



# Bachelorarbeit

zum Thema

„Entwicklung einer Anwendung für die optimale Anlegung neuer  
Telekommunikationsanschlüsse“

Betreuer: G. Kuhne,  
PYUR

Betreuer: Prof. Dr. rer. nat. habil. Dr. phil. Schenke,  
Hochschule Merseburg (FH)

Verfasser: Kliarskyi, Kyrlyo  
Fachbereich: Ingenieur- und Naturwissenschaften  
Studiengang: BAIN

## Abstract

Das Ziel in der vorliegenden Arbeit ist es, eine Methode zu entwerfen und zu entwickeln, die eine Menge von neuen Telekom-Anschlüssen optimal anlegt. Die Aufgabe sollte richtig formalisiert werden und als ein besonderer Fall eines Algorithmuses betrachtet werden. Dafür sind Kriterien wie die erforderliche Zeit, der Radius der Suche nach schon angelegten Trassen und Länge der Trassen zu beachten. Nach den Versuchen wurde so ein System implementiert und ausgewertet. Es stellte sich heraus, dass die App den Anforderungen entspricht. Ein Visualisierungsscript wurde vorbereitet, um die Ergebnisse darzustellen. Trotzdem es gibt Verbesserungsmöglichkeiten, die Leistungen des Systems verbessern können.

Schlagwörter: GIS, Clustering, .NET, C#, path-finding, Minimaler Spannbaum, Prim Algorithmus, Graphhopper,

## Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelorarbeit von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

## Abbildungsverzeichnis

Abb 1.1 Wie die Unternehmen in Deutschland für Digitalisierung sich interessieren

Abb. 1.2 Prognose zur Steigerung der Bruttowertschöpfung ausgewählter Branchen durch Digitalisierung

Abb. 1.3 Kundenzahlen Breitband-Anbieter Q4 2020 ( Quelle: DSLWEB Breitband Report 2020, Breitband Report 2020 - Wachstum im Krisenjahr (dslweb.de))

Abb. 1.4 Use-Case Diagramm

Abb. 2.1 Vergleich von Routing-Engines [5]

Abb. 2.2 Die Sprache, die bei Abfragen benutzt wird heißt - Overpass QL(Query Language).

Abb. 2.3 Algorithmus zum Lösen des Problems der Verbindungen zwischen verschiedenen Bundesländer.

Abb. 2.5 Die Unternehmen, die GH benutzen [8]

Abb. 2.6 So werden Geodaten in Datenbank gespeichert.

Abb. 2.7 Wie die räumliche Daten entschlüsselt werden können

Abb. 2.8 Repräsentierung von Geodaten in SQL Server Management Studio

Abb. 2.9 Häufigkeitsverteilung von Trassensegmenten

Abb. 2.10 Wie ein Gleichgewicht in Zerlegung erreicht wird

Abb. 2.11 Die Formel zur Berechnung den Abständen auf einer gebogenen Oberfläche.

Abb. 2.12 Wie ein Spannbaum aussieht

Abb. 2.13 Die Erklärung von Prim Algorithmus

Abb. 2.14 Klassendiagramm

Abb. 2.15 Wie die Anfrage bei Graphhopper aussieht

Abb. 2.16 Antwort vom Server

Abb. 3.1 Clustering. Schritt 1 - Repräsentierung von Eingangspunkten

Abb. 3.2 Clustering. Schritt 2 – jedem Punkt wird eine Linie(Trassensegment) zugewiesen, die am nächsten liegt

Abb. 3.3 Clustering. Schritt 2 – es werden zwei Ansätze verglichen.

Abb. 3.4 Clustering. Schritt 3. Jeder Trasse wird eine Menge von Punkten zugewiesen.

Abb. 3.5 Allgemeine Vorgehensweise bei der Implementierung

Abb. 3.6 Wie die Rückgabedaten aussehen

Abb. 3.7 Ein Teil des Skriptes für die Demonstration der neuangelegten Wege und schon vorhandenen Trassensegmenten

Abb. 3.8 Anzahl der Punkte – 11, Trassensuchradius – 105 Meter, Gesamtlänge – 4566 Meter.

Abb. 3.9 Anzahl der Punkte – 11, Trassensuchradius – 300 Meter, Gesamtlänge – 2086 Meter.

Abb. 3.10 Anzahl der Punkte – 11, Trassensuchradius – 600 Meter, Gesamtlänge – 1899 Meter.

Abb. 3.11 Letztendliche Performance bei der Veröffentlichung

Abb. 4.1 ein Beispiel mit Punkten in der Stadt Halle

Abb. 4.2 Detaillierter Aussicht 1

Abb. 4.3 Detaillierter Aussicht 2

Abb. 4.4 Detaillierter Aussicht 3

Abb. 4.5 Detaillierter Aussicht 4

## Inhaltsverzeichnis

Abstract .....	2
Erklärung.....	3
Abbildungsverzeichnis.....	4
Inhaltsverzeichnis .....	6
1 Einführung .....	8
1.1 Problemstellung (Soll-Ist Analyse).....	11
1.2 Ziel der Arbeit.....	12
1.3 Methodik .....	13
1.4 Gliederung und Aufbau.....	13
1.5 Grundlagen.....	14
1.6 Grundbegriffe.....	14
Zusammenfassung.....	15
2 Methodik und Vorgehen .....	16
2.1 Problemlösungsansatz.....	16
2.2 Auswahl einer Routing-Engine.....	17
2.3 Overpass Api.....	19
2.4 Graphhopper als eine alternative Routing Engine .....	21
2.5 Struktur von Eingangsdaten .....	22
2.6 Berechnungen bei der gebogenen Oberfläche .....	26
2.7 Algorithmus und Versuche .....	27
2.8 Formale Beschreibung des minimalen Spannbaums [10].....	28
2.9 Prim Algorithmus.....	28
2.10 Kruskal Algorithmus.....	29
2.11 Klassendiagramm.....	31
2.12 Anfrage-Schnittstellen .....	32
Zusammenfassung.....	33
3 Implementierung.....	34
3.1 Clustering Algorithmus.....	34
3.2 Anwendung von Prim Algorithmus .....	36
3.3 Letztendliche JSON Rückgabe .....	38
3.4 Visualisierung .....	39
3.5 Ein Auszug aus Versuchen .....	40
Zusammenfassung.....	43

4 Evaluierung.....	44
4.1 Analyse der berechneten Wege.....	44
Zusammenfassung.....	48
5 Zusammenfassung und Ausblick.....	49
5.1 Zusammenfassung.....	49
5.2 Ausblick .....	50
Literaturverzeichnis .....	51

## **1 Einführung**

Wir leben in der Zeit der vierten industriellen Revolution. "Industrie 4.0" ist eine Reihe von Merkmalen und Konzepten für die Unternehmensorganisation. Es gibt vier Grundprinzipien [1]:

Vernetzung: Die an der Produktion beteiligten Informationsknoten werden über Telekommunikationsnetze vernetzt und verwaltet.

Informationstransparenz: Sensoren ermöglichen die Übertragung von Informationssystemen digitaler Fabrikmodelle aus der realen Welt in die virtuelle Welt.

Technische Assistenz: Hilfesysteme bieten den Menschen gut visualisierte und umfassende Informationen. Dies ermöglicht eine schnellere Entscheidungsfindung, da der Überblick über das System klarer wird. Dazu gehört auch echte Hilfe, insbesondere physische Unterstützung durch Mitarbeiter, vor allem in gefährlichen Situationen.

Dezentralisierte Entscheidungen: Cyber-physische Systeme können viele Entscheidungen selbst treffen und müssen nur im Falle eines Fehlers die Kontrolle abgeben. Dazu gehören vor allem verschiedene Fehlfunktionen. Die meiste Zeit ist das System jedoch unabhängig [2].

Die Digitalisierung ist ein aktuelles Thema für alle Länder, insbesondere für Deutschland. Abbildung 1.1 zeigt den Trend, dass immer mehr Unternehmen die Industrie 4.0-Technologie einsetzen.



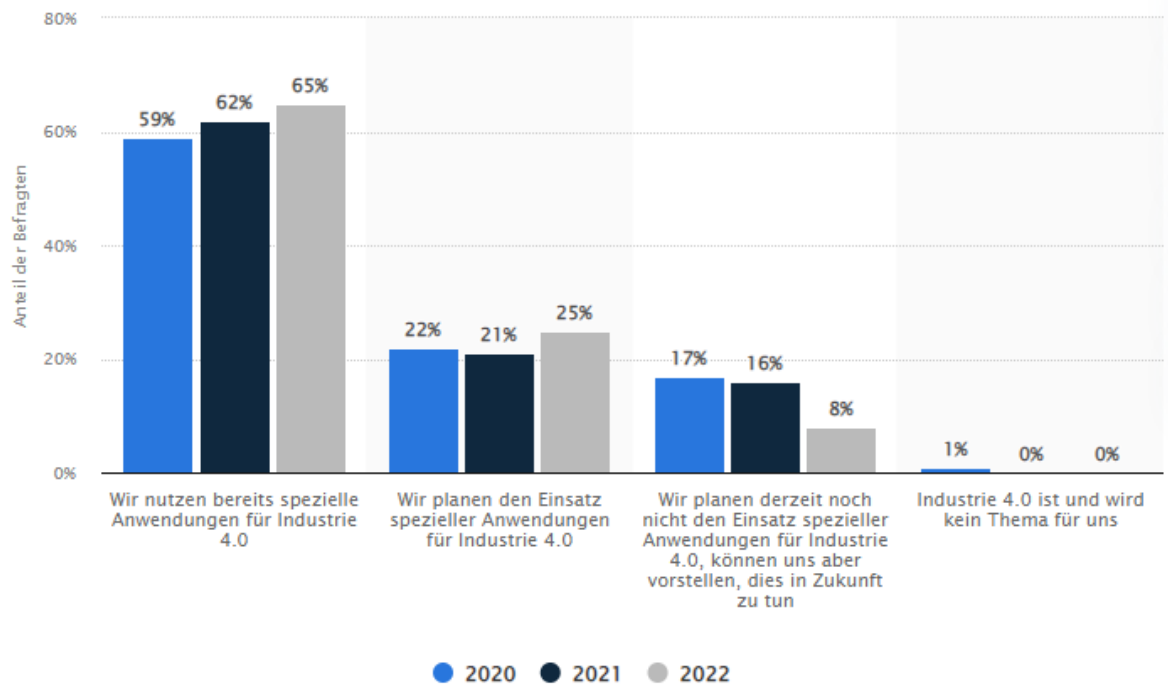


Abb. 1.1 Wie die Unternehmen in Deutschland für Digitalisierung sich interessieren[12]

Vergleicht man darüber hinaus, welche Branchen für Industrie 4.0 am wichtigsten sind, so zeigt sich ein Trend zur Informations- und Kommunikationstechnologie als wichtigster Branche. (Abb. 1.2)

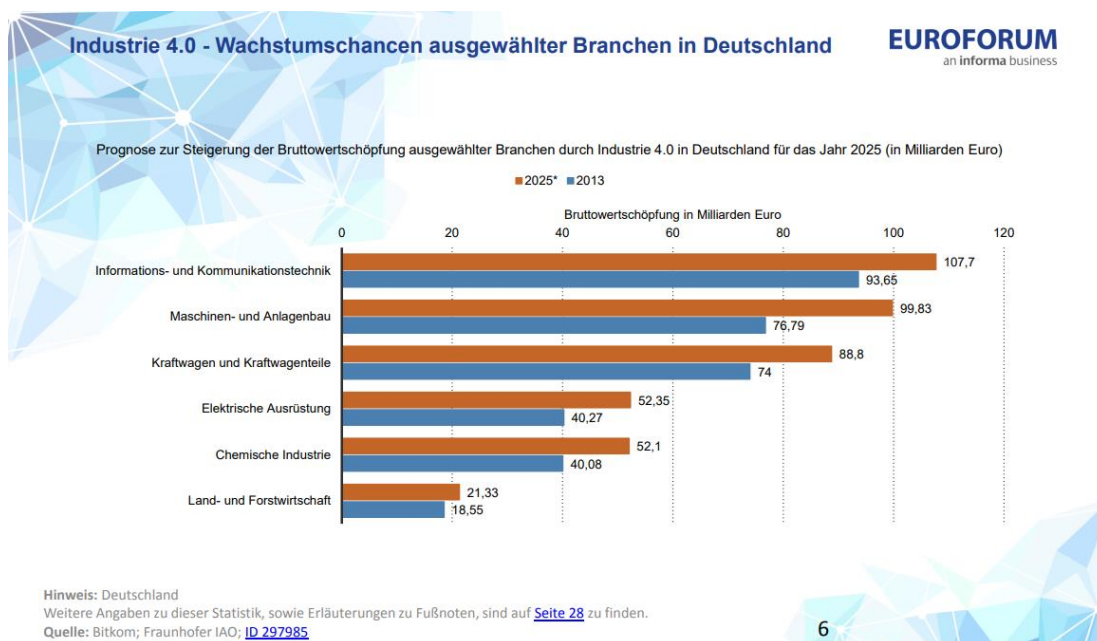


Abb. 1.2 Prognose zur Steigerung der Bruttowertschöpfung ausgewählter Branchen durch Digitalisierung [13]

Darüber hinaus können viele andere Aspekte der Industrie 4.0, wie vernetzte Produkte, Mensch-Maschine-Interaktion, Big Data, Cloud Computing usw., ohne hochwertige Kommunikationstechnologie kaum gut umgesetzt werden.

Portal dslweb.de berichtet: „Die Zahl der Internet-Kundenverträge von Tele Columbus(PYUR) ist auch im Q3 2021 weiter angestiegen - wenngleich sich das Wachstum im Vergleich zu den vorangegangenen Quartalen leicht abgeschwächt hat. Unterm Strich konnte der Kabelnetzbetreiber zwischen Anfang Juli und Ende September 4.000 zusätzliche Internetverträge gewinnen, deren Gesamtbestand hat damit auf 620.000 zugelegt.

In den ersten 9 Monaten des laufenden Geschäftsjahres konnte Tele Columbus mit seiner Marke PYUR so insgesamt 18.000 Netto-Neuverträge in dieser Sparte gewinnen. Hier machte sich nach Unternehmensangaben vor allem die Anfang des Jahres gestarteten Vertriebsmaßnahmen sowie die gesunkene Kündigungsrate bemerkbar. Seine ursprüngliche Zielmarke habe der Anbieter damit sogar übererfüllt.

PYUR hat dabei nicht nur zusätzliche Internetkunden gewonnen, diese griffen auch vermehrt zu höherwertigen Produkten. Der Umsatz im Segment "Internet & Telefon" lag in den ersten neun Monaten 2021 mit 126,72 Mio. Euro so auch um 3,4 Prozent über dem Vorjahresniveau.

