Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

CH-basierte Darstellung von Straßennetzwerken mit optimierter Entfaltungsreihenfolge

Sophia Heim

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Beginn am:	25. Juli 2023
Beendet am:	19. Dezember 2023

Kurzfassung

Echtzeitdarstellungen von Karten sind ohne die Auslassung und Vereinfachung von Kartenelementen nicht performant möglich. Ein Ansatz, um Performanz zu ermöglichen, ist der Einsatz von Kontraktionshierarchien. Um eine visuell ansprechende Darstellung zu garantieren, muss eine Kontraktionshierarchie jedoch bei jeder Anfrage nach einem Kartenausschnitt abhängig von dem gewünschten Detaillierungsgrad in einem gewissen Umfang wieder entpackt werden. Da die Entscheidung, welche Elemente nacheinander entpackt werden sollen, sehr zeitintensiv ist, sollte diese bereits in einem Vorverarbeitungsschritt, vor der eigentlichen Anfragezeit, getroffen werden.

Ziel dieser Arbeit ist die Bestimmung von optimalen Entpackreihenfolgen in einem Vorverarbeitungsschritt mittels verschiedener Entpackstrategien und unter Nutzung unterschiedlicher Fehlermetriken. Mithilfe dieser Reihenfolgen werden anschließend bei einer Anfrage die darzustellenden Kanten in Echtzeit ermittelt. Um dies zu ermöglichen, wird eine Graphdatenstruktur durch das Einlesen einer Kontraktionshierarchie aufgebaut. Anschließend werden für die enthaltenen Kanten verschiedene Fehler vorberechnet. Mit diesen Fehlern können im Anschluss optimale Entpackreihenfolgen für verschiedene Entpackstrategien berechnet und in Dateien geschrieben werden. Bei einer Anfrage wird die benötigte Datei mit der optimalen Entpackreihenfolge eingelesen und die darzustellenden Kanten durch einen Entpackprozess basierend auf der eingelesenen Reihenfolge in Echtzeit ermittelt.

Zudem wird eine Schnittstelle definiert, über welche Anfragen gestellt werden können. Ebenfalls wird eine Weboberfläche zur Demonstration der Funktionalität implementiert. Des Weiteren besteht die Möglichkeit, die nach einer Anfrage identifizierten Kanten als Ausgabedatei zu exportieren. Abschließend werden die entwickelten Verfahren auf mehreren Testgraphen auf einem Testsystem analysiert und der durch die verschiedenen Entpackreihenfolgen resultierende visuelle Eindruck verglichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

API Application Programming Interface. 18

BW Baden-Württemberg. 69

CH Kontraktionshierarchie. 17

HTTP Hypertext Transfer Protocol. 61

ID Identifikator. 25

IRM IERS Reference Meridian. 22

JSON JavaScript Object Notation. 21

OSM OpenStreetMap. 18

WGS 84 World Geodetic System 1984. 21

1 Einleitung

Zu Beginn dieser Arbeit wird die Motivation und Notwendigkeit des Themas erläutert. Anschließend wird ein Überblick über die Inhalte der nachfolgenden Kapitel dieser Arbeit gegeben.

1.1 Motivation

Typischerweise werden Karten als vorgerenderte Kacheln für feste Zoomlevel zur Verfügung gestellt. Diese Kacheln benötigen jedoch eine sehr lange Vorberechnungsdauer und sind deshalb nicht für den Einsatz in Echtzeitkarten geeignet. Für Kartendarstellungen in Echtzeit, die nicht auf vorgerenderten Kacheln basieren, können aufgrund der benötigten Performanz nicht alle Karteninformationen im Detail dargestellt werden. Daher ist es zwingend erforderlich, dass Kartenelemente teils vereinfacht oder sogar ganz ausgelassen werden. Es soll dennoch weiterhin eine visuell ansprechende Darstellung gewährleistet werden. Das Ziel ist daher, eine sowohl performante als auch visuell ansprechende Darstellung der Karteninformationen zu erreichen, trotz der Vereinfachung und Auslassung von Kartenelementen.

Möchte man beispielsweise einen Kartenausschnitt des Straßennetzwerks von Deutschland darstellen, dann ist es lediglich erforderlich, alle relevanten Autobahnen und Bundesstraßen darzustellen. Kleinere Dorfstraßen oder Landstraßen können ganz ausgelassen werden. Des Weiteren müssen die dargestellten Straßen nur mit ausreichender Qualität, nicht aber mit einer Genauigkeit von wenigen Metern dargestellt werden.

Mithilfe von vorberechneten Kontraktionshierarchien (CHs), einer Datenstruktur, um Anfragen nach kürzesten Pfaden zu beschleunigen, konnten in [FSS17] bereits Auslassungen und Vereinfachungen ermöglicht werden. In der genannten Arbeit kann bei einer Anfrage ein Kartenausschnitt und ein Kartendetaillierungsgrad angegeben werden und die darzustellenden Elemente werden daraufhin on-the-fly bestimmt. Neben der Entscheidung, welche Kartenelemente überhaupt dargestellt werden sollen, wird zusätzlich durch eine rekursive Entpackroutine und einen Fehlerparameter on-the-fly berechnet, in welcher Detailliertheit diese verbleibenden Kartenelemente darzustellen sind. Diese Auswertung passiert jedoch zur Laufzeit, ist relativ zeitintensiv und führt nicht notwendigerweise zum bestmöglichen visuellen Ergebnis.

1.2 Problemstellung

In dieser Arbeit wird untersucht, inwiefern optimale Entpackreihenfolgen von CHs schon in einem Vorverarbeitungsschritt berechnet werden können. Dadurch sind die Entpackreihenfolgen zur Anfragezeit bereits bekannt und bei der Anfrage selbst muss lediglich die Entpackung bis zur gewünschten Detailliertheit on-the-fly erfolgen. Hierfür werden verschiedene Entpackstrategien untersucht, welche jeweils unterschiedliche Fehlermetriken nutzen.

Schließlich wird der visuelle Eindruck der verschiedenen Entpackreihenfolgen und der unterschiedlichen Fehlermetriken verglichen. Zur Visualisierung kann zum einen ein von dem Institut für Formale Methoden der Informatik von der Arbeitsgruppe Algorithmik bereitgestellter OpenGL-Viewer für Kartendaten verwendet werden. Andererseits wird in dieser Arbeit ebenfalls eine Webanwendung mit einem Backend Server entwickelt, welche Echtzeitanfragen erlaubt.

1.3 Gliederung der Arbeit

Diese Arbeit ist wie folgt strukturiert:

Kapitel 2 - Grundlagen: Das zweite Kapitel beschreibt die notwendigen Grundlagen, welche zum Verständnis dieser Arbeit benötigt werden. Dies umfasst geografische und technologische Grundlagen, die Definitionen von Graphen und Kontraktionshierarchien und eine kurze Einführung in das OpenStreetMap (OSM)-Projekt.

Kapitel 3 - Einlesen der Daten und Grapherstellung: Im dritten Kapitel werden die von dieser Arbeit als Eingabe benötigten Dateien und die Abbildung des Inhalts dieser Dateien auf eine eigene Graphdatenstruktur beschrieben.

Kapitel 4 - Berechnung der optimalen Entpackreihenfolgen: Das vierte Kapitel behandelt die verschiedenen Möglichkeiten der Vorberechnung von optimalen Entpackreihenfolgen für alle Kanten, die in dem Graphen aus dem vorherigen Kapitel enthalten sind.

Kapitel 5 - Entpacken von Shortcuts: Im fünften Kapitel werden die aus dem vierten Kapitel vorberechneten optimalen Entpackreihenfolgen genutzt, um gegebene Kanten für eine gegebene Anzahl an Schritten zu entpacken.

Kapitel 6 - Visualisierung der entpackten Shortcuts: Das sechste Kapitel beschreibt unterschiedliche Möglichkeiten, um die aus dem Entpackprozess in Kapitel fünf resultierende Menge von Kanten zu visualisieren. Hierbei wird die Generierung einer Ausgabedatei, die Implementierung eines Application Programming Interface (API) und die Erstellung eines Frontends beschrieben.

Kapitel 7 - Komponentenkommunikation: In diesem Kapitel wird die Kommunikation der in dieser Arbeit entwickelten und verwendeten Komponenten dargestellt.

Kapitel 8 - Auswertung: Das achte Kapitel evaluiert die in dieser Arbeit implementierten Verfahren anhand von drei Testgraphen auf einem Testsystem. Es wird sowohl die Vorberechnung als auch der Entpackprozess analysiert und der visuelle Eindruck der verschiedenen Verfahren verglichen.

Kapitel 9 - Zusammenfassung und Ausblick: Im letzten Kapitel werden schließlich die Ergebnisse dieser Arbeit zusammengefasst und ein kurzer Ausblick für mögliche weitere Forschung oder Erweiterungen dieser Arbeit gegeben.

2 Grundlagen

Dieses Kapitel behandelt die benötigten Grundlagen, um alle Aspekte der folgenden Arbeit zu verstehen. Dies umfasst zunächst geografische Grundlagen und die für diese Arbeit relevanten Prinzipien des OSM-Projekts. Anschließend folgt die allgemeine Definition von Graphen und eine Beschreibung von CHs als spezielle Graphen. Abschließend werden noch die Grundlagen der in dieser Arbeit genutzten Technologien erklärt, wie Leaflet, den OpenGL-Viewer und die JavaScript Object Notation (JSON).

2.1 Geografische Grundlagen

Um zu verstehen, wie geometrische Daten in dieser Arbeit visualisiert werden und welche Berechnungen dazu auf diesen Daten ausgeführt werden müssen, werden im Folgenden die Grundlagen von geografischen Koordinatensystemen, im Speziellen dem World Geodetic System 1984 (WGS 84), und projizierten Koordinatensystemen mit der Mercator-Projektion beschrieben.

2.1.1 Geografisches Koordinatensystem

Ein geografisches Koordinatensystem nutzt eine dreidimensionale sphärische Oberfläche, um Positionen auf der Erdoberfläche zu bestimmen. Dabei ist jede Position auf der Erde durch einen Längengrad (longitude) und einen Breitengrad (latitude) definiert. Der Äquator hat den Breitengrad 0 und teilt die Erde in einen nördlichen und einen südlichen Teil. Positionen nördlich des Äquators haben die Breitengrade $\in$ [0;+90°] und Positionen südlich des Äquators die Breitengrade $\in$ [-90°;0]. Der Nullmeridian hat den Längengrad 0 und teilt die Erde in einen östlichen und einen westlichen Teil. Meridiane sind diejenigen Linien, die in senkrechter Richtung nach Norden beziehungsweise Süden verlaufen [IBMa]. Positionen östlich des Nullmeridians haben die Längengrade $\in$ [0;+180°] und Positionen westlich des Nullmeridians die Längengrade $\in$ [-180°;0] [IBMa]. Eine entsprechende Veranschaulichung ist in Abbildung 2.1 zu finden.

Der Äquator und der Nullmeridian definieren jeweils eine Ebene. Der Breitengrad eines bestimmten Punktes ist der Winkel zwischen der durch den Äquator definierten Ebene und derjenigen Linie, die durch das Erdzentrum und diesen Punkt verläuft (siehe ϕ in Abbildung 2.2). Der Längengrad eines bestimmten Punktes ist der Winkel zwischen der durch den Nullmeridian definierten Ebene und der Ebene, die durch denjenigen Meridian definiert ist, der diesen Punkt schneidet (siehe λ in Abbildung 2.2) [Gim].

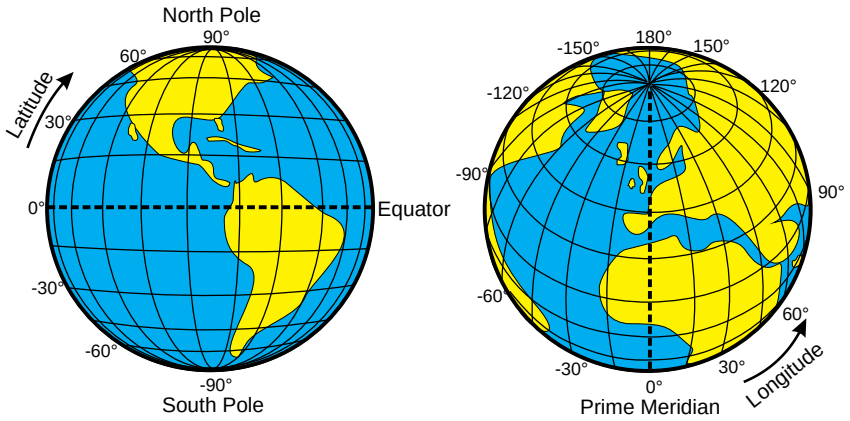


Abbildung 2.1: Längen- und Breitengrade auf der Erdoberfläche [Cond]

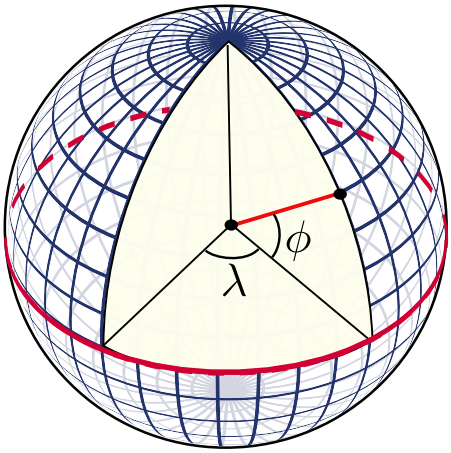


Abbildung 2.2: Berechnung der Längen- und Breitengrade auf einer Sphäre [Mer]

2.1.2 World Geodetic System 1984

Das WGS 84 ist ein globales geodätisches Referenzsystem für die Erde. Das im WGS 84 genutzte Referenzellipsoid hat den IERS Reference Meridian (IRM) als Nullmeridian. Es besitzt als große Halbachse $a \approx 6.378$ km und als kleine Halbachse $b \approx 6.357$ km. Hierbei ist die große Halbachse der größte Radius einer Ellipse und die kleine Halbachse der kleinste Radius einer Ellipse [Pag]. Dieses genutzte Referenzellipsoid kann als Sphäre mit Radius $R \approx 6.371$ km approximiert werden [GW14].

2.1.3 Projizierte Koordinatensysteme

Ein geografisches Koordinatensystem (siehe Abschnitt 2.1.1) kann auf ein projiziertes Koordinatensystem abgebildet werden [Gim]. Dazu werden die geografischen Koordinaten (also die Längen- und Breitengrade) auf kartesische Koordinaten (meist x- und y-Koordinaten) abgebildet. In einem kartesischen Koordinatensystem lassen sich Distanzen und Flächeninhalte deutlich einfacher berechnen. Es gibt verschiedene Arten der Projektionen, abhängig von der genutzten Projektionsoberfläche. Projektionsoberflächen können beispielsweise Kegel, Zylinder oder planare Oberflächen sein. Jede Projektion (außer trivialen Projektionen, wie beispielsweise der Identitätsprojektion) verzerrt bestimmte räumliche Eigenschaften, wie die Distanz, die Fläche, die Form, die Richtung und Kombinationen hiervon.

Man unterscheidet in flächentreue, konforme, äquidistante, richtungstreue oder azimutale Projektionen. Flächentreue Projektionen bewahren die Flächen, verzerren aber die Form, die Winkel und die Skala. Konforme Projektionen bewahren alle Winkel und die Form für kleine Flächen, verzerren aber die Fläche der Karte. Äquidistante Projektionen bewahren die Distanzen zwischen bestimmten Punkten, da sie die Skala eines bestimmten Datensatzes beibehalten. Außerhalb des Datensatzes wird die Skala aber verzerrt. Richtungstreue oder azimutale Projektionen bewahren die Richtung von einem Punkt zu allen anderen Punkten, indem sie einige Großkreisbögen beibehalten.

Im Folgenden wird die Mercator-Projektion beschrieben, die zur Gruppe der konformen Projektionen gehört [IBMb].

Mercator Projektion

Um die geografischen Daten von dem WGS 84 auf einer zweidimensionalen Karte visualisieren zu können, kann das geografische Koordinatensystem (wie in Abschnitt 2.1.1 beschrieben) auf ein kartesisches Koordinatensystem abgebildet werden. Die Mercator-Projektion ist eine winkeltreue Zylinderprojektion [Sei23]. Richtungen, Winkel und Formen werden beibehalten, jedoch wird die Fläche in Richtung der Pole zunehmend verzerrt [Arcb]. Es existieren verschiedene Versionen der Mercator-Projektion. In dieser Arbeit wird die Standard Mercator-Projektion verwendet, bei der die Zylinderachse identisch mit der Erdachse ist [EC17]. Eine entsprechende Visualisierung befindet sich in Abbildung 2.3.

Um die Mercator-Projektion zu berechnen, werden die geografischen Koordinaten als Bogenmaße benötigt. Falls diese als Gradmaße vorliegen, müssen die Gradmaße (*latitude* und *longitude*) zunächst in Bogenmaße (φ und λ) umgerechnet werden [Stub]:

$$\rho_{rad} : [-90°; 90°] \times [-180°; 180°] \rightarrow [-\frac{\pi}{2}; \frac{\pi}{2}] \times [-\pi; \pi] : \begin{pmatrix} \text{latitude} \\ \text{longitude} \end{pmatrix} \mapsto \begin{pmatrix} \varphi \\ \lambda \end{pmatrix} \tag{2.1}$$

mit geografischer Breite φ:

$$\varphi = latitude \cdot \frac{\pi}{180°} \tag{2.2}$$

und geografischer Länge λ:

$$\lambda = longitude \cdot \frac{\pi}{180°} \tag{2.3}$$

Die Mercator-Projektion selbst ist dann wie folgt definiert:

$$\rho_{\text{Merc}} : [-\frac{\pi}{2}; \frac{\pi}{2}] \times [-\pi; \pi] \to \mathbb{R}^2 : \begin{pmatrix} \varphi \\ \lambda \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.4}$$

mit

$$x = R(\lambda - \lambda_0) \tag{2.5}$$

und

$$y = R \cdot ln[tan(\frac{\pi}{4} + \frac{\varphi}{2})] \tag{2.6}$$

wobei $\lambda_0 \in [-\pi; \pi]$ die geografische Länge des Zentralmedians angibt und R den Radius der Sphäre angibt, die das Ellipsoid approximiert ($R \approx 6.371$ km, siehe Abschnitt 2.1.2). Meistens, wie auch in dieser Arbeit, wird der IRM als Zentralmedian gewählt, wofür $\lambda_0 = 0$ gilt [Jor22].

Sowohl die Umrechnung von dem Gradmaß in das Bogenmaß, als auch die Mercator-Projektion selbst sind bijektiv, können also rückgängig gemacht werden [Wei23b].

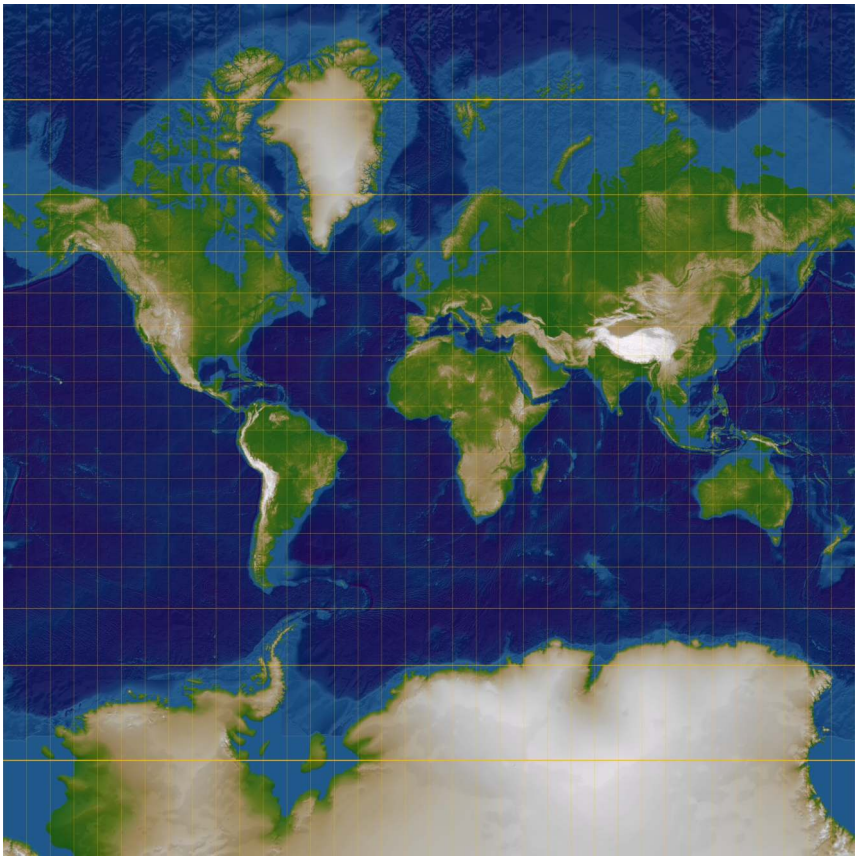


Abbildung 2.3: Standard Mercator-Projektion [Roh]

2.2 OpenStreetMap

OSM ist ein Projekt, das geografische Daten der ganzen Welt erstellt und verbreitet [Con22b]. Eine bearbeitbare Karte der ganzen Welt wird hierbei von Freiwilligen gebaut [Con22a]. OpenStreetMap Daten sind offene Daten, die von der OpenStreetMap Stiftung (OSMF) unter der Open Data Commons Open Database-Lizenz (ODbL) lizenziert sind [Conc]. In den Daten sind unter anderem Knoten und Pfade enthalten [Con23a]. Knoten sind Punkte im Raum mit geografischer Position, die als Koordinatenpaar von einem Längen- und einem Breitengrad gemäß dem WGS 84 gespeichert sind [Con23b]. Pfade sind eine geordnete Liste solcher Knoten [Con23c].

2.3 Graphentheorie

Ein gerichteter Graph $G(V, E)$ besteht aus einer Knotenmenge V und einer Kantenmenge $E \subseteq \{(u, v)|u, v \in V\}$, wobei eine Kante von Knoten u zu Knoten v führt. Bei einem gewichteten Graphen existiert zusätzlich eine Kostenfunktion $c : E \to \mathbb{R}^+$. Wenn der Graph ein Straßennetzwerk darstellt, könnten die Kosten zum Beispiel die Reisezeit für ein Auto darstellen.

Ein Pfad von v nach v' ist definiert als eine Folge von Knoten $(v_0, ..., v_k)$, wobei $v_0 = v$, $v_k = v'$, $(v_i, v_{i+1}) \in E$ und $v_i \neq v_{i+1}$ [Räc19]. Ein Polygon ist ein Pfad bestehend aus mindestens drei Knoten, wobei zusätzlich $v = v'$ gelten muss [Ste]. Knoten und Kanten können auch mit weiteren Attributen (zum Beispiel geografische Koordinaten für Knoten und Kosten für Kanten) versehen werden, um die Aussagekraft von Graphen zu verstärken.

Im Rahmen dieser Arbeit ist jeder Knoten durch einen unter allen Knoten eindeutigen Identifikator (ID) und jede Kante durch eine unter allen Kanten eindeutige ID gekennzeichnet.

2.4 Kontraktionshierarchien

Gegeben sei ein Graph G, wie in Abschnitt 2.3 beschrieben. Nun soll G so vorverarbeitet werden, dass kürzeste Pfade von einem Startknoten zu einem Zielknoten sehr schnell, aber dennoch optimal beantwortet werden können. In einem Vorverarbeitungsschritt wird hierfür die Kantenmenge E um eine Kantenmenge $\tilde{E}$, die sogenannte Menge der Shortcuts, erweitert. Ein Shortcut repräsentiert einen kürzesten Pfad im Originalgraph G. Außerdem wird jedem Knoten $v \in V$ ein $level(v)$ zugewiesen, wobei $level : V \to \mathbb{N}$. Es resultiert ein neuer Graph $G^*(V, E^*)$, wobei $E^* = E \cup \tilde{E}$. Es gilt dann $c^* := c(e)$, wenn $e \in E$, ansonsten ist $c^*(e)$ die Länge des kürzesten Pfades, den e repräsentiert.

Die zentrale Operation der Vorverarbeitungsphase ist die sogenannte Knotenkontraktion. Ziel ist, einen Knoten v (und seine adjazenten Kanten) aus G zu entfernen, ohne die Distanzen von kürzesten Pfaden zwischen den anderen Knoten zu beeinträchtigen. Falls dies dennoch der Fall sein sollte, müssen Shortcuts zwischen Nachbarn von v hinzugefügt werden, wie im Folgenden erklärt wird.

Bei der Kontraktion von Knoten v wird folgendermaßen vorgegangen: Betrachtet werden alle Pfade zwischen jeweils zwei Nachbarn u und w von v, wobei $(u, v) \in E$ und $(v, w) \in E$. Führt der kürzeste Pfad von u zu w über v, so muss ein Shortcut $s = (u, w)$ hinzugefügt werden, mit $c(s) = c((u, w)) = c((u, v)) + c((v, w))$.

Hierbei können drei Fälle auftreten, die mit dem Beispiel in Abbildung 2.4 verdeutlicht werden sollen:

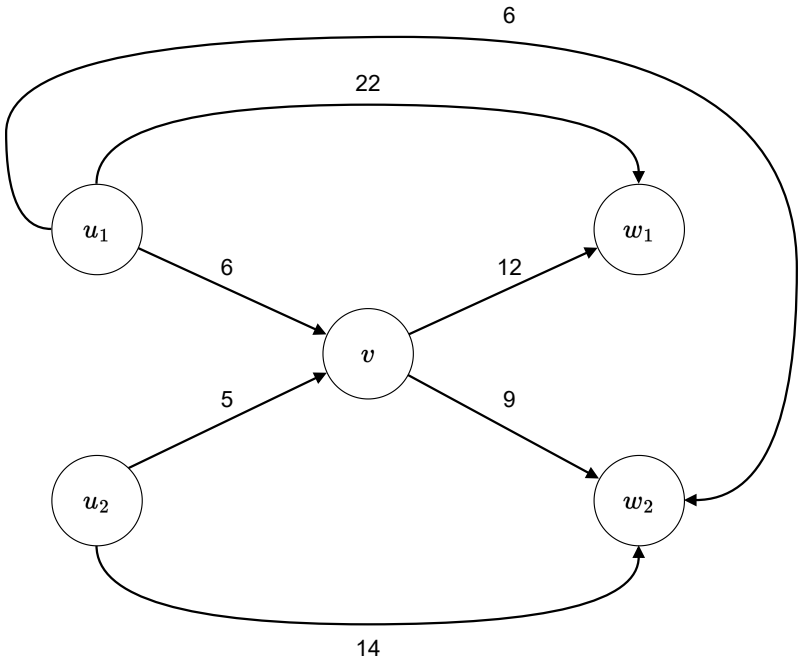


Abbildung 2.4: Kontraktion eines Knotens v beim Aufbau einer Kontraktionshierarchie

Erster Fall: Die Kosten des einzufügenden Shortcuts $s_{1,1} = (u_1, w_1)$ wären $c(s_{1,1}) = c((u_1, v)) + c((v, w_1)) = 6 + 12 = 18$. Der einzige andere Pfad von u_1 zu w_1 ist (u_1, w_1) mit $c((u_1, w_1)) = 22 > c(s_{1,1})$. Der kürzeste Pfad von u_1 zu w_1 geht also tatsächlich über den Knoten v. Daher muss der Shortcut $s_{1,1}$ hinzugefügt werden.

Zweiter Fall: Die Kosten des einzufügenden Shortcuts $s_{1,2} = (u_1, w_2)$ wären $c(s_{1,2}) = c((u_1, v)) + c((v, w_2)) = 6 + 9 = 15$, aber der kürzeste Pfad von u_1 zu w_2 ist (u_1, w_2) mit $c((u_1, w_2)) = 6 \leq c(s_{1,2})$. Daher muss kein Shortcut hinzugefügt werden.

Dritter Fall: Die Kosten des einzufügenden Shortcuts $s_{2,1} = (u_2, w_1)$ wären $c(s_{2,1}) = c((u_2, v)) + c((v, w_1)) = 5 + 12 = 17$. Da kein anderer Pfad von u_2 zu w_1 existiert, muss der Shortcut $s_{2,1}$ hinzugefügt werden.

