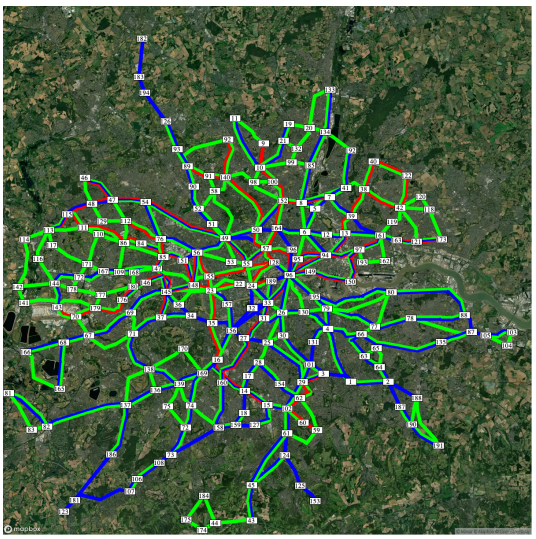
(c) Zoomlevel 13



(d) Zoomlevel 12

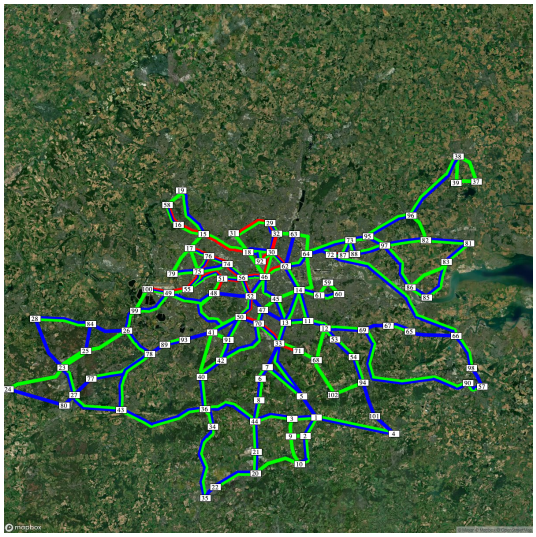


(e) Zoomlevel 11

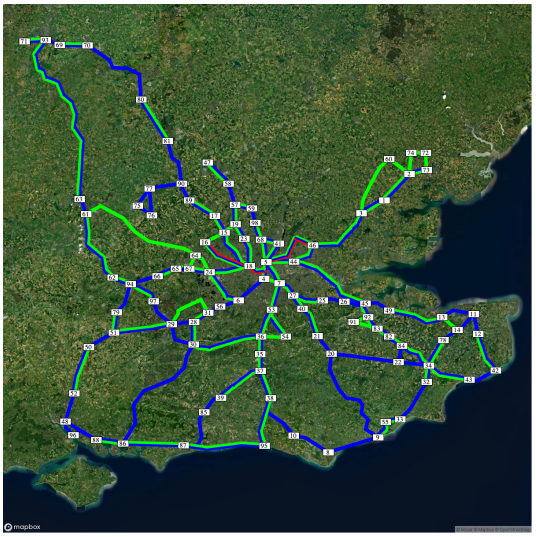


(f) Zoomlevel 10

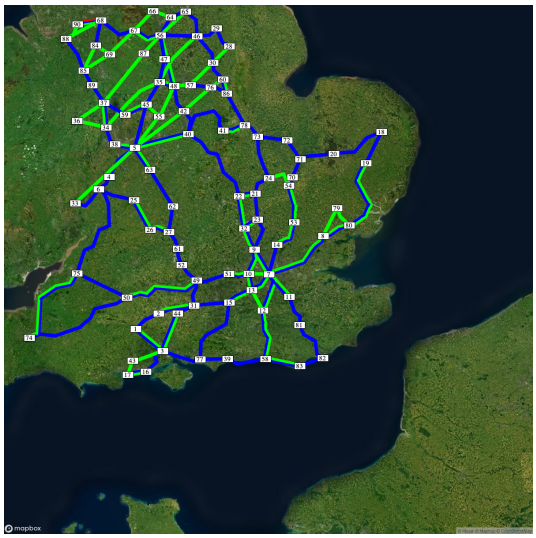
Abbildung 4.5: Generierte Spielpläne für verschiedene Zoomlevel