Der Gesamtumsatz von Tele Columbus jedoch ist im gleichen Zeitraum um 1,8 Prozent auf 350,3 Mio. Euro abgesunken. Problemfelder waren hier vor allem der weiter rückläufige TV-Umsatz (- 2,9% auf 176,87 Mio. €) sowie die niedrigeren Umsätze aus dem Bauleistungsgeschäft. Auch insgesamt geht die Rechnung für Tele Columbus derzeit (noch) nicht auf: Für den 9-Monats-Zeitraum 2021 hat sich das negative Finanzergebnis auf 55,35 Mio. Euro verschlechtert (Vorjahr: 53,31 Mio. €), der Fehlbetrag für die Geschäftsperiode betrug 48,89 Mio. Euro (Vorjahr: 33,45 Mio. €)“ (Quelle - Tele Columbus im Q3 2021: Der nächste Umbruch ist schon eingeleitet - dslweb.de)

Diese Arbeit muss Bauleistungsgeschäft verbessern, denn durch die Erneuerung von Software müssen die Aufwände reduziert werden.

Darüber hinaus gibt es in Deutschland viele weitere Telekommunikationsdienstleister, die einigen Bundesländern voraus sind. Diese Konkurrenten (insbesondere Telekom, o2) haben jedoch eine führende Position bei den Internetanschlüssen. (Abb. 1.3)

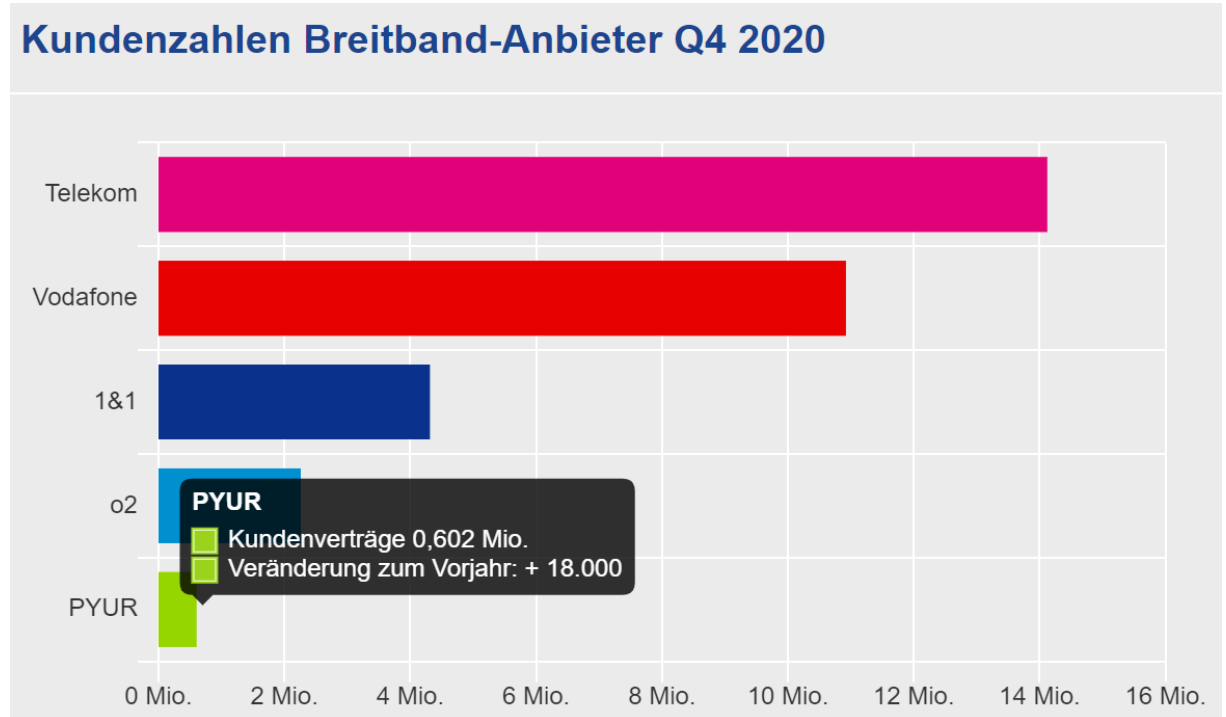


Abb. 1.3 Kundenzahlen Breitband-Anbieter Q4 2020 [3]

Wie aus der Abb. 1.3 zu ersehen ist, ist PYUR nicht führend, aber es erhält jedes Jahr Tausende von neuen Anfragen für neue Verbindungen.

Daher ist es für PYUR sehr wichtig, den Abstand zu den Wettbewerbern zu minimieren und sich stärker an der Digitalisierung zu beteiligen.

Dies bedeutet, dass viele Kunden an Internetverbindungsdiensten interessiert sind.

Es gibt jedoch noch viele Orte, an denen keine Leitungen verlegt sind. Oft befinden sie sich in Dörfern, Weilern oder abgelegenen Orten in Städten.

Darüber hinaus sind wir nicht an jeder möglichen Verbindung interessiert, sondern an der bestmöglichen Verbindung.

### 1.1 Problemstellung (Soll-Ist Analyse)

Ein wichtiger Geschäftsprozess, der sich auf die Gesamtqualität des Geschäfts auswirkt, ist die digitale Ermittlung neuer Verbindungen. Der Zweck dieses Geschäftsprozesses ist

es, Adressen für Verbindungen zu sammeln und Verbindungen zu bestehenden Verbindungen (Pfaden) herzustellen.

Das in Entwicklung befindliche System muss ein anderes ersetzen, da es die Anforderungen des Geschäftsprozesses nicht ausreichend erfüllt. Dies ist die Itinero-Bibliothek. [14]

Der Funktionsumfang dieser Bibliothek ist sehr begrenzt. Sie bietet nur Pfade für eine mögliche Adresse. Wenn wir also Pfade für viele Adressen erstellen wollten, müssten wir Pfade für jede Adresse einzeln erstellen, nach einander.

Dies hat eine Reihe von Nachteilen: zumindest ist dieser Ansatz aus Sicht des Nutzers nicht optimal. Der Benutzer müsste immer wieder die gleichen Arbeitsschritte durchführen, um die Aufgabe zu erledigen.

Der zweite Aspekt, der sich aus dem ersten Punkt ergibt, sind die Zeitkosten. Da die Leistung der Lösung selbst nicht die beste ist, wird die Bearbeitung jeder einzelnen Adresse viel Zeit in Anspruch nehmen und somit unerwünschte Kosten für das Unternehmen verursachen.

Und der dritte und fatalste Punkt ist, dass die Pfade, die für einzelne Adressen gefunden werden, brauchen mehr Kosten, als wenn die Pfade für eine bestimmte Gruppe von Adressen, die zuerst gefunden würden. Mit anderen Worten: Die Summe der einzelnen berechneten Pfadlängen ist in der Regel größer als die Summe der mit Blick auf die Gesamtheit der Pfadlängen von Adressen.

All diese Nachteile des aktuellen Stands zwingen uns, eine neue Anwendung zu entwickeln, die den Anforderungen besser entspricht.

## **1.2 Ziel der Arbeit**

Die Hauptaufgabe der Arbeit besteht darin, den Algorithmus und die Architektur der Anwendung zu durchdenken und die beste Lösung für das Problem der optimalen Anlegung neuer Verbindungen zu finden. Die erste besteht darin, die bestehenden Funktionen in kürzerer Zeit auszuführen, und die zweite darin, optimale Pfade für mehrere Adressen in einer einzigen Sitzung zu ermöglichen.

Relevantes Teilproblem besteht darin, einen optimalen Routing-Engine für unsere App auszusuchen und zu adaptieren.

Zentrale Frage besteht darin, was für ein Algorithmus soll man nutzen, um ein optimales und ausgeglichenes Ergebnis zu bekommen.

Mit der Firma wurde besprochen, dass das Prozess muss insgesamt nicht mehr als 5 Minuten betragen. Als die maximale Punkteanzahl wird 50 angenommen.

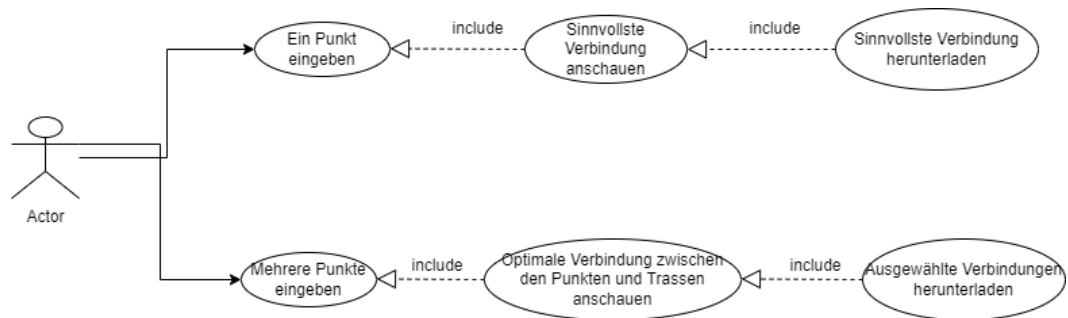


Abb. 1.4 Use-Case Diagramm

### 1.3 Methodik

Das Problem lässt sich dadurch lösen, dass man zunächst ein Überblick über verschiedene Routing-Engines macht und testet, wie es für die Anwendung passt.

Für Algorithmus-Problem, soll man zunächst die Aufgabe formalisieren, also klar mathematisch definieren, welche Lösung gesucht wird und wie es gesucht werden muss. Also, auf welches algorithmische Problem läuft unser Problem hinaus bzw. auf welches bestehende Problem kann es reduziert werden?

### 1.4 Gliederung und Aufbau

Die Arbeit wird folgendermaßen strukturiert:

In der Einführung es werden die erforderliche Grundlagen genannt. Sie müssen erwähnt werden, weil das Thema ziemlich eng zum GIS-Kontext passt. Außerdem in der Arbeit werden häufig die Begriffe auftauchen, die intern in der Firma genutzt werden (z.B. Trassensegment).

Danach wird die Herangehensweise zur Problemlösung allgemein vorgeschlagen. Das Problem wird in bestimmte Unterprobleme eingeteilt.

Nachdem die Konkrete Punkte zum Lösen genannt sind, kommt der Kern der Arbeit, nämlich die Kapitel, wo jedes Problem grundsätzlich betrachtet und analysiert ist. Es wird gezeigt, wie im Laufe der Entwicklung verschiedene Problem-Lösungs-Ansätze gewechselt wurden.

Sobald es gezeigt wird, wie die Probleme gelöst werden, kommt der Kapitel namens „Implementierung.“ Es dient dafür, um zu zeigen, wie die im vorherigen Kapitel genannte abstrakte Lösungsansätze im Quellcode umgesetzt werden können. Ohne das jede Zeile des Quellcodes penibel zu analysieren, werden die wichtigsten Schlussfolgerungen gezeigt und untermauert.

## 1.5 Grundlagen

In dem Kapitel werden Grundlagen für diese Arbeit genannt.

Es ist wichtig, weil ohne expliziter Akzent auf diese Begriffe es konnte für einen Leser oder Begutachter zu Missverständnisse kommen. Es ist auch interessant, weil die Branche konnte einigermaßen spezifisch sein, und es können einige interessante Punkte aufkommen.

Der Inhalt dieses Kapitels verknüpft sich mit dem vorhergehenden Kapitel dadurch, dass die Begriffe und Einzelheiten, die schon genannt wurde, werden jetzt ausführlich geklärt.

## 1.6 Grundbegriffe

Mögliche Knoten:

- **Haus** – gilt als ein Endpunkt.
- **Verteiler** (z.B. Y-Verteiler) – ein Punkt, wo ein Trassensegment geteilt werden kann und wo anders gerichtete Trassensegmente erzeugt werden.
- **Schacht** – ein Ort, wo ein Punkt (Muffe, Verteiler) tief gebaut ist.

Mögliche Kanten:

- **Trassensegment/Lines(Linien)** – eine gerade Verbindung zwischen zwei Knoten.
- **Trasse** – eine Sammlung von Trassensegmenten.

**Linestring** – ein Datentyp in SQL Server, wo die raumbezogenen Daten gespeichert werden können.

**Luftlinie** – der minimale Abstand zwischen zwei Punkten. In der Praxis kommt normalerweise nicht vor, da viele Hindernisse sich auf dem Weg befinden.

**Gerouteter Weg** – der minimale Abstand zwischen zwei Punkten, der in der Praxis realisierbar ist, also mit der Rücksicht von aller möglichen Hindernissen.

**Routing Engine** – eine Software, die in unserem Kontext die optimale Verbindung zwischen zwei Punkten in der angegebenen Reihenfolge zur Verfügung stellt. Die Ausgabe wird normalerweise in JSON Format dargestellt und enthält eine Sammlung von Punkten, die leiten aus Punkt A nach Punkt B. Nach Bedarf können zusätzliche nützliche Informationen gelistet werden wie Länge einzelner Abschnitte, Anzahl von Schritten, Annotationen (Metadaten für jede Koordinate entlang der Streckengeometrie), alternative Wege usw. [4]

## Zusammenfassung

In diesem Kapitel wurden die Gründe gelistet, warum so ein Problem gelöst werden muss. Diese Motivation lässt sich durch zwei verschiedenen Perspektiven betrachten.

Erste – innerliche Motivation, also, die Begründung durch die Digitalisierung. Es ist ein sehr wichtiges Thema und wir müssen anstreben, Digitalisierung weiterverbreiten.

Wurde gezeigt, dass unser Thema – Verlegung neuer Anschlüsse – ein wichtiger Aspekt davon ist und große Bedeutung hat.

Der zweite Aussichtspunkt – Interesse des Unternehmens. Wurde bewiesen, dass eine optimale Verlegung neuer Anschlüsse, ein wichtiger Geschäftsprozess ist und soll den Abstand bis Konkurrenten minimieren.

Ein Ansatz zur Problemlösung wurde dargestellt. Die Grundbegriffe wurden gelistet.

## 2 Methodik und Vorgehen

### 2.1 Problemlösungsansatz

In diesem Kapitel werden alle Probleme der Arbeit allgemein und ohne Einzelheiten vorgestellt. Vor allem, wie überhaupt so ein Problem lässt an sich heranschieben.

Begonnen mit der Eingangsdaten unter Rücksicht der Anforderung seitens der Firma wird nach und nach gezeigt, wie so ein Problem sich lösen lässt.

Zunächst wird dem Leser gezeigt, welche Struktur die Eingangsdaten haben und wie man damit umgehen und in dem Kode einsetzen kann, welche Unterschiede gibt es bei den Berechnungen zwischen flacher und gebogener Oberfläche, anschließend wird ein optimaler Routing-Engine ermittelt und abschließend das Problem wird mathematisch betrachtet und formuliert. Es wird gezeigt zu welchem algorithmischem Problem das Problem lässt sich ableiten und welche Algorithmen gibt es, um das Problem zu lösen.