Um eine CH aufzubauen, werden nach und nach alle Knoten aus G kontrahiert. Die Auswahl eines Knotens v basiert hierbei auf einer Priorität. Eine Priorität ist eine lineare Kombination von Faktoren mit Koeffizienten. In einem Kontraktionsschritt wird immer der Knoten mit der aktuell geringsten Priorität kontrahiert. Beispielsweise könnte der Knoten mit der aktuell kleinsten Kantendifferenz die niedrigste Priorität besitzen. Die Kantendifferenz ist die Anzahl der Shortcuts, die beim Kontrahieren dieses Knotens eingefügt werden müssen, minus die Anzahl der Kanten, die während der Kontraktion gelöscht werden. Bei gleicher Kantendifferenz kann die Knoten-ID als Tiebreak fungieren. Dem i-ten kontrahierten Knoten v wird $level(v) = i$ zugewiesen.

Eine effektive Strategie ist es, iterativ eine Menge von unabhängigen (nicht adjazenten) Knoten simultan zu kontrahieren, von denen jeder die Anzahl der Kanten so wenig wie möglich erhöht. Simultan kontrahierte Knoten bekommen alle dasselbe *level* zugewiesen [GSSD]. Es stellt sich heraus, dass die Levels der Knoten die Relevanz für kürzeste Pfade im Netzwerk sehr genau repräsentieren. Knoten, die später kontrahiert wurden, also ein hohes Level besitzen, sind meist Knoten, die in der Mitte von vielen langen kürzesten Pfaden auftreten (gemäß der Kostenfunktion) [FSS17].

Wie Anfragen nach kürzesten Pfaden auf dem Graph G^* durchgeführt werden, ist in dieser Arbeit nicht von Bedeutung, da es in dieser Arbeit lediglich um die Darstellung von Graphen geht.

Beispiel:
Gegeben sei der Graph G in Abbildung 2.5, der formal wie folgt definiert ist: $G(V, E)$ mit $V = \{0, 1, 2, 3, 4\}$, $E = \{(0, 1), (1, 2), (2, 3), (3, 4)\}$ und $c((0, 1)) = 5$, $c((1, 2)) = 3$, $c((2, 3)) = 4$, $c((3, 4)) = 2$.

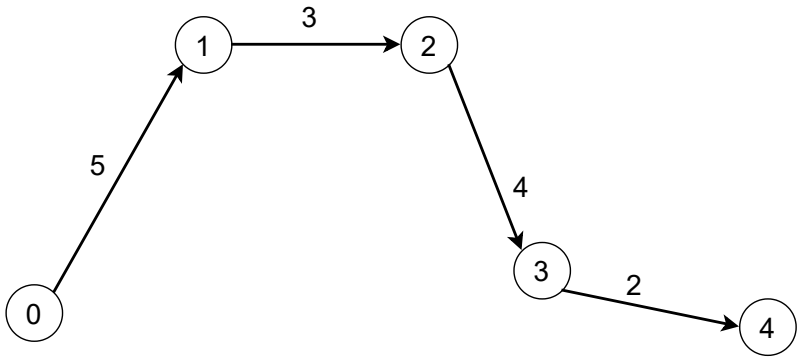


Abbildung 2.5: Beispiel eines Graphen

Eine mögliche aus dem Graph in Abbildung 2.5 durch simultane Knotenkontraktion entstehende CH G^* könnte wie in Abbildung 2.6 dargestellt aussehen, die formal wie folgt definiert ist: $G^*(V, E^*)$, wobei $E^* = E \cup \tilde{E}$ mit $\tilde{E} = \{(0, 2), (2, 4), (0, 4)\}$, $c^*(e) = c(e)$ für alle Originalkanten $e \in E$ und $c^*((0, 2)) = 8$, $c^*((2, 4)) = 6$, $c^*((0, 4)) = 14$. Außerdem gilt $level(0) = 3$, $level(1) = 1$, $level(2) = 2$, $level(3) = 1$ und $level(4) = 3$, da zuerst Knoten 1 und 3, dann Knoten 2 und dann Knoten 0 und 4 kontrahiert wurden.

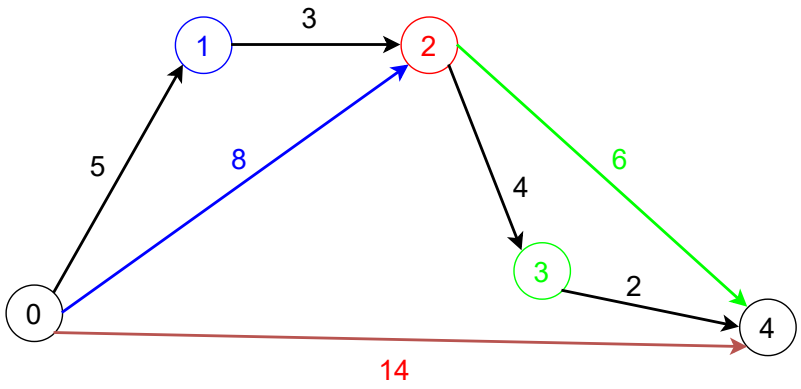


Abbildung 2.6: Beispiel einer Kontraktionshierarchie

2.5 Verwendete Technologien

Um die implementierungstechnische Umsetzung dieser Arbeit nachvollziehen zu können, ist es notwendig, die Grundlagen einiger der genutzten Technologien zu verstehen. Dies umfasst die Grundlagen von Leaflet, von dem OpenGL-Viewer und von JSON.

2.5.1 Leaflet

Leaflet ist die führende Open-Source JavaScript-Bibliothek für interaktive Karten. Sie wurde mit den Prinzipien Einfachheit, Performanz und Benutzbarkeit entworfen und läuft effizient auf allen großen Plattformen [Agaa].

Leaflet API

Leaflet erlaubt die Erstellung einer Karte auf einer Webseite und die Manipulation dieser durch verschiedene Methoden. Auf dieser Karte können verschiedene Schichten platziert werden. Beispiele für Schichten sind die Benutzeroberfläche, Raster oder Vektoren [Agab]. Meist, wie auch in dieser Arbeit, wird zunächst eine Karte erstellt, worauf eine Raster-Schicht für Kacheln gesetzt wird. Auf diese Raster-Schicht wird wiederum eine Vektor-Schicht für geografische Daten (zum Beispiel als GeoJSON-Objekt, siehe Abschnitt 2.5.3) gesetzt. Auf die Vektor-Schicht wird dann noch eine Schicht für die Benutzeroberfläche gesetzt (siehe Abschnitt 6.2.2). Durch das Übereinandersetzen von verschiedenen Schichten kann eine Mischung verschiedener Funktionalitäten erreicht werden.

Web Map Tile Service

Der Web Map Tile Service (WMTS) ist ein Service, der von der Open Geospatial Consortium (OGC) spezifiziert wurde, um digitale Karten im Web über gecachte Bildkacheln bereitzustellen [Arca]. Damit Karten für Anwendungen schnell zur Verfügung gestellt werden können, werden die Kartenkacheln serverseitig einmalig erzeugt und gespeichert [Geo]. Leaflet benutzt diesen Service für den sogenannten Tile-Layer, der Teil der Raster-Schicht ist [Agac].

2.5.2 OpenGL Viewer

Der OpenGL-Viewer von dem Institut für Formale Methoden der Informatik von der Arbeitsgruppe Algorithmik [SHi] kann Kartendaten visualisieren, die dem im Folgenden beschriebenen Format folgen:

In der ersten Zeile befindet sich die Anzahl der Knoten und in der zweiten Zeile befindet sich die Anzahl der Kanten. Ab Zeile 3 werden alle Informationen über die Knoten in der Datei aufgelistet. Jede Zeile ist folgendermaßen aufgebaut:

Breitengrad Längengrad

Nach den Knotendaten folgen alle Informationen über die Kanten in der Datei. Jede Zeile ist folgendermaßen aufgebaut:

StartknotenID ZielknotenID Malbreite Farbe

In Tabelle 2.1 sind die in den Zeilen vorkommenden Variablen näher spezifiziert.

Inhalt der Datei	**Beschreibung**
Breitengrad	Breitengrad dieses Knotens
Längengrad	Längengrad dieses Knotens
StartKnotenID	ID des Startknotens dieser Kante
ZielknotenID	ID des Zielknotens dieser Kante
Malbreite	Breite der Linie, mit der diese Kante visualisiert werden soll
Farbe	Farbe der Linie, mit der diese Kante visualisiert werden soll

Tabelle 2.1: Dateiformat von GL-Dateien

Dies bedeutet, dass die Knoten implizit durch ihre Position in der Datei IDs zugewiesen bekommen, die von den Kanten referenziert werden. Die Knoten IDs beginnen bei 0.

Für die CH in Abbildung 2.6 könnte die GL-Datei wie in Listing 2.1 dargestellt aussehen.

Listing 2.1 Beispiel einer GL-Datei

```
5               //Anzahl der Knoten
7               //Anzahl der Kanten
0 0             //Information des ersten Knotens (Knoten mit impliziter ID 0)
3 5
7 5
9 1
13 0            //Information des letzten Knotens (Knoten mit impliziter ID 4)
0 1 1 3         //Information der ersten Kante
0 2 1 3
0 4 1 3
1 2 1 3
2 3 1 3
2 4 1 3
3 4 1 3          //Information der letzten Kante
```

2.5.3 JavaScript Object Notation

JSON ist ein kompaktes, programmiersprachenunabhängiges und menschlich lesbares Datenaustauschformat [Conb].

GeoJSON

GeoJSON basiert auf JSON und ist ein Format, um geografische Datenstrukturen zu kodieren. Unterstützt werden Geometrien wie Punkte (*Point*), Linien (*LineString*), Polygone (*Polygon*) sowie mehrteilige Typen dieser Geometrien (wie *MultiPoint*, *MultiLineString*, *MultiPolygon*) [Cona]. GeoJSON nutzt das WGS 84 als geografisches Koordinaten-Referenzsystem [BDD+16]. Heute wird GeoJSON beispielsweise von Leaflet unterstützt.

In Listing 2.2 ist ein *MultiLineString*, bestehend aus drei Linien dargestellt, wobei die erste Linie aus drei Punkten, die zweite Linie aus zwei Punkten und die dritte Linie aus vier Punkten besteht. Die dazu passende Visualisierung könnte wie in Abbildung 2.7 dargestellt aussehen.

Listing 2.2 Beispiel eines GeoJSON-Objektes bestehend aus drei Linien

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [2.0, 2.0], [3.0, 3.0], [4.0, 3.0]
          ],
          [
            [1.0, 1.0], [1.0, 4.0]
          ],
          [
            [3.0, 1.0], [3.0, 2.0], [4.0, 2.0], [4.0, 1.0]
          ]
        ]
      }
    }
  ]
}
```

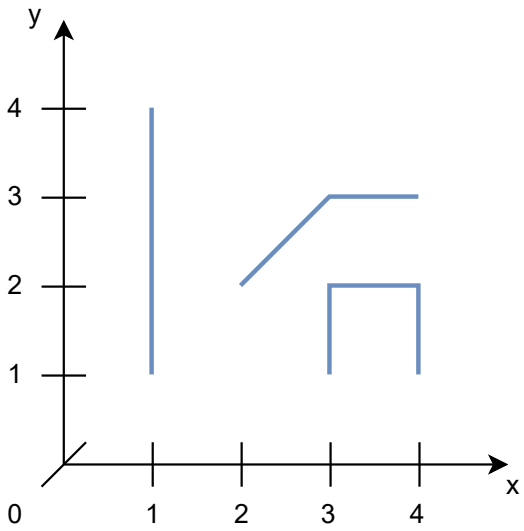


Abbildung 2.7: Visualisierung eines beispielhaften GeoJSON-Objektes

3 Einlesen der Daten und Grapherstellung

Zu Beginn müssen zunächst die benötigten Eingabedateien in den Hauptspeicher eingelesen werden. Dazu wird der Java *BufferedReader* genutzt. Gegeben sind eine SCH- und eine dazugehörige RANGES-Datei. Der Inhalt und das Format dieser Dateien, sowie das Einlesen und Speichern der eingelesenen Daten in einer eigenen Graphdatenstruktur, werden im Folgenden näher beschrieben.

3.1 SCH Datei

Die SCH-Dateien definieren vorberechnete CHs (siehe Abschnitt 2.4). Hierbei ist zu beachten, dass diese CHs aus Straßennetzwerken aufgebaut wurden und für das Routing darauf benutzt werden können. Der Graph, der das jeweilige Straßennetzwerk darstellt, ist außerdem gerichtet, enthält also sowohl Vorwärts- als auch Rückwärtskanten. Die in der Datei enthaltenen Graphdaten basieren dabei auf OSM-Daten, daher sind Koordinaten gemäß dem WGS 84 gespeichert [FSS17].

Im Folgenden wird das Format einer SCH-Datei beschrieben:

Nach einigen Platzhalterzeilen folgt der eigentliche Inhalt der Datei ab Zeile 11. In Zeile 11 befindet sich die Anzahl an Knoten und in Zeile 12 die Anzahl der Kanten. Ab Zeile 13 werden alle Informationen über die Knoten im Datensatz aufgelistet. Jede Zeile ist folgendermaßen aufgebaut:

nodeID nodeID2 latitude longitude elevation level

Für die Zwecke dieser Arbeit sind nur *nodeID*, *latitude*, *longitude* und *level* von Bedeutung. In Tabelle 3.1 sind diese Variablen näher spezifiziert.

Variable	**Beschreibung**
nodeID	ID für diesen Knoten, durch den dieser eindeutig gekennzeichnet ist
latitude	Breitengrad dieses Knotens
longitude	Längengrad dieses Knotens
level	definiert, ab welchem Zoomlevel dieser Knoten (beziehungsweise eine Kante, die mit diesem Knoten verbunden ist) visualisiert werden soll

Tabelle 3.1: Format der Knoteninformationen von SCH-Dateien

Jede Kante ist nur auf bestimmten Zoomlevels überhaupt relevant. Das Zoomlevel bestimmt also den Detaillierungsgrad der relevanten Kanten.

Nach den Knotendaten folgen alle Informationen über die Kanten im Datensatz. Jede Zeile ist folgendermaßen aufgebaut:

srcIDX trgIDX cost type maxspeed child1 child2

Für die Zwecke dieser Arbeit sind nur *srcIDX*, *trgIDX*, *cost*, *child1* und *child2* von Bedeutung, wobei *cost* ausschließlich für die Kosten-Fehlermetrik, siehe Abschnitt 4.2.1, relevant ist. In Tabelle 3.2 sind diese Variablen näher spezifiziert.

Variable	Beschreibung
srcIDX	ID des Startknotens dieser Kante
trgIDX	ID des Zielknotens dieser Kante
cost	Kosten, welche diese Kante verursacht. Für diese Arbeit sind die Kosten einer Kante eine Approximation für die Länge dieser Kante.
child1	ID der ersten Kante, die beim Entpacken dieser Kante entsteht
child2	ID der zweiten Kante, die beim Entpacken dieser Kante entsteht

Tabelle 3.2: Format der Kanteninformationen von SCH-Dateien

Besitzen *child1* und *child2* einer Kante den Wert -1, dann ist diese Kante eine Originalkante und kann nicht weiter entpackt werden. Im anderen Fall ist diese Kante ein Shortcut, der zwei Kindkanten besitzt und entpackt werden kann. Entpacken eines Shortcuts bedeutet in unserem Fall, dass nicht mehr der Shortcut selbst visualisiert wird, sondern seine beiden Kindkanten. Der Shortcut wird also durch seine beiden Kindkanten ersetzt. Die Kanten besitzen im Gegensatz zu den Knoten keine expliziten, sondern nur implizite IDs. Die i-te Kante im Datensatz hat ID i. Dies ist unbedingt nötig, da mit *child1* und *child2* auf andere Kanten verwiesen wird [FMI22].

Für die CH in Abbildung 2.6 könnte die SCH-Datei wie in Listing 3.1 dargestellt aussehen.

Listing 3.1 Beispiel einer SCH-Datei

```
#                   //Platzhalter
...
#

5                   //Anzahl der Knoten
7                   //Anzahl der Kanten
0 100 0 0 0 3       //Information des ersten Knotens
1 101 3 5 0 1
2 102 7 5 0 2
3 103 9 1 0 1
4 104 13 0 0 3      //Information des letzten Knotens
0 1 5 3 50 -1 -1    //Information der ersten Kante
0 2 8 3 50 0 3
0 4 14 3 50 1 5
1 2 3 3 50 -1 -1
2 3 4 3 50 -1 -1
2 4 6 3 50 4 6
3 4 2 3 50 -1 -1    //Information der letzten Kante
```

3.2 RANGES Datei

Wie in Abschnitt 3.1 erwähnt, ist jede Kante nur auf bestimmten Zoomlevels überhaupt relevant. Später in Abschnitt 5.2.1 wird beschrieben, wie genau man die auf einem gegebenen Zoomlevel relevanten Kanten mittels den Levels aus der SCH-Datei identifiziert. Man hat hierbei jedoch das Problem, dass zum einen teilweise die physische Realität sehr schlecht repräsentiert wird und zum anderen die Position und Darstellung von Kreuzungen im Straßennetzwerk verändert wird. Diese Probleme kann man versuchen zu lösen, indem man Shortcuts (bis zum Originalpfad) entpackt. Da dies jedoch zu sehr vielen Kanten führt, können gewisse Kanten wieder in die Shortcuts, aus denen sie entstanden sind, eingepackt werden. Zum einen können Knoten mit weniger als drei Nachbarn wieder kontrahiert werden, da nur ein Knoten mit mehr als zwei Nachbarn eine Kreuzung darstellt. Zum anderen muss der entstehende Shortcut ein ästhetisches Kriterium erfüllen, zum Beispiel nicht zu lang sein oder nicht zu sehr von dem Originalpfad abweichen. Geschehen diese Berechnungen in einem Vorverarbeitungsschritt, so resultiert für jede Kante ein lückenloses Intervall an Levels, auf denen diese Kante relevant ist [Erw].

Eine RANGES-Datei ist immer genau einer SCH-Datei zugehörig. Die RANGES-Dateien enthalten alle Informationen darüber, welche Kanten bei welchem Zoomlevel vor dem Start des Entpackprozesses visualisiert werden sollten. Pro Kante existiert eine Zeile:

edgeID levelStart levelEnd

In Tabelle 3.3 sind diese Variablen näher spezifiziert.

Variable	**Beschreibung**
edgeID	ID der zu visualisierenden Kante
levelStart	Level, ab welchem diese Kante visualisiert wird
levelEnd	Level, bis zu welchem diese Kante visualisiert wird

Tabelle 3.3: Dateiformat von RANGES-Dateien

Das bedeutet, dass die Kante mit ID *edgeID* zwischen den Zoomlevels *levelStart* und *levelEnd* visualisiert werden sollte. Hierbei bedeutet ein großes Zoomlevel, dass wenig Kanten visualisiert werden sollen und ein kleines Zoomlevel, dass viele Kanten visualisiert werden sollen. Insbesondere gilt $levelStart \geq levelEnd$. Wenn sowohl *levelStart* als auch *levelEnd* für eine Kante den Wert -1 haben, soll diese Kante nie visualisiert werden. Die Kanten-IDs referenzieren hierbei die Kanten aus der zugehörigen SCH-Datei.

Eine fiktive RANGES-Datei könnte wie in Listing 3.2 dargestellt aussehen.

Listing 3.2 Beispiel einer RANGES-Datei

```
0 179 179       //Kante mit ID 0 wird nur auf Zoomlevel 179 visualisiert
1 -1 -1         //Kante mit ID 1 wird nie visualisiert
...
40 178 0        //Kante mit ID 40 wird zwischen Zoomlevel 178 und 0 visualisiert
...
229 178 176     //Kante mit ID 229 wird zwischen Zoomlevel 178 und 176 visualisiert
...
554 0 0         //Kante mit ID 554 wird nur auf Zoomlevel 0 visualisiert
...
```

3.3 Graph Datenstruktur

Die aus den SCH- und RANGES-Dateien in den Hauptspeicher eingelesenen Informationen werden in einer Datenstruktur abgespeichert. Ob es sich beim Einlesen der SCH-Datei noch um Knoteninformationen oder schon um Kanteninformationen handelt, wird über die Anzahl der Werte pro Zeile unterschieden. Für die Knoten sind es sechs Werte, für die Kanten sieben Werte pro Zeile.

Die Gesamtanzahl der Knoten sowie alle anderen Knoteninformationen werden in einem *Nodes*-Objekt gespeichert. Die Gesamtanzahl der Kanten sowie alle anderen Kanteninformationen werden in einem *Edges*-Objekt gespeichert. Ein *Graph*-Objekt besteht aus jeweils einem *Nodes*- und einem *Edges*-Objekt und verbindet dann Knoten- und Kanteninformationen. Alle Informationen, die einmal pro Knoten beziehungsweise Kante existieren, werden in jeweils einem Array (im *Nodes*- beziehungsweise im *Edges*-Objekt) gespeichert, wobei der Eintrag an Index i die Information des Knotens beziehungsweise der Kante i angibt. Die Größen der Arrays werden durch die zuvor eingelesene Anzahl von Knoten und Kanten bestimmt.

In Abbildung 3.1 ist die resultierende *Graph*-Datenstruktur dargestellt.

Für den Graphen aus Abbildung 2.6 sehen die Arrays im *Nodes*-Objekt wie in Abbildung 3.2 dargestellt aus, wobei die Arrays eine Größe von $nodeCount = 5$ besitzen. Die blau eingefärbten Einträge stellen hierbei die Informationen für den Knoten mit ID 1 dar. Dieser hat einen Längengrad von 3, einen Breitengrad von 5 und ist auf dem Level 1.

Die Arrays im *Edges*-Objekt sehen für den Graphen aus Abbildung 2.6 wie in Abbildung 3.3 aus, wobei die Arrays eine Größe von $edgeCount = 7$ besitzen. Die blau eingefärbten Einträge stellen hierbei die Informationen für die Kante mit ID 2 dar. Diese verbindet den Knoten mit ID 0 mit dem Knoten mit ID 4, mit Kosten von 14. Man kann diese Kante entpacken, da *isShortcut* auf *true* gesetzt ist. Dabei entstehen die Kanten mit ID 1 und ID 5.

Die *Graph*-Klasse bietet einen *Getter* für das zu einem Graphen gehörende *Nodes*-Objekt und einen *Getter* für das zu einem Graphen gehörende *Edges*-Objekt an, die dann jeweils *Getter* und *Setter* für alle gespeicherten und zu speichernden Daten anbieten.

Laufzeit: Um die *Graph*-Datenstruktur zu füllen, müssen lediglich einmal über die SCH- und Ranges-Dateien gelesen werden. Sei V die Anzahl der Knoten und sei E die Anzahl der Kanten (wie in Zeile 11 und 12 der SCH-Datei angegeben), dann kann die SCH-Datei in $O(V + E)$ und die RANGES-Datei in $O(E)$ gelesen werden. Insgesamt können beide Dateien in $O(V + E)$ gelesen werden.

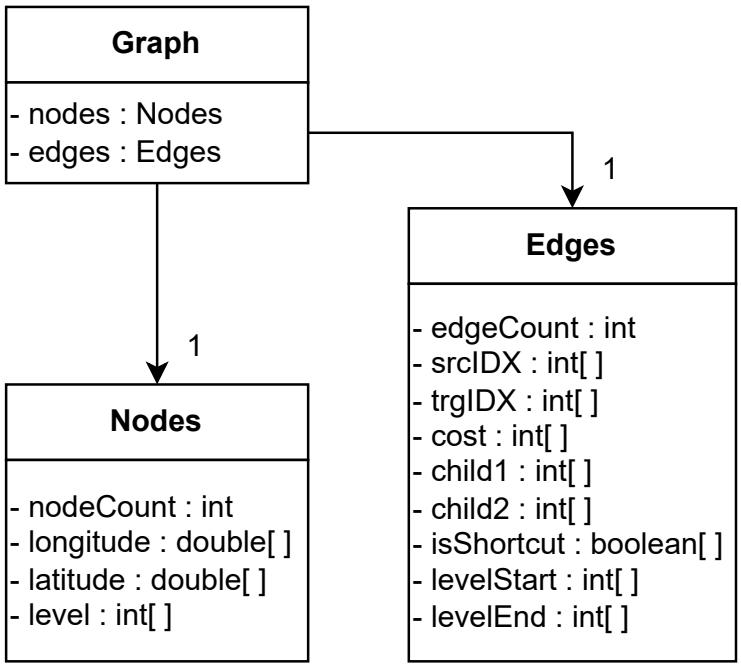


Abbildung 3.1: Klassendiagramm der Graphdatenstruktur

longitude	longitude	level
0	0	3
3	5	1
7	5	2
9	1	1
13	0	3

Abbildung 3.2: Arrays im *Nodes*-Objekt für einen beispielhaften Graphen

srcIDX	trgIDX	cost	child1	child2	isShortcut
0	1	5	-1	-1	0
0	2	8	0	3	1
0	4	14	1	5	1
1	2	3	-1	-1	0
2	3	4	-1	-1	0
2	4	6	4	6	1
3	4	2	-1	-1	0

Abbildung 3.3: Arrays im *Edges*-Objekt für einen beispielhaften Graphen

4 Berechnung der optimalen Entpackreihenfolgen

Für jeden im Graphen aus Abschnitt 3.3 enthaltenen Shortcut soll eine optimale Entpackreihenfolge in einem Vorverarbeitungsschritt berechnet werden. Durch die Kombination von unterschiedlichen Fehlermetriken und unterschiedlichen Modi ergeben sich verschiedene optimale Entpackreihenfolgen. Im Folgenden befindet sich daher eine Erklärung, wie genau der Fehler eines Shortcuts berechnet wird, eine Auflistung und Definition aller in dieser Arbeit genutzten Fehlermetriken und Modi, sowie das Vorgehen bei der tatsächlichen Vorberechnung aller optimalen Entpackreihenfolgen.

4.1 Fehlerberechnung

Ein Shortcut stellt in dieser Arbeit immer eine Approximation einer Grad-2-Kette, bestehend aus Originalkanten, dar. Der Fehler eines Shortcuts wird immer in Bezug auf den von ihm approximierten Originalpfad berechnet. Um den originalen Pfad zu bestimmen, der durch einen gewissen Shortcut approximiert wird, wird dieser Shortcut vollständig entpackt. Das bedeutet, dass eine Kante so lange durch ihre beiden Kindkanten ersetzt wird, bis keine Kante mehr Kindkanten besitzt, also nur noch Originalkanten verbleiben. Dies kann beispielsweise mithilfe eines Stacks implementiert werden, der mit dem vollständig zu entpackenden Shortcut initialisiert wird. Anschließend wird immer das oberste Element von dem Stack entnommen und die beiden Kindkanten dieser Kante auf den Stack gepusht, falls das entnommene Element ein Shortcut war.

Laufzeit: Sei E' die maximale Anzahl an Kanten auf dem Originalpfad, der von einem einzigen Shortcut approximiert wird (abhängig von dem gegebenen Eingabegraph). Die Anzahl von Entpackschritten bis zur Erreichung des Originalpfades ist dann für diesen Shortcut durch E' begrenzt. Aufgrund der genutzten Stack-Datenstruktur ist das Ersetzen eines Shortcuts durch seine beiden Kindkanten in $O(1)$ möglich. Aufgrund der Nutzung von Arrays ist das Finden der beiden Kindkanten eines Shortcuts ebenfalls in $O(1)$ möglich. Die Gesamtlaufzeit der vollständigen Entpackung eines Shortcuts ist daher in $O(E')$.

Berechnungen werden grundsätzlich im kartesischen Koordinatensystem durchgeführt. Dazu werden zunächst die im *Graph*-Objekt gespeicherten geografischen Koordinaten mittels der Mercator-Projektion (siehe Abschnitt 2.1.3) auf kartesische Koordinaten abgebildet. Für diese Zwecke werden die *Java 2D*-Klassen *Point2D* und *Line2D* genutzt. Diese bieten unter anderem die Möglichkeit, Knoten als *Point2D*-Objekt und Kanten als *Line2D*-Objekt zu speichern und Distanzen zwischen diesen zu berechnen [Ora20] [Ora23].

4.2 Fehlermetriken

Die Fehlermetrik definiert, wie genau der Fehler zwischen einem Shortcut und dem von ihm approximierten Originalpfad berechnet wird. Die in dieser Arbeit verwendeten Fehlermetriken sind im Folgenden aufgeführt.

4.2.1 Kosten

Der Fehler eines Shortcuts kann definiert werden als die Kosten dieser Kante, die als Approximation für die Länge dieser Kante dienen können. Die Kosten für jede Kante können aus der SCH-Datei entnommen werden.

Laufzeit: Das Auslesen der Kosten für eine Kante ist in $O(1)$ möglich, da nach dem Einlesen des Eingabegraphen in den Hauptspeicher die Kosten jeder Kante in einem Array gespeichert sind.

4.2.2 Hausdorff

Die Hausdorff-Distanz von einer Menge A zu einer Menge B ist die maximale Distanz der Menge A zum nächstgelegenen Punkt in der Menge B. Gegeben sind also zwei Mengen von Punkten. Formal ist die Hausdorff-Distanz von A zu B definiert als

$$h(A, B) = max_{a \in A}\{min_{b \in B}\{d(a, b)\}\}$$

wobei $d(a, b)$ irgendeine Metrik zwischen den Punkten a und b darstellt [GB]. Beispielsweise könnte $d(a, b)$ die euklidische Distanz zwischen $a = (x_1, y_1)$ und $b = (x_2, y_2)$ darstellen [Stua], mit

$$\rho_{euclid} : (\mathbb{R} \times \mathbb{R})^2 \to \mathbb{R} : (x_1, y_1) \times (x_2, y_2) \mapsto \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.1}$$

Sei für die Zwecke dieser Arbeit A eine endliche Menge von Punkten und sei B die unendliche Menge aller Punkte, die auf einem Liniensegment liegen.

Für die Zwecke dieser Arbeit wird dann die Hausdorff-Distanz von einer Menge von Punkten $\{P_i | 0 \leq i \leq n\}$ zu einem Liniensegment p definiert als

$$h(\{P_i | 0 \leq i \leq n\}, p) = max\{d(P_i, p) | 0 \leq i \leq n\}$$

wobei $d(P_i, p)$ die euklidische Distanz von Punkt P_i zum Liniensegment p angibt. Der Fehler eines Shortcuts kann dann definiert werden als die Hausdorff-Distanz von der Menge von Punkten auf dem Originalpfad zum Shortcut selbst.

In Abbildung 4.1 ist ein Shortcut mit zugehörigem Originalpfad dargestellt, der die Hausdorff-Berechnung verdeutlichen soll.

Gegeben seien die Originalkanten a, b, c, d und der Shortcut e. Seien A, B, C, D, E die Knoten auf dem zu e gehörenden Originalpfad. Es gelten $d(A, e) = 0$, $d(B, e) = 2$, $d(C, e) = 3$, $d(D, e) = 2$ und $d(E, e) = 0$. Nach obiger Formel lässt sich die Hausdorff-Distanz h zwischen der Menge von Punkten auf dem Originalpfad $\{A, B, C, D, E\}$ zum Shortcut e berechnen als $h(\{A, B, C, D, E\}, e) = max\{0, 2, 3, 2, 0\} = 3$.

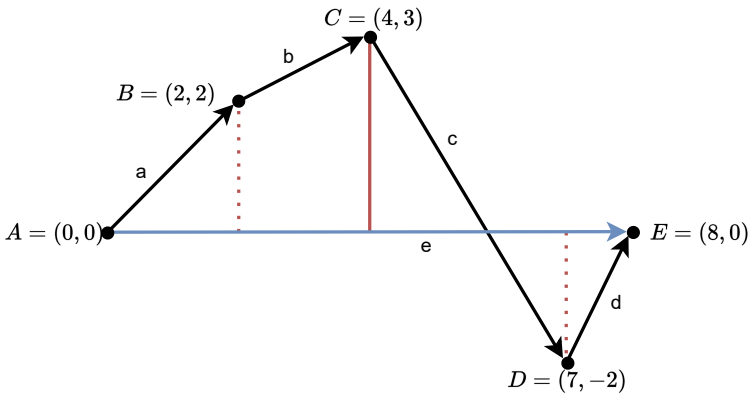


Abbildung 4.1: Beispiel zur Berechnung der Hausdorff-Distanz

Es existieren auch Shortcuts, die im selben Punkt starten und enden (zum Beispiel, wenn ein Kreisverkehr durch einen einzigen Shortcut approximiert wird). In diesem Fall wird die Hausdorff-Distanz von einer Menge von Punkten $\{P_i | 0 \leq i \leq n\}$ zu einem Punkt P definiert als

$$h(\{P_i | 0 \leq i \leq n\}, P) = max\{d(P_i, P) | 0 \leq i \leq n\}$$

wobei $d(P_i, P)$ die euklidische Distanz von Punkt P_i zum Punkt P angibt. Der Fehler eines Shortcuts kann dann definiert werden als die Hausdorff-Distanz von der Menge von Punkten auf dem Originalpfad zum Startpunkt (oder Endpunkt) des Shortcuts.

Laufzeit: Die Menge der Punkte auf dem Originalpfad kann, wie in Abschnitt 4.1 beschrieben, in $O(E')$ berechnet werden. Für jeden dieser Punkte kann die Distanz zum Shortcut in $O(1)$ berechnet werden. Das Maximum einer Menge mit E' Elementen kann in $O(E')$ identifiziert werden. Die Gesamtlaufzeit zum Berechnen der Hausdorff-Distanz beläuft sich daher für einen Shortcut auf $O(E')$ und für alle Shortcuts auf $O(E \cdot E')$.

4.2.3 Fréchet

Gegeben seien zwei Pfade, die je aus einer Sequenz von Punkten bestehen. Dann lässt sich die diskrete Fréchet-Distanz anschaulich folgendermaßen erklären:

Eine Person geht mit ihrem Hund spazieren. Dabei bewegt sich die Person auf dem einen Pfad und der Hund auf dem anderen Pfad vorwärts. Zu jedem Zeitpunkt bewegt sich genau einer der beiden genau einen Punkt weiter vorwärts, bis beide am Ende ihres jeweiligen Pfades ankommen. Gesucht ist die kürzeste benötigte Leine von der Person zum Hund, sodass sowohl die Person als auch ihr Hund jeweils ihren Pfad, je Punkt für Punkt, entlang laufen können [Kri21].

Der Fehler eines Shortcuts kann dann definiert werden als die diskrete Fréchet-Distanz zwischen der Menge von Punkten auf dem Originalpfad und einer generierten Menge von Punkten auf dem Shortcut selbst.

Hierbei ist zu beachten, dass für die Zwecke dieser Arbeit auf dem Shortcut künstlich genau so viele Punkte uniform verteilt generiert werden, wie sich auch auf dem dazugehörigen Originalpfad befinden.

Die diskrete Fréchet-Distanz kann dann, wie in Algorithmus 4.1 dargestellt, mittels dynamischer Programmierung berechnet werden [Kri21].

Algorithmus 4.1 Berechnung der Fréchet-Distanz mittels dynamischer Programmierung