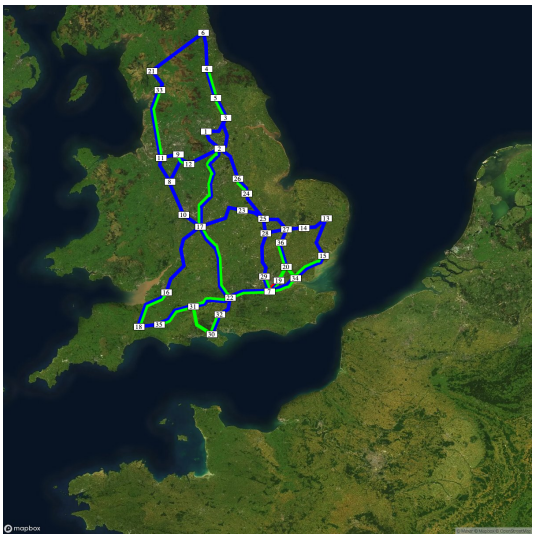
(g) Zoomlevel 9



(h) Zoomlevel 8



(i) Zoomlevel 7



(j) Zoomlevel 6

(k) Zoomlevel 5

Abbildung 4.5: Generierte Spielpläne für verschiedene Zoomlevel

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Überblick über die Grundlagen und die Generalisierungsschritte eines voll automatisierten Spielplanserstellungsprozesses für digitale Varianten von *Scotland Yard* gegeben. Die in der WGS84 Standard Projektion vorliegenden, bezogenen Daten öffentlicher Verkehrsmittel von OpenStreetMap wurden durch die Web Mercator Projektion auf 2-dimensionale Koordinaten abgebildet, die sich auf der durch Mapbox bezogenen Satellitenkarte positionsgetreu visualisieren lassen. Es wurde das interne Datenmodell der herangezogenen Daten definiert, sowie zu erfüllende und leicht überprüfbare Qualitätskriterien in Bezug auf dieses Datenmodell formalisiert. Daraufhin wurden die Konzepte der einzelnen Verarbeitungsschritte erläutert, die neben der Verwendung von bereits bekannten Algorithmen (Douglas-Peucker Algorithmus, Bentley-Ottmann Algorithmus) und Metriken (Hausdorff Metrik, Fréchet Metrik), eine Anpassung dieser, als auch die Entwicklung eigener Algorithmen erforderten. Außerdem wurde ermittelt wie sich die einzelnen Qualitätskriterien durch die beschriebenen Generalisierungsschritte garantieren lassen und infolgedessen ein übersichtlicher Spielplan am Ende des Generierungsprozesses zu erwarten ist.

Ein Wunsch der Spielergemeinde von *Scotland Yard*, welcher der Spielplangenerierungsprozess nicht erfüllt, ist die automatisierte Erstellung von Spielplänen, die weit über Stadtgrenzen hinaus gehen und weitaus größere Bereiche der realen Welt abbilden, wie z.B. größere Länder. Die Erstellung solcher Spielpläne bringt weitere Herausforderungen mit sich, die erweiterte Zusammenhangsanalysen der Routen von Netzwerken öffentlicher Verkehrsmittel und tiefgreifendere Generalisierungsprozesse erfordern.

Ausblick

Es steht in Planung, die begleitende Implementierung der Arbeit in Kombination mit der verwendeten digitalen *Scotland Yard* Variante unter OpenStreetMap Games[1] bereitzustellen, um die entwickelte Arbeit einer breiten Masse von Spielern zugänglich zu machen und infolgedessen ein ausgiebiges Feedback zu erhalten. Darüber hinaus ist geplant den Quellcode der Implementierung über Github zu veröffentlichen, um zukünftigen Entwicklern von automatisierten Generalisierungsprozessen oder Interessenten, die von den gewählten Ansätzen dieser Arbeit überzeugt wurden, die Wiederverwendbarkeit der Implementierung zu ermöglichen.

[1] https://wiki.openstreetmap.org/wiki/Games

Literaturverzeichnis

[ABB95] H. Alt, B. Behrends, J. Blömer. „Approximate matching of polygonal shapes“. In: *Annals of Mathematics and Artificial Intelligence* 13.3 (1995), S. 251–265 (zitiert auf S. 32, 33).