Das Problem ist komplex. Das heißt, es mussten andere Unterprobleme gelöst werden, bevor dir gezeigt wird.

Listung einzelner Problemen, die auf dem Weg zur Lösung entstanden sind.

1) Auswahl einer Routing-Engine.

während die Anwendung läuft, wird ständig nach Verknüpfungen gesucht. Dafür muss dieser Prozess sehr schnell ablaufen, damit der ganze Algorithmus nicht zu langsam ist. Im Internet kann man viele Open-Source Lösungen finden, die das machen. Um eine Richtige Lösung zu treffen, muss man verschiedene Kriterien vergleichen.

2) Struktur von Eingangsdaten erfassen

Die von Firma zur Verfügung gestellte Daten über Trassensegmenten werden auf dem SQL Server gespeichert. Die Daten werden in hexadezimaler Form dargestellt. Es sollten zusätzliche Manipulationen ausgeführt werden, um mit den Daten umgehen zu können. Weiter werden die Einzelheiten erklärt.

3) GIS Grundlagen

Die Erde ist eine Kugel (Ein Geoid). Das heißt, herkömmliche Ansätze und Formeln zur Berechnung werden einigermaßen Ungenauigkeiten aufweisen. Dafür muss man auf die gebräuchliche euklidische Geometrie verzichten und andere Ansätze nutzen.



4) Formalisieren der Aufgabe, Auswahl vom Algorithmus, Zersplitterung von Lines, Durchführung der Versuche

In diesem Abschnitt wird die Aufgabe formalisiert. Dafür werden verschiedene algorithmische Problemen betrachtet und das Passende ausgewählt. Dann werden die möglichen Lösungen dieses Problems auch betrachtet. Abschließend wird das gefundene Algorithmus so angepasst, dass die entstehende Lösung unseren Zwecken entspricht.

## 2.2 Auswahl einer Routing-Engine.

Zunächst betrachten wir die am häufigsten genutzte Routing-Engines.

Einige Bemerkungen zur Tabelle:

- Je mehr Sternen desto besser
- Activity – dieser Wert spiegelt Wartungsfreundlichkeit und Engagement von Benutzern wieder
- Performance – ein subjektiver Wert, der stellt dar, wie schnell die Api-Aufrufe ausgeführt werden können
- RAM Requirements zeigt wie viel Arbeitsspeicher für Build und Load einzelner Lösung gebraucht wird (nicht Echtzeit)
- Customizability – inwiefern lässt sich die Lösung anpassen
- Flexibility zeigt wie leicht lassen sich die einzelnen Api-Aufrufe anpassen (z.B. Autobahne vermeiden usw.)

	OSRM	Graphhopper	Valhalla	Openrouteservice
<b>Profiles</b>				
Car	✓	✓	✓	✓
Truck	✗	✗	✓	✓
Pedestrian	✓	✓	✓	✓
Bike	✓	✓	✓	✓
Transit	✗	✓	✓	✗
Others	-	other bikes motorcycle hiking wheelchair	taxi bus scooter motorcycle	other bikes wheelchair hiking
<b>API</b>				
Routing	✓	✓	✓	✓
Isochrones	✗	✓	✓	✓
Matrix	✓	✗	✓	✓
Map Matching	✓	✓	✓	✗
Optimization	✗	✗	✓	✗
<b>Features</b>				
Turn restrictions	✓	✓	✓	✗
Avoid locations/polygons	✗	✓	✓	✓
Dynamic vehicle attributes	✗	✓	✓	✓
Alternative routes	✓	✓	✓	✓
Round trips	✗	✓	✗	✓
Time awareness	✗	✗	✓	✗
Elevation	✗	✓	✓	✓
<b>Activity</b> <sup>2</sup>	***	*****	*****	*
<b>Performance</b> <sup>3</sup>	*****	****	***	****
<b>RAM requirements</b> <sup>4</sup>	*	***	****	*
<b>Customizability</b> <sup>5</sup>	***	***	**	*
<b>Flexibility</b> <sup>6</sup>	*	***	****	*

Abb. 2.1 Vergleich von Routing-Engines [5]

Die zeitlichen Anforderungen laut der Forma sind:

- max. 2 Sekunden um den Weg für eine Adresse zu finden
- max. 5 Minuten um den Weg für viele Adressen zu finden.

Aus diesem Grund wurde Performance als ein wichtiges Kriterium gewählt.

Um Kosten zu minimieren, soll man Arbeitsspeicherverbrauch minimieren. Darum möchten wir auch eine Lösung haben, die den Arbeitsspeicher am wenigsten verbraucht.

Laut der Firma, müssen nur die Wege betrachtet werden, die für Fußgänger erreichbar sind. Deswegen für das Routing wird Pedestrian(Fußgänger) Profil genutzt.

Unterstützung von Routing für Fußgänger ist auch ein Muss für eine passende Bibliothek.

Wenn man die Voraussetzungen vergleicht, wird klar, dass die OSRM Bibliothek am besten für unsere Zwecke passt.

Am Anfang der Entwicklung wurde diese Bibliothek eingesetzt aber es stellte sich heraus, dass sie gravierende Nachteile aufweist. Vor allem - manchmal konnte die Bibliothek den Weg zwischen zwei Punkten nicht aufbauen. Laut der Versuchen, es kommt häufig vor, wenn zwischen Punkten Abstand mehr als 1 km ist und dazwischen Autobahnen sich befinden. Nach der Suche nach Problemlösung wurde klargeworden, dass es ein Bug ist.

Bemerkung: der Bug entstand nur wenn die Bibliothek auf Basis von der Karte des ganzen Deutschlands gebaut wurde([germany-latest.osm.pbf](#)). Wenn man aber den Weg für zwei gleiche dieselben Punkten gebaut hat mithilfe von OSRM Bibliothek, die auf Basis von der Karte des einzelner Bundeslandes (z.b. [sachsen-anhalt-latest.osm.pbf](#)) gebaut wurde, dann aber konnte dieser Weg gefunden werden.

So eine Situation hat gezwungen, den Ausweg zu suchen. Zunächst wurde beschlossen, dass wir die OSRM Bibliothek weiter benutzen und dann müssen aber irgendwie so eine Situation lösen können, wenn Punkt A in Bundesland A' liegt und Punkt B in Bundesland B'.

## 2.3 Overpass Api

Als ein möglicher Ausweg konnte man Folgendes machen. Es gibt eine Open-Source Lösung namens Overpass API. Es dient dafür, nützliche und sinnvolle Informationen

aus der OSM-Dateien zu extrahieren. Zum Beispiel, wenn wir alle Bushaltestellen in bestimmtem Bereich bekommen möchten, könnten wir den folgenden Ausdruck benutzen. (Abb 2.2)

```
node(50.745,7.17,50.75,7.18)[highway=bus_stop];  
out;
```

Abb. 2.2 Die Sprache, die bei Abfragen benutzt wird heißt - Overpass QL(Query Language).

Bei unserer Situation konnte so eine Lösung sehr hilfreich sein. Alle möglichen Punkte, zu denen man den Weg anlegen konnte, könnten mittels der Lösung gefunden werden. Das vorgeschlagene Verfahren für ein sicheres Aussuchen der gerouteten Wege sah so aus. (Abb. 2.3)

1. Zunächst wird die Luftlinie zwischen Punkten A und B halbiert.
2. Innerhalb vom bestimmten Radius R werden mithilfe Overpass die Punkte ausgesucht, mit denen es möglich ist, eine Route zu bauen.
3. Aus diesen Punkten wird ein Punkt C ausgesucht, so dass die Summe von Luftlinien von Punkt A nach Punkt B und von Punkt C nach Punkt P minimal ist.
4. Gerouteter Weg von C nach B wird gespeichert.
5. Der ganze Vorgang wird wiederholt mit Punkt D und der geroutete Weg von A nach D wird gespeichert.
6. Der Prozess wiederholt sich bis die Länge der Luftlinie zwischen Punkten weniger einiger bestimmten Länge ist. (ein Meter war ein Vorschlag). Hier – Punkte F und R.
7. Dann die Punkten F und E werden verknüpft (weil der Abstand so wie so sehr klein ist) und somit wird der ganze geroutete Weg von A nach B gebaut.

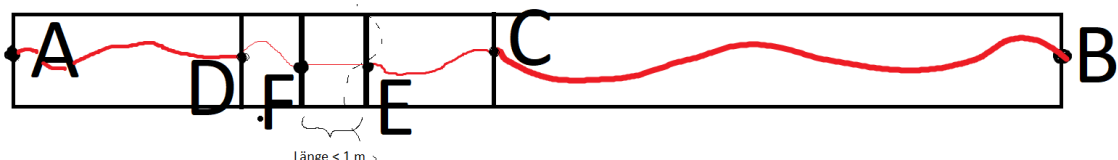


Abb. 2.3 Algorithmus zum Lösen des Problems der Verbindungen zwischen verschiedenen Bundesländern. Strichlinie – die Grenze zwischen Bundesländern. Rote Linie – gerouteter Weg. A und B – Anfangspunkte. C, D, E, F – Zwischenpunkte.

Es fällt auf, dass die Lösung ziemlich unsicher aussieht. Vor allem, Es kann Situationen geben, wenn die Knoten um Wege zu bauen nicht gefunden werden können. Außerdem die Komplexität der Lösung wäre gestiegen.

Aus diesem Grund wurde beschlossen anstatt OSRM eine andere Bibliothek zu nutzen.

## 2.4 Graphhopper als eine alternative Routing Engine

Auf die Website [6, 7] wird behauptet, es sei eine schnelle und Open-Source-Routing-Engine. Wegen Apache-Lizenz ist es freundlich für Geschäftsintegration. Es ist in Java geschrieben und hat eine API-Schnittstelle, die die Integration im Projekt erleichtert. Die Aufrechterhaltung und Community der Bibliothek ist wirklich aktiv. Auf dem Portal „Routexl.com“ [6] wurde ein Vergleich zwischen OSRM, Graphhopper und Gosmore durchgeführt.

```
Total legs: 10.000
Total time Gosmore: 671.423sec OSRM: 28.682sec GraphHopper: 23.586sec
Success Gosmore: 97.6% OSRM: 98% GraphHopper: 99.1%
Too slow Gosmore: 0% OSRM: 0% GraphHopper: 0%
Too fast Gosmore: 0% OSRM: 0% GraphHopper: 0%
```

Abb. 2.4 Vergleich von OSRM und GH

Wie man sieht, diese Lösung ist schneller als OSRM.



Abb. 2.5 Die Unternehmen, die GH benutzen [8]

Außerdem GH wird bei sehr bekannten Unternehmen benutzt wie „Deutsche Bahn“, „Flixbus“, „Komoot“ und andere. D.h. die Bibliothek hat eine gute Qualität und funktioniert sicher.

## 2.5 Struktur von Eingangsdaten

Abb. 2.6 So werden Geodaten in Datenbank gespeichert.

```

1 SELECT TOP (1000) [geo].[STAsText()]
2 FROM [master].[dbo].[Geos]
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
10
```

Dieser Befehl wird weiter in der Anwendung benutzt, um die Geodaten zu bekommen.  
(Abb. 2.8)



Abb. 2.8 Repräsentierung von Geodaten in SQL Server Management Studio

Auf dem folgenden Bild wird die Häufigkeitsverteilung von Trassenlängen gezeigt.  
(Abb. 2.9)

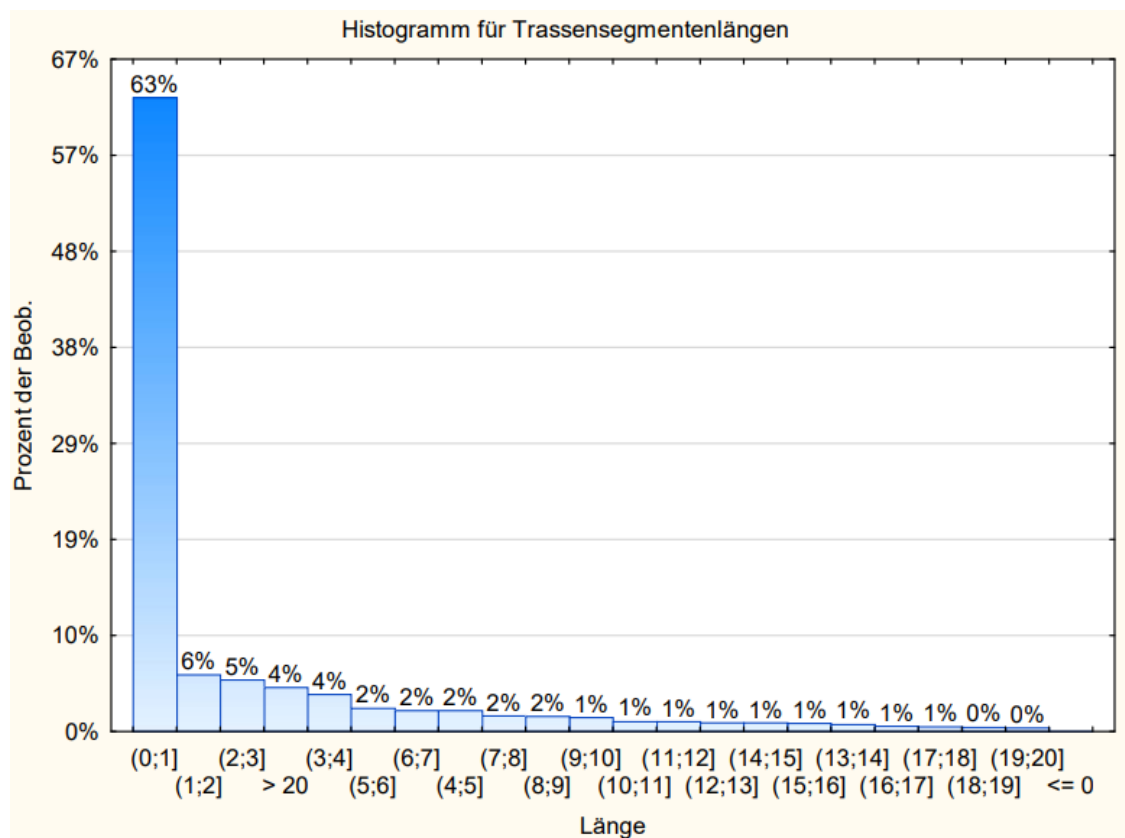


Abb. 2.9 Häufigkeitsverteilung von Trassensegmenten

Diese Verteilung spielt eine große Rolle aus zwei Gründen. Erstens, es zeigt die innerliche Struktur von unseren Geodaten. Zweitens, man merkt, dass mehr als die Hälfte von Trassensegmenten hat die Länge weniger als ein Meter. Ursprünglich bei der Entwicklung die Fähigkeit, Trassensegmenten in kleinere Segmenten zu teilen war ein wichtiger Aspekt.