```
procedure computeFrechet(shortcutId)
    o[] ← getPointsOnOriginalPath(shortcutId)
    numberOfPoints ← o.length()
    s[] ← getPointsUniformlyDistributedOnShortcut(shortcutId)
    d[][] ← d[numberOfPoints][numberOfPoints]
    d[0][0] ← euclid(o[0], s[0])                                  // Fill the first entry
    for i from 1 to numberOfPoints do                  // Fill the first column with distances
        d[i][0] ← max(d[i – 1][0], euclid(o[i], s[0]))
    end for
    for j from 1 to numberOfPoints do                     // Fill the first row with distances
        d[0][j] ← max(d[0][j – 1], euclid(o[0], s[j]))
    end for
    for i from 1 to numberOfPoints do
        for j from 1 to numberOfPoints do          // Fill the remaining entries with distances
            d[i][j] ← max(min(d[i – 1][j], d[i][j – 1], d[i – 1][j – 1]), euclid(o[i], s[j]))
        end for
    end for
    return d[numberOfPoints – 1][numberOfPoints – 1]
end procedure
```

$euclid(o[i], s[j])$ beschreibt dabei die euklidische Distanz zwischen dem i-ten Punkt auf dem Originalpfad und dem j-ten Punkt auf dem Shortcut. Die entsprechende Formel ist in Gleichung (4.1) definiert, wobei $o[i] = (x_1, y_1)$ und $s[j] = (x_2, y_2)$ ist.

In Abbildung 4.2 ist ein Shortcut mit zugehörigem Originalpfad dargestellt, der die Variablenbelegung in der Fréchet-Berechnung verdeutlichen soll.

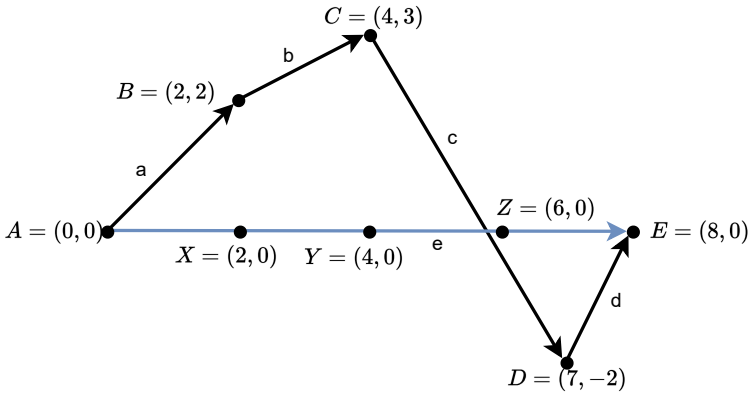


Abbildung 4.2: Beispiel zur Variablenbelegung in der Berechnung der Fréchet-Distanz

Gegeben seien die Originalkanten a, b, c, d und der Shortcut e. Seien A, B, C, D, E die Knoten auf dem zu e gehörenden Originalpfad. Auf e werden (zusätzlich zum Startknoten A und Zielknoten D) die Knoten X, Y und Z generiert. Es gelten daher $o[0] = A$, $o[1] = B$, $o[2] = C$, $o[3] = D$, $o[4] = E$ und $s[0] = A$, $s[1] = X$, $s[2] = Y$, $s[3] = Z$, $s[4] = E$ und $numberOfPoints = 5$.

Laufzeit: Die Menge der Punkte auf dem Originalpfad kann, wie in Abschnitt 4.1 beschrieben, in $O(E')$ berechnet werden. Die Generierung der Punkte auf dem Shortcut ist durch die Anzahl der Punkte auf dem Originalpfad begrenzt und daher ebenfalls in $O(E')$ möglich. Die Einträge in der Matrix werden durch die Berechnung von euklidischen Distanzen, Abfragen von bereits existierenden Werten in der Matrix und durch das Finden von Minimums und Maximums über eine Menge mit konstant vielen Elementen berechnet, was alles in $O(1)$ möglich ist. Es müssen |Punkte auf dem Originalpfad|^2 viele Einträge berechnet werden, was in $O(E'^2)$ möglich ist. Die Gesamtlaufzeit zum Berechnen der Fréchet-Distanz beläuft sich daher für einen Shortcut auf $O(E'^2)$ und für alle Shortcuts auf $O(E \cdot E'^2)$.

4.2.4 Flächeninhalt

Der Fehler eines Shortcuts kann definiert werden als der Flächeninhalt zwischen dem Originalpfad und dem Shortcut selbst.

Dazu werden zunächst die Schnittpunkte zwischen dem Originalpfad und dem Shortcut berechnet. Eine entsprechende Formel ist im nächsten Unterkapitel zu finden. Anschließend werden die Flächeninhalte der so entstehenden Polygone berechnet und aufsummiert. Der Flächeninhalt eines Polygons, bestehend aus den Punkten $P_0, P_1, ..., P_n$, wobei $P_i = (x_i, y_i)$ und $P_0 = P_n$, kann wie folgt mit der Trapezoid-Formel berechnet werden [Con23d]:

$$A = \frac{1}{2} \sum_{i=1}^{n} (y_i + y_{i+1})(x_i - x_{i+1}) \tag{4.2}$$

Der Pseudocode zur Berechnung des Flächeninhalts ist in Algorithmus 4.2 dargestellt.

Algorithmus 4.2 Berechnung des Flächeninhalts zwischen einem Shortcut und dem von dem Shortcut approximierten Originalpfad

```
procedure computeAreaBetweenShortcutAndOriginalPath(shortcutId)
    polygons[] ← getPolygons(shortcutId)
    sum ← 0
    for polygon in polygons do                    // Iterate over all polygons
        sum ← sum + computeAreaPolygon(polygon)
    end for
    return sum
end procedure
```

In Abbildung 4.3 ist ein Shortcut mit zugehörigem Originalpfad dargestellt, der die Flächeninhalt-Berechnung verdeutlichen soll.

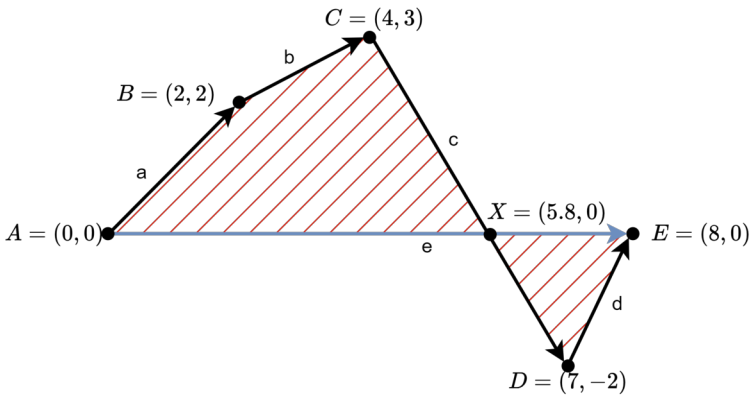


Abbildung 4.3: Beispiel zur Berechnung des Flächeninhalts

Gegeben seien die Originalkanten a, b, c, d und der Shortcut e. Seien A, B, C, D, E die Knoten auf dem zu e gehörenden Originalpfad. Nun werden zunächst die Schnittpunkte zwischen e und dem Originalpfad berechnet. Neben dem Startpunkt A und dem Endpunkt E des Shortcuts sei dies nur der Punkt $X = (5.8, 0)$. Es entstehen folglich zwei Polygone. Das erste Polygon besteht aus den Punkten A, B, C, X, das zweite Polygon besteht aus den Punkten X, D, E. Es werden jetzt die Flächeninhalte dieser beiden Polygone berechnet und anschließend addiert.

Laufzeit: Zur Berechnung der Schnittpunkte zwischen dem Originalpfad und dem Shortcut und zur Berechnung der so entstehenden Polygone, muss einmal über die Punkte auf dem Originalpfad iteriert werden, was, wie in Abschnitt 4.1 beschrieben, in $O(E')$ möglich ist. Die Zeit zur Berechnung des Flächeninhalts eines Polygons ist durch die Anzahl der Punkte, die dieses Polygon definieren, beschränkt. Die Anzahl der Punkte, die alle gefundenen Polygone definieren, ist wiederum durch die Anzahl der Punkte auf dem Originalpfad beschränkt. Daher beläuft sich die Gesamtlaufzeit zum Berechnen des Flächeninhalts zwischen einem Shortcut und seinem Originalpfad auf $O(E')$ und zwischen allen Shortcuts und ihren Originalpfaden auf $O(E \cdot E')$.

Berechnung des Schnittpunkts zweier Linien

Gegeben seien zwei Linien *line1* und *line2*, wobei *line1* durch die Punkte (x_1, y_1) und (x_2, y_2) und *line2* durch die Punkte (x_3, y_3) und (x_4, y_4) geht.

Zuerst wird mittels der Methode *intersectsLine* der *Line2D*-Klasse überprüft, ob sich diese zwei Liniensegmente schneiden. Falls das der Fall ist, kann der Schnittpunkt (x, y) der durch *line1* und *line2* definierten unendlich langen Linien berechnet werden [Wei23a], mit

$$x = \frac{(x_1 \cdot y_2 - y_1 \cdot x_2) \cdot (x_3 - x_4) - (x_1 - x_2) \cdot (x_3 \cdot y_4 - y_3 \cdot x_4)}{(x_1 - x_2) \cdot (y_3 - y_4) - (y_1 - y_2) \cdot (x_3 - x_4)} \tag{4.3}$$

und

$$y = \frac{(x_1 \cdot y_2 - y_1 \cdot x_2) \cdot (y_3 - y_4) - (y_1 - y_2) \cdot (x_3 \cdot y_4 - y_3 \cdot x_4)}{(x_1 - x_2) \cdot (y_3 - y_4) - (y_1 - y_2) \cdot (x_3 - x_4)} \tag{4.4}$$

4.2.5 Distanz

Der Fehler eines Shortcuts kann definiert werden als die Distanz des zugehörigen Originalpfades. Diese Distanz ist in dieser Arbeit definiert als die Summe der Distanzen zwischen jeweils zwei benachbarten Knoten auf diesem Pfad. Initial müssen hierfür die Distanzen der Originalkanten berechnet werden. Die Distanz einer Originalkante ergibt sich aus der euklidischen Distanz zwischen Start- und Zielpunkt dieser Kante. Die entsprechende Formel ist in Gleichung (4.1) definiert, wobei (x_1, y_1) Startpunkt und (x_2, y_2) Zielpunkt sind.

Anschließend kann für einen Shortcut i mit Kindkanten *child1* und *child2* dessen Fehler als $distance(i) = distance(child1) + distance(child2)$ berechnet werden. Voraussetzung hierfür ist, dass die Fehler von *child1* und *child2* jeweils bereits zuvor berechnet wurden.

Der entsprechende Pseudo-Code ist in Algorithmus 4.3 zu finden, wobei $euclid(edge)$ die euklidische Distanz von dem Startpunkt zum Zielpunkt der Kante *edge* berechnet.

Algorithmus 4.3 Berechnung der Distanzen als Fehlermetrik

```
procedure computeDistances
    error[] ← error[edges.getEdgeCount()]
    for edge in edges do                                  // Compute errors of all original edges
        if !isShortcut(edge) then                         // Check if the current edge is no shortcut
            error[edge] ← euclid(edge)
        else
            error[edge] ← −1                              // i.e. this error has not been computed yet
        end if
    end for
    change ← true
    while change do                                       // While still errors are being computed
        change ← false
        for edge in edges do                              // Compute errors of remaining edges
            if error[edge] = −1 then                      // The error of edge has not been computed yet
                child1 ← getChild1(edge)
                child2 ← getChild2(edge)
                if error[child1] ! = −1 and error[child2] ! = −1 then
                    error[edge] ← error[child1] + error[child2]          // Compute the error
                    change ← true
                end if
            end if
        end for
    end while
    return error[]
end procedure
```

In Abbildung 4.4 ist ein Shortcut mit zugehörigem Originalpfad dargestellt, der die Distanz-Berechnung verdeutlichen soll.

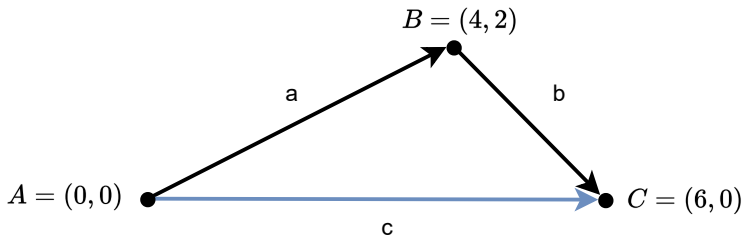


Abbildung 4.4: Beispiel zur Berechnung der Distanz

Gegeben seien die Originalkanten a und b sowie der Shortcut c. Es gelten $distance(a) = \sqrt{(0-4)^2 + (0-2)^2} = \sqrt{20}$, $distance(b) = \sqrt{(4-6)^2 + (2-0)^2} = \sqrt{8}$ und $distance(c) = distance(a) + distance(b) = \sqrt{20} + \sqrt{8} \approx 7.3$

Laufzeit: Die euklidische Distanz einer Originalkante kann in $O(1)$ berechnet werden. Um initial die Distanzen aller Originalkanten zu berechnen, muss einmal über alle Kanten iteriert werden, was in $O(E)$ läuft, wobei E die Anzahl aller Kanten definiert. Anschließend wird so lange über alle Kanten iteriert, wie noch Distanzen berechnet werden müssen. Die Anzahl der Iterationen ist durch die Anzahl von Kanten auf dem längsten Originalpfad begrenzt, da dies die maximale Tiefe des Entpackprozesses eines Shortcuts darstellt und in jeder Iteration alle Distanzen von Shortcuts einer gewissen Tiefe berechnet werden können. Daher ist die Anzahl der Iterationen durch E' beschränkt. Zur Überprüfung, ob die Distanz eines Shortcuts noch berechnet werden muss, ob sie aktuell berechnet werden kann und zur tatsächlichen Berechnung dieser, sind ausschließlich Zugriffe auf Arrays nötig, was in $O(1)$ geschehen kann. Die Gesamtlaufzeit zum Berechnen der Distanzen aller Kanten beläuft sich daher auf $O(E \cdot E')$.

4.3 Modi

Der Modus gibt an, wie die berechneten Fehler genutzt werden, um denjenigen Shortcut zu bestimmen, der als Nächstes entpackt werden soll. Die in dieser Arbeit verwendeten Modi sind im Folgenden aufgeführt.

4.3.1 Größter/Kleinster Fehler

Aus einer Menge von Shortcuts, die alle entpackbar wären, wird im nächsten Schritt derjenige Shortcut entpackt, der unter all diesen Shortcuts den größten (beziehungsweise kleinsten) Fehler, basierend auf einer bestimmten Fehlermetrik, zum Originalpfad besitzt. Wenn mehrere Shortcuts als Fehler den größten (beziehungsweise kleinsten) Fehler besitzen und der größte (beziehungsweise kleinste) Fehler somit nicht eindeutig ist, dann wird im nächsten Schritt derjenige Shortcut entpackt, dessen überbrückter Knoten unter all diesen Shortcuts die kleinste ID besitzt.

4.3.2 Größte/Kleinste Fehlerreduktion

Aus einer Menge von Shortcuts, die alle entpackbar wären, wird im nächsten Schritt derjenige Shortcut entpackt, der unter all diesen Shortcuts die größte (beziehungsweise kleinste) Fehlerreduktion, basierend auf einer bestimmten Fehlermetrik, besitzt. Für jeden dieser Shortcuts wird dafür sowohl der Fehler dieses Shortcuts berechnet, als auch dieser Shortcut einmal entpackt und der totale Fehler der beiden Kindkanten berechnet. Anschließend wird die Differenz dieser beiden Fehler berechnet, also der Fehler des Shortcuts minus der totale Fehler seiner beiden Kindkanten. Im nächsten Schritt wird dann derjenige Shortcut mit der größten (beziehungsweise kleinsten) Differenz entpackt.

Wenn mehrere Shortcuts als Fehlerreduktion die größte (beziehungsweise kleinste) Fehlerreduktion besitzen und die größte (beziehungsweise kleinste) Fehlerreduktion somit nicht eindeutig ist, dann wird im nächsten Schritt derjenige Shortcut entpackt, dessen überbrückter Knoten unter all diesen Shortcuts die kleinste ID besitzt.

Um den totalen Fehler der beiden Kindkanten $child1$ und $child2$ einer Kante zu berechnen, gibt es wie im Folgenden beschrieben unterschiedliche Möglichkeiten, die je nach Fehlermetrik sinnvoll (oder auch nicht sinnvoll) sein können.

Summe

Der totale Fehler der beiden Kindkanten $child1$ und $child2$ einer Kante kann als Summe der Fehler der Kindkanten berechnet werden. Dies ist beispielsweise beim Flächeninhalt, sowie bei der Kosten- und Distanz-Metrik sinnvoll.

$$error(child1, child2) = error(child1) + error(child2)$$

Maximum

Der totale Fehler der beiden Kindkanten $child1$ und $child2$ einer Kante kann als Maximum der Fehler der Kindkanten berechnet werden. Dies ist beispielsweise bei Hausdorff sinnvoll.

$$error(child1, child2) = max(error(child1), error(child2))$$

4.3.3 Zufällig

Aus einer Menge von Shortcuts, die alle entpackbar wären, wird im nächsten Schritt zufällig einer dieser Shortcuts gewählt und entpackt.

4.4 Vorberechnung

In einem Vorberechnungsschritt werden, nach Einlesen der gegebenen Eingabedateien in den Hauptspeicher, die optimalen Entpackreihenfolgen für alle Kanten für jeweils alle Kombinationen von Metrik und Modus vorberechnet und in einer Datei abgespeichert. Die Berechnung für einen bestimmten Shortcut ist in Abschnitt 4.4.1 näher erläutert. Es resultiert am Ende für jede Metrik-Modus-Kombination eine Datei, die für alle Kanten die optimalen Entpackreihenfolgen (für die gegebene Kombination) enthält. Für den Modus Zufällig existiert nur eine Datei, da diese unabhängig von der Fehlermetrik ist. Bei späteren Anfragen wird dann jeweils die richtige Datei in den Hauptspeicher eingelesen, falls sie nicht bereits eingelesen ist. Pro Datei befindet sich in Zeile *i* die optimale Entpackreihenfolge für den Shortcut mit ID *i*. Hierfür sind, durch Leerzeichen getrennt, die IDs des nächsten zu entpackenden Shortcuts aufgelistet. Für das Beispiel in Abbildung 2.6 und Listing 3.1 könnte eine Datei wie in Listing 4.1 dargestellt aussehen.

Listing 4.1 Beispieldatei einer optimalen Entpackreihenfolge

```
-       //Kante mit ID 0 ist eine Originalkante, kann also nicht entpackt werden
1       //Kante mit ID 1 ist ein Shortcut, der genau einmal entpackt werden kann
2 1 5   //erst wird Shortcut mit ID 2, dann mit ID 1, dann mit ID 5 entpackt
-       //Kante mit ID 3 ist eine Originalkante
-       //Kante mit ID 4 ist eine Originalkante
5       //Kante mit ID 5 ist ein Shortcut, der genau einmal entpackt werden kann
-       //Kante mit ID 6 ist eine Originalkante
```

4.4.1 Berechnung der optimalen Entpackreihenfolge für einen bestimmten Shortcut

Um die optimale Entpackreihenfolge für einen Shortcut, die in einer Liste *optimalUnpackOrder* gespeichert wird, zu bestimmen, wird dieser Shortcut initial einer Menge *shortcutsToBeUnpacked* hinzugefügt. Nun wird nach und nach immer ein Shortcut aus *shortcutsToBeUnpacked* entnommen, entpackt, seine ID *optimalUnpackOrder* hinzugefügt und seine Kindkanten *shortcutsToBeUnpacked* hinzugefügt. Dies wird so lange wiederholt, bis *shortcutsToBeUnpacked* leer ist, da nur noch Originalkanten verbleiben. Welcher Shortcut jeweils aus *shortcutsToBeUnpacked* entnommen wird, hängt von gegebener Fehlermetrik (siehe Abschnitt 4.2) und gegebenem Modus (siehe Abschnitt 4.3) ab. Der entsprechende Pseudo-Code ist in Algorithmus 4.4 zu finden.

Algorithmus 4.4 Berechnung einer optimalen Entpackreihenfolge für einen bestimmten Shortcut

```
procedure computeOptimalUnpackOrderSingleShortcut(relevantEdgeId, metric, mode)
    shortcutsToBeUnpacked.add(relevantEdgeId)
    while shortcutsToBeUnpacked is not empty do
        currentShortcut ← extractShortcut(shortcutsToBeUnpacked, metric, mode)
        child1 ← getChild1(currentShortcut)
        child2 ← getChild2(currentShortcut)
        if isShortcut(child1) then
            shortcutsToBeUnpacked.add(child1)
        end if
        if isShortcut(child2) then
            shortcutsToBeUnpacked.add(child2)
        end if
        optimalUnpackOrder.add(currentShortcut)
    end while
    return optimalUnpackOrder
end procedure
```

Laufzeit: Die Anzahl der Kanten auf dem zu einem Shortcut gehörenden Originalpfad ist, wie in Abschnitt 4.1 beschrieben, durch $O(E')$ begrenzt. Für einen gegebenen Shortcut ist daher die Anzahl der Kanten, die aus *shortcutsToBeUnpacked* entnommen werden können, bis diese Menge leer ist, ebenfalls durch E' beschränkt. Für jeden in *shortcutsToBeUnpacked* enthaltenen Shortcut kann dessen Fehler in $O(1)$ abgefragt werden, da die vorberechneten Fehler in einem Array abgespeichert sind. Die Identifikation der zu entnehmenden Kante läuft daher auch in $O(E')$. Die Gesamtlaufzeit zur Berechnung der optimalen Entpackreihenfolge beläuft sich daher für einen Shortcut auf $O(E'^2)$ und für alle Shortcuts auf $O(E \cdot E'^2)$.

5 Entpacken von Shortcuts

Nachdem für alle Shortcuts die optimalen Entpackreihenfolgen für jede Metrik-Modus-Kombination vorberechnet wurden, können diese genutzt werden, um Shortcuts für eine gegebene Anzahl von Schritten zu entpacken. Anhand der Parameter Metrik und Modus wird zunächst die vorberechnete Datei, welche die optimalen Entpackreihenfolgen für diese Metrik-Modus-Kombination enthält, in den Hauptspeicher eingelesen. Es muss außerdem spezifiziert werden, ob ein einzelner Shortcut entpackt werden soll oder ob alle Shortcuts eines gegebenen Zoomlevels entpackt werden sollen. Für einen einzelnen Shortcut ist das Vorgehen in Abschnitt 5.1 beschrieben, für alle Shortcuts eines gegebenen Zoomlevels in Abschnitt 5.2. Des Weiteren befinden sich in Abschnitt 5.3 Optimierungen, die den Entpackprozess betreffen.

5.1 Einzelner Shortcut

Ein einzelner Shortcut wird gemäß der zuvor, abhängig von der Metrik-Modus-Kombination, berechneten optimalen Entpackreihenfolge für die gegebene Anzahl von Schritten entpackt. Falls die gegebene Anzahl an Schritten größer ist als die maximale Anzahl von Entpackschritten für diesen Shortcut, dann wird dieser Shortcut vollständig entpackt. Es resultiert eine Menge von zu visualisierenden Kanten, in der sich sowohl Originalkanten als auch Shortcuts befinden können.

Realisiert werden kann dies beispielsweise über eine Menge *edgesToBeVisualisized*, die initial mit dem zu entpackenden Shortcut initialisiert wird. Nun wird nach und nach der laut der optimalen Entpackreihenfolge als Nächstes zu entpackende Shortcut in dieser Menge gesucht, entpackt und seine beiden Kindkanten wieder dieser Menge hinzugefügt. Das wird für die gegebene Anzahl von Entpackschritten wiederholt. Der entsprechende Java-Code befindet sich in Listing 5.1.

Listing 5.1 Java-Code für das Entpacken eines einzelnen Shortcuts

```
//initialize the list of edges that should be visualized by the given shortcut to start the
unpacking process
edgesToBeVisualized.add(optimalUnpackOrderSingleShortcut[0]);
//unpacks the shortcut unpackingSteps times
//(or until the original path is reached, if the number of unpacking steps is greater than
times the given shortcut can be unpacked)
for (int i = 0; i < Math.min(unpackingSteps, optimalUnpackOrderSingleShortcut.length); i++) {
    //get the next shortcut that should be unpacked according to the optimal unpack order
    int nextShortcutToUnpack = optimalUnpackOrderSingleShortcut[i];
    //remove this shortcut from the edges that should be visualized and replace it with its
two children:
    //get the index of this shortcut in the list of edges that should be visualized
    int index = edgesToBeVisualized.indexOf(nextShortcutToUnpack);
    //remove this shortcut
    edgesToBeVisualized.remove(index);
    //get the first and second child of this shortcut and add it to the list of edges that
should be visualized at the position of the removed shortcut
    edgesToBeVisualized.add(index, edges.getSingleChild1(nextShortcutToUnpack));
    edgesToBeVisualized.add(index + 1, edges.getSingleChild2(nextShortcutToUnpack));
}
```

Laufzeit: Die gegebene Anzahl von Entpackschritten ist durch die Anzahl an Kanten auf dem zum Shortcut gehörenden Originalpfad, also E', begrenzt. Die Anzahl der in jeder Iteration in *edgesToBeVisualisized* befindlichen Kanten ist ebenfalls durch E' begrenzt. Die Gesamtlaufzeit der Entpackung eines Shortcuts für eine gegebene Anzahl von Entpackschritten ist daher in $O(E'^2)$.

5.2 Alle Shortcuts

Um alle Shortcuts für ein gegebenes Zoomlevel zu entpacken, werden zunächst diejenigen Shortcuts identifiziert, die auf dem gegebenen Zoomlevel relevant sind. Anschließend wird zuvor beschriebenes Vorgehen für alle diese identifizierten Shortcuts wiederholt. Die Identifikation der auf dem gegebenen Zoomlevel relevanten Kanten ist abhängig davon, ob mit den Knotenlevels aus der SCH-Datei oder mit den Levelintervallen aus der RANGES-Datei gearbeitet wird. Der Unterschied wird in Abschnitt 5.2.1 näher erläutert.

Laufzeit: Wie in Abschnitt 5.2.1 genauer beschrieben wird, können alle auf einem gegebenen Zoomlevel relevanten Kanten in $O(E)$ berechnet werden. Wie in Abschnitt 5.1 beschrieben, läuft die Entpackung eines Shortcuts für eine gegebene Anzahl von Entpackschritten in $O(E'^2)$. Die Gesamtlaufzeit der Entpackung aller auf einem gegebenen Zoomlevel relevanten Shortcuts für eine gegebene Anzahl von Entpackschritten ist daher in $O(E \cdot E'^2)$.

5.2.1 Identifikation der auf einem Zoomlevel relevanten Kanten

Die Menge der auf einem Zoomlevel relevanten Kanten wird abhängig von der gewählten Eingabedatei identifiziert.

SCH Datei

Wie in Abschnitt 3.1 beschrieben, besitzt jeder Knoten ein Level. Eine Originalkante ist genau dann auf einem gegebenen Zoomlevel relevant, wenn die Levels der durch diese Kante verbundenen Knoten größer oder gleich als das Zoomlevel sind. Für einen Shortcut muss zusätzlich gelten, dass das Level des überbrückten Knotens kleiner als das Zoomlevel ist [FSS17]. Damit der von einem Shortcut überbrückte Knoten nicht mehrfach berechnet werden muss, wird dem *Edges*-Objekt ein Array *bridgedNode* hinzugefügt, welches die ID des von Kante *i* überbrückten Knotens nach erstmaligem Berechnen an Index *i* abspeichert. Den von dem Shortcut überbrückten Knoten kann man berechnen, indem man überprüft, in welchem Knoten die Kindkanten dieses Shortcuts übereinstimmen.

In Abbildung 5.1 ist ein Shortcut mit zugehörigem Originalpfad dargestellt, der dieses Prinzip verdeutlichen soll.

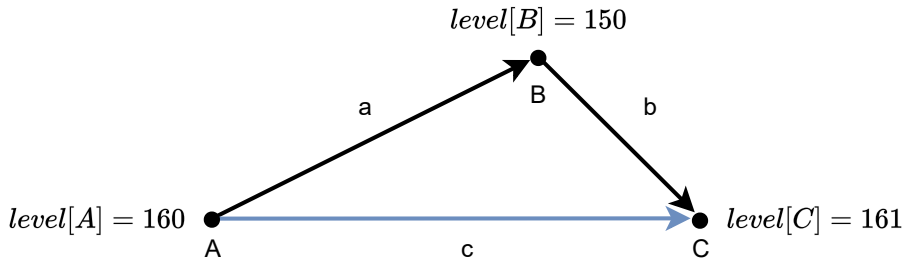


Abbildung 5.1: Berechnung des benötigten Zoomlevels zur Visualisierung eines Shortcuts mit gegebenen Knotenlevels aus einer SCH-Datei

Gegeben seien die Originalkanten a, b und der Shortcut c. Seien A, B, C die Knoten auf dem zu c gehörenden Originalpfad. Sei $level[A] = 160$, $level[B] = 150$ und $level[C] = 161$. Damit Shortcut c visualisiert wird, muss also $level[A] = 160 \geq zoomlevel$, $level[C] = 161 \geq zoomlevel$ und $level[B] = 150 < zoomlevel$ gelten. Shortcut c würde somit für alle Zoomlevel von 151 bis 160 visualisiert werden.

RANGES Datei

Wie in Abschnitt 3.2 beschrieben, besitzt jede Kante ein Start- und ein Endlevel. Eine Kante ist genau dann auf einem gegebenen Zoomlevel relevant, wenn sich das Zoomlevel zwischen dem Start- und Endlevel (jeweils inklusive) befindet.

Laufzeiten: Der Zugriff auf das Level eines Knotens beziehungsweise auf das Start- und Endlevel einer Kante ist durch die Nutzung von Arrays in $O(1)$ möglich. Der durch einen Shortcut überbrückte Knoten kann daher ebenfalls in $O(1)$ berechnet werden. Für eine Kante kann demnach in $O(1)$ berechnet werden, ob sie auf gegebenem Zoomlevel relevant ist. Die Gesamtlaufzeit zur Identifikation aller auf gegebenem Zoomlevel relevanten Kanten ist daher in $O(E)$.

5.3 Optimierungen

Um dieselben Berechnungen nicht mehrmals durchzuführen, den visuellen Eindruck zu verbessern und die Laufzeit des Entpackprozesses zu verbessern, können unterschiedliche Optimierungen vorgenommen werden. Diese Optimierungen werden im Folgenden beschrieben.

5.3.1 Unpacked Array

In der SCH-Datei existieren Kanten, die Kindkante von mehr als einer anderen Kante sind. Der entsprechende Pseudo-Code, um herauszufinden, für wie viele Kanten dies der Fall ist, ist in Algorithmus 5.1 zu finden. Optimierend wird die Schleife über die Suche nach Elternkanten abgebrochen, sobald mehr als eine Elternkante gefunden wurde.

Algorithmus 5.1 Berechnung der Anzahl der Kanten, welche mehrere Elternkanten besitzen

```
procedure countNumberOfSharedEdges
    originalWithMoreParents ← 0
    shortcutsWithMoreParents ← 0
    for edge in edges do                                          // Iterate over all edges
        counter ← 0
        for possibleParent in edges do                            // Iterate over all edges
            if getChild1(possibleParent) = edge or getChild2(possibleParent) = edge then
                counter ← counter + 1
                if counter > 1 then
                    if isShortcut(edge) then
                        shortcutsWithMoreParents ← shortcutsWithMoreParents + 1
                    else
                        originalWithMoreParents ← originalWithMoreParents + 1
                    end if
                    break                  // Optimization: Continue with the next edge in edges
                end if
            end if
        end for
    end for
    return (shortcutsWithMoreParents, originalEdgesWithMoreParents)
end procedure
```

Laufzeit: Im Worst Case muss für jede Kante über alle anderen Kanten iteriert werden. Die asymptotische Laufzeit dieses Algorithmus beträgt demnach $O(E^2)$.

Für eine getestete SCH-Datei, die Graphdaten des Regierungsbezirks Stuttgart enthält, sind 25.349 der insgesamt 2.292.297 Originalkanten und 75.394 der insgesamt 1.847.179 Shortcuts Kindkante von mehr als einer anderen Kante. Für alle anderen getesteten SCH-Dateien ist dies ebenfalls der Fall, die genauen Anzahlen wurden jedoch aufgrund der quadratischen Laufzeit des Programms nicht berechnet.

Wenn also die Relevanz von Kanten auf einem bestimmten Zoomlevel über die Level in der SCH-Datei bestimmt werden, kann es beim Entpacken von Shortcuts dazu kommen, dass ein gewisser Shortcut schon zuvor aufgrund einer anderen Grad-2-Kette entpackt wurde und jetzt nochmals entpackt werden müsste. Da die Kindkanten jedoch schon existieren, muss der Shortcut nicht nochmals entpackt werden. Um diesen Fall abzufangen, kann ein Array *unpacked* verwendet werden, welches für jeden möglichen Shortcut festhält, ob dieser bereits entpackt wurde. Falls der Shortcut mit ID i bereits entpackt wurde, ist der Eintrag an Index i im Array auf *true* gesetzt, ansonsten auf *false*. Shortcuts, die im Array den Wert *true* besitzen, müssen im Entpackprozess nicht nochmal entpackt werden, was unnötige Berechnungen erspart. Zusätzlich müssen diese aus der finalen Menge der zu visualisierenden Kanten entfernt werden. Durch diese Entfernung kann vermieden werden, dass ein Teilpfad in unterschiedlichen Detaillierungsstufen visualisiert wird.

Der gerade beschriebene Fall wird in Abbildung 5.2 verdeutlicht.

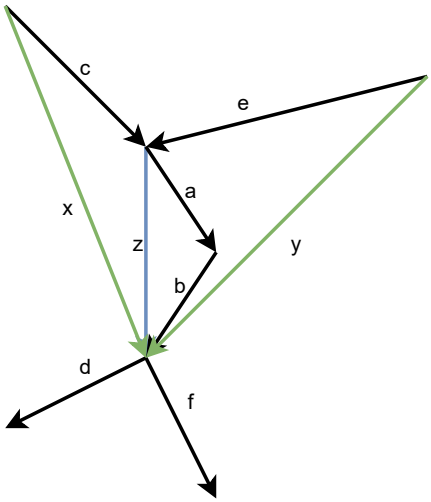


Abbildung 5.2: Darstellung zweier Grad-2-Ketten, welche sich beim Aufbau der Kontraktionshierarchie einen Shortcut teilen

Angenommen, es existiert eine Grad-2-Kette bestehend aus den Kanten c, a, b, d und eine andere Grad-2-Kette bestehend aus Kanten e, a, b, f, wobei in beiden Grad-2-Ketten die Originalkanten a und b durch denselben Shortcut z approximiert werden. Außerdem werden c und z durch x und e und z durch y approximiert. Wenn jetzt beim Entpackprozess für die erste Grad-2-Kette z zu a und b entpackt wird, muss dies für die zweite Grad-2-Kette nicht nochmals geschehen, z kann aus dieser Grad-2-Kette entfernt werden, da Kindkanten a und b schon aufgrund der anderen Grad-2-Kette existieren. Genauer gesagt sollte z sogar entfernt werden, da sonst dieser Teilpfad in unterschiedlichen Detaillierungsstufen visualisiert wird.

Laufzeit: Da Zugriffe auf ein Array in $O(1)$ erfolgen, wird die asymptotische Laufzeit nicht verschlechtert.

5.3.2 Visualisieren von Shortcuts nur in ihrer detailliertesten existierenden Version

Trotz zuvor beschriebener Optimierung kann es immer noch dazu kommen, dass ein bestimmter Teilpfad in unterschiedlichen Detaillierungsstufen visualisiert wird. Dies kann passieren, wenn ein zu visualisierender Shortcut nie explizit entpackt wurde, aber seine beiden Kindkanten durch das Entpacken anderer Shortcuts entstanden sind. Damit solche Shortcuts nur in ihrer detailliertesten existierenden Version visualisiert werden, kann in einer Schleife, unter Zuhilfenahme des *unpacked*-Arrays, über alle noch nicht entpackten Shortcuts iteriert werden. In jeder Iteration werden dann diejenigen Shortcuts als entpackt markiert, deren beide Kindkanten bereits als entpackt markiert wurden. Dies wird so lange wiederholt, bis keine neuen Markierungen mehr hinzukommen. Anschließend kann dann über alle zu visualisierenden Kanten iteriert werden und diejenigen Shortcuts entfernt werden, deren beide Kindkanten als entpackt markiert sind.

Das gerade beschriebene Problem wird in Abbildung 5.3 verdeutlicht.

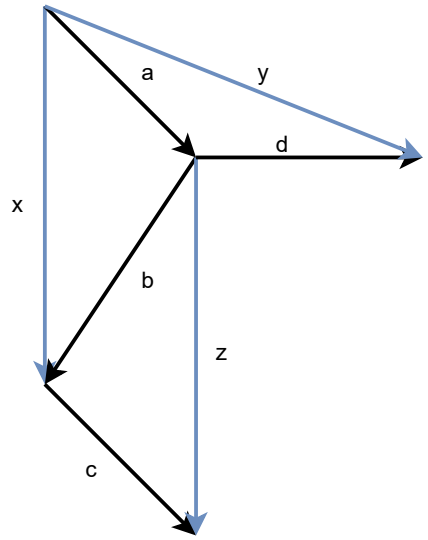


Abbildung 5.3: Darstellung der Problematik, dass eine Kante mehrere Elternkanten besitzt

Angenommen zu Beginn sollen Shortcuts x, y und z visualisiert werden. Nun wird im nächsten Schritt zuerst y zu a und d entpackt und anschließend z zu b und c entpackt. Es würden also x, a, d, b und c visualisiert werden. Von x bestehen jetzt aber unterschiedliche Detaillierungsstufen. Obiges Vorgehen sorgt dafür, dass nicht mehr Shortcut x, sondern nur noch dessen Kindkanten a und b visualisiert werden.

Laufzeit: Die Anzahl der Iterationen, um die Markierungen in dem *unpacked*-Array nach oben zu propagieren, ist durch die Anzahl von Kanten auf dem längsten Originalpfad, also E', begrenzt. Dies ist der Fall, da dieser Wert die maximale Tiefe des Entpackprozesses eines Shortcuts darstellt und in jeder Iteration die *unpacked*-Werte von Kanten einer gewissen Tiefe berechnet werden können. Die Iteration über alle zu visualisierenden Kanten und das Entfernen der Shortcuts geschieht dann in $O(E)$. Die Gesamtlaufzeit der Optimierung, dass Shortcuts nur in ihrer detailliertesten existierenden Version visualisiert werden, ist daher in $O(E \cdot E')$.

5.3.3 Verbesserung der Laufzeit beim Entpacken

Die quadratische Laufzeit des Entpackens in Abschnitt 5.1 kann optimiert werden. Dazu wird eine Priority-Queue *edgesToBeUnpacked* mit dem zu entpackenden Shortcut initialisiert. Eine andere Priority-Queue *firstElementsOfOptimalUnpackOrder* wird mit den ersten |Entpackschritten| Elementen der optimalen Entpackreihenfolge für diesen Shortcut initialisiert. Die Priorität basiert hierbei auf der kleinsten Kanten-ID. Nun werden iterativ jeweils die beiden ersten IDs der Priority-Queues verglichen. Sind diese gleich, so werden beide entfernt, der zugehörige Shortcut entpackt und dessen Kindkanten *edgesToBeUnpacked* hinzugefügt. Sind diese unterschiedlich, so wird die ID nur aus *edgesToBeUnpacked* entfernt und einer Liste *edgesToBeVisualized* hinzugefügt. Dieses Vorgehen wird so lange wiederholt, bis *firstElementsOfOptimalUnpackOrder* leer ist. Alle dann noch in *edgesToBeUnpacked* verbleibenden IDs werden ebenfalls *edgesToBeVisualized* hinzugefügt. *edgesToBeVisualized* enthält jetzt die IDs aller final zu visualisierenden Kanten. Der entsprechende Java-Code ist in Listing 5.2 zu finden.

Diese Optimierung ist möglich, da die Kanten-IDs im Entpackprozess monoton steigend sind. Das bedeutet, dass die IDs der Kindkanten eines Shortcuts immer beide größer sind als die ID des Shortcuts selbst.

Mit dieser Optimierung ist jedoch die in Abschnitt 5.3.1 erwähnte Optimierung und damit auch die in Abschnitt 5.3.2 erwähnte Optimierung nicht mehr problemlos möglich, da Kanten jetzt nicht mehr genau in der Reihenfolge entpackt werden, wie sie in der optimalen Entpackreihenfolge aufgelistet sind. Die erwähnten Optimierungen sind jedoch sowieso nur relevant, wenn mit den unoptimierten Levels aus der SCH-Datei gearbeitet wurde.

Laufzeit: Die gegebene Anzahl von Entpackschritten sowie die Anzahl der Elemente in den beiden Priority-Queues sind durch die Anzahl an Kanten auf dem zum Shortcut gehörenden Originalpfad, also E', begrenzt. Das Einfügen und Entnehmen von Elementen aus der Java *PriorityQueue* hat daher eine Laufzeit von $O(\log(E'))$. Das initiale Befüllen von *firstElementsOfOptimalUnpackOrder*, sowie das Entpacken des Shortcuts, wie auch das schlussendliche Leeren von *edgesToBeUnpacked* läuft demnach in $O(E' \cdot \log(E'))$. Die Gesamtlaufzeit des optimierten Entpackens eines Shortcuts beläuft sich daher ebenfalls auf $O(E' \cdot \log(E'))$.

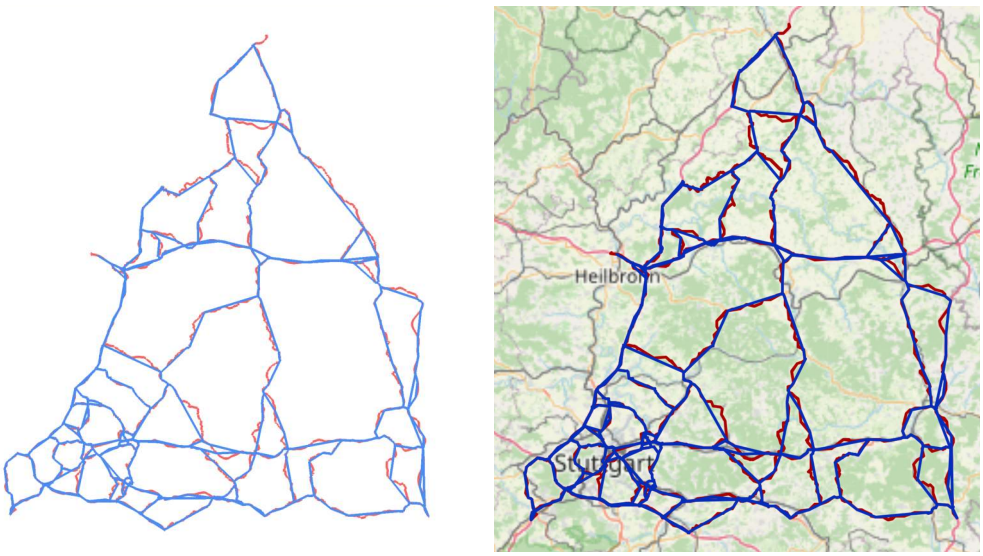
Listing 5.2 Optimierter Java-Code für das Entpacken eines einzelnen Shortcuts

```
//list that contains the edges that should be visualized
ArrayList<Integer> edgesToBeVisualized = new ArrayList<>();
//initialize the priority-queue which later contains the edges that are either going to be
unpacked (i.e. remove this edge and add its children again to this queue) or
//that will be finally visualized (and therefore removed from the queue and added to the
edgesToBeVisualized-list)
PriorityQueue<Integer> edgesToBeUnpacked = new PriorityQueue<>();
//initialize the priority queue by the given shortcut to start the unpacking process
edgesToBeUnpacked.add(optimalUnpackOrderSingleShortcut[0]);

//unpack the shortcut unpackingSteps times
//(or until the original path is reached, if the number of unpacking steps is greater than
times the given shortcut can be unpacked):

//therefore copy the first unpackingSteps elements (or all, if unpackingSteps is too big) from
 the optimal unpack order into a priority-queue firstElementsOfOptimalUnpackOrder
PriorityQueue<Integer> firstElementsOfOptimalUnpackOrder = new PriorityQueue<>();
for (int i = 0; i < Math.min(unpackingSteps, optimalUnpackOrderSingleShortcut.length); i++) {
    firstElementsOfOptimalUnpackOrder.add(optimalUnpackOrderSingleShortcut[i]);
}

//unpack the shortcut
while (!firstElementsOfOptimalUnpackOrder.isEmpty()) {
    //compare the first element in edgesToBeUnpacked with the first element in
firstElementsOfOptimalUnpackOrder
    int edgeToBeUnpacked = edgesToBeUnpacked.poll();
    //if the first elements are equal, then this shortcut gets unpacked and its children get
added to edgesToBeUnpacked
    if (edgeToBeUnpacked == firstElementsOfOptimalUnpackOrder.peek()) {
        firstElementsOfOptimalUnpackOrder.poll();
        edgesToBeUnpacked.add(edges.getSingleChild1(edgeToBeUnpacked));
        edgesToBeUnpacked.add(edges.getSingleChild2(edgeToBeUnpacked));
    } else {
        //if they are unequal, then the edge get removed and added to the edgesToBeVisualized-
list
        edgesToBeVisualized.add(edgeToBeUnpacked);
    }
}
//if the shortcut got unpacked for the defined number, the remaining edges in
edgesToBeUnpacked get also added to the edgesToBeVisualized-list
while (!edgesToBeUnpacked.isEmpty()) {
    edgesToBeVisualized.add(edgesToBeUnpacked.poll());
}
```

6 Visualisierung der entpackten Shortcuts

Um die Approximationsgüte verschiedener optimaler Entpackreihenfolgen zu vergleichen, ist es notwendig, die aus dem Entpackprozess resultierenden Kanten zu visualisieren. Im Rahmen dieser Arbeit ist dies zum einen durch den in Abschnitt 2.5.2 beschriebenen OpenGL-Viewer und zum anderen über eine API, welche visualisierbare GeoJSON-Objekte (siehe Abschnitt 2.5.3) zurückgibt, möglich. Diese beiden Möglichkeiten werden in Abschnitte 6.1 und 6.2 beschrieben. Zur Beurteilung der Approximationsgüte sollten außerdem nicht nur die approximierten Pfade, sondern auch die dazugehörigen Orginalpfade visualisiert werden. Wie man den zu einem Shortcut gehörenden Originalpfad erhält, wurde bereits in Abschnitt 4.1 beschrieben.

Entsprechende Darstellungen zu den Möglichkeiten zur Visualisierung von entpackten Shortcuts sind in Abbildung 6.1 zu finden.



(a) OpenGL-Viewer **(b)** Frontend

Abbildung 6.1: Möglichkeiten zur Visualisierung von entpackten Shortcuts

6.1 Generierung einer Ausgabedatei

Um die aus dem Entpackprozess resultierenden Kanten und deren Originalpfade mittels des OpenGL-Viewers zu visualisieren, muss diese Menge von zu visualisierenden Kanten in das in Abschnitt 2.5.2 beschriebene GL-Format überführt werden. Dies funktioniert, indem zunächst für jede dieser Kanten deren Start- und Zielknoten mit in die Datei aufgenommen werden. Man merkt sich hierbei, welche IDs diese Knoten laut der Datei besitzen. Anschließend kann dann bei den Kanteninformationen in der Datei auf diese gemerkten IDs verwiesen werden. Das sorgt jedoch möglicherweise dafür, dass dieselben Knoten mehrfach in der Datei enthalten sind.

Den aus dem Entpackprozess resultierenden Kanten kann in der Datei außerdem eine andere Farbe zugewiesen werden als den zugehörigen Originalpfaden.

Um die Ausgabedatei so klein wie möglich zu halten, kann zusätzlich noch die im Folgenden beschriebene Optimierung durchgeführt werden.

6.1.1 Vermeidung von Knoten-Duplikaten in der Ausgabedatei

Damit die resultierende GL-Datei so wenig redundante Information wie möglich enthält, insbesondere also keine Knoten mehrfach vorkommen, kann man bei der Generierung der Ausgabedatei eine Abbildung zur Hilfe nehmen. Diese Abbildung speichert für jeden Knoten, der bereits in die Datei aufgenommen wurde, welche ID dieser Knoten laut der Datei besitzt. Um dann eine Menge von Kanten in das in Abschnitt 2.5.2 beschriebene GL-Format zu überführen, wird zunächst für jede Kante geprüft, ob deren Startknoten bereits in die Datei aufgenommen wurden. Falls dieser Knoten noch nicht der Datei hinzugefügt wurde, wird der Knoten jetzt der Datei hinzugefügt und in der Abbildung wird gespeichert, welche ID dieser Knoten laut der Datei besitzt. Selbiges wiederholt man dann für den Zielknoten jeder Kante. Anschließend kann dann bei den Kanteninformationen in der Datei die Abbildung genutzt werden, um für eine Kante auf die IDs ihrer verbundenen Knoten zu verweisen. Diese Methode verringert somit potenziell die Größe der resultierenden GL-Datei, besitzt jedoch eine schlechtere Laufzeit durch die Nutzung der zusätzlichen Abbildung.

6.2 Application Programming Interface

Um auch Anfragen für entpackte Shortcuts in Echtzeit machen zu können, wird eine API implementiert. Diese API ist eine Schnittstelle zum Backend und gibt auf eine Anfrage ein visualisierbares GeoJSON-Objekt zurück. Im Folgenden wird daher das dazu benötigte Server und Query Handling beschrieben. Der Server bietet zusätzlich ein eigenes Frontend an, welches diese Schnittstelle nutzt. Dieses Frontend wird ebenfalls im Folgenden beschrieben.

6.2.1 Server und Query Handling

Als Schnittstelle zum Backend wird ein Javalin-Server implementiert. Dieser Server beantwortet Hypertext Transfer Protocol (HTTP) GET Requests durch zwei verschiedene Handler. Der Status Handler benötigt keine Parameter und gibt dem Aufrufer zurück, ob der Server in Betrieb ist, indem er einen Status Code 200 mit der JSON-Nachricht ”Server is up and running” zurückgibt. Der Query Handler benötigt Anfrageparameter und gibt die aus diesen Parametern berechnete, zu visualisierende Menge an Kanten als GeoJSON-Objekt zurück.

Parameter für Queries

Die API des Servers kann genutzt werden, indem man diese mittels eines HTTP GET Requests anfragt. In Tabelle 6.1 sind die zu spezifizierenden Parameter für die Anfrage aufgelistet. Wird ein Parameter nicht explizit spezifiziert, so wird ein Standardwert für den jeweiligen Parameter für die Anfrage verwendet. Eine Anfrage könnte wie in Listing 6.1 dargestellt aussehen. Dabei ist zu beachten, dass die Reihenfolge der Parameter wie folgt ist:

Metric, Zoomlevel, File, Mode, Shortcut ID, Unpacking Steps, Original Edges

Listing 6.1 Beispielhafter HTTP GET Request für den Server

```
http://<ip>:8080/query/0/150/false/0/-1/5/true
```

Parameter	Standard	Beschreibung
Metric	0	0: Hausdorff (siehe Abschnitt 4.2.2), 1: Fréchet (siehe Abschnitt 4.2.3), 2: Flächeninhalt (siehe Abschnitt 4.2.4), 3: Kosten (siehe Abschnitt 4.2.1), 4: Distanz (siehe Abschnitt 4.2.5)
		unerlaubt: *Metric* < 0 und *Metric* > 4
ZoomLevel	-1	Zoomlevel der zu entpackenden Shortcuts oder, *maxZoomLevel*, wenn *ZoomLevel*$= -1$ und *Shortcut ID* $= -1$, wobei *maxZoomLevel* maximales *level* eines Knotens, wenn *File* = *SCH*, maximales *levelStart* einer Kante, wenn *File* = *RANGES*
		unerlaubt: *ZoomLevel* > *maxZoomLevel* und unerlaubt: *ZoomLevel* < *minZoomLevel*, wobei *minZoomLevel* minimales *level* eines Knotens, wenn *File* = *SCH*, minimales *levelEnd* einer Kante, wenn *File* = *RANGES*
Mode	0	0: Größter Fehler (siehe Abschnitt 4.3.1), 1: Größte Fehlerreduktion, Summe (siehe Abschnitt 4.3.2), 2: Größte Fehlerreduktion, Maximum (Abschnitt 4.3.2), 3: Kleinster Fehler (siehe Abschnitt 4.3.1), 4: Kleinste Fehlerreduktion, Summe (siehe Abschnitt 4.3.2), 5: Kleinste Fehlerreduktion, Maximum (siehe Abschnitt 4.3.2), 6: Zufällig (siehe Abschnitt 4.3.3)
		unerlaubt: *Mode* < 0 und *Mode* > 6
Unpacking Steps	20	wie oft die zu entpackenden Shortcuts entpackt werden sollen
Shortcut ID	-1	welcher Shortcut entpackt werden soll, -1, wenn alle auf gegebenem Zoomlevel relevanten Shortcuts entpackt werden sollen
		unerlaubt: *Shortcut ID* < 0 und *Shortcut ID* $> (|\text{Kanten}| - 1)$
File	*true*	gemäß welcher Eingabedatei die auf dem gegebenen Zoomlevel relevanten Shortcuts identifiziert werden sollen (siehe Abschnitt 5.2.1). *SCH* $\widehat{=}$ *true*, *RANGES* $\widehat{=}$ *false*
Original Edges	*true*	ob die Originalkanten, die von den zu entpackenden Shortcuts approximiert werden, ebenfalls visualisiert werden sollen (siehe Abschnitt 4.1)

Tabelle 6.1: Beschreibung der Parameter für Serveranfragen und deren Standardwerte

Query Handling

Jede Anfrage durchläuft fünf Schritte, bevor diese von dem Server beantwortet wird: Parsing, Entscheidung, was zu entpacken ist, Einlesen der richtigen Datei, Erstellung und Rückgabe des GeoJSON-Objektes.

Parsing
Zuerst werden alle Parameter aus der Anfrage geparst. Wenn ein Parameter einen unerlaubten Wert enthält oder kein Wert spezifiziert wurde, wird er durch den in Tabelle 6.1 definierten Standardwert ersetzt.

Entscheidung, was zu entpacken ist
Nun wird anhand des Parameters *Shortcut ID* unterschieden, ob ein einzelner Shortcut oder alle Shortcuts auf einem gegebenen Zoomlevel visualisiert werden sollen, abhängig davon, ob bei der Anfrage eine *Shortcut ID* angegeben wird.

Einlesen der richtigen Datei
Als Nächstes wird überprüft, ob sich diejenige Datei, welche die vorberechneten optimalen Entpackreihenfolgen für die angefragte Metrik-Modus-Kombination enthält, aktuell im Hauptspeicher befindet. Falls das nicht der Fall ist, wird diese jetzt in den Hauptspeicher eingelesen.

Erstellung des GeoJSON-Objektes
Zuletzt werden die zu visualisierenden Kanten gemäß den spezifizierten Parametern berechnet und als GeoJSON-Objekt zurückgegeben. Dazu werden die, wie in Abschnitte 5.1 und 5.2, identifizierten Kanten einem GeoJSON-Objekt hinzugefügt. Hierfür wird die JSON Bibliothek verwendet, um ein *JSONObject* zu erstellen. Dieses kann zur Visualisierung eines einzelnen Shortcuts und dessen Originalpfads beziehungsweise zur Visualisierung aller Shortcuts und deren Originalpfaden verwendet werden und ist wie in Listing 6.2 dargestellt aufgebaut.

Hierbei gibt *properties* an, ob sich im zugehörigen *geometry*-Objekt Informationen über entpackte Shortcuts (Wert 0) oder Originalpfade (Wert 1) befinden. So kann später unterschieden werden, welche Kanten zu approximierten Pfaden und welche Kanten zu Originalpfaden gehören. Das ist beispielsweise notwendig, wenn diese in verschiedenen Farben visualisiert werden sollen.

Das *coordinates*-Array für Kanten des Originalpfades enthält ein *JSONArray* für jeden Originalpfad, der von einem Top-Level-Shortcut approximiert wird. Ein Top-Level-Shortcut ist hierbei ein Shortcut, der auf dem angegebenen Zoomlevel relevant ist. In jedem dieser *JSONArray*s befindet sich für jeden Knoten auf diesem Originalpfad ein weiteres Array, worin sich dessen *longitude* und *latitude* befinden.

Das *coordinates*-Array für Kanten von approximierten Pfaden enthält für jede zu visualisierende Kante, die aus dem Entpackprozess resultiert, genau ein *JSONArray*. Jedes dieser *JSONArray*s besteht wiederum aus genau zwei Arrays, wovon das erste die *longitude* und *latitude* des Startknotens und das zweite die *longitude* und *latitude* des Zielknotens dieser Kante enthalten.

Zu beachten ist hier, dass das GeoJSON-Objekt nur zu Originalpfaden gehörende Kanten enthält, wenn dies bei der Anfrage gewünscht war. Waren keine zugehörigen Originalpfade gewünscht, so ist im GeoJSON-Objekt nur ein *Feature* enthalten.

Listing 6.2 Aufbau des GeoJSON-Objektes, welches der Backend-Server als Antwort auf eine vorausgegangene Anfrage zurücksendet

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [longitude1, latitude1], [longitude2, latitude2],...
          ],
          [
            [longitude1, latitude1], [longitude2, latitude2],...
          ],...
        ]
      },
      "properties": {
        "shortcutOrOriginalEdges": "1"
      }
    },
    { "type": "Feature",
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [longitude1, latitude1], [longitude2, latitude2]
          ],
          [
            [longitude1, latitude1], [longitude2, latitude2]
          ],...
        ]
      },
      "properties": {
        "shortcutOrOriginalEdges": "0"
      }
    }
  ]
}
```

Rückgabe des GeoJSON-Objektes
Nachdem die *FeatureCollection* vollständig gefüllt ist, kann sie mit einem *JSONParser* in einen *JSONString* geparst werden, der dann als Antwort auf die vorige Anfrage zurückgegeben wird.

6.2.2 Frontend

Der gerade beschriebene Server stellt ebenfalls ein eigenes Frontend zur Verfügung, das die eben definierte Schnittstelle nutzt. Die Kartendarstellung und die Funktionalität des Frontends werden im Folgenden beschrieben.

Kartendarstellung

Für die Kartendarstellung wird die Open-Source JavaScript-Bibliothek Leaflet verwendet, die bereits genauer in Abschnitt 2.5.1 beschrieben wurde. Es wird eine Kartenkomponente definiert, die den ganzen Bildschirm einnimmt. Auf diese Karte wird ein Tile-Layer gesetzt, worauf wiederum eine GeoJSON-Komponente gesetzt wird.

Die Kartenkomponente zeigt initial den Regierungsbezirk Stuttgart (49.1°, 9.75°) bei einem Zoomlevel von 9. Diese Ansicht kann geändert werden, indem man den Linksklick gedrückt hält und die Maus bewegt. Das Zoomlevel kann mithilfe des Mausrades geändert werden.

Der Tile-Layer ist an einen Tile-Server des Instituts für Formale Methoden der Informatik der Universität Stuttgart (https://tiles.fmi.uni-stuttgart.de/z/x/y.png) gebunden, der die entsprechenden Kacheln für das aktuelle Ansichtsfenster, das von der Kartenkomponente definiert wird, liefert. Es kann jedoch theoretisch jeder gängige Tile-Server eingesetzt werden.

Um eine Menge von Kanten zu visualisieren, wird eine GeoJSON-Komponente verwendet, die verschiedene GeoJSON-Objekte visualisieren kann.

In der rechten oberen Ecke wird außerdem angezeigt, ob der Server gerade in Betrieb ist.

Eine entsprechende Darstellung des Frontends befindet sich in Abbildung 6.2.

Abbildung 6.2: Darstellung des Frontends

Funktionalität

Über das von dem Server bereitgestellte Frontend ist es einem Nutzer möglich, Anfragen an das Backend durch die Nutzung von Eingabefeldern und Knöpfen zu stellen. Das Frontend visualisiert dann schließlich das in der Antwort des Servers enthaltene GeoJSON-Objekt. Des Weiteren wird dem Nutzer die Verfügbarkeit des Servers angezeigt, welche im Hintergrund mittels periodischer Heartbeat-Anfragen ermittelt wird. Um dies zu ermöglichen, nutzt das Frontend die API des Servers. Die Funktionalität des Frontends ist im Folgenden genauer beschrieben.

Einstellungen
Metric, *Zoomlevel*, *Mode*, *Unpacking Steps* und *Shortcut ID* sind als Textfelder realisiert, in die der Nutzer seine gewünschten Werte eintragen kann. *Original Edges* ist als Knopf realisiert, der von *aktiviert (grün)* zu *deaktiviert (rot)* wechselt. Wenn dieser Knopf gedrückt wird, ändern sich dessen Farbe und Kennzeichnung. *File* ist ebenfalls als Knopf realisiert, hier ändert sich durch Drücken jedoch lediglich die Kennzeichnung und nicht die Farbe.

Erstellung einer Anfrage und Verarbeitung der Antwort
Sobald der SEND-Knopf gedrückt wird, werden die von dem Nutzer spezifizierten Werte der Eingabefelder genommen und ein HTTP GET Request erstellt. Wurde ein Wert nicht explizit gesetzt, dann wird für diesen Parameter der in Tabelle 6.1 angegebene Standardwert bei der Anfrage verwendet. Diese Anfrage wird an das Backend geschickt und die ankommende Antwort wird der GeoJSON-Komponente zugewiesen. Die Leaflet Bibliothek dekodiert das GeoJSON-Objekt der Antwort und visualisiert dieses schließlich. Hierbei werden die approximierten Pfade in Blau dargestellt. Falls bei der Anfrage Originalpfade gewünscht waren, werden diese in Rot dargestellt. Wird eine weitere Anfrage getätigt, dann wird die vorherige GeoJSON-Komponente entfernt und das in der Antwort neu enthaltene GeoJSON-Objekt visualisiert.

Verfügbarkeitsprüfung des Servers
Der Serverstatus wird periodisch alle zehn Sekunden abgefragt (und der Status Text entsprechend angepasst), indem ein HTTP GET Request an das Backend geschickt wird. Ist der Status Code 200, so ist der Server in Betrieb und beantwortet HTTP GET Requests.

7 Komponentenkommunikation

Für den Zweck dieser Arbeit wurden verschiedene Komponenten entwickelt. In Abbildung 7.1 ist die Interaktion aller relevanten Komponenten als Gesamtbild dargestellt.

Beim Start des Programms werden zunächst in einem Vorverarbeitungsschritt mithilfe des File-Readers die SCH- und RANGES-Dateien des Eingabegraphen in den Hauptspeicher eingelesen. Außerdem wird geprüft, ob alle optimalen Entpackreihenfolgen für diesen Graphen bereits vorhanden sind. Ansonsten werden diese vorberechnet und in eine entsprechende Datei geschrieben. Je nachdem wie das Programm gestartet wurde, wird entweder ein Server gestartet oder eine GL-Ausgabedatei mithilfe des FileWriters für die beim Programmstart angegebenen Parameter geschrieben. Die GL-Datei kann anschließend beispielsweise durch den OpenGL-Viewer visualisiert werden. Wurde ein Server gestartet, so können durch API Clients HTTP GET Requests direkt an die von dem Server bereitgestellten Handler gestellt werden. Des Weiteren kann ein Nutzer das von dem Server bereitgestellte Frontend verwenden, welches die benötigten Kacheln von einem Tile-Server mittles HTTP GET Requests anfragt und ebenfalls die von dem Server bereitgestellten Handler nutzt. In diesem Fall verarbeitet das Frontend die zurückkommenden Antworten. Der QueryHandler nimmt hierzu den ResponseWriter zur Hilfe, um eine Anfrage zu beantworten. Sowohl der FileWriter als auch der ResponseWriter nutzen die vorberechneten optimalen Entpackreihenfolgen aus den Dateien.

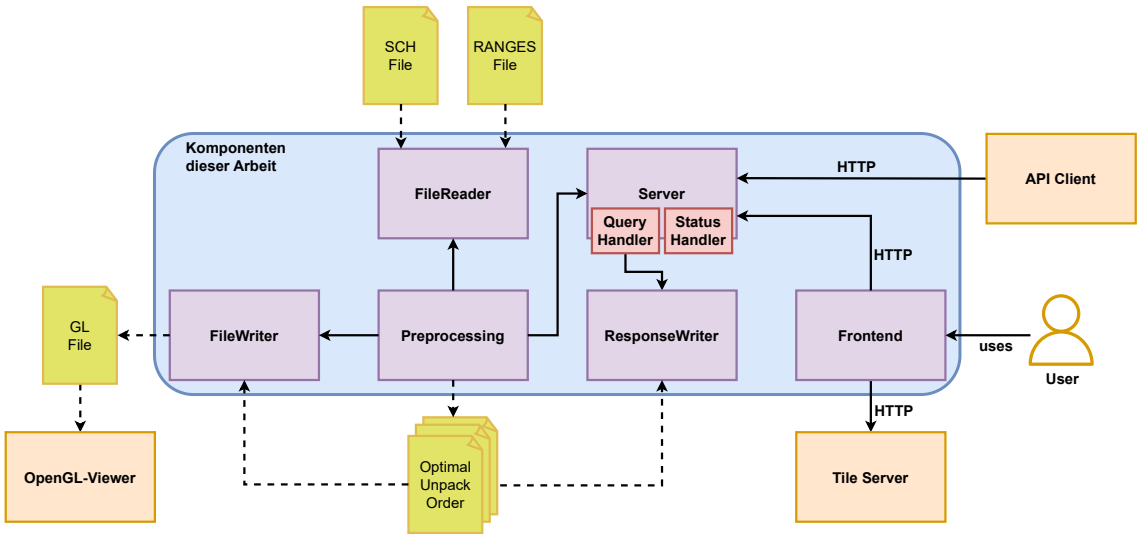


Abbildung 7.1: Gesamtbild der Interaktion zwischen den verschiedenen entwickelten oder genutzten Komponenten

8 Auswertung

Die in dieser Arbeit implementierten und getesteten Verfahren werden auf einem Testsystem für drei Eingabegraphen evaluiert. Hierzu sind die Systemeigenschaften sowie die Eigenschaften der getesteten Graphen nachfolgend aufgeführt. Es wird zum einen die Vorberechnung der optimalen Entpackreihenfolgen anhand der benötigten Zeiten ausgewertet. Zum anderen wird das Verhalten der Fehler im Entpackprozess näher betrachtet. Des Weiteren werden dynamische Anfragen an den Server geschickt. Hier werden die Berechnungszeiten im Backend sowie die Größen der Antworten genauer betrachtet. Schließlich wird noch der visuelle Eindruck der Antworten des Servers bei verschiedenen Entpackreihenfolgen verglichen.

8.1 Testsystem

Das Testsystem läuft auf Ubuntu 64-bit 22.04.3 LTS. Es besitzt einen Intel i5-9500 Prozessor, 64GB DDR4 RAM, eine Intel UHD Graphics 630 (GT2) Grafikeinheit und eine KIOXA NVMe SSD mit 512GB Speicher. Der Vorberechnung wurden 60GB RAM zugewiesen.

8.2 Getestete Graphen

In Tabelle 8.1 sind die Eigenschaften der drei getesteten Eingabegraphen tabellarisch aufgelistet. Die Anzahl der Knoten und Kanten wurde dabei aus der jeweiligen SCH-Datei ausgelesen. Die Anzahl der Shortcuts kann durch Zählen der Markierungen in dem *isShortcut*-Array berechnet werden und die Anzahl der Originalkanten kann durch die Differenz der Anzahl aller Kanten und der Shortcuts berechnet werden.

Eigenschaft	**Stuttgart**	**Baden-Württemberg (BW)**	**Deutschland**
SCH-Dateigröße in MB	180	781	4.411
RANGES-Dateigröße in MB	53	231	1.293
Anzahl der Knoten	1.132.113	4.688.703	25.115.477
Anzahl der Kanten	4.139.476	17.268.751	93.202.367
Anzahl der Originalkanten	2.292.297	9.458.680	50.774.208
Anzahl der Shortcuts	1.847.179	7.810.071	42.428.159

Tabelle 8.1: Eigenschaften der getesteten Eingabedateien

8.3 Einlesen der Daten und Berechnung der optimalen Entpackreihenfolgen

Zur Auswertung der Vorberechnung der optimalen Entpackreihenfolgen werden die Zeiten zum Einlesen der Eingabedateien in den Hauptspeicher, zur Vorberechnung der Fehler, zur tatsächlichen Vorberechnung der optimalen Entpackreihenfolgen, zum Schreiben der optimalen Entpackreihenfolgen in eine Datei und schließlich zum Einlesen dieser vorberechneten optimalen Entpackreihenfolgen aus der entsprechenden Datei in den Hauptspeicher gemessen und verglichen.

8.3.1 Einlesen der Eingabedateien und Vorberechnung der Fehler

Um mit einem der drei Eingabegraphen arbeiten zu können, müssen zuerst dessen SCH- und RANGES-Dateien in den Hauptspeicher eingelesen werden. Um daraufhin dann die optimalen Entpackreihenfolgen für alle in diesem Graph enthaltenen Shortcuts zu berechnen, werden zunächst die Fehler für jede im Graph enthaltene Kante für alle möglichen Fehlermetriken vorberechnet. Die entsprechenden Dauern für die drei Eingabegraphen sind in Tabelle 8.2 tabellarisch aufgelistet. Die Fehlermetrik 3 (Kosten) muss hierbei nicht vorberechnet werden, da die Kosten direkt aus der entsprechenden SCH-Datei eingelesen werden. Ebenfalls werden die zuvor berechneten asymptotischen Laufzeiten in der Tabelle aufgelistet. Die Zeiten zur Vorberechnung der Fehler spiegeln weitgehend die asymptotischen Laufzeiten wider, welche in Abschnitte 3.3 und 4.2.2 bis 4.2.5 berechnet wurden. Wie in den erwähnten Abschnitten definiert, ist V die Anzahl der Knoten, E die Anzahl der Kanten und E' die maximale Anzahl an Kanten auf einem Originalpfad, der von einem einzigen Shortcut approximiert wird.

	Stuttgart	**BW**	**Deutschland**	**Laufzeit**
Einlesen der Eingabedateien	3.674 ms	9.924 ms	50.711 ms	$O(V+E)$
Metrik 0 (Hausdorff)	3.398 ms	13.454 ms	77.442 ms	$O(E \cdot E')$
Metrik 1 (Fréchet)	49.298 ms	348.115 ms	2.131.169 ms	$O(E \cdot E'^2)$
Metrik 2 (Flächeninhalt)	2.365 ms	12.746 ms	78.714 ms	$O(E \cdot E')$
Metrik 3 (Kosten)	-	-	-	-
Metrik 4 (Distanz)	383 ms	1.916 ms	11.169 ms	$O(E \cdot E')$

Tabelle 8.2: Zeitmessung des Einlesens der getesteten Eingabedateien in den Hauptspeicher und der Vorberechnung der Fehler für die verschiedenen Metriken

8.3.2 Tatsächliche Berechnung der optimalen Entpackreihenfolgen

Nachdem die SCH- und RANGES-Dateien für einen Eingabegraph in den Hauptspeicher eingelesen sind und alle Fehler vorberechnet wurden, können jetzt die optimalen Entpackreihenfolgen für alle Metrik-Modus-Kombinationen vorberechnet werden. Die entsprechenden Dauern für die drei Eingabegraphen sind in Tabelle 8.3 tabellarisch aufgelistet. Für den Modus Zufällig existiert nur eine Datei, da dieser Modus unabhängig von der Fehlermetrik ist.

Obwohl die asymptotische Laufzeit für all diese Vorberechnungen dieselbe ist (siehe Abschnitt 4.4.1), lassen sich doch Unterschiede in den gemessenen Zeiten erkennen. Die vergleichsweise hohen Berechnungszeiten sind in der Tabelle in Rot hervorgehoben. Eine mögliche Ursache könnte sein, dass diese Metrik-Modus-Kombinationen gleichmäßiger entpacken und sich daher über einen längeren Zeitraum viele Shortcuts in *shortcutsToBeUnpacked* befinden, über die iteriert werden muss, um den als Nächstes zu entpackenden Shortcut zu identifizieren. Würde im Gegensatz dazu ein Teilpfad zunächst bis auf den Originalpfad entpackt werden, würden sich im Schnitt weniger Shortcuts gleichzeitig in *shortcutsToBeUnpacked* befinden. Stellt man sich den Entpackprozess wie in Abbildung 8.1 dargestellt als Baum vor, so besteht der Unterschied darin, ob die Shortcuts Ebene für Ebene (0-1-2-3-4-5-6), oder in die Tiefe (0-1-3-4-2-5-6) entpackt werden.

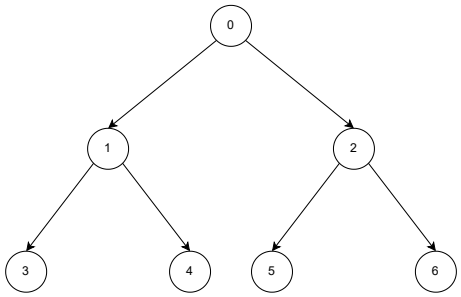


Abbildung 8.1: Darstellung des Entpackprozesses eines beispielhaften Shortcuts als Baum

8.3.3 Schreiben der optimalen Entpackreihenfolgen in eine Datei

Nachdem eine optimale Entpackreihenfolge für eine spezifische Metrik-Modus-Kombination berechnet wurde, wird diese sofort in eine entsprechende Datei geschrieben. Die entsprechenden Dauern für die drei Eingabegraphen sind ebenfalls in Tabelle 8.3 tabellarisch aufgelistet. Die fertige Datei ist für den Stuttgart-Graph 116 MB, für den BW-Graph 588 MB und für den Deutschland-Graph 3.736 MB groß. Diese Größe ist pro Eingabegraph für jede Metrik-Modus-Kombination dieselbe. Das ist der Fall, da für jeden Shortcut bis zur vollständigen Entpackung die optimale Entpackreihenfolge berechnet wird. Unabhängig von der Metrik-Modus-Kombination ist also für jeden Shortcut die Liste der Shortcut-IDs, die nacheinander entpackt werden, gleich groß. Daher sind die Schreibdauern pro Eingabegraph in derselben Größenordnung.

8.3.4 Einlesen der optimalen Entpackreihenfolgen aus einer Datei

Befindet sich eine zum Entpacken benötigte optimale Entpackreihenfolge für eine spezifische Metrik-Modus-Kombination aktuell nicht im Hauptspeicher, so muss diese aus der entsprechenden zuvor berechneten und geschriebenen Datei in den Hauptspeicher eingelesen werden. Die entsprechenden Dauern für die drei Eingabegraphen sind ebenfalls in Tabelle 8.3 tabellarisch aufgelistet.

Es lässt sich erkennen, dass die Einlesezeiten pro Eingabegraph in derselben Größenordnung liegen. Dies war auch zu erwarten, da die Dateien pro Eingabegraph exakt dieselbe Größe besitzen.

M-M-K	**Berechnung** (in ms)			**Schreiben** (in ms)			**Einlesen** (in ms)		
	ST	**BW**	**DE**	**ST**	**BW**	**DE**	**ST**	**BW**	**DE**
0-0	16.190	132.452	764.886	806	3.756	19.801	1.447	3.403	14.661
0-1	4.875	27.521	157.554	748	3.848	21.163	1.099	3.018	13.191
0-2	6.407	33.240	196.138	744	5.002	20.433	1.215	4.181	12.620
0-3	3.374	15.543	91.355	748	3.406	18.610	951	2.875	12.762
0-4	3.592	16.116	101.756	812	4.215	18.440	1.493	3.743	12.106
0-5	3.243	15.747	106.666	682	3.174	18.558	1.026	2.901	12.094
1-0	12.500	102.604	644.563	773	3.272	20.880	1.079	3.105	14.013
1-1	3.595	17.436	108.378	710	3.317	18.640	1.029	3.041	11.921
1-2	4.528	24.475	157.702	1.195	3.129	20.498	833	3.294	11.992
1-3	2.638	13.907	92.701	770	3.217	20.999	828	3.167	12.228
1-4	4.324	22.224	147.445	768	3.264	21.209	1.044	3.233	13.652
1-5	3.695	18.401	118.364	766	3.266	23.827	830	3.082	13.810
2-0	17.473	149.540	867.110	784	3.429	18.658	837	3.269	12.276
2-1	7.261	40.689	254.151	727	4.144	20.461	832	4.216	15.177
2-2	11.016	74.130	441.774	770	3.262	18.428	850	3.393	12.558
2-3	2.975	15.285	94.594	672	4.412	20.104	829	2.872	12.275
2-4	3.190	16.892	106.547	771	5.439	18.499	990	2.870	15.042
2-5	3.218	19.985	107.728	769	4.229	20.876	1.195	2.869	11.919
3-0	16.267	131.787	773.556	740	3.287	21.988	837	3.356	13.485
3-1	18.080	306.154	1.829.119	1.049	3.288	18.635	826	2.875	12.204
3-2	21.421	180.281	1.157.844	840	3.119	19.674	1.051	3.941	13.328
3-3	2.641	14.443	91.020	771	3.468	19.156	826	3.066	13.305
3-4	17.382	316.110	1.888.758	921	3.113	19.957	846	3.123	13.881
3-5	4.121	17.223	110.174	757	3.340	18.536	1.042	3.034	11.951
4-0	15.104	125.476	742.626	767	3.282	23.076	837	3.283	12.244
4-1	17.534	299.882	1.794.639	766	3.392	18.657	1.007	5.142	12.968
4-2	20.200	171.278	1.042.982	769	3.962	24.126	836	2.919	12.280
4-3	2.993	14.345	91.148	764	3.285	20.790	828	3.571	14.998
4-4	17.765	300.174	1.802.693	768	3.303	21.489	827	2.868	12.225
4-5	3.221	17.252	107.914	1.175	3.313	21.511	829	3.097	12.639
x-6	1.733	8.539	52.190	947	3.887	19.641	866	3.262	13.794

Tabelle 8.3: Zeitmessung in Millisekunden für die Berechnung und für das Schreiben der optimalen Entpackreihenfolgen in eine Datei und für das Einlesen der optimalen Entpackreihenfolgen aus einer Datei in den Hauptspeicher für den Stuttgart-Graph (ST), den Baden-Württemberg-Graph (BW) und den Deutschland-Graph (DE) für jede Metrik-Modus-Kombination (M-M-K)

8.4 Verhalten der Fehler im Entpackprozess

In Tabelle 8.4 ist tabellarisch aufgetragen, wie viele der Shortcuts beim einmaligen Entpacken den aktuellen Fehler bezüglich einer gewissen Fehlermetrik verschlechtern. Dazu wird jeweils ein Shortcut entpackt, sein eigener Fehler und der Fehler seiner beiden Kindkanten berechnet. Der totale Fehler der einmaligen Entpackung ergibt sich dann entweder aus der Summe oder dem Maximum der Fehler der beiden Kindkanten. Ist dieser totale Fehler der Kindkanten größer als der Fehler des zu entpackenden Shortcuts, so wird der Fehler bei dieser Entpackung verschlechtert. Der entsprechende Pseudocode zur Berechnung der Anzahl der Kanten, für welche dieser Fall eintritt, ist in Algorithmus 8.1 zu finden.

Algorithmus 8.1 Berechnung der Anzahl der Kanten, welche den Fehler beim einmaligen Entpacken erhöhen

```
procedure countHowOftenErrorIncreasesWhileUnpacking(metric, sum)
    count ← 0
    for edge in edges do                                   // Iterate over all edges
        if isShortcut(edge) then                           // Check if the current edge is a shortcut
            errorShortcut ← getError(edge, metric)
            child1 ← getChild1(edge)
            child2 ← getChild2(edge)
            errorChild1 ← getError(child1, metric)
            errorChild2 ← getError(child2, metric)
            if sum then
                totalError ← errorChild1 + errorChild2
            else
                totalError ← max(errorChild1, errorChild2)
            end if
            if totalError > errorShortcut then
                count ← count + 1
            end if
        end if
    end for
    return count
end procedure
```

Strategie	Fehlerberechnung	Stuttgart	BW	Deutschland
Metrik 0 (Hausdorff)	Summe	124.858	526.013	3.095.326
	Maximum	99.297	424.397	2.508.081
Metrik 1 (Fréchet)	Summe	402.843	1.767.069	9.981.848
	Maximum	191.668	867.561	4.972.220
Metrik 2 (Flächeninhalt)	Summe	80.267	340.230	2.023.951
	Maximum	55.477	237.591	1.437.131
Metrik 3 (Kosten)	Summe	0	0	0
	Maximum	0	0	0
Metrik 4 (Distanz)	Summe	0	0	0
	Maximum	0	0	0

Tabelle 8.4: Anzahl der Kanten, die den Fehler beim einmaligen Entpacken erhöhen, für verschiedene Metriken

Aufgrund der Tatsache, dass der Fehler für keine der Metriken 0 (Hausdorff), 1 (Fréchet) oder 2 (Flächeninhalt) beim Entpackprozess monoton fallend ist, kann es mit Modus 1 (Größte Fehlerreduktion, Summe) und Modus 2 (Größte Fehlerreduktion, Maximum) (siehe Abschnitt 4.3.2) zu Problemen kommen. Ein Shortcut, der im nächsten Schritt den Fehler minimal verschlechtert, in den folgenden Schritten den Fehler aber drastisch reduzieren würde, wird erst entpackt, sobald kein anderer Shortcut mehr auch nur eine minimale Verbesserung bringt, also sobald das Entpacken jedes anderen Shortcuts den Fehler nur noch mehr verschlechtern würde. Das kann dazu führen, dass im Entpackprozess gewisse Pfade schon sehr detailliert visualisiert werden, während andere Pfade noch sehr grob approximiert werden.

Bei den Metriken 3 (Kosten) und 4 (Distanz) wird der Fehler im Entpackprozess nie größer. Das liegt daran, dass der Fehler einer Kante immer genau der Summe der Fehler der beiden Kindkanten entspricht. Somit ist der Fehler immer mindestens so groß wie die Summe beziehungsweise das Maximum der Fehler der beiden Kindkanten.

8.5 Dynamische Evaluation des Entpackens von Shortcuts

Um die Berechnungszeiten im Backend sowie die Größe der Antwort bei Anfragen zu evaluieren, wurden verschiedene HTTP GET Requests an den Server geschickt. Die entsprechenden Zeiten wurden hierbei mit der Methode *System.currentTimeMillis()* der Java *System* Klasse gemessen.

Da im Entpackprozess im Backend die jeweilige angefragte Metrik-Modus-Kombination keine Rolle mehr spielt, sondern nur die jeweils richtige vorberechnete Datei in den Hauptspeicher eingelesen werden muss, wurden die Anfragen stellvertretend für Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler) gestellt.

Aufgrund der in Abschnitte 3.2, 5.3.1 und 5.3.2 angesprochenen Probleme wurde in diesen Anfragen ausschließlich mit den Kantenlevels aus der RANGES-Datei gearbeitet. Außerdem werden keine zugehörigen Originalpfade visualisiert.

Es werden für jeden Eingabegraph Anfragen über jeweils alle Zoomlevel und für unterschiedliche Anzahlen an Entpackschritten gestellt. Für die Anzahl von Entpackschritten werden die Werte $0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 50$ und 100 getestet.

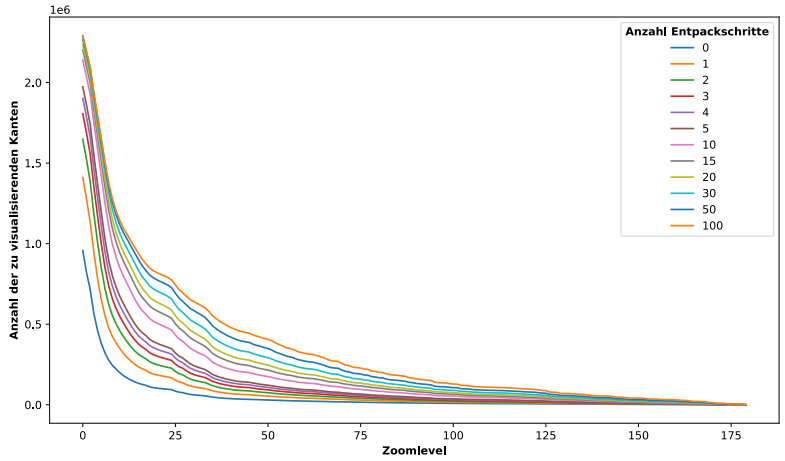
Hierbei wird ein Timeout von 120 Sekunden eingesetzt. Bei einem Timeout werden die Anfragen für den aktuellen Wert der Anzahl von Entpackschritten abgebrochen und es wird mit dem nächsten Wert, wieder über jeweils alle Zoomlevel, fortgefahren. Ebenfalls werden die Anfragen für den aktuellen Wert der Anzahl von Entpackschritten abgebrochen und es wird mit dem nächsten Wert fortgefahren, wieder über jeweils alle Zoomlevel, wenn die Antwort nicht zu einem *JSONString* geparst werden kann, da sie zu groß ist.

In Abbildung 8.2 sind die Kurven für die Anzahl der zu visualisierenden Kanten, in Abbildung 8.3 für die zum Entpacken benötigte Zeit, in Abbildung 8.4 für die zum Bauen der GeoJSON-Antwort benötigte Zeit und in Abbildung 8.5 für die totale Backend Zeit zum Bauen der Antwort dargestellt.

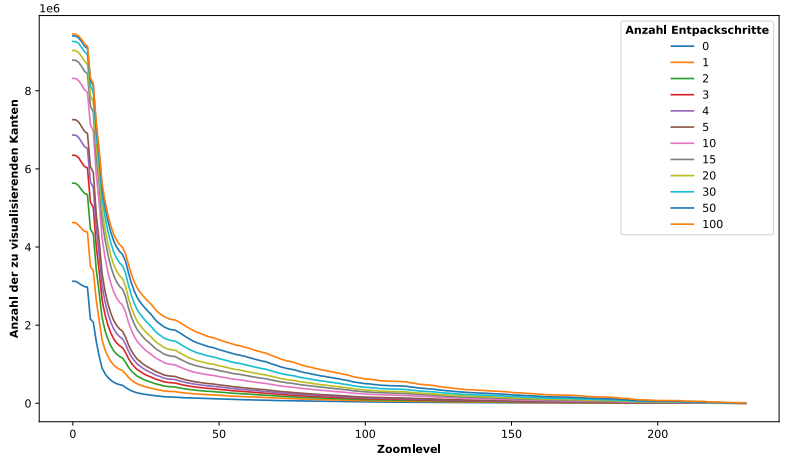
Aufgrund der Art und Weise, wie das GeoJSON-Objekt erstellt wird (siehe Abschnitt 6.2.1), ist die Größe der Antwort ausschließlich von der Anzahl der zu visualisierenden Kanten abhängig. Daher sieht der Verlauf einer Kurve, welche die Größe der Antwort für alle Zoomlevel darstellt, exakt so aus wie der Verlauf der Kurve, welche die Anzahl der zu visualisierenden Kanten für alle Zoomlevel darstellt.

Es lässt sich erkennen, dass sowohl mit fallendem Zoomlevel, als auch mit steigender Anzahl von Entpackschritten zum einen die Anzahl der zu visualisierenden Kanten und damit auch die Größe der Antwort und zum anderen die gemessenen Zeiten steigen.

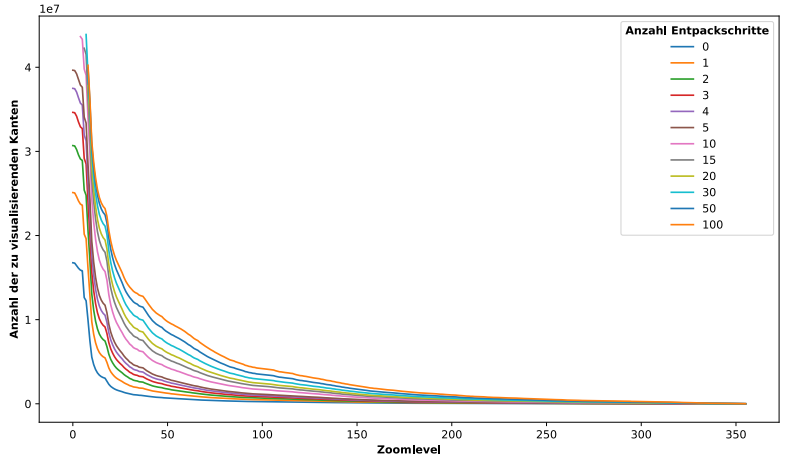
Um zu demonstrieren, dass die Metrik-Modus-Kombination bei Anfragen keinerlei Auswirkung hat, sind in Abbildung 8.6 noch die entsprechenden Kurven für den Stuttgart-Graph für Metrik 2 (Flächeninhalt) und Modus 1 (Größte Fehlerreduktion, Summe) dargestellt. Es lässt sich erkennen, dass die Größenordnungen der gemessenen Werte dieselben sind.



(a) Stuttgart-Graph

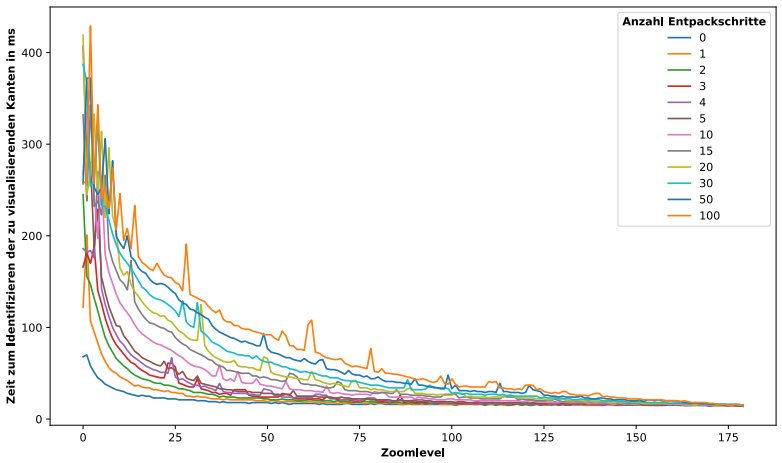


(b) Baden-Württemberg-Graph

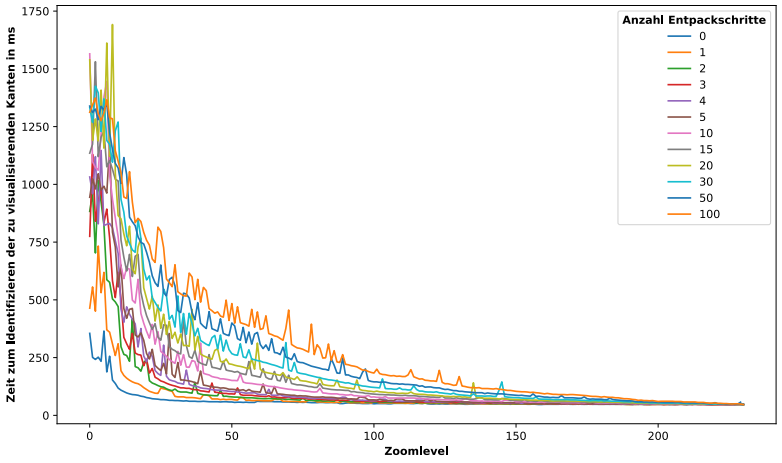


(c) Deutschland-Graph

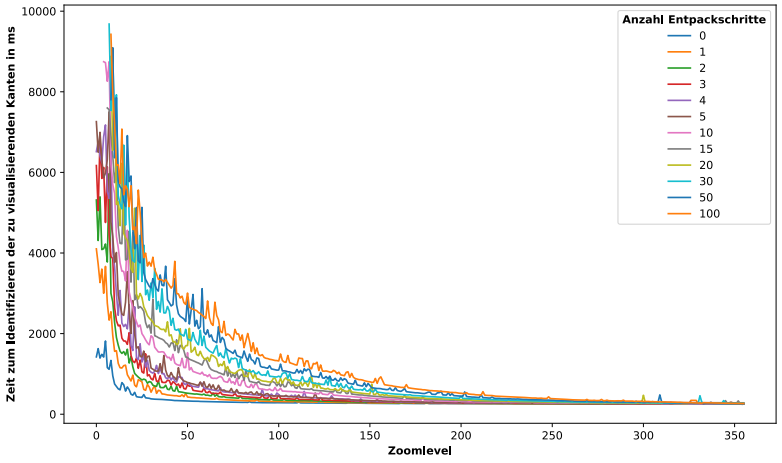
Abbildung 8.2: Anzahl der zu visualisierenden Kanten für alle Zoomlevel für verschiedene Anzahlen von Entpackschritten für alle Testgraphen für Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler)



(a) Stuttgart-Graph

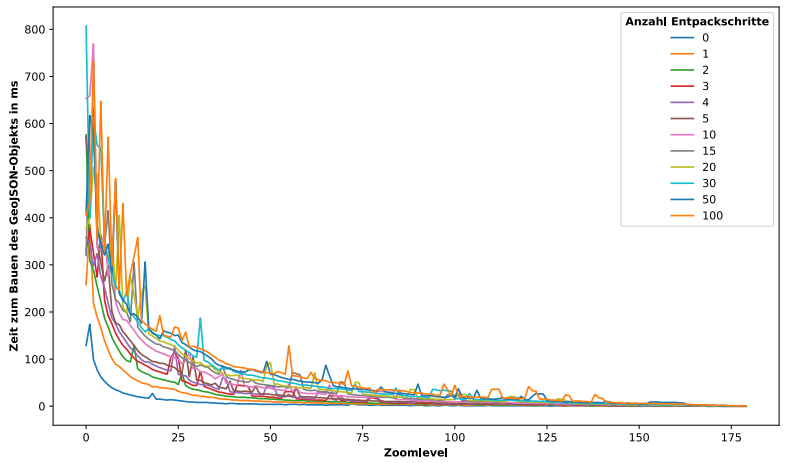


(b) Baden-Württemberg-Graph

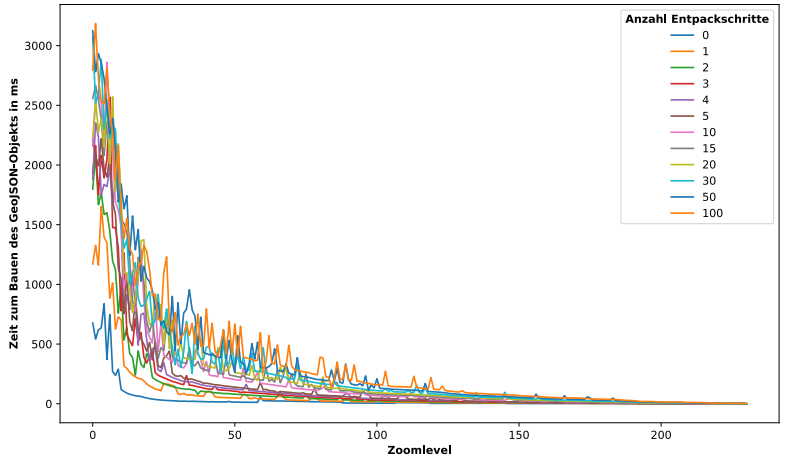


(c) Deutschland-Graph

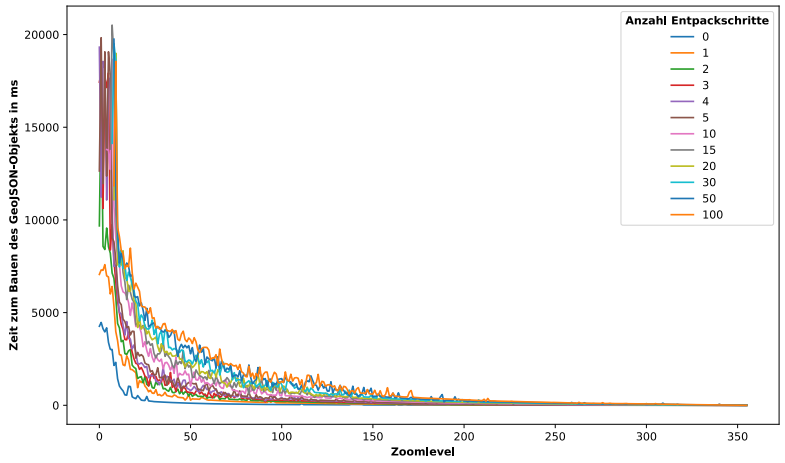
Abbildung 8.3: Zeit, die zum Entpacken benötigt wird, bis alle zu visualisierenden Kanten identifiziert sind für alle Zoomlevel für verschiedene Anzahlen von Entpackschritten für alle Testgraphen für Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler)



(a) Stuttgart-Graph

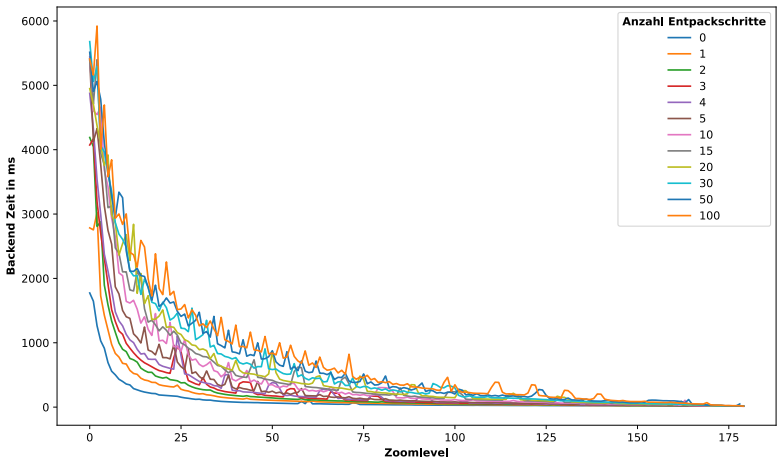


(b) Baden-Württemberg-Graph

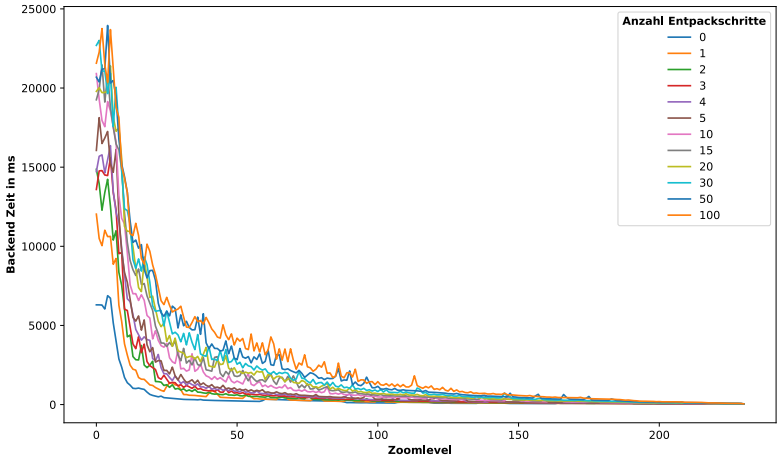


(c) Deutschland-Graph

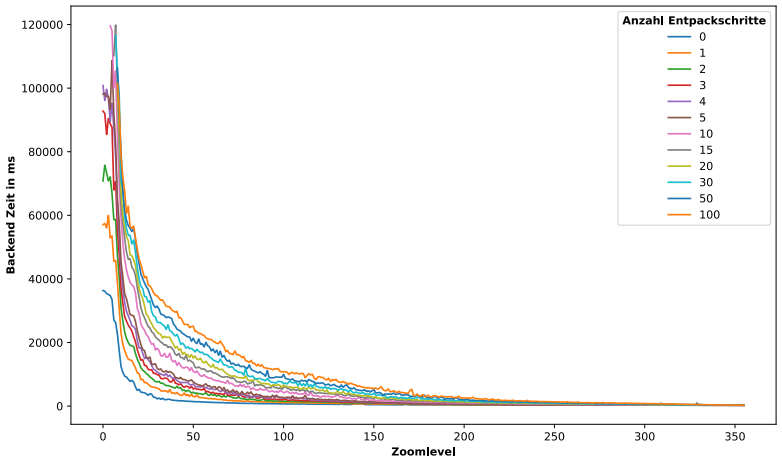
Abbildung 8.4: Zeit zum Bauen der GeoJSON-Antwort für alle Zoomlevel für verschiedene Anzahlen von Entpackschritten für alle Testgraphen für Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler)



(a) Stuttgart-Graph

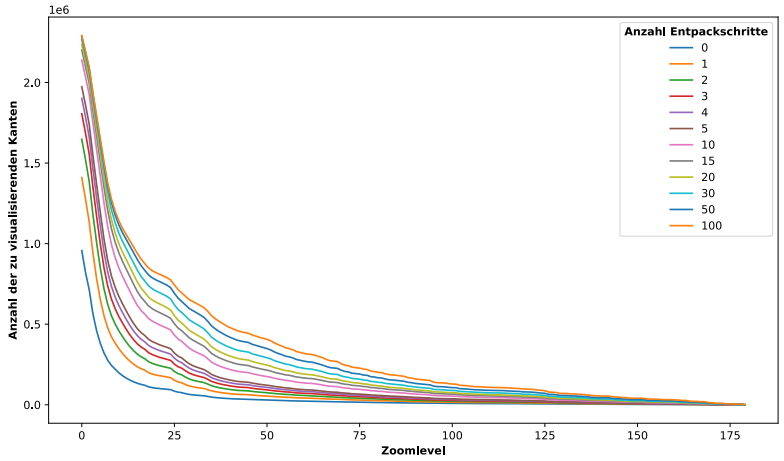


(b) Baden-Württemberg-Graph

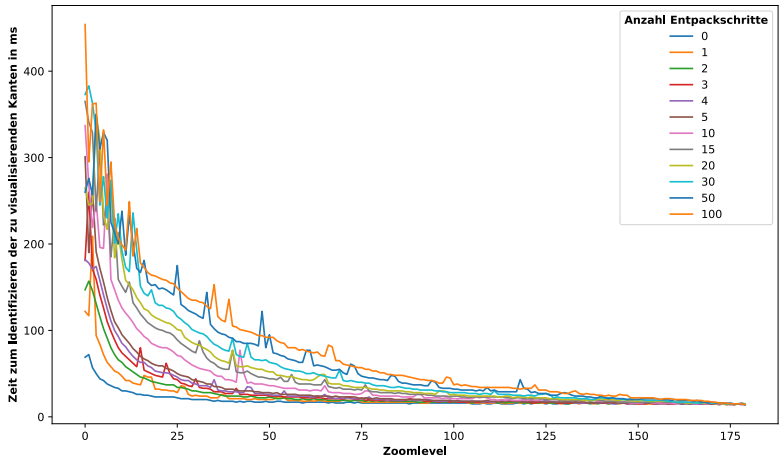


(c) Deutschland-Graph

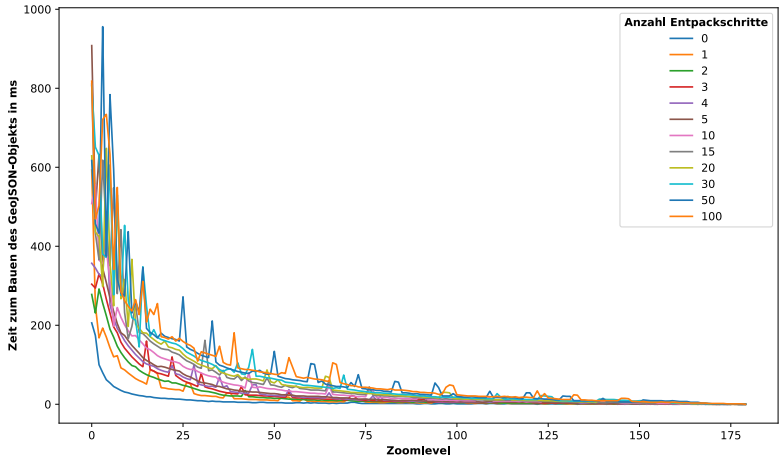
Abbildung 8.5: Totale Backend Zeit zum Bauen der Antwort für alle Zoomlevel für verschiedene Anzahlen von Entpackschritten für alle Testgraphen für Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler)



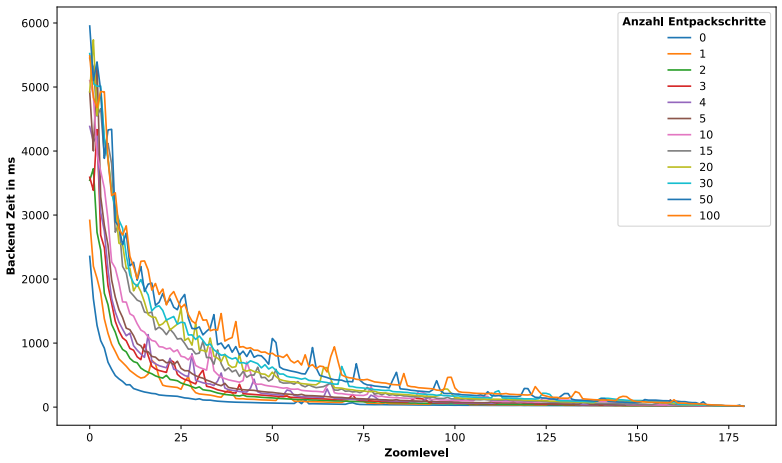
(a) Anzahl der zu visualisierenden Kanten



(b) Zeit, die zum Entpacken benötigt wird, bis alle zu visualisierenden Kanten identifiziert sind



(c) Zeit zum Bauen der GeoJSON-Antwort



(d) Totale Backend Zeit zum Bauen der Antwort

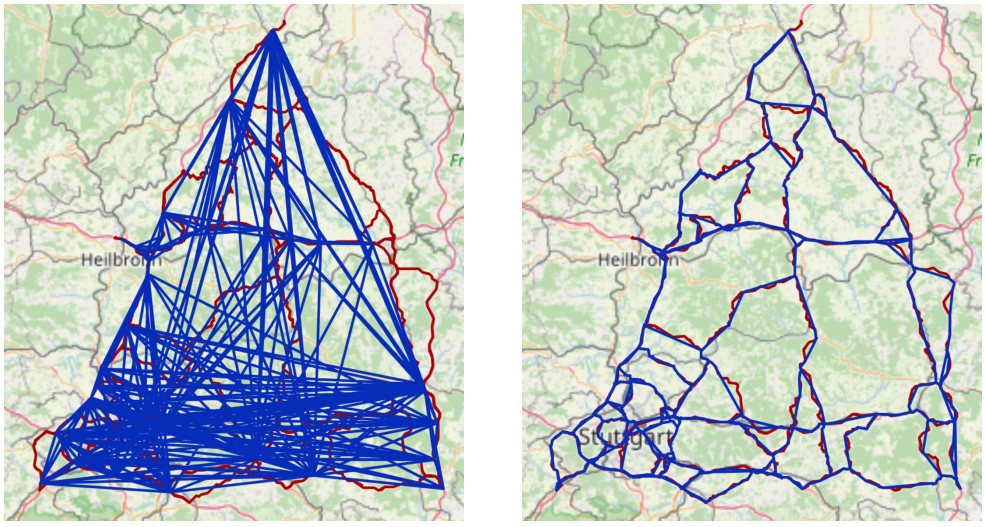
Abbildung 8.6: Messungen der Berechnungszeiten im Backend und der Antwort-Größen für alle Zoomlevel für verschiedene Anzahlen von Entpackschritten für den Stuttgart-Graph für Metrik 2 (Flächeninhalt) und Modus 1 (Größte Fehlerreduktion, Summe)

8.6 Anschauungsbeispiele zum Vergleich des visuellen Eindrucks verschiedener Strategien

Im Folgenden werden die unterschiedlichen Metriken und Modi in Bezug auf ihren visuellen Eindruck verglichen.

In Abschnitte 3.2, 5.3.1 und 5.3.2 wurde erwähnt, dass das Identifizieren der relevanten Kanten auf einem gegebenen Zoomlevel auf Basis der gegebenen Level in der SCH-Datei zu Problemen führen kann. Daher sollte mit den gegebenen Level-Intervallen aus der RANGES-Datei gearbeitet werden.

Dies wird in Abbildung 8.7 nochmals durch zwei Bilder verdeutlicht, welche die beispielhaft auf Zoomlevel 150 identifizierten relevanten Kanten für 0 Entpackschritte, einmal für die SCH-Datei und einmal für die RANGES-Datei, zeigen.

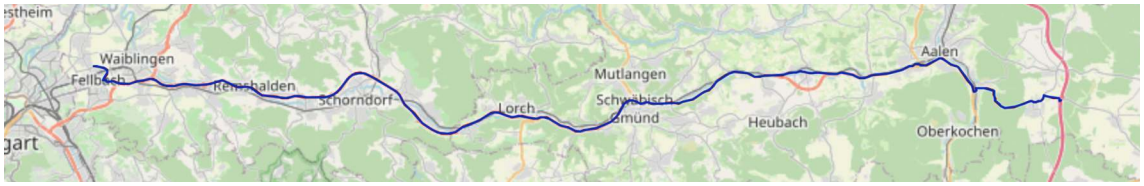


(a) SCH **(b)** RANGES

Abbildung 8.7: Vergleich der visualisierten Antwort für die SCH- und die RANGES-Dateien des Stuttgart-Graphen für eine Anfrage mit Zoomlevel 150 und 0 Entpackschritten

Im Folgenden wird daher nur noch mit den Levels aus der RANGES-Datei gearbeitet.

Zunächst wird der visuelle Eindruck des Shortcuts, der die Bundesstraße 29 (B29) approximiert, für alle Metrik-Modus-Kombinationen und jeweils 200 Entpackschritte verglichen. Dieser Shortcut besitzt 1.387 Kanten auf dem Originalpfad. In Abbildung 8.8 ist der Vergleich aller Modi für Metrik 0 (Hausdorff), in Abbildung 8.9 für Metrik 1 (Fréchet), in Abbildung 8.10 für Metrik 2 (Flächeninhalt), in Abbildung 8.11 für Metrik 3 (Kosten) und in Abbildung 8.12 schließlich für Metrik 4 (Distanz) dargestellt.



(a) Modus 0 (Größter Fehler)

(b) Modus 1 (Größte Fehlerreduktion, Summe)

(c) Modus 2 (Größte Fehlerreduktion, Maximum)

(d) Modus 3 (Kleinster Fehler)

(e) Modus 4 (Kleinste Fehlerreduktion, Summe)

(f) Modus 5 (Kleinste Fehlerreduktion, Maximum)

(g) Modus 6 (Zufällig)

Abbildung 8.8: Vergleich aller Modi für den Shortcut, der die B29 approximiert, für Metrik 0 (Hausdorff) und jeweils 200 Entpackschritte

(a) Modus 0 (Größter Fehler)

(b) Modus 1 (Größte Fehlerreduktion, Summe)

(c) Modus 2 (Größte Fehlerreduktion, Maximum)

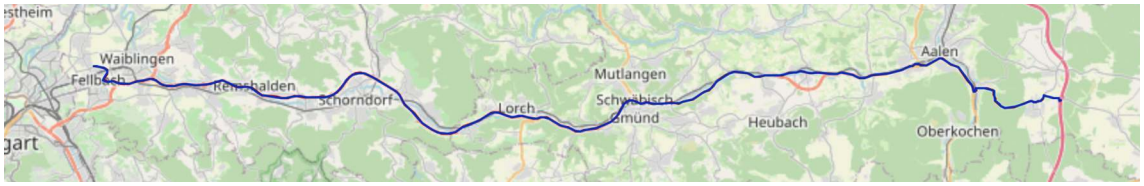
(d) Modus 3 (Kleinster Fehler)

(e) Modus 4 (Kleinste Fehlerreduktion, Summe)

(f) Modus 5 (Kleinste Fehlerreduktion, Maximum)

(g) Modus 6 (Zufällig)

Abbildung 8.9: Vergleich aller Modi für den Shortcut, der die B29 approximiert, für Metrik 1 (Fréchet) und jeweils 200 Entpackschritte



(a) Modus 0 (Größter Fehler)

(b) Modus 1 (Größte Fehlerreduktion, Summe)

(c) Modus 2 (Größte Fehlerreduktion, Maximum)

(d) Modus 3 (Kleinster Fehler)

(e) Modus 4 (Kleinste Fehlerreduktion, Summe)

(f) Modus 5 (Kleinste Fehlerreduktion, Maximum)

(g) Modus 6 (Zufällig)

Abbildung 8.10: Vergleich aller Modi für den Shortcut, der die B29 approximiert, für Metrik 2 (Flächeninhalt) und jeweils 200 Entpackschritte

(a) Modus 0 (Größter Fehler)

(b) Modus 1 (Größte Fehlerreduktion, Summe)

(c) Modus 2 (Größte Fehlerreduktion, Maximum)

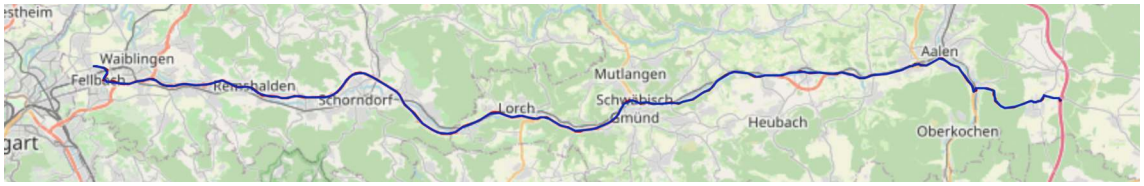
(d) Modus 3 (Kleinster Fehler)

(e) Modus 4 (Kleinste Fehlerreduktion, Summe)

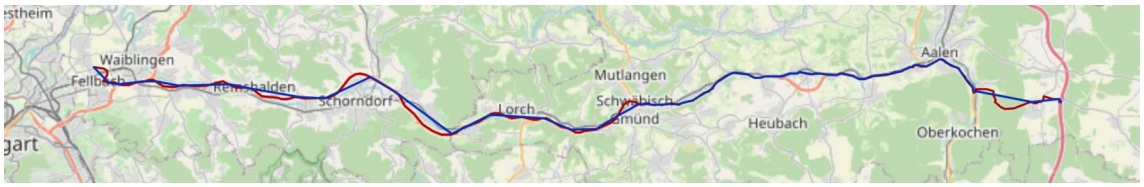
(f) Modus 5 (Kleinste Fehlerreduktion, Maximum)

(g) Modus 6 (Zufällig)

Abbildung 8.11: Vergleich aller Modi für den Shortcut, der die B29 approximiert, für Metrik 3 (Kosten) und jeweils 200 Entpackschritte



(a) Modus 0 (Größter Fehler)



(b) Modus 1 (Größte Fehlerreduktion, Summe)

(c) Modus 2 (Größte Fehlerreduktion, Maximum)

(d) Modus 3 (Kleinster Fehler)

(e) Modus 4 (Kleinste Fehlerreduktion, Summe)

(f) Modus 5 (Kleinste Fehlerreduktion, Maximum)

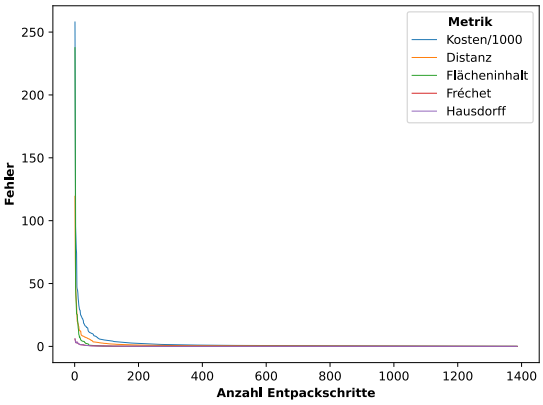
(g) Modus 6 (Zufällig)

Abbildung 8.12: Vergleich aller Modi für den Shortcut, der die B29 approximiert, für Metrik 4 (Distanz) und jeweils 200 Entpackschritte

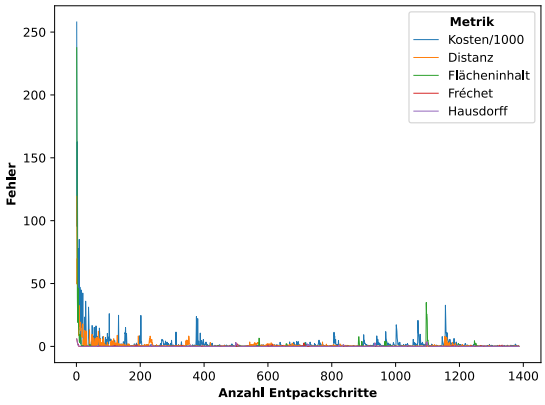
Die gezeigten Bilder bestätigen die Vermutung aus Abschnitt 8.3.2, dass diejenigen Metrik-Modus-Kombinationen, welche eine höhere Vorberechnungsdauer hatten, den Shortcut tendenziell gleichmäßiger entpacken als die Metrik-Modus-Kombinationen mit geringeren Vorberechnungsdauern. Ausgenommen von dieser Beobachtung ist Modus 6 (Zufällig). Gleichmäßig bedeutet hierbei, dass die Detailliertheit der visualisierten Kanten möglichst gleich ist.

Die gezeigten Bilder bestätigen ebenfalls die Vermutung aus Abschnitt 8.4, dass bei Modus 1 (Größte Fehlerreduktion, Summe) und Modus 2 (Größte Fehlerreduktion, Maximum) aufgrund der Nicht-Monotonie des Fehlers bei dem Entpackprozess bei gewissen Fehlermetriken die Gefahr besteht, dass nach dem Entpacken gewisse Pfade schon sehr detailliert visualisiert werden, während andere Pfade noch sehr grob approximiert werden.

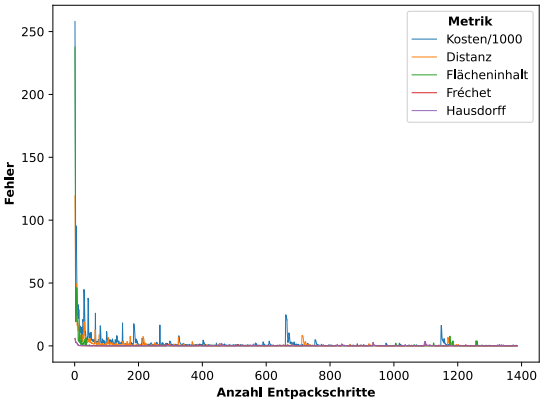
Es lässt sich erkennen, dass für jede Metrik der visuelle Eindruck für Modus 0 (Größter Fehler) am besten ist, falls der Detaillierungsgrad aller visualisierten Kanten möglichst gleich sein soll. Dies wird ebenfalls deutlich, wenn man, wie in Abbildung 8.13 dargestellt, die Fehlerentwicklung der aktuell entpackten Kante im vollständigen Entpackprozess dieses Shortcuts betrachtet. Die Fehler der Metrik 3 (Kosten) wurde hierbei skaliert.



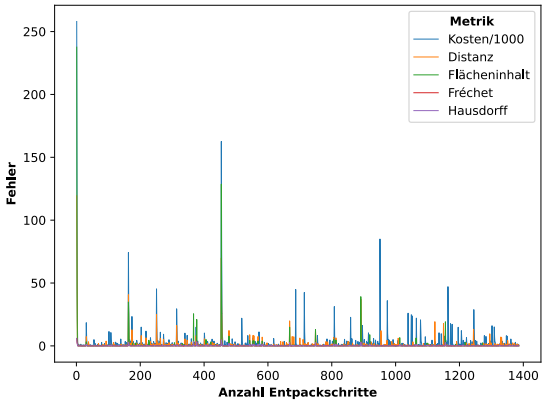
(a) Modus 0
(Größter Fehler)



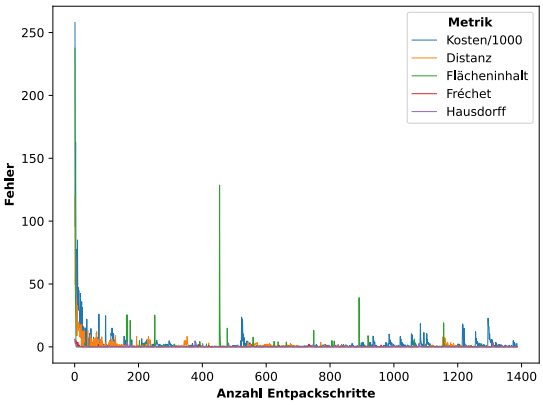
(b) Modus 1
(Größte Fehlerreduktion, Summe)



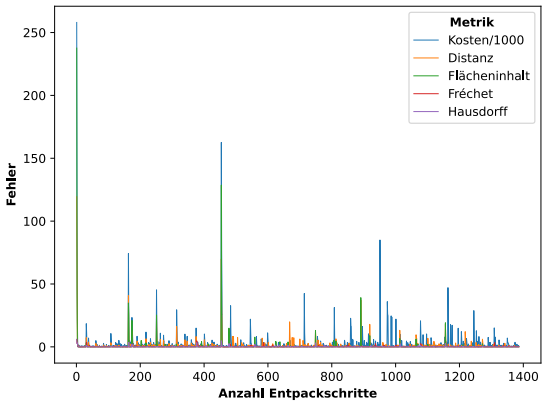
(c) Modus 2
(Größte Fehlerreduktion, Maximum)



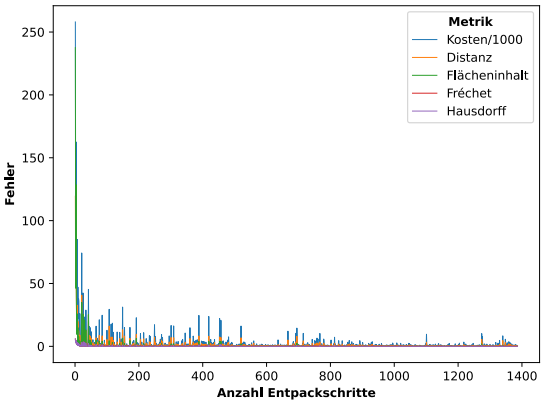
(d) Modus 3
(Kleinster Fehler)



(e) Modus 4
(Kleinste Fehlerreduktion, Summe)



(f) Modus 5
(Kleinste Fehlerreduktion, Maximum)



(g) Modus 6
(Zufällig)

Abbildung 8.13: Fehler der jeweils aktuell entpackten Kante für die vollständige Entpackung des Shortcuts, der die B29 approximiert

Um die verschiedenen Metriken zu vergleichen, wird im Folgenden Modus 0 (Größter Fehler) fixiert. Es wird für diesen Vergleich beispielhaft ein weiterer Shortcut näher betrachtet. In Abbildung 8.14 ist dieser Shortcut unentpackt dargestellt. Dieser Shortcut besitzt 1416 Kanten auf dem Originalpfad.

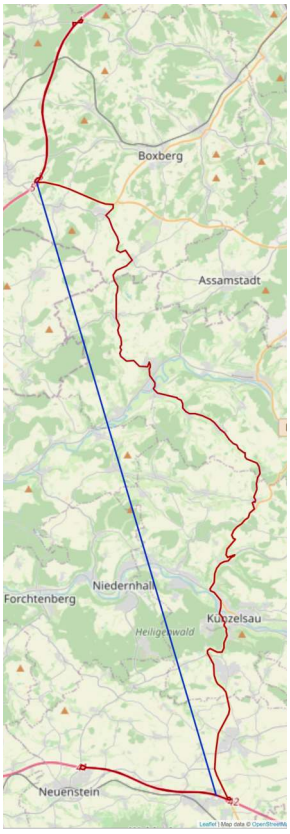
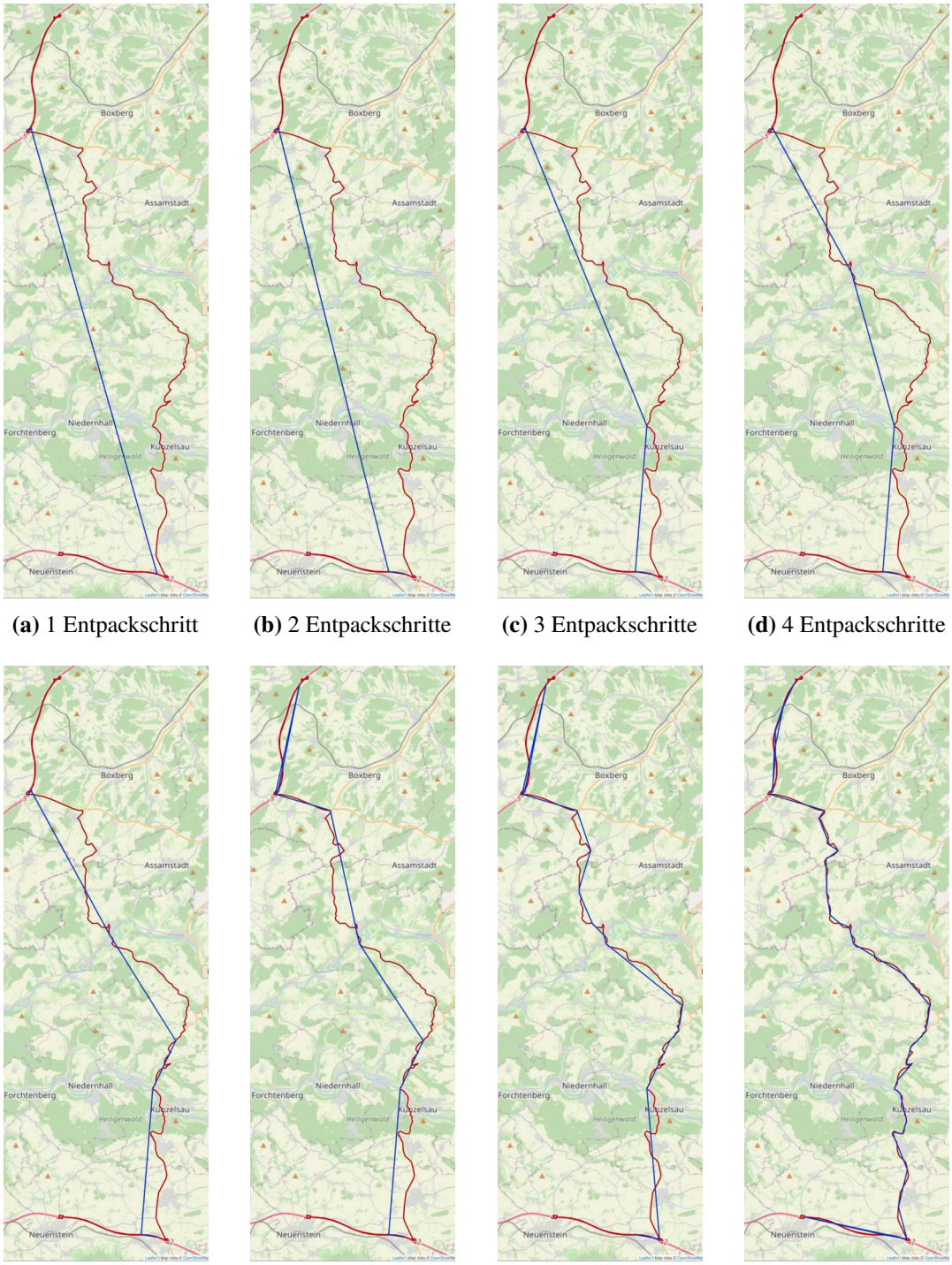


Abbildung 8.14: Beispielhafter Shortcut zum Vergleich der Metriken für 0 Entpackschritte

Für die verschiedenen Metriken werden die Visualisierungen dieses Shortcuts jeweils für 1, 2, 3, 4, 5, 10, 20 und 50 Entpackschritte dargestellt. Für Metrik 0 (Hausdorff) sind diese in Abbildung 8.15, für Metrik 1 (Fréchet) in Abbildung 8.16, für Metrik 2 (Flächeninhalt) in Abbildung 8.17, für Metrik 3 (Kosten) in Abbildung 8.18 und für Metrik 4 (Distanz) in Abbildung 8.19 dargestellt.

In Abbildung 8.20 ist für diesen Shortcut noch die Fehlerentwicklung der aktuell entpackten Kante im Entpackprozess bis zur 50. Entpackung dargestellt, wobei der Fehler der Metrik 3 (Kosten) wieder skaliert wurde.

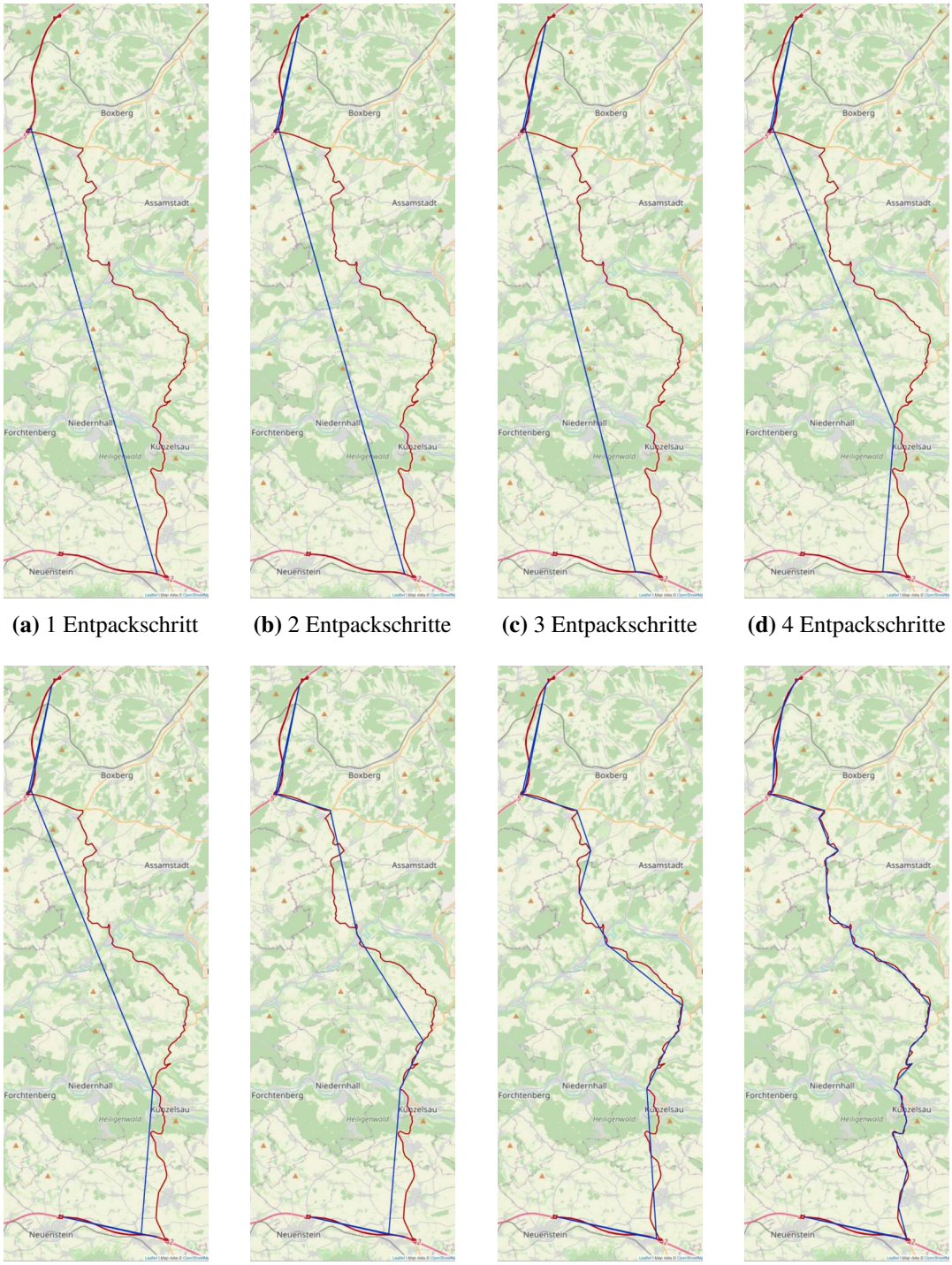
Es lassen sich minimale visuelle Unterschiede im Entpackprozess der verschiedenen Metriken erkennen, beispielsweise wie früh die im gegebenen Beispiel enthaltenen Abzweigungen entpackt werden. Alles in allem sehen jedoch die Approximationen für jede Metrik ab der 50. Entpackung visuell schon sehr ansprechend aus und es sticht keine Metrik deutlich hervor.



(a) 1 Entpackschritt **(b)** 2 Entpackschritte **(c)** 3 Entpackschritte **(d)** 4 Entpackschritte

(e) 5 Entpackschritte **(f)** 10 Entpackschritte **(g)** 20 Entpackschritte **(h)** 50 Entpackschritte

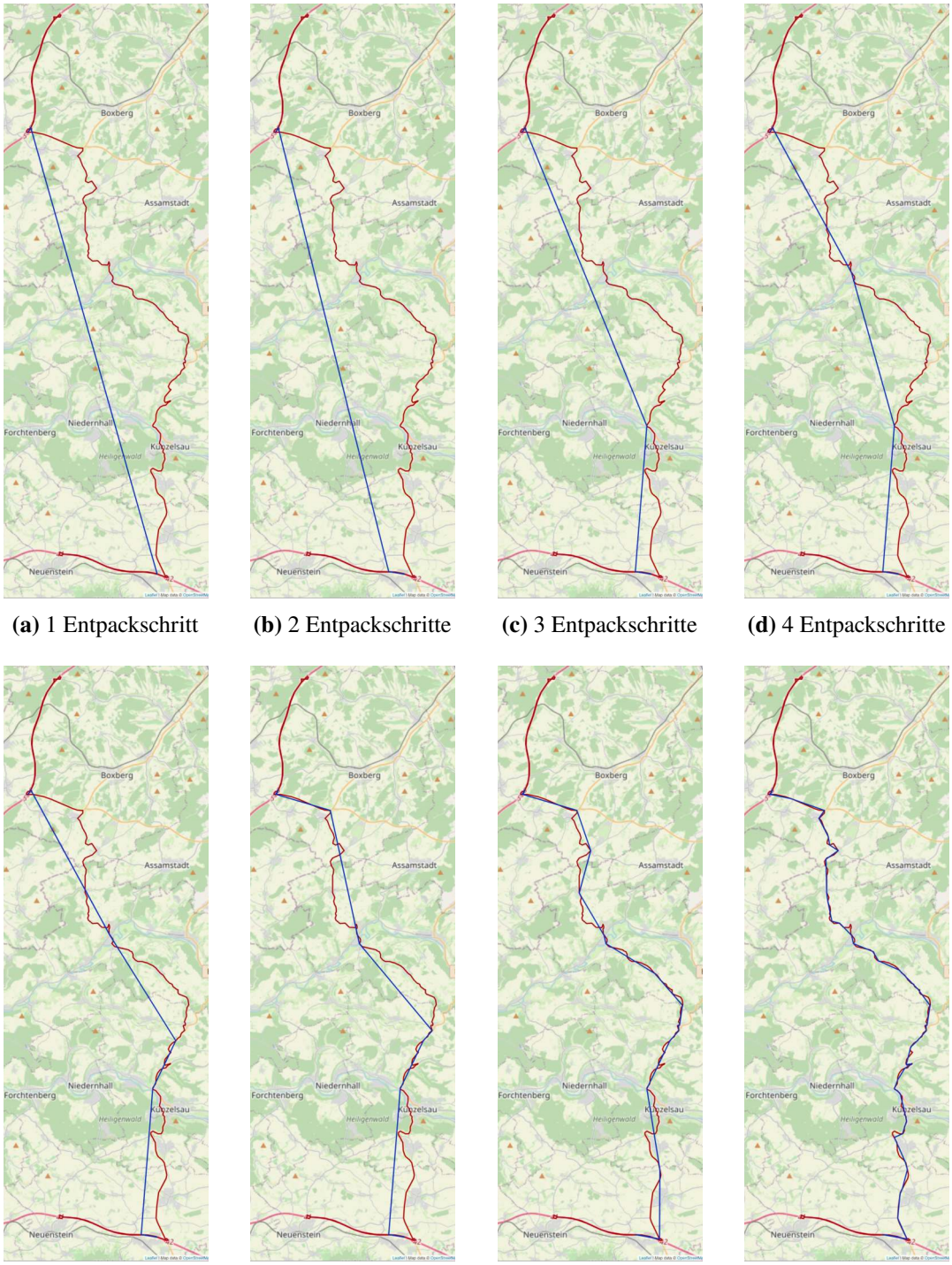
Abbildung 8.15: Beispielhafter Shortcut für verschiedene Anzahlen von Entpackschritten mit Metrik 0 (Hausdorff) und Modus 0 (Größter Fehler)



(a) 1 Entpackschritt **(b)** 2 Entpackschritte **(c)** 3 Entpackschritte **(d)** 4 Entpackschritte

(e) 5 Entpackschritte **(f)** 10 Entpackschritte **(g)** 20 Entpackschritte **(h)** 50 Entpackschritte

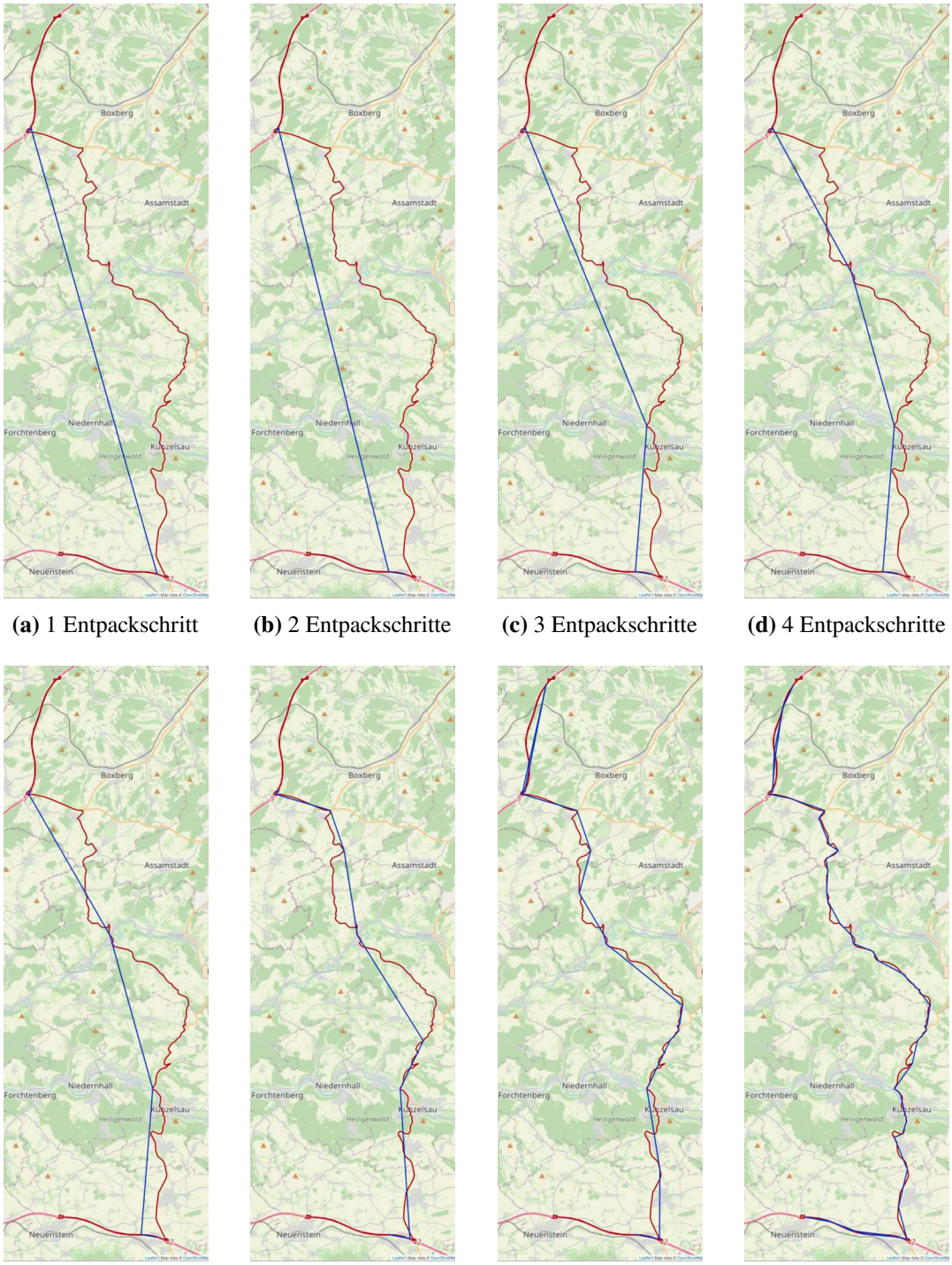
Abbildung 8.16: Beispielhafter Shortcut für verschiedene Anzahlen von Entpackschritten mit Metrik 1 (Fréchet) und Modus 0 (Größter Fehler)



(a) 1 Entpackschritt **(b)** 2 Entpackschritte **(c)** 3 Entpackschritte **(d)** 4 Entpackschritte

(e) 5 Entpackschritte **(f)** 10 Entpackschritte **(g)** 20 Entpackschritte **(h)** 50 Entpackschritte

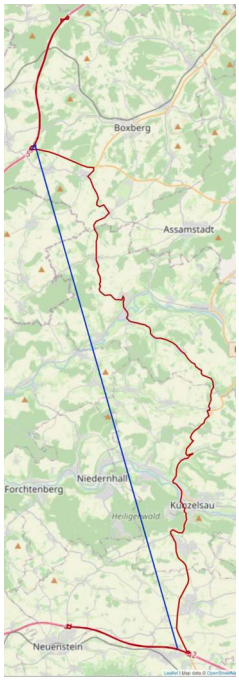
Abbildung 8.17: Beispielhafter Shortcut für verschiedene Anzahlen von Entpackschritten mit Metrik 2 (Flächeninhalt) und Modus 0 (Größter Fehler)



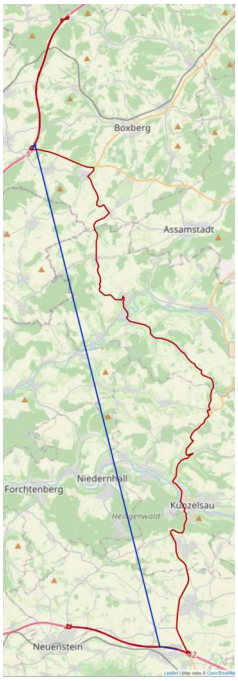
(a) 1 Entpackschritt **(b)** 2 Entpackschritte **(c)** 3 Entpackschritte **(d)** 4 Entpackschritte

(e) 5 Entpackschritte **(f)** 10 Entpackschritte **(g)** 20 Entpackschritte **(h)** 50 Entpackschritte

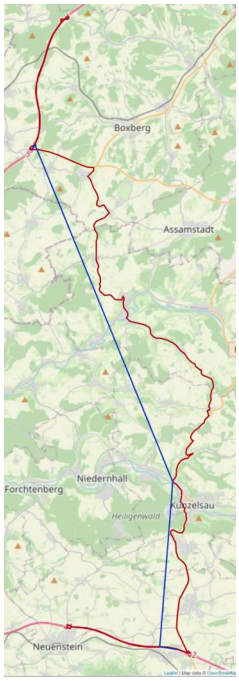
Abbildung 8.18: Beispielhafter Shortcut für verschiedene Anzahlen von Entpackschritten mit Metrik 3 (Kosten) und Modus 0 (Größter Fehler)



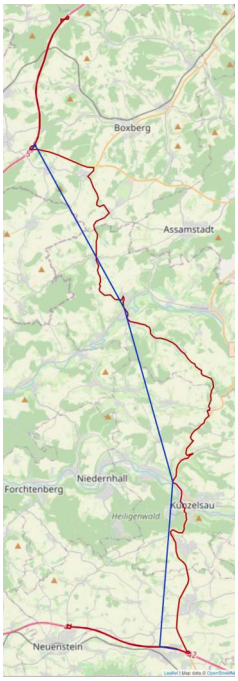
(a) 1 Entpackschritt



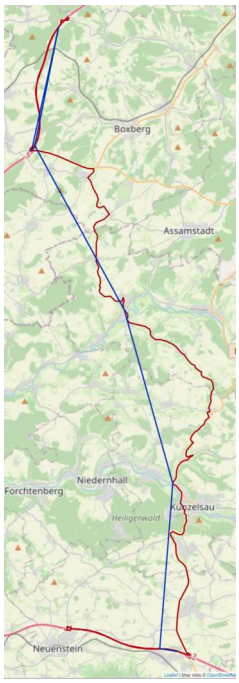
(b) 2 Entpackschritte



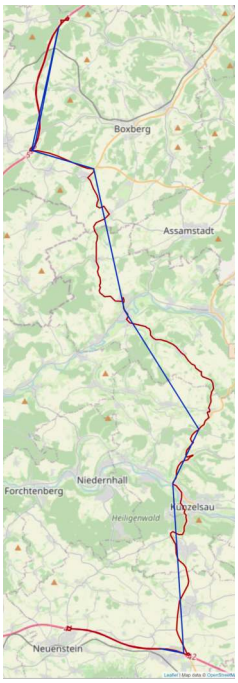
(c) 3 Entpackschritte



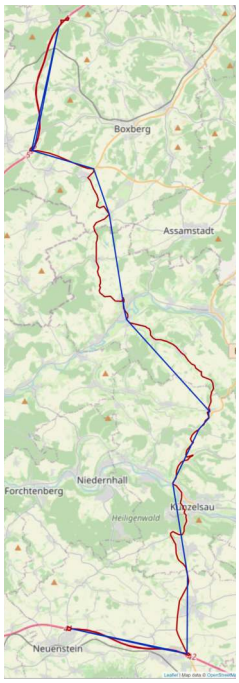
(d) 4 Entpackschritte



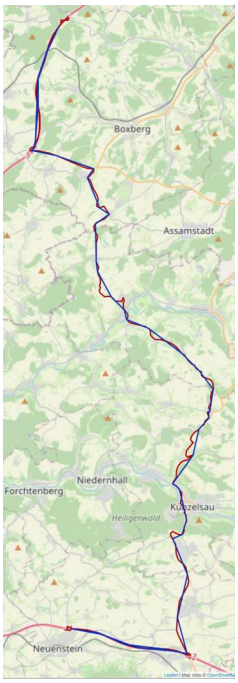
(e) 5 Entpackschritte



(f) 10 Entpackschritte



(g) 20 Entpackschritte



(h) 50 Entpackschritte

Abbildung 8.19: Beispielhafter Shortcut für verschiedene Anzahlen von Entpackschritten mit Metrik 4 (Distanz) und Modus 0 (Größter Fehler)

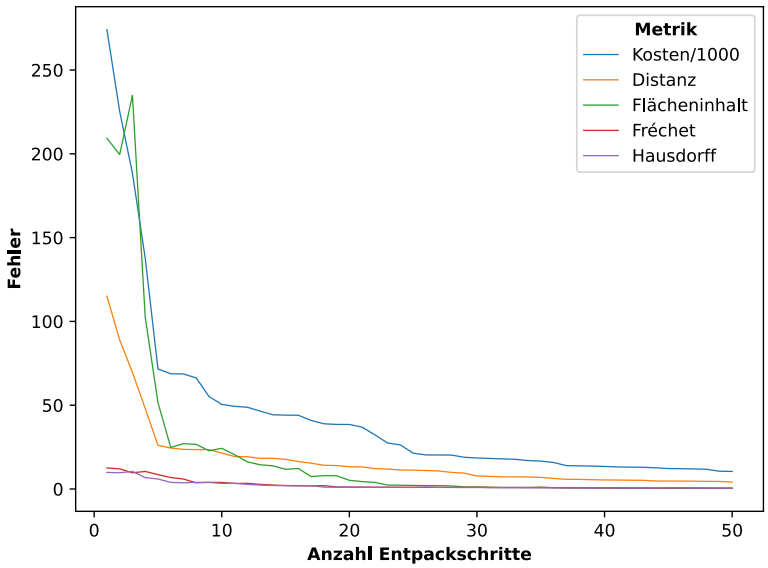


Abbildung 8.20: Fehler der jeweils aktuell entpackten Kante bis zur 50. Entpackung des beispielhaften Shortcuts für Modus 0 (Größter Fehler)

9 Zusammenfassung und Ausblick

Dieses letzte Kapitel enthält eine Zusammenfassung der wichtigsten Ergebnisse dieser Arbeit. Außerdem wird ein Ausblick über mögliche zukünftige Forschung und Erweiterungen dieser Arbeit gegeben.

9.1 Zusammenfassung

Zu Beginn dieser Arbeit wurden die zum Verständnis notwendigen Grundlagen beschrieben. Diese umfassen sowohl die geografischen und technologischen Grundlagen als auch die Definitionen von Graphen und CHs. Im Anschluss wurde das Format der Eingabedateien und eine Abbildung der enthaltenen Informationen auf eine eigene Graphdatenstruktur erläutert. Unter Zuhilfenahme von dieser Graphdatenstruktur wurde anschließend ein Algorithmus entworfen, welcher die optimalen Entpackreihenfolgen für verschiedene Kombinationen von Fehlermetriken und Modi vorberechnen und in eine Datei schreiben kann. Nachfolgend wurde ebenfalls ein Algorithmus konzipiert, welcher einen einzelnen Shortcut oder eine Menge an Shortcuts für eine gegebene Anzahl von Schritten entsprechend einer zuvor aus einer Datei eingelesenen Entpackreihenfolge entpackt. Die nach dem Entpackprozess resultierenden Kanten können als GL-Datei mit einem OpenGL-Viewer oder über eine API mit dem selbst entwickelten Frontend dargestellt werden. Nach Abschluss der Entwicklung wurden alle in dieser Arbeit entworfenen oder genutzten Komponenten in einem Komponentendiagramm aufgezeigt und deren Rollen und Interaktionen beschrieben. Schlussendlich wurden auf einem Testsystem für unterschiedliche Testgraphen verschiedenste Dauern, Größen und Anzahlen bei den Vorberechnungen und bei Echtzeitanfragen gemessen und der entstehende visuelle Eindruck anhand von Darstellungsbeispielen verglichen.

Die Analysen zeigen hierbei, dass die Zeiten für die Vorberechnungen der verschiedensten Entpackreihenfolgen in zwei Kategorien unterteilt werden können. Die rechenintensiveren Metrik-Modus-Kombinationen entpacken eher gleichmäßig und die Detailliertheit der visualisierten Kanten ist sehr ähnlich. Daher liefern diese Entpackreihenfolgen eher visuell ansprechende Darstellungen, da der gesamte visualisierte Pfad am ehesten dem Originalpfad entspricht. Im Gegensatz dazu resultieren die rechenärmeren Metrik-Modus-Kombinationen eher in Entpackreihenfolgen, welche sich auf einen Teilpfad konzentrieren und diesen bereits sehr weit entpacken. Ein Teilpfad wird daher schon sehr detailliert dargestellt, während der andere Teil noch sehr grob approximiert dargestellt wird.

Es lässt sich außerdem erkennen, dass derjenige Modus, in welchem immer der Shortcut mit dem derzeit größten Fehler als Nächstes entpackt wird, über alle Metriken den besten visuellen Eindruck ermöglicht. Während bei gleichem Modus zwar Unterschiede zwischen den einzelnen Metriken erkennbar sind, so ist jedoch nicht entscheidbar, welche Metrik den besten visuellen Eindruck liefert.

Die Auswertung zeigt zusätzlich, dass mit Blick auf die akzeptablen Vorberechnungsdauern und Größen der entstandenen Dateien, deren Einlesedauern und der Entpackdauern, Echtzeitanfragen mit dem in dieser Arbeit entwickelten Vorgehen durchaus möglich sind.

9.2 Ausblick

Während in dieser Arbeit bereits einige Metrik-Modus-Kombinationen betrachtet wurden, sind noch weitere Fehlermetriken und Modi denkbar. Interessant wäre, Metriken basierend auf Metadaten des Graphen zu definieren oder Kombinationen von bereits existierenden Metriken zu nutzen. Ein Metadatum könnte beispielsweise die erlaubte Höchstgeschwindigkeit auf Straßen sein, wonach Straßen mit höheren Höchstgeschwindigkeiten weniger detailliert dargestellt werden könnten. Des Weiteren wären auch Modi realisierbar, welche nicht nur die aktuellen und nächsten Fehler, sondern alle bis zur vollständigen Entpackung eines Shortcuts auftretenden Fehler betrachten.

Ebenfalls von Interesse wäre die Darstellung von Kanten basierend auf weiterem Kontext, anstelle von lediglich einem gegebenen Kartenausschnitt und gewünschtem Detaillierungsgrad. Ein Beispiel hierfür wäre Lokalität, sodass Kanten in der näheren Umgebung deutlich detaillierter dargestellt werden könnten als weiter entfernte Kanten.

Schlussendlich wäre auch denkbar, sich bereits beim Aufbau der CH und nicht erst beim Entpacken an einer visuell ansprechenden Darstellung zu orientieren.

Literaturverzeichnis

[Agaa] V. Agafonkin. *An open-source JavaScript library for interactive maps*. URL: https://leafletjs.com/ (zitiert auf S. 28).

[Agab] V. Agafonkin. *Documentation - leaflet - a JavaScript library for interactive maps*. URL: https://leafletjs.com/reference.html (zitiert auf S. 28).

[Agac] V. Agafonkin. *Documentation - leaflet - a JavaScript library for interactive maps*. URL: https://leafletjs.com/reference.html#tilelayer-wms (zitiert auf S. 28).

[Arca] ArcGIS. *ArcGIS server*. URL: https://enterprise.arcgis.com/de/server/latest/publish-services/windows/wmts-services.htm (zitiert auf S. 28).

[Arcb] ArcGIS. *Mercator-projektion*. URL: https://pro.arcgis.com/de/pro-app/latest/help/mapping/properties/mercator.htm (zitiert auf S. 23).

[BDD+16] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, S. Hagen. *The GeoJSON Format*. RFC 7946. Aug. 2016. DOI: 10.17487/RFC7946. URL: https://www.rfc-editor.org/info/rfc7946 (zitiert auf S. 30).

[Cona] G. Contributors. *Geojson*. URL: https://geojson.org/ (zitiert auf S. 30).

[Conb] J. Contributors. *Introducing json*. URL: https://www.json.org/json-en.html (zitiert auf S. 30).

[Conc] O. Contributors. URL: https://www.openstreetmap.org/copyright (zitiert auf S. 25).

[Cond] W. Contributors. *File:latitude and longitude of the Earth.svg*. URL: https://commons.wikimedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg (zitiert auf S. 22).

[Con22a] O. Contributors. *About openstreetmap*. 2022. URL: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap (zitiert auf S. 25).

[Con22b] O. Contributors. *Main page*. 2022. URL: https://wiki.openstreetmap.org/ (zitiert auf S. 25).

[Con23a] O. Contributors. *Elements*. 2023. URL: https://wiki.openstreetmap.org/wiki/Elements (zitiert auf S. 25).

[Con23b] O. Contributors. *Node*. 2023. URL: https://wiki.openstreetmap.org/wiki/Node (zitiert auf S. 25).

[Con23c] O. Contributors. *Way*. 2023. URL: https://wiki.openstreetmap.org/wiki/Way (zitiert auf S. 25).

[Con23d] W. Contributors. *Shoelace formula*. Nov. 2023. URL: https://en.wikipedia.org/wiki/Shoelace_formula (zitiert auf S. 43).

[EC17] W. Emery, A. Camps. „Chapter 7 - Orbital Mechanics, Image Navigation, and Cartographic Projections“. In: *Introduction to Satellite Remote Sensing*. Hrsg. von W. Emery, A. Camps. Elsevier, 2017, S. 565–596. ISBN: 978-0-12-809254-5. DOI: https://doi.org/10.1016/B978-0-12-809254-5.00007-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128092545000075 (zitiert auf S. 23).

[Erw] J. Erwerle. *Efficient Preprocessing for Unified Route Planning and Graph Rendering* (zitiert auf S. 35).

[FMI22] FMI. *Useful Stuff*. März 2022. URL: https://fmi.uni-stuttgart.de/alg/research/stuff/ (zitiert auf S. 34).

[FSS17] S. Funke, N. Schnelle, S. Storandt. „Uran: A unified data structure for rendering and Navigation“. In: *Web and Wireless Geographical Information Systems* (2017), S. 66–82. DOI: 10.1007/978-3-319-55998-8_5 (zitiert auf S. 17, 27, 33, 53).

[GB] N. Grégoire, M. Bouillot. *Hausdorff distance between convex polygons*. URL: https://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html (zitiert auf S. 40).

[Geo] Geofabrik. *Tiles*. URL: https://www.geofabrik.de/de/maps/tiles.html (zitiert auf S. 28).

[Gim] M. Gimond. *Intro to GIS and Spatial Analysis*. URL: https://mgimond.github.io/Spatial/chp09_0.html (zitiert auf S. 21, 23).

[GSSD] R. Geisberger, P. Sanders, D. Schultes, D. Delling. „Contraction hierarchies: Faster and simpler hierarchical routing in Road Networks“. In: *Experimental Algorithms* (), S. 319–333. DOI: 10.1007/978-3-540-68552-4_24 (zitiert auf S. 27).

[GW14] GWG, WGSG. *Department of Defense World Geodetic System 1984: Its Definition and Relationships with Local Geodetic Systems*. 2014. URL: https://nsgreg.nga.mil/doc/view?i=4085&month=11&day=3&year=2021 (zitiert auf S. 22).

[IBMa] IBM. *Geographic coordinate system*. URL: https://www.ibm.com/docs/en/informix-servers/14.10?topic=data-geographic-coordinate-system (zitiert auf S. 21).

[IBMb] IBM. *Projected Coordinate System*. URL: https://www.ibm.com/docs/en/informix-servers/14.10?topic=data-projected-coordinate-system (zitiert auf S. 23).

[Jor22] Jorgen. *Mercator projections, a comparison*. Nov. 2022. URL: https://blog.studiobluepplanet.net/archives/972 (zitiert auf S. 24).

[Kri21] M. Krishnan. *Computing the discrete Fréchet distance using dynamic programming*. Jan. 2021. URL: https://muthu.co/computing-the-discrete-frechet-distance-using-dynamic-programming/ (zitiert auf S. 41, 42).

[Mer] P. Mercator. *File:latitude and longitude graticule on a sphere.svg*. URL: https://commons.wikimedia.org/wiki/File:Latitude_and_longitude_graticule_on_a_sphere.svg (zitiert auf S. 22).

[Ora20] Oracle. *Class Point2D*. Juni 2020. URL: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/geom/Point2D.html (zitiert auf S. 39).

[Ora23] Oracle. *Class Line2D*. Okt. 2023. URL: https://docs.oracle.com/javase/8/docs/api/java/awt/geom/Line2D.html (zitiert auf S. 39).

[Pag] J. Page. *Semi-major / semi-minor axis of an ellipse*. URL: https://www.mathopenref.com/ellipsesemiaxes.html (zitiert auf S. 22).

[Räc19] H. Räcke. *Fortgeschrittene Datenstrukturen*. 2019. URL: https://www14.in.tum.de/lehre/2019SS/ad/split/sec-Fortgeschrittene-Datenstrukturen-handout.pdf (zitiert auf S. 25).

[Roh] L. H. Rohwedder. URL: https://commons.wikimedia.org/wiki/File:Normal_Mercator_map_85deg.jpg (zitiert auf S. 24).

[Sei23] S. Seidel. *Infoblatt Kartenprojektionen*. 2023. URL: https://www.klett.de/alias/1037819#:~:text=Zylinderprojektionen%20k%C3%B6nnen%20fl%C3%A4chen%2D%20oder%20winkeltreu,Zylindermantel%20den%20Erdk%C3%B6rper%20am%20%C3%84quatorkreis (zitiert auf S. 23).

[SHi] N. Schnelle, C. Haag, invor. *Invor/simplestgraphrendering: A very simple opengl renderer for road-networks. mostly self-contained, but still needs GLFW and Glew.* URL: https://github.com/invor/simplestGraphRendering (zitiert auf S. 29).

[Ste] T. Steinfeld. *Polygone*. URL: https://mathepedia.de/Polygone.html (zitiert auf S. 25).

[Stua] Studyflix. *Euklidische distanz • definition und berechnung*. URL: https://studyflix.de/mathematik/euklidische-distanz-1972 (zitiert auf S. 40).

[Stub] Studyflix. *Gradmaß und Bogenmaß Berechnen • einfach erklärt*. URL: https://studyflix.de/mathematik/gradmass-und-bogenmass-berechnen-2644 (zitiert auf S. 23).

[Wei23a] E. Weisstein. *Line-line intersection*. 2023. URL: https://mathworld.wolfram.com/Line-LineIntersection.html (zitiert auf S. 44).

[Wei23b] E. Weisstein. *Mercator projection*. 2023. URL: https://mathworld.wolfram.com/MercatorProjection.html (zitiert auf S. 24).

Alle URLs wurden zuletzt am 17. 12. 2023 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