[AG92] H. Alt, M. Godau. „Measuring the resemblance of polygonal curves“. In: *Proceedings of the eighth annual symposium on Computational geometry*. 1992, S. 102–109 (zitiert auf S. 32, 33).

[AG95] H. Alt, M. Godau. „Computing the Fréchet distance between two polygonal curves“. In: *International Journal of Computational Geometry & Applications* 5.01n02 (1995), S. 75–91 (zitiert auf S. 33, 34, 37–39).

[Age87] U. S. D. M. Agency. *Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems*. Bd. 8350. Defense Mapping Agency, 1987 (zitiert auf S. 21).

[AKW01] H. Alt, C. Knauer, C. Wenk. „Matching polygonal curves with respect to the Fréchet distance“. In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 2001, S. 63–74 (zitiert auf S. 35).

[BFUY14] S. E. Battersby, M. P. Finn, E. L. Usery, K. H. Yamamoto. „Implications of web Mercator and its use in online mapping“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 49.2 (2014), S. 85–101 (zitiert auf S. 21, 22).

[BO79] J. L. Bentley, T. A. Ottmann. „Algorithms for reporting and counting geometric intersections“. In: *IEEE Transactions on computers* 28.09 (1979), S. 643–647 (zitiert auf S. 49).

[BW88] K. E. Brassel, R. Weibel. „A review and conceptual framework of automated map generalization“. In: *International Journal of Geographical Information System* 2.3 (1988), S. 229–244 (zitiert auf S. 14).

[Cha08] K.-T. Chang. *Introduction to geographic information systems*. Bd. 4. McGraw-Hill Boston, 2008 (zitiert auf S. 20).

[DP73] D. H. Douglas, T. K. Peucker. „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“. In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), S. 112–122 (zitiert auf S. 28).

[Foua] O. Foundation. *OpenStreetMap 64-Bit Integer*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/64-bit_Identifiers (zitiert auf S. 18).

[Foub] O. Foundation. *OpenStreetMap about*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap (zitiert auf S. 17).

[Fouc] O. Foundation. *OpenStreetMap copyright*. Accessed: 2022-03-17. URL: https://www.openstreetmap.org/copyright (zitiert auf S. 17).

[Foud] O. Foundation. *OpenStreetMap downloading data*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/Downloading_data (zitiert auf S. 17).

[Foue] O. Foundation. *OpenStreetMap Elemente*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/Elements (zitiert auf S. 18).

[Fouf] O. Foundation. *OpenStreetMap FAQ*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/FAQ#Why_OpenStreetMap.3F (zitiert auf S. 18).

[Foug] O. Foundation. *OpenStreetMap Node*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/Node (zitiert auf S. 18).

[Fouh] O. Foundation. *OpenStreetMap Node*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/Relation (zitiert auf S. 19, 20).

[Foui] O. Foundation. *OpenStreetMap Node*. Accessed: 2022-03-18. URL: https://wiki.openstreetmap.org/wiki/Public_transport (zitiert auf S. 19).

[Fouj] O. Foundation. *OpenStreetMap Planet.osm*. Accessed: 2022-03-17. URL: https://wiki.openstreetmap.org/wiki/Planet.osm (zitiert auf S. 17).

[Fouk] O. Foundation. *OpenStreetMap statistics*. Accessed: 2022-03-17. URL: https://planet.openstreetmap.org/statistics/data_stats.html (zitiert auf S. 17, 18).

[Foul] O. Foundation. *OpenStreetMap Way*. Accessed: 2022-03-18. URL: https://wiki.openstreetmap.org/wiki/Way (zitiert auf S. 19).

[Fré06] M. M. Fréchet. „Sur quelques points du calcul fonctionnel“. In: *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 22.1 (1906), S. 1–72 (zitiert auf S. 33).

[Geo] Geofabrik. *Geofabrik Download Server Greater London*. Accessed: 2022-04-03. URL: https://download.geofabrik.de/europe/great-britain/england/greater-london.html (zitiert auf S. 57).