Vor allem, weil am besten wäre es, wenn man ein Trassensegment mit einem Punkt verbindet, so dass die minimale Länge erreicht wird. Um dieser Punkt eindeutig entlang der Trasse zu finden, kann man theoretisch versuchen, jeden Punkt der Trassen mit dem Zielpunkt zu verbinden. Es ist aber sehr aufwendig von Komplexität her.

### **Zerlegung jeder Meter**

Eine mögliche alternative wäre es, die Trassen zu teilen. Also, eine Trasse als eine Sammlung von Punkten bestimmter Länge zu betrachten. Die Idee war, diese optimale Zerlegungsgrad zu finden, aber es sollte ausgeglichen erfolgen: die Trassen sollten in möglichst kleinere Stücke geteilte werden. Die Anzahl der Punkte bei der Zerlegung muss aber auch nicht zu groß sein, um Ressourcen nicht allzu verschwenden.

Es wurden 130 zufällige Trassen ausgewählt, und jede Trass hatte einen separaten Punkt, zu dem der Mindestabstand gezählt werden musste. Für jede Trasse wurden 14 Unterteilungen vorgenommen: alle Meter, alle zwei Meter, alle drei Meter und so weiter bis zu allen 14 Metern. Somit gab es 130x14 verschiedene Unterteilungen. Für jede Unterteilung wurden Messungen durchgeführt, um die Mindestentfernung für die Verbindung zu dem jeweiligen Punkt sowie die Anzahl der daraus resultierenden Segmente zu ermitteln. Anschließend wurden alle Unterteilungsfälle nach dem Grad ihrer Unterteilung sortiert. In der ersten Kategorie wurden zum Beispiel alle Linesplits pro Meter durchgeführt, in der letzten Kategorie alle 14 Meter. Anschließend wurde der Durchschnitt der kürzesten Entfernung zu einem einzelnen Punkt und die Anzahl der daraus resultierenden Segmente berechnet.

Die Ergebnisse des Experiments sind auf Abb. 2.10 zu sehen.



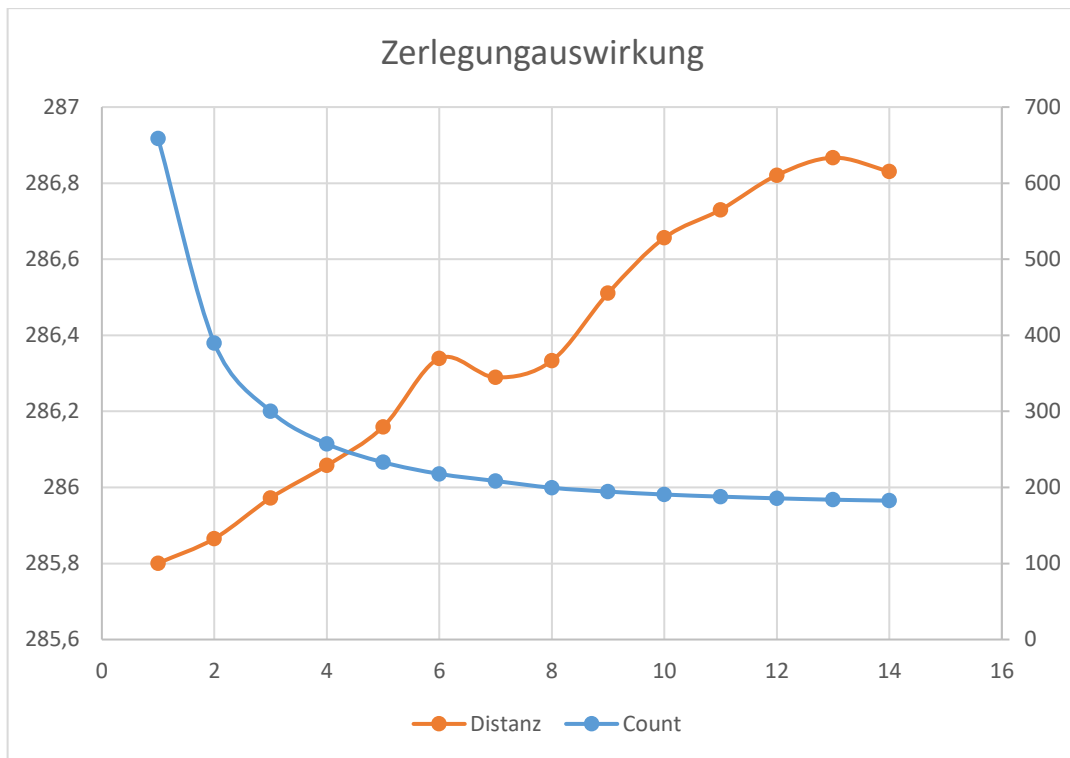


Abb. 2.10 Wie ein Gleichgewicht in Zerlegung erreicht wird

Wie man auf der Abb. 2.10 sieht, die Zerlegung einer Trasse jeden 1 Meter ergibt durchschnittlich 658,9 neue Punkte und der durchschnittliche minimale geroutete Weg ist 285,8 Meter. Bei der Zerlegung jede 14 Meter wiederum, wurden ungefähr 182,55 Punkte entstanden und die Länge beträgt 286,8.

Es wurde beschlossen für weitere Arbeit die Zerlegung jede 4 Meter zu machen, weil, wie aus auf der Abb. 2.10 zu sehen ist, ein Gleichgewicht zwischen Performance und Genauigkeit erreicht wird.

### Iterative Halbierung

Ein anderes Verfahren, der betrachtet wurde, war die iterative Halbierung. Der Sinn dieser Halbierung besteht darin, nicht eine allgemeine optimale Zerlegungswert zu finden, sondern jede Trasse so lange zu halbieren, bis die gewünschte Genauigkeit erreicht wird.

Wegen der zeitlichen Begrenzung konnte die gewünschte Qualität nicht erreicht werden.

Nachdem der Algorithmus letztendlich angepasst wurde und Komplexität reduziert wurde, wurde eindeutig gewesen, dass diese Zerlegung zurzeit nicht aktuell ist. Außerdem es wurde keine gut optimierte Zerlegung entworfen, die wirklich gut von Performance her war. Es wurde beschlossen die Zerlegung nicht durchzuführen und als eine weitere mögliche Verbesserung wahrnehmen. Bei der letztendlichen Version der App anstatt jeglicher Zerlegung wurde beschlossen, die Trassen mit den Punkten so zu verbinden: wurde versucht jeden Punkt mit Anfangspunkt der Trasse zu verknüpfen und die Trasse wurde verbunden mit dem Punkt, bei dem die Länge der Verknüpfung zu Anfangspunkt der Trasse minimal ist.

Aber das war eine Unterlassung meinerseits. Die Tatsache, dass wenn die Längen der Trassen gemessen wurden, die Trassen wurden schon jedes Mal zerlegt, wo es ein Knick gibt. Diese Zerlegung bei jedem Knick war damals nützlich, wenn die tatsächliche Anzahl der Geraden, aus denen ein Trassensegment entsteht, gemessen sollte. Und da es schon auf jegliche Zerlegung verzichtet wurde, liefert diese Graphik keine aktuellen Informationen über Trassen.

Aus diesem Grund bei der letztendlichen App sind die Abstände zwischen Trassen und dem Punkt, mit dem die Trasse verbunden ist, manchmal nicht möglichst klein. (Abb. 4.2)

## **2.6 Berechnungen bei der gebogenen Oberfläche**

Wenn es um GIS-Projekt handelt, man muss nicht vergessen, dass die Erde nicht eine Fläche ist. Es gibt eine andere Art von Geometrie, und zwar, Elliptische Geometrie (Sphärische Geometrie), die mit Figuren wie Kugel besser handelt, da es um eine positive Krümmung geht. Dabei wird angenommen, dass parallele Geraden schneiden sich.

In der Praxis (und wie es in der Firma vorbestimmt wurde) wird häufig eine vereinfachte Formel benutzt, um die Geodaten präzise auszuwerten. Es ist Haversine Formel [9]. Die Haversinus-Formel bestimmt die Großkreisdistanz zwischen zwei Punkten auf einer Kugel unter Berücksichtigung ihrer Längen- und Breitengrade.

$$\begin{aligned}
d &= 2r \arcsin \left( \sqrt{\text{hav}(\varphi_2 - \varphi_1) + (1 - \text{hav}(\varphi_1 - \varphi_2) - \text{hav}(\varphi_1 + \varphi_2)) \cdot \text{hav}(\lambda_2 - \lambda_1)} \right) \\
&= 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\varphi_2 - \varphi_1}{2} \right) + \left( 1 - \sin^2 \left( \frac{\varphi_2 - \varphi_1}{2} \right) - \sin^2 \left( \frac{\varphi_2 + \varphi_1}{2} \right) \right) \cdot \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right) \\
&= 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\varphi_2 - \varphi_1}{2} \right) + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right).
\end{aligned}$$

Abb. 2.11 Die Formel zur Berechnung den Abständen auf einer gebogenen Oberfläche.

$\varphi_1$  und  $\varphi_2$  sind Breiten von bzw. Anfang- und Endpunkte,  $\lambda_1$  und  $\lambda_2$  sind Längen von Anfang- und Endpunkte.

So eine einfache Formel lässt präzise Berechnungen bei der Erde durchführen.

## 2.7 Algorithmus und Versuche

Seitens Firma wurde hingewiesen, dass das gestellte Problem als minimaler Spannbaum betrachtet werden muss. Aus folgenden Gründen: laut der Aufgabestellung wir sollen ein System bauen, dass sich die minimale bei der Verbindung vieler Punkten ergibt. Das ganze Netz von Punkten kann als ein Graph betrachtet werden, wobei die Punkte als Knoten gelten. Es gibt aber eine Bemerkung: wir müssen ja nämlich die Punkte miteinander verbinden, aber letztendlich dar ganze Graph muss noch mit einem Trassensegment verbunden werden. D.h. zu diesem Netz muss auch ein Punkt aus einem Trassensegment gehören. Weiter wird gezeigt wie das Problem gelöst wurde.

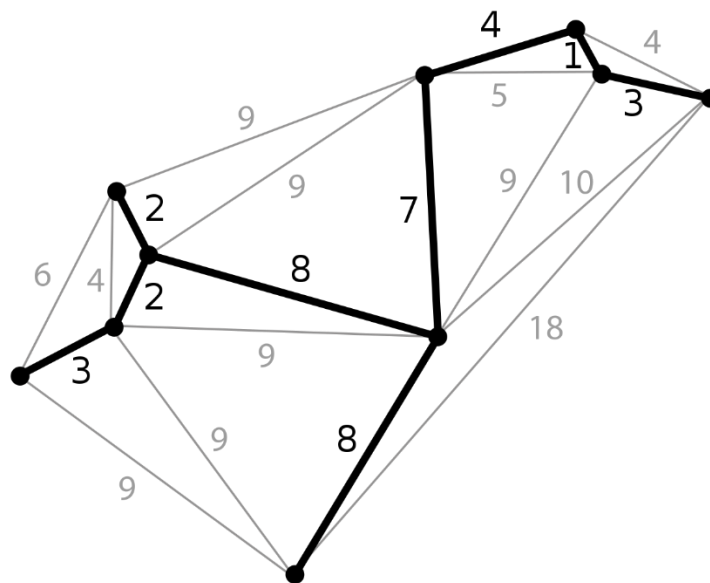


Abb. 2.12 Wie ein Spannbaum aussieht

Dann passt es zu minimalem Spannbaum. Wir möchten alle Knoten so verbinden, wobei minimaler Gewicht der Graph erreicht wird.

## 2.8 Formale Beschreibung des minimalen Spannbaums [10]

Nehmen wir an, dass  $G = (V, E, w)$  ein gewichteter Graph ist. Die Funktion  $w$  wird jeder Kante  $e \in E$  zugewiesen und weist ein reellwertiges Gewicht  $w(e)$  auf. Dann wir können ein Baum  $T$  betrachten, wobei  $T = (V, A, w)$  so, dass  $A \subseteq E$  und  $|A| = |V| - 1$ . Der Baum  $T$  wird dann als Spannbaum von  $G$  gekennzeichnet. Vorausgesetzt ist, dass  $G$  zusammenhängend ist. [10]

Wir können das Gewicht des Spannbaums  $w(T)$  durch  $w(T) = \sum_{e \in A} w(e)$  definieren. Also, ein Spannbaum  $T = (V, A, w)$  abgeleitet von  $G$  wird dann minimal sein, wenn es kein Spannbaum weniger Gewichtes gibt, oder anders ausgedrückt, falls  $w(T) \leq w(T')$  für alle mögliche Spannbäume  $T' = (V, A', w)$  abgeleitet von  $G$ .

Zur bekanntesten Algorithmen, die minimales Spannbaum lösen, gehören [11]:

- 1) Prim Algorithmus -  $O(v^2)$  - bei Adjazenzmatrix oder  $O(e + v \log v)$  bei Fibonacci-Heap.
- 2) Kruskal Algorithmus -  $O(e \log v)$ , wo  $e$  – die Anzahl von Kanten,  $v$  – Anzahl von Knoten.

## 2.9 Prim Algorithmus

Prim Algorithmus wird mit Auswahl eines Startknotens angefangen. Danach es wird sich verzweigen vom schon konstruierten Baum durch Auswahl neuer Kante und Knotens bei jeder Iteration. Die neue Kante verbindet den neuen Knoten mit vorherigem Baum. [11]:

Während Algorithmus ausgeführt wird, alle Kanten in Graph können in 3 Kategorien geteilt werden:

- Baumknoten - befinden sich schon im Baum
- Randknoten (fringe vertices) – nicht im Baum, aber angrenzend an einigen Knoten des Baums
- Ungesehene Knoten – alle anderen

Der Schlüsselmoment des Algorithmuses ist die Auswahl eines Knotens von Randkanten. Prim Algorithmus sucht immer einen Knoten minimaler Gewicht.

Der Algorithmus wird folgendermaßen ablaufen:

$\text{primMST}(G, n)$ :

Alle Knoten als „ungesehen“ markieren.

Willkürliche Auswahl eines Startknotens S. Der Knoten als Baumknoten markieren.

Alle Knoten, die angrenzend an dem Knoten S sind, als Randknoten markieren.

Bis es gibt Randknoten:

Auswahl eines Knotens minimaler Gewicht zwischen dem Baumknoten t und Randknoten v

V als Baumknoten markieren; Kante t zum Baum hinzufügen.

Alle Ungesehene Knoten angrenzend an v als Randknoten markieren.

### **Algorithmus Schema. Prim**

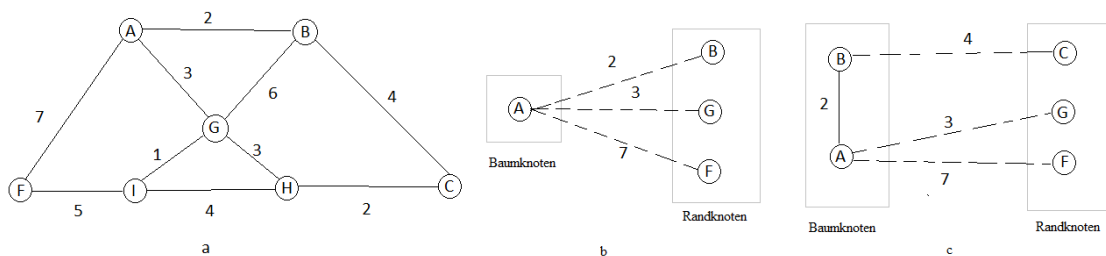


Abb. 2.13 Die Erklärung von Prim Algorithmus

Es wird eine Iteration gezeigt. Solide Linien zeigen die Verbindungen im Baum, gestrichelte – Kanten zu Randknoten. In a ist ein gewichteter Graph gebildet. In b werden das Baum und Randkanten gebildet nachdem Startknoten ausgewählt ist. In c wird gezeigt, dass nach der Auswahl von Kanten und Knoten BG ist nicht mehr zu sehen, da AG hat weniger Gewicht um G zu erreichen.

## **2.10 Kruskal Algorithmus**

Die allgemeine Vorgehensweise sieht so aus: bei jedem Schritt der Algorithmus wählt die verbleibende Kante mit dem niedrigsten Gewicht aus dem Graph aber verwirft die

Kanten, die einen Kreislauf mit schon ausgewählten Kanten macht [11]. Zu jedem Zeitpunkt die ausgewählten Kanten bilden ein Wald und nicht unbedingt ein Baum. Algorithmus endet, sobald alle Kanten gesehen wurden.

Der Algorithmus wird folgendermaßen ablaufen:

kruskalMST(G, n)

R=E// R – gebliebene Kanten

F=  $\emptyset$  //F ist ein Wald der Kanten

While (R ist nicht leer)

Die kürzeste Kante vw aus R verwerfen

If(vw macht keinen Kreislauf)

vw zu F hinzufügen

return F

### *Algorithmus Schema. Kruskal*

Es gibt mehrere Beispiele, wie der Prim Algorithmus angewendet werden kann. Es wurde beschlossen die Lösung mithilfe des Algorithmuses zu entwickeln. Als Eingabe wird Adjazenzmatrix genutzt, d.h. die Komplexität wird  $O(v^2)$  betragen, wo v – Anzahl der Knoten

## 2.11 Klassendiagramm

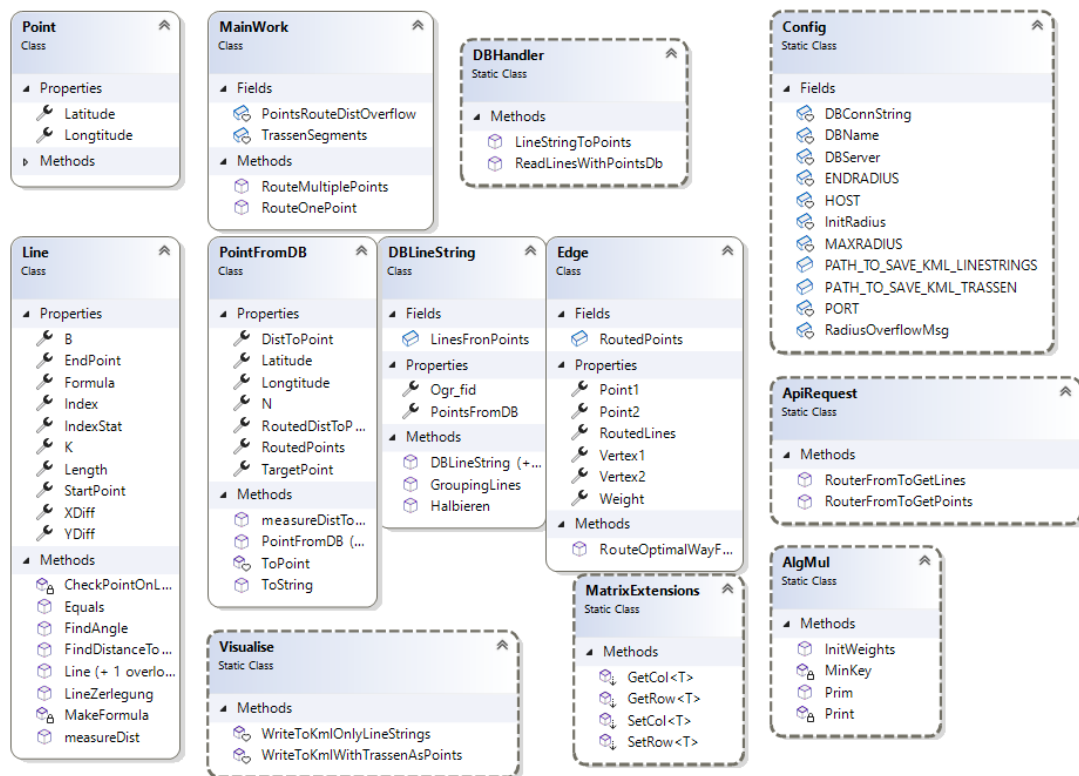


Abb. 2.14 Klassendiagramm

Während der Entwicklung wurden zahlreiche Klassen entworfen. Einige von Ihnen sind schon aussagekräftig (wie z.B. Config, ApiRequest, DBHandler, MainWork, Visualise, Point usw.).

Einige von Ihnen müssen ein wenig erklärt werden.

Line – es wurden am Anfang verschiedene Versuche durchgeführt, wobei die Linien als geometrische Objekten betrachtet wurden. Dafür waren Felder K und B nützlich, weil dabei konnten Sie eine Geradegleichung bilden( $y=Kx+B$ ). Aber am wichtigsten in dieser Klasse ist die Methode „FindDistanceToPoint“, weil es während der App-Ablauf, häufig die Abstände zwischen Punkten gemessen werden.

Edge – diese Klasse ist nützlich, wenn wir die gerouteten Wege schon gebaut haben.

Eine Instanz dieser Klasse beinhaltet Informationen über Anfangspunkt, Endpunkt und bestimmte Punkten oder Linien, die nach dem der Algorithmus seine Arbeit gemacht hat, die optimalste Verbindung zwischen diese zwei Punkten hat.

Algmul – die Klasse, die ein Graph in einen Minimalerspannbaum umwandelt, wobei die Methode Prim ausgeführt wird. Dafür müssen zunächst die Gewichte jedem Punkt zugewiesen werden. Dies macht Funktion InitWeights.

Die Klasse „MatrixExtensions“ ist eine Systemklasse (in C# als Erweiterungsmethode genannt), die erleichtert den Umgang mit anderen Datenstrukturen und fügt zusätzliches Funktionalität hinzu. Damit sind die Operationen mit Matrizen leichter aufzurufen.

## 2.12 Anfrage-Schnittstellen

```
http://localhost:8989/route?
point=52.533166321%2C13.40241577046
&point=52.52502065258%2C13.45254083286
&profile=foot
&instructions=false
&points_encoded=false
```

Abb. 2.15 Wie die Anfrage bei Graphhopper aussieht

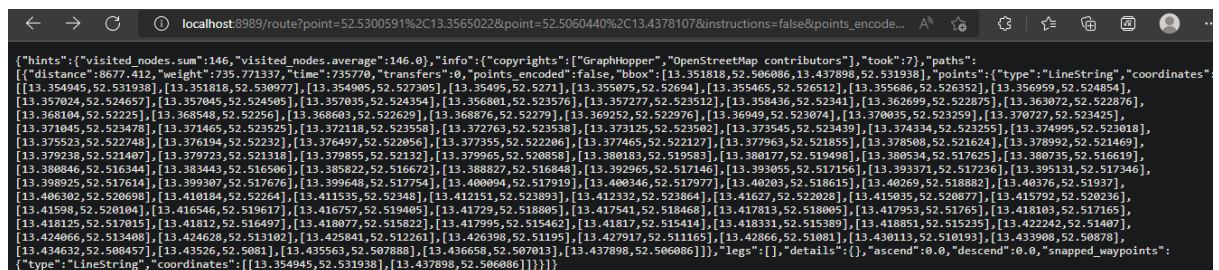
Point – Sammlung von Punkten, zwischen denen die Route gebaut werden muss. Es können auch mehr als 2 Punkten sein. Die Route wird in der Reihenfolge von angegebenen Punkten gebaut.

Profile – „foot“, da die Wege nur da gebaut werden müssen, wo es für einen Fußgänger erreichbar ist.

Instructions – dieser Parameter sagt, ob die Rückgabe die Anweisungen enthalten muss.

Points\_encoded – standardmäßig ist True. Dann werden die Rückgabedaten codiert und demzufolge weniger Ressourcen beim Transport verbrauchen. Wir aber brauchen die tatsächlichen Daten über Routing.

Antwort:



```
{
  "hints": {
    "visited_nodes.sum": 146,
    "visited_nodes.average": 146.0,
    "info": {
      "copyrights": [
        "GraphHopper",
        "OpenStreetMap contributors"
      ],
      "took": 7
    },
    "paths": [
      [
        [
          [
            13.354945, 52.531938, [13.351818, 52.530977], [13.354905, 52.527305], [13.354905, 52.5271], [13.355075, 52.52694], [13.355465, 52.526512], [13.355686, 52.526352], [13.356999, 52.524854], [13.357024, 52.524657], [13.357049, 52.524508], [13.357035, 52.524354], [13.356801, 52.523576], [13.357277, 52.523512], [13.358436, 52.52241], [13.364899, 52.522075], [13.363072, 52.522076], [13.368104, 52.52225], [13.368548, 52.52256], [13.368609, 52.522629], [13.368876, 52.52279], [13.369252, 52.522976], [13.36949, 52.523074], [13.370035, 52.523259], [13.370727, 52.523425], [13.371045, 52.523478], [13.371465, 52.523525], [13.372118, 52.523558], [13.372763, 52.523538], [13.373125, 52.523502], [13.373545, 52.523439], [13.374334, 52.523255], [13.374995, 52.523018], [13.375523, 52.522748], [13.376194, 52.52232], [13.376497, 52.522056], [13.377355, 52.522206], [13.377465, 52.522127], [13.377963, 52.521855], [13.378508, 52.521624], [13.378992, 52.521469], [13.379238, 52.521407], [13.379723, 52.521318], [13.379855, 52.52132], [13.379965, 52.520858], [13.380183, 52.519583], [13.380177, 52.519498], [13.380534, 52.517625], [13.380735, 52.516619], [13.380846, 52.516344], [13.383443, 52.516506], [13.385822, 52.516672], [13.388827, 52.516848], [13.392965, 52.517146], [13.393055, 52.517156], [13.393371, 52.517236], [13.395131, 52.517346], [13.398925, 52.517614], [13.399307, 52.517676], [13.399648, 52.517754], [13.400094, 52.517919], [13.400346, 52.517977], [13.40203, 52.518615], [13.40269, 52.518882], [13.40376, 52.51937], [13.406202, 52.520698], [13.410184, 52.52264], [13.411535, 52.52346], [13.412151, 52.523093], [13.41232, 52.523064], [13.41627, 52.522008], [13.415035, 52.520877], [13.415792, 52.520236], [13.41598, 52.520104], [13.416546, 52.519617], [13.416757, 52.519405], [13.41729, 52.518805], [13.417541, 52.518468], [13.417813, 52.518005], [13.417953, 52.51765], [13.418183, 52.517165], [13.418125, 52.517015], [13.41812, 52.516497], [13.418077, 52.515822], [13.417995, 52.515462], [13.41817, 52.515414], [13.418331, 52.515389], [13.418851, 52.515235], [13.422242, 52.51407], [13.424066, 52.513408], [13.424628, 52.513102], [13.425841, 52.512261], [13.426398, 52.51195], [13.427917, 52.511165], [13.42866, 52.51081], [13.430113, 52.510193], [13.433908, 52.50878], [13.434632, 52.508457], [13.43526, 52.5081], [13.435563, 52.507888], [13.436658, 52.507013], [13.437898, 52.506086]]],
        [13.354945, 52.531938], [13.437898, 52.506086]]
      ]
    ]
  },
  "distance": 8677.412,
  "weight": 735.771337,
  "time": 735770,
  "transfers": 0,
  "points_encoded": false,
  "bbox": [13.351818, 52.506086, 13.437898, 52.531938],
  "points": {
    "type": "LineString",
    "coordinates": [
      [
        [
          13.354945, 52.531938, [13.351818, 52.530977], [13.354905, 52.527305], [13.354905, 52.5271], [13.355075, 52.52694], [13.355465, 52.526512], [13.355686, 52.526352], [13.356999, 52.524854], [13.357024, 52.524657], [13.357049, 52.524508], [13.357035, 52.524354], [13.356801, 52.523576], [13.357277, 52.523512], [13.358436, 52.52241], [13.364899, 52.522075], [13.363072, 52.522076], [13.368104, 52.52225], [13.368548, 52.52256], [13.368609, 52.522629], [13.368876, 52.52279], [13.369252, 52.522976], [13.36949, 52.523074], [13.370035, 52.523259], [13.370727, 52.523425], [13.371045, 52.523478], [13.371465, 52.523525], [13.372118, 52.523558], [13.372763, 52.523538], [13.373125, 52.523502], [13.373545, 52.523439], [13.374334, 52.523255], [13.374995, 52.523018], [13.375523, 52.522748], [13.376194, 52.52232], [13.376497, 52.522056], [13.377355, 52.522206], [13.377465, 52.522127], [13.377963, 52.521855], [13.378508, 52.521624], [13.378992, 52.521469], [13.379238, 52.521407], [13.379723, 52.521318], [13.379855, 52.52132], [13.379965, 52.520858], [13.380183, 52.519583], [13.380177, 52.519498], [13.380534, 52.517625], [13.380735, 52.516619], [13.380846, 52.516344], [13.383443, 52.516506], [13.385822, 52.516672], [13.388827, 52.516848], [13.392965, 52.517146], [13.393055, 52.517156], [13.393371, 52.517236], [13.395131, 52.517346], [13.398925, 52.517614], [13.399307, 52.517676], [13.399648, 52.517754], [13.400094, 52.517919], [13.400346, 52.517977], [13.40203, 52.518615], [13.40269, 52.518882], [13.40376, 52.51937], [13.406202, 52.520698], [13.410184, 52.52264], [13.411535, 52.52346], [13.412151, 52.523093], [13.41232, 52.523064], [13.41627, 52.522008], [13.415035, 52.520877], [13.415792, 52.520236], [13.41598, 52.520104], [13.416546, 52.519617], [13.416757, 52.519405], [13.41729, 52.518805], [13.417541, 52.518468], [13.417813, 52.518005], [13.417953, 52.51765], [13.418183, 52.517165], [13.418125, 52.517015], [13.41812, 52.516497], [13.418077, 52.515822], [13.417995, 52.515462], [13.41817, 52.515414], [13.418331, 52.515389], [13.418851, 52.515235], [13.422242, 52.51407], [13.424066, 52.513408], [13.424628, 52.513102], [13.425841, 52.512261], [13.426398, 52.51195], [13.427917, 52.511165], [13.42866, 52.51081], [13.430113, 52.510193], [13.433908, 52.50878], [13.434632, 52.508457], [13.43526, 52.5081], [13.435563, 52.507888], [13.436658, 52.507013], [13.437898, 52.506086]]],
        [13.354945, 52.531938], [13.437898, 52.506086]]
      ]
    ]
  },
  "legs": [],
  "details": {},
  "ascend": 0.0,
  "descend": 0.0,
  "snapped_waypoints": [
    [
      [
        [
          13.354945, 52.531938, [13.437898, 52.506086]]
        ]
      ]
    ]
  ]
}
```

Abb. 2.16 Antwort vom Server



Wie es zu sehen ist, bei dem Schlüssel „points“ in der Antwort sind die geroutete Punkte zu sehen. Außerdem der Schlüssel „distance“ auch wichtig ist, weil damit die Länge des gerouteten Wegs zu sehen ist.

## **Zusammenfassung**

In diesem Kapitel wurde dem Lesern gezeigt, wie die in vorherigem Kapitel erwähnte Probleme gelöst wurden. Ein eigener Ansatz wurde entworfen und beschrieben, um die weitere Arbeit mit dem OSRM-Routing Engine fortzusetzen. Nach der Begründung der Schwierigkeit der Implementierung wurde ein anderes, aber nicht weniger von Performance her leistungsfähiger Routing-Engine vorgeschlagen. Den Auftrag wurde algorithmisch formalisiert und als ein minimaler Spannbaum betrachtet. Dazu wurden 2 Algorithmen zur Verfügung gestellt und letztendlich den Prim Algorithmus für die weitere Implementierung ausgewählt.

### 3 Implementierung

In diesem Kapitel kann der Leser über die notwendigen Implementierungsdetails erfahren: wie der Algorithmus in der App aussieht, welche Datenstrukturumwandlungen passieren. Dazu werden ein paar Versuche gezeigt, die illustrieren, was für ein Gleichgewicht in der Variablen bei der Implementierung erreicht werden sollte wie hat es mit Performance zu tun.

Wenn es um Implementierung geht, es muss zunächst entschieden werden, welche Datenstrukturumwandlungen durchgeführt werden müssen. Was zum Anfang klar ist, dass als Eingabe gilt eine Menge von Punkten und als Ausgabe – irgendwelche, noch nicht definierte Art und Weise von gerouteten Wegen zu Trassensegmenten.

Während der Entwicklung nach den zahlreichen Versuchen wurde herausgefunden, wie die Daten umgewandelt werden müssen, um den Auftrag optimal und korrekt lösen zu können. Um das zu erreichen, gibt es zwei großen Schritten. Erster wurde „Clustering“ genannt. Bei dem zweiten Schritt erfolgt die Anwendung und Benutzung von Prim Algorithmus.

#### 3.1 Clustering Algorithmus.

Der Schritt „Clustering“ ist auch geteilt, und zwar, in drei Schritten.

Clustering Schritt 1. Wir wissen, dass für die Eingabe gilt eine Menge von Punkten. Die Datenstruktur, die den Zustand widerspiegelt ist logischerweise eine List von Punkten. (Abb. 3.1)

Datenstruktur:  
List<Point>



Abb. 3.1 Clustering. Schritt 1 - Repräsentierung von Eingangspunkten

Clustering Schritt 2. Bei diesem Schritt für jeden Punkt wird eine Suche aus der Datenbank ausgeführt. In einem bestimmten Radius werden Trassensegmente gesucht. Dann zu jedem Punkt wird so ein Trassensegment zugewiesen, deren Entfernung minimal ist. (Abb. 3.2) Die Datenstruktur wird dabei „Dictionary<Point, Line>“.

Datenstruktur:  
Dictionary<Point, Line>

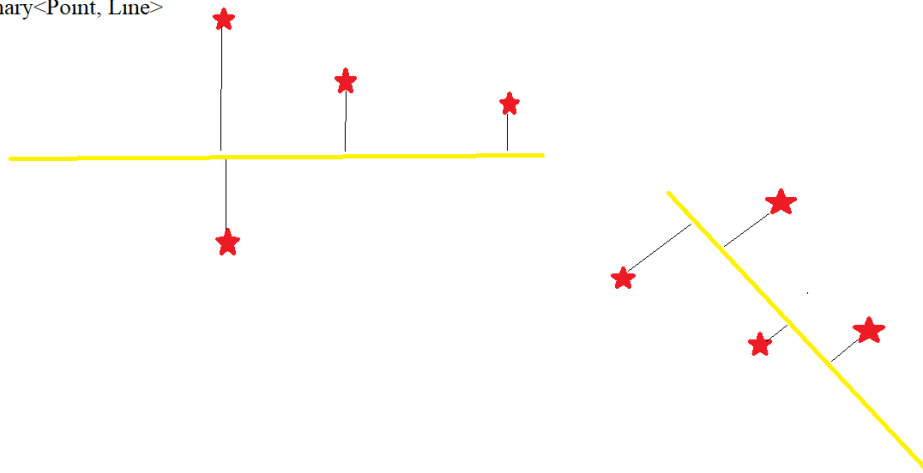


Abb. 3.2 Clustering. Schritt 2 – jedem Punkt wird eine Linie(Trassensegment) zugewiesen, die am nächsten liegt

Bei diesem Schritt wurde auch eine Technik durchgeführt, die ermöglicht, neue Punkten mit den schon klusterisierten Punkten zu verbinden. Die gesamte Länge neuer Trassen wird dann reduziert. Auf dem Abb. 3.3 ist zu sehen, wie stark die Länge neuer Wege reduziert wird.



Abb. 3.3 Clustering. Schritt 2 – es werden zwei Ansätze verglichen. Bei erstem (links) die Punkte werden nur mit Trassen verbunden. Bei dem zweiten (rechts) werden auch

die schon mit Trassen verbundene Punkte berücksichtigt. Allgemein die zweite Lösung weist bessere Ergebnisse auf. Die gerouteten Wege werden kürzer, falls es ein Punkt gibt, der Nähe als eine Trasse liegt.

Aber solche Datenstruktur ist nicht das, was zu erreicht ist. Wünschenswert wiederum ist, dass für jedes Trassensegment die Menge von Punkten zugewiesen wird. Also, die Punkte sind jetzt durch Trassensegmenten verteilt. Das ist sehr praktisch, weil eine optimale Verbindung zwischen Punkten wird genau im Rahmen dieses Trassensegmentes erfolgen. So ein Verfahren stellt sicher, dass die gesamte Länge der neuangelegten Wege minimal ist und Punkten aus einem weitentfernten Cluster werden mit dem Trassensegment nicht verbinden, also nur mit naheliegendem Trassensegment. Die Datenstruktur dabei wird als Dictionary<Line, List<Point>> gekennzeichnet. (Abb. 3.4)

Datenstruktur:  
Dictionary<Line, List<Point>>

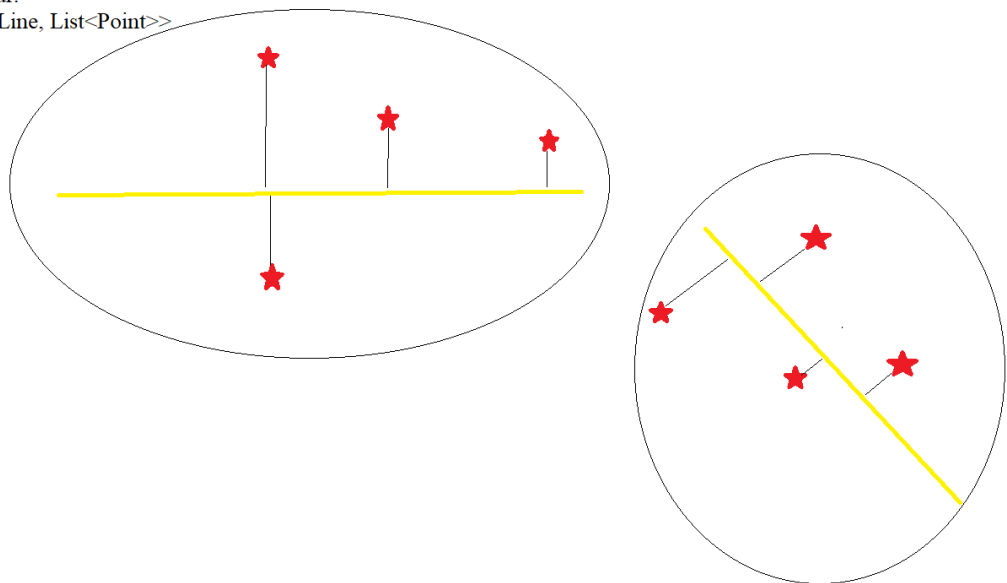


Abb. 3.4 Clustering. Schritt 3. Jeder Trasse wird eine Menge von Punkten zugewiesen. Nun die Punkte werden klasterisiert und jeder Cluster kann als ein separates Graph betrachtet werden.

### 3.2 Anwendung von Prim Algorithmus

Den zweiten Schritt bei der Implementierung ist die richtige Anpassung und Nutzung von Prim Algorithmus.

Zuvor ging es nur um die Positionierung der Punkten. Hier geht es um Darstellung der Punkten in Form von Graph.

Dieser Schritt ist auch verteilt und zwar, in 2 Unterschritte.

**Bei ersten Schritt** jeder Punktenklaster wird als ein Graph dargestellt. Funktion „InitWeights(List<Point> points)“ wird ausgeführt, um den Knoten des Graphes die Gewichte zuzuweisen.

Wichtig zu erwähnen ist, dass bei jedem Punktenklaster wird ein zusätzlicher Punkt zugewiesen. Dies ist ein Punkt, der auf dem Trassensegment liegt. Dieser Punkt ist dafür da, um die Klusterpunkte mit der Trasse zu verbinden. Die Länge dieser Verbindung muss minimal sein.

**Bei dem zweiten Schritt** wird Prim Algorithmus angewendet. Die Menge der Punkten in Form eines Graphes landet in Funktion „Prim(double[,] graph, List<Point> points)“ und die Menge der gerouteten Wegen wird ausgegeben.

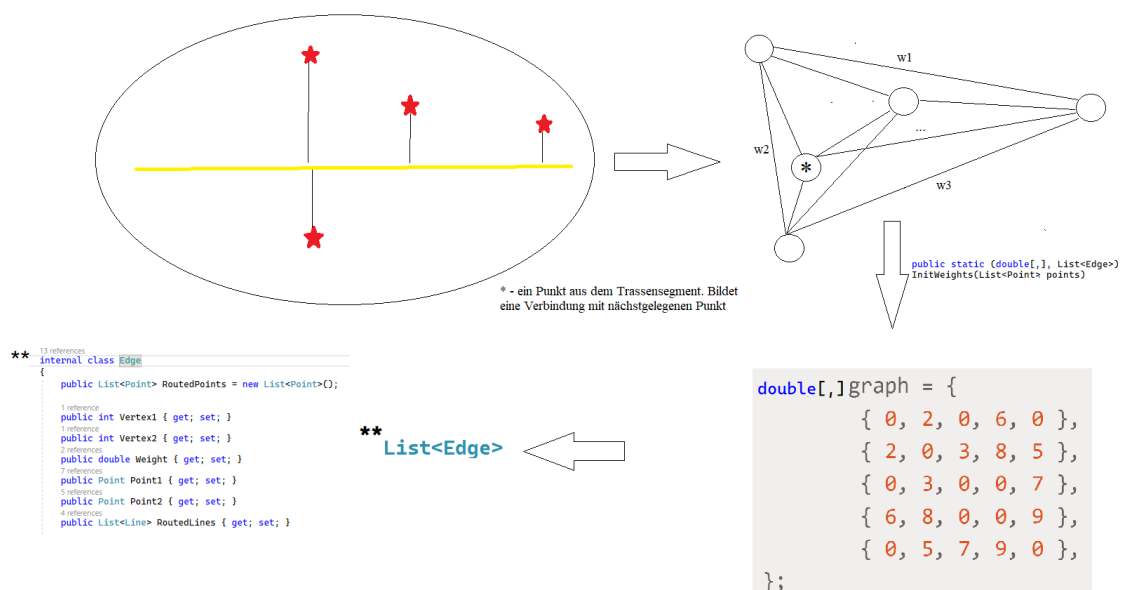


Abb. 3.5 Allgemeine Vorgehensweise bei der Implementierung

Dann der Graph wird in Form von Adjazenzmatrix dargestellt.

Durch den Algorithmus wird eine Liste von Instanzen der Edge-Klasse gewonnen (Abb. 3.5). Es beinhaltet Informationen über Anlegung neuer Wege. Zu jedem Punkt ist ein

Edge zugewiesen. Edge hat Felder „RoutedPoints“ und „RoutedLines“. D.h. es gibt eine Auswahl, die Wege mit Punkten oder mit Linien darzustellen.

### 3.3 Letztendliche JSON Rückgabe

Um die Rückgabe Übersichtlich zu gestalten, werden die gesammelte Informationen in JSON umgewandelt. Es setzt sich aus Schlüsseln „NewPoint“, „OldPoint“, „Length“, „Status“. Bei der Schlüsseln „NewPoint“, „OldPoint“ handelt sich darum, aus welchen Punkten lässt sich ein neuer Weg bauen. „Length“ spiegelt bzw. die Länge des Weges. Bei „Status“ es geht darum, ob der Weg erfolgreich gebaut wurde. Es können zwei mögliche Werte gelten: „OK“ und „Trassen not found in {Config.MAXRADIUS}“. Die Konstante „Config.MAXRADIUS“ wird am Anfang angegeben und entspricht einer grenzwertigen Wert von Radius, wo die Trassensegmente gesucht werden.

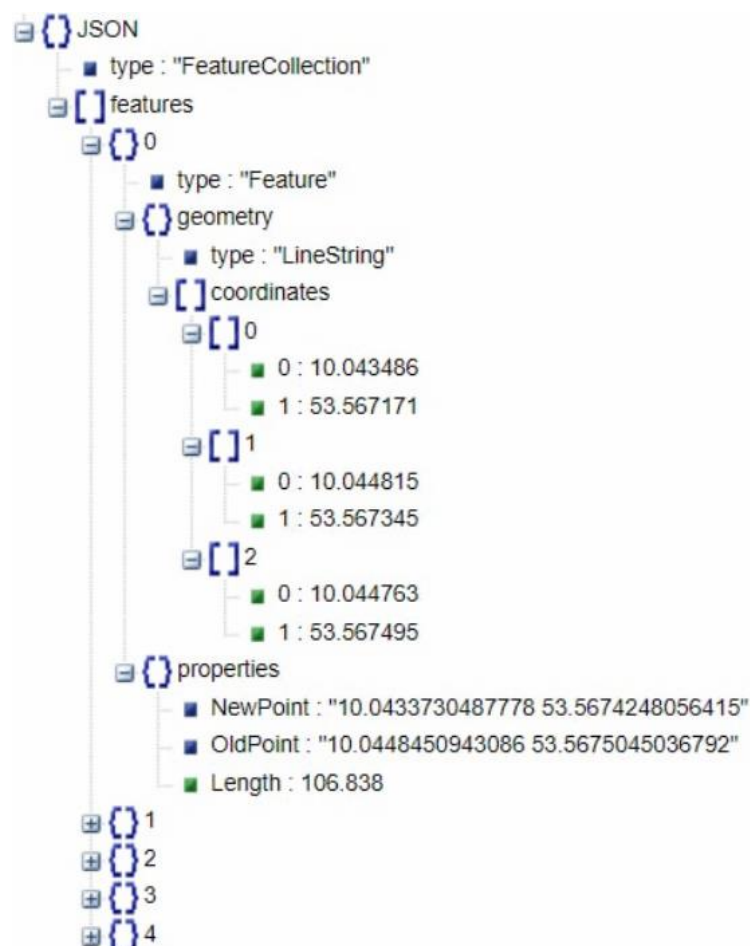


Abb. 3.6 Wie die Rückgabedaten aussehen

Wie in der Firma besprochen wurde, dieser Wert wird standardmäßig zu 5000 Metern gesetzt. Das heißt, die neuangelegten Wege, deren Länge mehr als dieser Wert ist, sind nicht zu berücksichtigen und werden als fehlerhaft gekennzeichnet.

Ausnahmsweise bei manchen Städten je nach Geschäftsbedarf kann dieser Wert reduziert oder vergrößert werden.

### 3.4 Visualisierung

Wenn es um Visualisierung geht, ist „Google Earth pro“ sehr praktisch. Die von Google entwickelte Sprache „KML“ lässt komplexe und umfangreiche Skripte zu programmieren. Der Nachteil ist, dass die Sprache sehr empfindlich ist, d.h. jegliche unnötige oder redundante Leer-Zeichen oder Tabs werden zu einem Fehler führen.

Bei der Evaluierung oder Auswertung der Korrektheit der Anwendung wurde immer dieses Tool genutzt, um zu testen, ob die neuangelegten Wege wirklich optimal sind. Da es gibt keine strenge mathematische Begründung der Korrektheit der Anwendung, die von Script generierte Scripts wurden benutzt um sicher zu sein, dass die Wege mindestens optisch optimal sind.

Um zusätzliche Infos von Wegen zu liefern, bei der Script wird auch die Gesamtlänge der neuen Wege gezeigt. Es kann praktisch sein, wenn einige Veränderungen vorgenommen wurden und ist auszuwerten, ob die Veränderung einen positiven Einfluss hat. Dann sind Gesamtlängen zu vergleichen.

```

foreach (var line in listLines)
{
    var middle = new List<string>
    {
        "<Placemark>",
        "<name>TrassenSegment</name>",
        "<LineString>",
        "<extrude>1</extrude>",
        "<tessellate>1</tessellate>",
        "<coordinates>",
        $"{{line.StartPoint.Longitude.ToString().Replace(',', ' ')},{line.StartPoint.Latitude.ToString().Replace(',', ' ')}}",0 {line
        "</coordinates>",
        "</LineString>",
        "<Style>",
        "<LineStyle>",
        "<color>#ff0000ff</color>",
        "<width>4</width>",
        "</LineStyle>",
        "</Style>",
        "</Placemark>",
    };
    linesToWriteBeg.AddRange(middle);
}

foreach (var line in _TrassenSegments)
{
    var TrassSegm = new List<string>
    {
        "<Placemark>",
        "<name>TrassenSegment</name>",
        "<LineString>",
        "<extrude>1</extrude>",
        "<tessellate>1</tessellate>",
        "<coordinates>",
        $"{{line.StartPoint.Longitude.ToString().Replace(',', ' ')},{line.StartPoint.Latitude.ToString().Replace(',', ' ')}}",0 {line
        "</coordinates>",
        "</LineString>",
        "<Style>",
        "<LineStyle>",
        "<color>ff1ce2d9</color>",
        "<width>10</width>",
        "</LineStyle>",
        "</Style>",
        "</Placemark>",
    };
    linesToWriteBeg.AddRange(TrassSegm);
}

var end = new List<string>
{
    "</Document>",
    "</kml>"
};

```

Abb. 3.7 Ein Teil des Skriptes für die Demonstration der neuangelegten Wege und schon vorhandenen Trassensegmenten

### 3.5 Ein Auszug aus Versuchen

Während der Entwicklung wurden zahlreiche Versuche durchgeführt. Neben den wichtigsten Schritten der Entwicklung konnte man die richtige Auswahl von Parametern nennen. Und zwar, ein Radius, in welchem die Trassensegmente aus der Datenbank gesucht werden.

Bei einem kleinen Radius es konnte so sein, dass es nicht optimale Trassensegmente gefunden werden und als Ergebnis wird die gesamte Verbindung zwischen allen Punkten sehr lang.

Auf Abb. 3.8, 3.9, 3.10 werden Versuche mit 11 gleichen Punkten durchgeführt. Es wird nur Radius geändert. Davon abgesehen wird auch die Gesamtlänge sich variieren.





Abb. 3.8 Anzahl der Punkte – 11, Trassensuchradius – 105 Meter, Gesamtlänge – 4566 Meter.

Auf dem Abb. 3.8 ist zu sehen, dass das Radius zu klein ist. Es werden sehr lange Verbindungen gebaut. Demzufolge ist die Gesamtlänge so groß – 4566 Meter. Der Radius von 105 Meter wird dann als zu unpräzise gezeichnet.

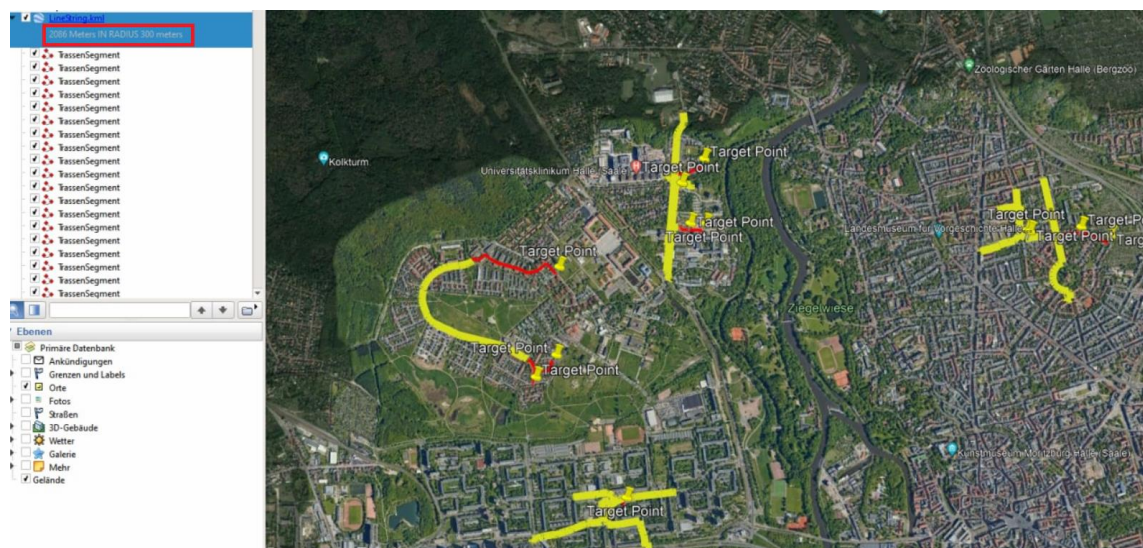


Abb. 3.9 Anzahl der Punkte – 11, Trassensuchradius – 300 Meter, Gesamtlänge – 2086 Meter.

Au dem Abb. 3.9 ist zu sehen, dass die Gesamtlänge gravierend gesunken hat. Mehr als doppelt weniger Länge. (4566 und 2086 -> in 2.18 Mal). Es passiert, weil der Radius 300 Meter beträgt. Die Verbindungen sehen schon viel mehr präziser aus.

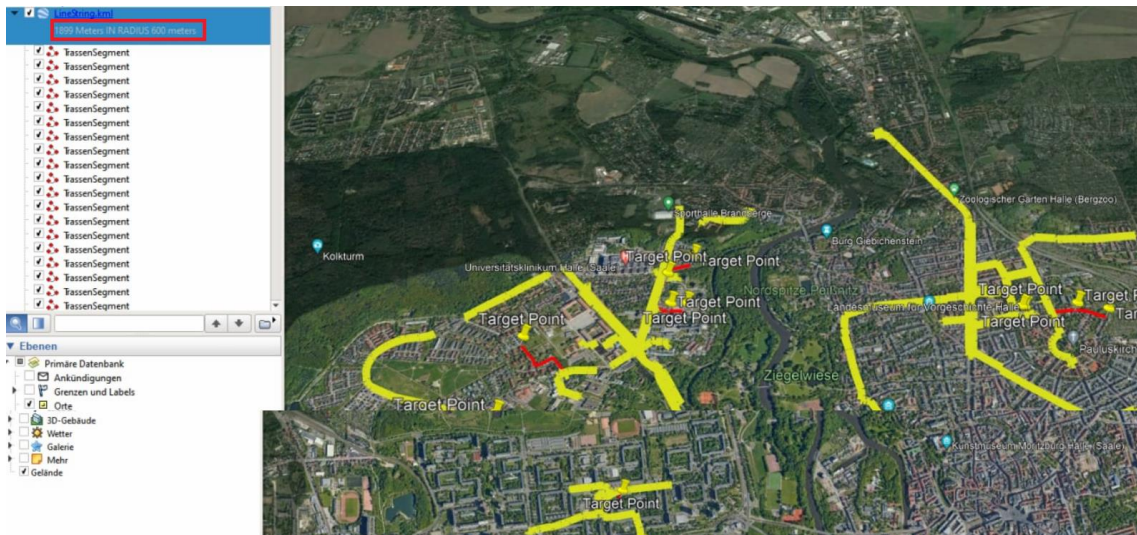


Abb. 3.10 Anzahl der Punkte – 11, Trassensuchradius – 600 Meter, Gesamtlänge – 1899 Meter.

Auf dem Abb. 3.10 fällt auf, dass bei dem Radius von 600 Meter wird die Genauigkeit noch besser. Der Radius macht dabei 600 Meter aus. Aber der Unterschied ( $2086/1899 > 1.09$  Mal) ist jetzt nicht so gravierend.

Trotzdem die Fläche, wo die Trassen gesucht werden, wird dabei von 300 bis 600 Meter steigen. Die Komplexität bei der Vergrößerung des Radiuses wird quadratisch zunehmen, da es von Fläche des dabei resultierenden Kreises abhängt.

Deswegen wurde beschlossen, der Radius bei 500 Meter zu halten und dabei die Ressourcen ein wenig zu sparen.

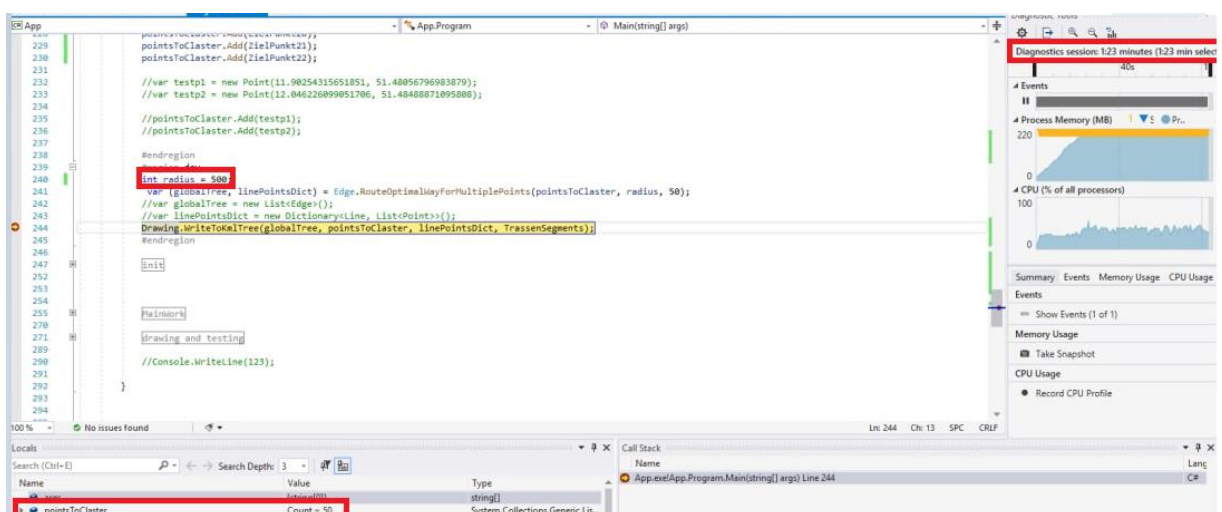


Abb. 3.11 Letztendliche Performance bei der Veröffentlichung

Auf Abb. 3.11 ist ein Debugprozess gezeigt. Dabei sieht man, dass Radius, wo die Trassen gesucht werden beträgt 500 Meter. Anzahl der Punkte, für die die Wege gesucht werden, macht 50 aus. Die erforderliche Zeit, um das Prozess abzuschließen beträgt 1 Minute und 23 Sekunden. Dabei wird die erforderliche Genauigkeit nicht verletzt. Von Performance her ist so ein Gleichgewicht auch sehr gut.

## Zusammenfassung

In diesem Kapitel wurde das algorithmische Thema aus dem vorherigem Kapitel fortgesetzt, es wird aber mehr Einzelheiten über Implementierung liefern. Wurde erzählt, welche Datentransformationen passieren sowie wie das Ergebnis gebaut wird und sich visualisieren lässt. Wurde gezeigt, welche Variablen den Einfluss auf die Performance der Anwendung hat.



## 4 Evaluierung

In der Arbeit handelte sich um Vergleichung und Aussuchung optimaler Wege. Dafür mussten viele Berechnungen durchgeführt werden. Während der Entwicklung wurde häufig die Zeit gemessen.

Um die Entwicklung schneller durchzuführen, wurde bei der Firma angefordert, dass der Entwicklungsprozess auf einem Server erfolgt.

Mithilfe der „Remote Desktop“ ich konnte einfach mit anderem Rechner verbunden sein und die Entwicklung da fortsetzen. Es war besonders hilfreich, wenn die großen Datenmengen heruntergeladen werden mussten. Die Internetverbindung auf dem Rechner war sehr schnell ( $>1000$  Mb pro sek) und ich konnte wirklich viel Zeit sparen und die Zeit für die nützliche Entwicklung oder Projektierung benutzen.

Aber der wichtigste Vorteil davon war, dass die allgemeine Leistungsfähigkeit bei der Entwicklung gravierend gestiegen ist. Infolge einen doppelten Intel-Xeon CPU, konnten die Berechnungen in etwas bis zu 10 mal schneller erfolgen, wie auf einem Laptop mit einem mobilen CPU.

### 4.1 Analyse der berechneten Wege

Als eine Referenz um die Korrektheit der Anwendung zu bewerten wurden 50 zufälligen Punkten in der Stadt Halle in verschiedenen Wohnviertel ausgewählt und die gerouteten Wege gebaut. Abb. 4.1 liefert Informationen über den ganzen Ausblick.

Abb. 4.2, 4.3, 4.4, 4.5 zeigen jeden Kluster detaillierter.

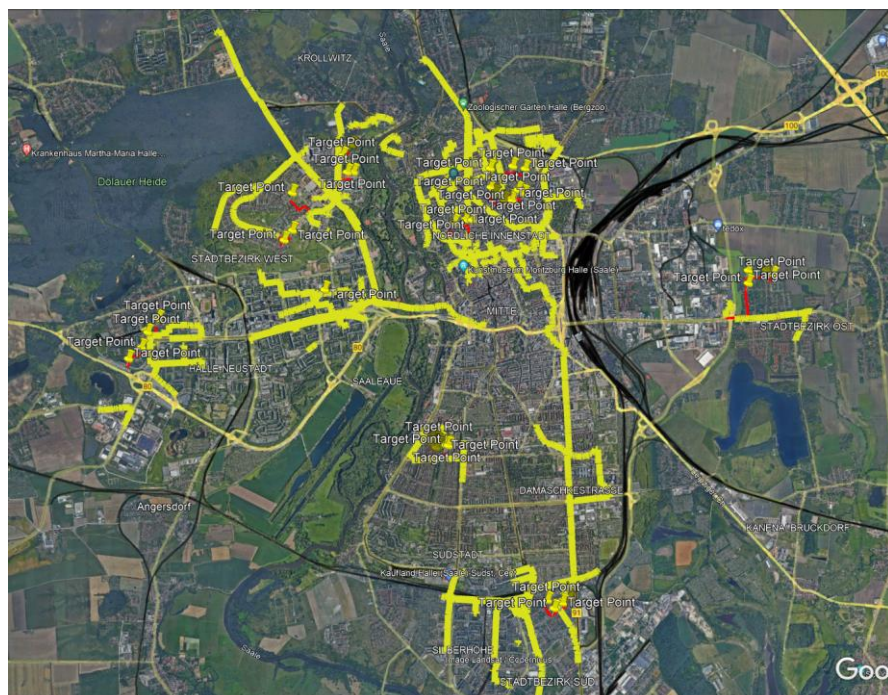


Abb. 4.1 Ein Beispiel mit Punkten in der Stadt Halle

Es fällt auf, dass die Trassen wurde korrekt gefunden. Das heißt, wo eine Sammlung von Punkten gibt, wird es nach den Trassen gesucht und wo es keine Punkte gibt, wird nicht gesucht. So funktioniert es bei der letztendlichen Version. Aber während der Entwicklung wurden auch verschiedene Experimente durchgeführt, wobei es zunächst ein Zentrum von allen Punkten gesucht wurde und dann mit einem sehr großen Radius nach alle Punkten gesucht, die innerhalb dieses Kreises liegen. Von Performance her war gar nicht optimal, zeigte aber eine sehr genaue Verbindung zwischen Trassen und Punkten.

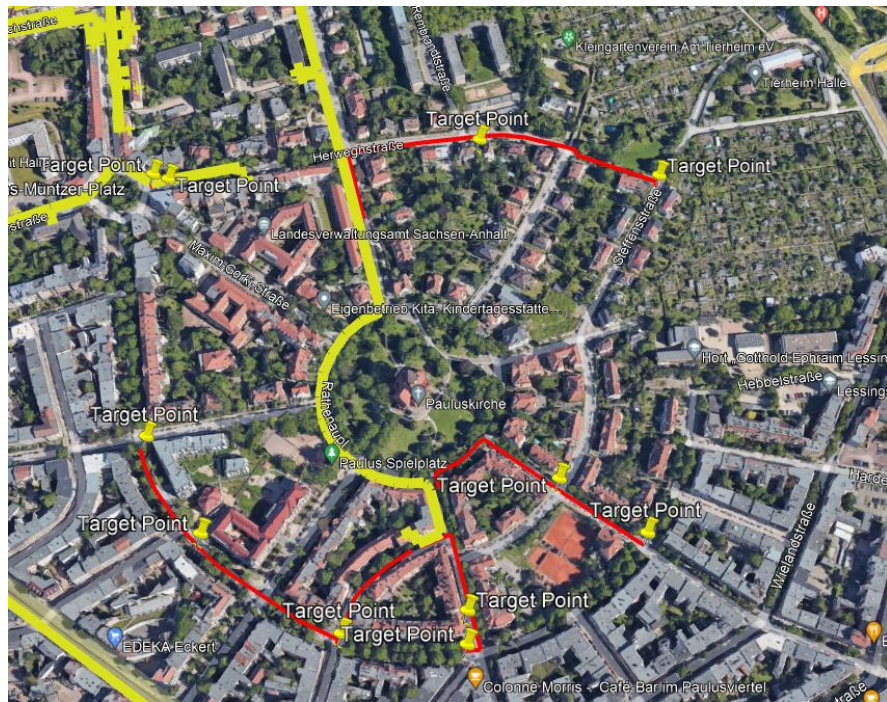


Abb. 4.2 Detaillierter Aussicht 1

Auf der Abb. 4.2 sieht man eine Sammlung von Punkten, die sich rechts auf dem gesamten Bild befinden. In diesem Bereich sind 4 Punkten anzuschließen. Der Abstand zwischen Punkten ist weniger als die Entfernung bis Trasse. Deswegen sind alle 4 Punkte miteinander verknüpft.

Im allgemeinen es sieht wie eine sehr optimale Verbindung aus. Die einzige Bemerkung ist: die Verbindung zwischen die Trasse und den Punkt links unten konnte besser gebaut werden. Der Punkt ist mit der Trasse links verbunden, konnte aber mit der trasse unten verbunden sein.





Auf der Abb. 4.3 wird eine ungewöhnliche Situation gezeigt. Die Straßen sind um ein Gebäude herum. Trotzdem sind die Punkten auch optimal verbunden.



Abb. 4.4 Detaillierter Aussicht 3



Auf der Abb. 4.4 fällt auf, dass die Punkten richtig verknüpft sind. Die Punkten unten links wurden auch elegant und richtig verbunden.



Abb. 4.5 Detaillierter Ausschnitt 4

Auf der Abb. 4.4 es gibt sehr viele Punkten nebeneinander. Dabei funktioniert der Algorithmus auch gut. Anstatt jeden Punkt separat mit einer Trassen zu verbinden, werden sie optimal verteilt (je nach Straße), miteinander verknüpft und dann mit einer Trasse verbunden.

Die auf den Bildern gezeigt Ergebnisse zeigen gute Qualität der Verbindungen. Die Klusterisierung der Punkte erfolgt richtig. Sie werden korrekt gruppiert. Bei einigen Stellen kommt so eine Situation vor, wenn die schon miteinander verbundenen Punkte mit einer Trasse nicht komplett korrekt verknüpft ist.

Es liegt daran, dass der Abstand bis der Trassen wird nur am Anfang und am Ende der Trasse gemessen. Die richtige Zerlegung der Trassen sollte das Problem lösen, wurde aber nicht komplett implementiert.

## **Zusammenfassung**

In diesem Kapitel wird ein Fall wie die Anwendung funktioniert, gezeigt. Die 5 Punkte in der Stadt Halle wurden mit der Trassen verbunden. Einige Gruppen der Punkte wurden analysiert.

Nach der Analyse wurde festgestellt, dass die Anwendung entspricht den Anforderungen seitens Firma. Die Genauigkeit der Verbindungen ist akzeptabel, konnte aber an einigen Stellen durch die korrekte Zerlegung der Trassen verbessert werden.



## **5 Zusammenfassung und Ausblick**

In diesem Kapitel wird die ganze Arbeit zusammengefasst. Außerdem es wird auch der Ausblick zu lesen.

### **5.1 Zusammenfassung**

In der Zielstellung der Arbeit steht „Den Algorithmus und die Architektur der Anwendung zu durchdenken und die beste Lösung für das Problem der optimalen Anlegung neuer Verbindungen zu finden“.

In der Einführung ist begründet, warum die Arbeit aktuell und wichtig ist. Wurde gezeigt, dass die diese Anwendung soll der Firma helfen, die Aufgaben schneller zu lösen und somit die Zeit bzw. die Ressourcen zu sparen.

Bei dem Kapitel Grundlagen wurden die Grundbegriffe erklärt.

In der Methodik wurde gezeigt, welche Teilprobleme zu lösen sind. Dann wurde jedes Problem ausführlich erklärt und die entsprechende Lösung vorgeschlagen. Dazu zählen die Auswahl und Einstellung einer Routing-Engine, der richtiger Umgang mit Eingangsdaten, Besonderheiten bei den Berechnungen bei der Erde im Vergleich zu flache Oberfläche, Formalisierung der Aufgabe zu einer algorithmischen Aufgabe und das Vorschlagen eines richtigen Algorithmus zur Lösung dieser Aufgabe.

Weiter kommt der Kapitel „Implementierung“. Es setzt sich aus Clustering von Punkten und das Umsetzen von Algorithmus in der Programmiersprache C#. Anschließend wurde gezeigt, welche Variablen experimentell angepasst werden sollten und was für ein Einfluss es auf die Performance und Qualität der Ergebnisse hat. Dieses Gleichgewicht wurde vorgeschlagen und die Ergebnisse wurden gezeigt.

In dem Kapitel Evaluierung die Ergebnisse wurden ausgewertet. Es wurde festgestellt, dass die Qualität der Anwendung genügend ist. Es weist eine hohe Effizienz aus, obwohl es viele Eingangspunkte gibt. Die neuangelegten Wege sind sehr präzise und genau. Die Klusterisierung erfolgt richtig und die Punkte sind richtig verteilt. Es gibt trotzdem einige Schwachstellen, und zwar, die Verknüpfung mit der Trasse erfolgt nicht immer optimal. Im Weiteren es sollte durch eine Effiziente Trassenzerlegung verbessert werden.

## 5.2 Ausblick

Wenn ich für die Arbeit mehr Zeit hätte, würde ich die Architektur der App so gestalten, dass es keine Bindung zu einer festen Datenbank gibt, sondern so dass der Benutzer selber die Quelle für seine Trassen auswählen konnte. Dann wäre die App nicht nur für eine Firma praktisch, sondern konnte anderen Firmen verkauft werden. Dazu sollte auch die Behandlung mit Trassen sehr flexibel erfolgen. Es mussten verschiedene Datenformate berücksichtigt werden usw.

Außerdem als eine gute Verbesserungsmöglichkeit, ich würde ein Interface zur Verfügung zu stellen, anstatt KML-Dateien zu generieren.

Es sollte auch die Auswahl verschiedener Routing Engines berücksichtigt werden, da bei verschiedenen Herausforderungen (z.B. nur Wege für Autos bauen oder wenn Höheunterschied auch zählen würde), hätte verschiedene Routing-Engines unterschiedliche Bedeutung.

Die Effizienz der App konnte auch durch das Nutzen von Fibonacci Heap anstatt Adjazenzmatrix erreicht werden.

Eine richtige Verteilung der Trassen wurde in dieser Arbeit nicht erreicht. Wenn sie aber erreicht werden könnte, dann wäre die Genauigkeit der Verbindungen gravierend gestiegen. Wenn ich das machen würde, dann würde ich diese Idee mit Halbierung durchsetzen.

## Literaturverzeichnis

- [1] M. Hermann, T. Pentek, B. Otto: Design Principles for Industrie 4.0 Scenarios. In: 2016 49th Hawaii International Conference on System Sciences (HICSS). 1. Januar 2016, S. 3928–3937, doi:10.1109/HICSS.2016.488 (ieee.org [abgerufen am 22. August 2016]).
- [2] Norbert Gronau, Marcus Grum, Benedict Bender: Determining the optimal level of autonomy in cyber-physical production systems. In: 2016 IEEE 14th International Conference on Industrial Informatics (INDIN). IEEE, Poitiers, France 2016, ISBN 978-1-5090-2870-2, S. 1293–1299, doi:10.1109/INDIN.2016.7819367 (ieee.org [abgerufen am 17. Januar 2020]).
- [3] DSLWEB Breitband Report 2020, Breitband Report 2020 - Wachstum im Krisenjahr (dslweb.de))
- [4] OSRM API Documentation (project-osrm.org)
- [5] Open Source Routing Engines And Algorithms - An Overview » GIS•OPS (gis-ops.com)
- [6] <https://www.routexl.com/blog/openstreetmap-router-graphhopper-osrm-gosmore/>
- [7] <https://graphhopper.com/>
- [8] <https://www.graphhopper.com/open-source/>
- [9] R. W. Sinnott, "Virtues of the Haversine," Sky and Telescope, vol. 68, no. 2, 1984, s. 159
- [10] Norbert Blum, „Theoretische Informatik“ s 252
- [11] Sara Baase, “Computer Algorithms: Introduction to Design and Analysis” s. 390-391, 412
- [12] <https://de.statista.com/statistik/daten/studie/830769/umfrage/bedeutung-von-industrie-40-in-deutschland/>
- [13] Bitkom; Fraunhofer IAO; ID 297985
- [14] (<https://www.itinero.tech/>).