[Hau20] F. Hausdorff. „Grundzüge der Mengenlehre“. In: *Bull. Amer. Math. Soc* 27 (1920), S. 116–129 (zitiert auf S. 32).

[HKR93] D. P. Huttenlocher, G. A. Klanderman, W. J. Rucklidge. „Comparing images using the Hausdorff distance“. In: *IEEE Transactions on pattern analysis and machine intelligence* 15.9 (1993), S. 850–863 (zitiert auf S. 32).

[HS92] J. E. Hershberger, J. Snoeyink. „Speeding up the Douglas-Peucker line-simplification algorithm“. In: (1992) (zitiert auf S. 31).

[HT73] J. Hopcroft, R. Tarjan. „Algorithm 447: efficient algorithms for graph manipulation“. In: *Communications of the ACM* 16.6 (1973), S. 372–378 (zitiert auf S. 54).

[IGN] IGNF. *CartAGen Webseite*. Accessed: 2022-03-16. URL: https://ignf.github.io/CartAGen/ (zitiert auf S. 14).

[Joc08] Jochen Corts. *Spiel des Jahres 1983: SCOTLAND YARD*. Accessed: 2022-03-16. 2008. URL: https://www.spiel-des-jahres.de/spiel-des-jahres-1983-scotland-yard/ (zitiert auf S. 13).

[Mapa] Mapbox. *Mapbox Projektion*. Accessed: 2022-03-20. URL: https://docs.mapbox.com/help/glossary/projection/ (zitiert auf S. 17).

[Mapb] Mapbox. *Mapbox Static Images API*. Accessed: 2022-03-20. URL: https://docs.mapbox.com/api/maps/ (zitiert auf S. 17, 27).

[MSP+15] S. Malys, J. H. Seago, N. K. Pavlis, P. K. Seidelmann, G. H. Kaplan. „Why the Greenwich meridian moved“. In: *Journal of Geodesy* 89.12 (2015), S. 1263–1272 (zitiert auf S. 20, 21).

[NP82] J. Nievergelt, F. P. Preparata. „Plane-sweep algorithms for intersecting geometric figures“. In: *Communications of the ACM* 25.10 (1982), S. 739–747 (zitiert auf S. 49).

[Pel] S. Pelletier. *Computing the Fréchet distance between two polygonal curves*. Accessed: 2022-03-24. URL: http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2002/StephanePelletier/ (zitiert auf S. 24, 34–36, 38).

[SBB+14] L. V. Stanislawski, B. P. Buttenfield, P. Bereuter, S. Savino, C. A. Brewer. „Generalisation operators“. In: *Abstracting geographic information in a data rich world*. Springer, 2014, S. 157–195 (zitiert auf S. 14).

[SH76] M. I. Shamos, D. Hoey. „Geometric intersection problems“. In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. IEEE. 1976, S. 208–215 (zitiert auf S. 49).

[SI] I. C. on Surveying, M. (ICSM). *Projektionsfiguren*. Accessed: 2022-03-19. URL: https://www.icsm.gov.au/education/fundamentals-mapping/projections (zitiert auf S. 22).

[Sny87] J. P. Snyder. *Map projections–A working manual*. Bd. 1395. US Government Printing Office, 1987 (zitiert auf S. 21, 23).

[TLD19] G. Touya, I. Lokhat, C. Duchêne. „CartAGen: An open source research platform for map generalization“. In: *International Cartographic Conference 2019*. Bd. 2. 2019, S. 1–9 (zitiert auf S. 14).

[Wika] Wikipedia. *Douglas-Peucker Algorithmus*. Accessed: 2022-03-22. URL: https://de.wikipedia.org/wiki/Douglas-Peucker-Algorithmus (zitiert auf S. 29, 30).

[Wikb] Wikipedia. *Ellipsoid*. Accessed: 2022-03-19. URL: https://de.wikipedia.org/wiki/Ellipsoid (zitiert auf S. 21).

[Wikc] Wikipedia. *Geographische Koordinaten im Gradnetz der Erdkugel*. Accessed: 2022-03-19. URL: https://de.wikipedia.org/wiki/Geographische_Koordinaten (zitiert auf S. 20).

Alle URLs wurden zuletzt am 04. 04. 2022 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift