Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 51

Contraction Hierarchies für Kontinuierliche Graphsimplifizierung mit Qualitätsgarantien

Tobias Rupp

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Beginn am:	28. Juli 2015
Beendet am:	27. Januar 2016
CR-Nummer:	H.5.4

Kurzfassung

Moderne Navigationsdienste können kürzeste Pfade berechnen und diese dann auf Straßenkarten anzeigen. Als zugrunde liegende Datenstruktur für beide Aufgaben kann eine Contraction Hierarchy verwendet werden.
Ursprünglich waren Contraction Hierarchies dazu konzipiert, die Suche nach kürzesten Pfaden zu beschleunigen. In dieser Arbeit wurde untersucht, wie sich Contraction Hierarchies aufbauen lassen, sodass sie sich besser für kontinuierlich vereinfachte Darstellungen eignen. Dazu sollten vor allem die groben Straßenverläufe erhalten bleiben und topologische Inkonsistenzen wie Überschneidungen vermieden werden.

Diese Anforderungen wurden formalisiert und in heuristischen Vereinfachungsalgorithmen zum Aufbau von Contraction Hierarchies umgesetzt. Für kleine Eingaben wurden mithilfe ganzzahliger linearer Programme garantiert optimale Lösungen berechnet.

Damit konnten in empirischen Vergleichen auf dem Deutschlandgraphen Qualitätsgewinne nahe dem Optimum für vereinfachte Darstellungen von Contraction Hierarchies nachgewiesen werden. Außerdem mussten keine längeren Berechnungszeiten für kürzeste Pfade hingenommen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

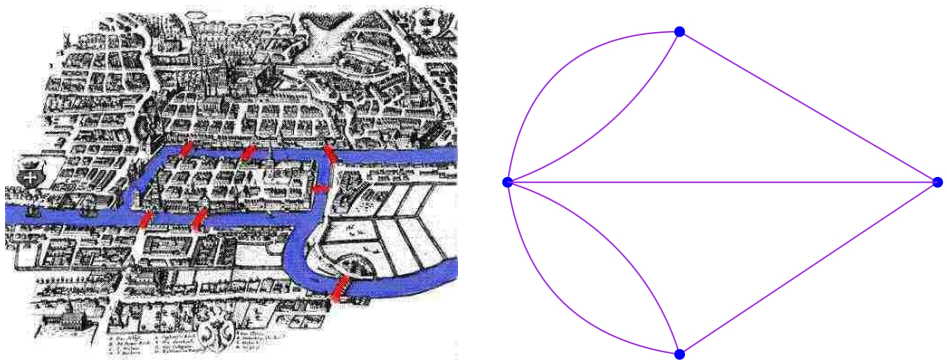
Verzeichnis der Algorithmen

1. Einleitung

Seit es bewegliche Lebewesen gibt, wurden diese vor die Herausforderung gestellt, sich in bekannten oder unbekannten Umgebungen zurechtzufinden. Im Tierreich hat die Evolution verschiedenste Lösungen wie beispielsweise das Legen von Duftspuren für Ameisenstraßen hervorgebracht.

Ähnlich wie Ameisen haben auch wir Menschen von Trampelpfaden bis zu Autobahnen Straßen und Möglichkeiten hervorgebracht, uns über dieselben auszutauschen. Da bei Menschen die visuelle Wahrnehmung stark ausgeprägt ist, wurden schon mindestens seit der Antike visuelle Darstellungen in Form von Landkarten angefertigt. Heutige computergestützte Navigationssysteme stellen Landkarten nicht nur interaktiv dar, sondern können uns außerdem das Finden von kürzesten Pfaden abnehmen.

Um die mathematischen Grundlagen dieser Systeme zu verstehen, gehen wir zum Anfang des 18. Jahrhunderts zurück. Damals war Leonhard Euler vor die Aufgabe gestellt, das Königsberger Brückenproblem aus Abbildung 1.1 zu lösen. Um einerseits alle wichtigen Information zur Lösung des Problems zu berücksichtigen und andererseits die unwichtigen für eine saubere mathematische Analyse wegzulassen, erstellte er eine abstrakte Karte von Königsberg. Mit dieser Karte konnte er mathematisch beweisen, dass es keinen Weg gab, der über alle Brücken von Königsberg genau einmal führte und etablierte damit die Graphentheorie [Pao11].



(a) Köngisberg zu Eulers Zeit mit den rot hervorgehobenen Brücken (Quelle: [Pao11])

(b) Repräsentation als abstrakter Graph (Quelle: [Sch15])

Abbildung 1.1.: Königsberger Brückenproblem: Existiert ein Weg, der über alle Brücken genau einmal führt?

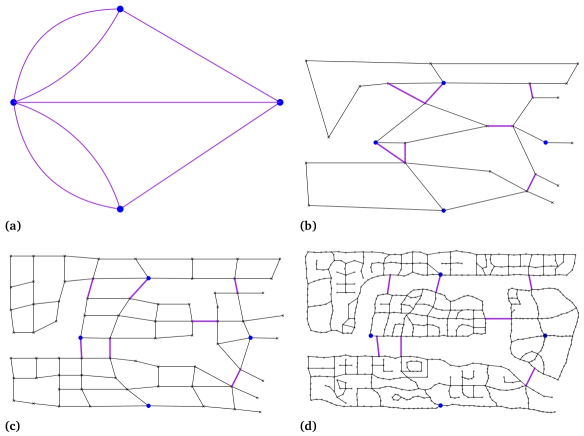


Abbildung 1.2.: Graphen von Königsberg (Quelle: [Sch15])
Wir können Königsberg neben Eulers einfachem Graphen aus (a) auch in unterschiedlich detaillierten Graphen darstellen, wobei (d) das komplette Straßennetz darstellt.

Solche abstrakten Graphen können verschiedenste Strukturen wie z.B. Stromnetzwerke oder Arbeitsablaufpläne modellieren. Im Prinzip sind Graphen nur Mengen von Knoten und Kanten, wobei Knoten beliebige Objekte darstellen können, die durch Kanten untereinander verbunden sind. Im Kontext von Navigationsanwendungen werden Straßennetzwerke ähnlich wie in Eulers Ansatz modelliert. Die Knoten entsprechen dabei geografischen Punkten und die Kanten den Straßenabschnitten, die diese verbinden.

Die Suche von kürzesten Pfaden ist in diesen Graphen formalisierbar und kann durch Computer algorithmisch umgesetzt werden. Wenn die Knoten mit geografischen Koordinaten gespeichert sind, können wir Graphen in die Ebene projizieren und als Landkarten verwenden. Beispielsweise könnte Abbildung 1.2 (d) als Karte von Königsberg verwendet werden. Selbst die weniger detaillierten Graphen (c), (b) bis hin zu (a) könnten im Rahmen einer größeren Karte von z.B. ganz Preußen die groben Strukturen der Stadt Königsberg skizzieren. Solche Landkarten mit kleineren Maßstäben profitieren sogar von vereinfachten Darstellungen, da die Einzeichnung aller Details sie unübersichtlich machen würde. Deswegen beschäftigt sich die Kartografie auch seit ihren Anfängen mit der Frage, welche Strukturen für solche Karten wichtig genug sind, um eingezeichnet zu werden.

Ähnlich wie Menschen sich auf Karten mit weniger Details schneller orientieren können, können Algorithmen vereinfachte Graphen schneller analysieren und kürzeste Pfade finden. Trotz dieser Gemeinsamkeiten ist es für heutige Navigationssysteme normal, die Graphen für Darstellung und Analyse getrennt zu speichern, um die Lösung der Einzelaufgaben besser optimieren zu können. Für die Analyse ist es beispielsweise am einfachsten, wenn lange Straßen zwischen zwei Kreuzungen jeweils nur als eine Kante dargestellt werden.
Um sich als Mensch in einem angezeigten Straßengraphen orientieren zu können, ist jedoch eine lange Straße mit ihrem charakteristischen Formverlauf ein leicht wiedererkennbares

Merkmal. Für eine vereinfachte Darstellung sollte diese Straße also durch so viele Kanten repräsentiert werden, das ihr Formverlauf grob erhalten bleibt.
Durch die getrennte Vereinfachung der Graphen, kann es außerdem zu Inkonsistenzen zwischen der analytischen Suche nach einem kürzesten Pfad und seiner Darstellung führen. Weiterhin wird dabei viel Speicherplatz bzw. bei Client-Server-Anwendungen eine höhere Datenübertragungsrate benötigt, die sich mit einer kompakteren Datenstruktur wesentlich reduzieren ließen.

1.1. Motivation

Vermutlich ist die Diskrepanz zwischen Darstellung und Analyse in praktischen Anwendungssystemen bis auf Ausnahmen wie in [AS01; Sch15], ein Stück weit auch der theoretischen Forschung geschuldet. Dort werden klassischerweise die Gebiete getrennt behandelt, um jeweils noch minimale Verbesserungen zu erzielen und nachzuweisen. Die Forschung an Kompromissen zwischen Darstellung und Analyse erscheint nicht attraktiv, da es zu erwarten ist, dass die Ergebnisse hinter denen der einzelnen Teildisziplinen zurückbleiben.
Aber gerade für die Praxis hat das Ausloten solcher Kompromisse hohe Priorität.

Weiterhin sollten heutige interaktiven Karten im Gegensatz zu statischen herkömmlichen Landkarten beliebige Maßstäbe annehmen können. Wiederrum beschäftigt sich die meiste Forschung zu vereinfachten Darstellungen speziell mit Erstellen von Letzteren, wobei flexiblere Ansätze [Van01; MKVV06] in der Minderheit sind. Bei der Realisierung von interaktiven Karten wird deswegen meist auf statische zurückgegriffen. Da diese Darstellungen durch Redundanzen noch mehr Speicherplatz brauchen und der Maßstab nicht beliebig fein gewählt werden kann, besteht Bedarf an einer kontinuierlichen Vereinfachung.

Als Datenstruktur für kontinuierliche Vereinfachungen werden wir eine Contraction Hierarchy (CH) verwenden, die alle benötigten Graphen kompakt enthält. Normalerweise werden CHs konstruiert um Anfragen nach kürzesten Pfaden effizienter zu beantworten. Für diesen Zweck optimierte CHs wurden in [Sch15] schon für die gleichzeitige Verwendung für Darstellung und Suche nach kürzesten Pfaden verwendet. Dort wurden vereinfachte Graphen aus der CH entnommen, die aber zu viele Mängel als Darstellung hatten, um direkt angezeigt zu werden. Zur Laufzeit der Anwendung mussten deshalb Anpassungen vorgenommen werden, um diese Mängel zu kompensieren.

Um sich als Darstellung möglichst gut zu eignen, sollte ein vereinfachter Graph neben der Formerhaltung von Straßen keine Überschneidungen einführen, die im Ursprungsgraph nicht vorhanden sind. Außerdem sollten parallele Straßenabschnitte wie die zwei Richtungsfahrbahnen einer Autobahn auch nach deren Vereinfachung als parallel zu erkennen sein.

Wir werden in dieser Arbeit untersuchen, wie sich in der Vorverarbeitungszeit CHs aufbauen lassen, sodass diese Mängel minimiert werden. Damit können qualitativ höherwertige Dar-

stellungen in einem Anwendungsprogramm gezeigt werden, bzw. wird zur Laufzeit weniger Anpassung nötig.

Wir werden die obigen Mängel in formale Fehlerkriterien umsetzen. Zur konkreten Konstruktion einer CH werden wir unterschiedliche Algorithmen mit Heuristiken vorstellen. Wir erstellen außerdem ganzzahlige lineare Programme (ILPs), die für kleine Eingaben optimale Lösung finden können. Vereinfachungen wurden auch schon in [CvDH14; FMM+; Mil] durch ILPs modelliert.
Nach der Konstruktion einer CH werden wir ihre Qualität als kontinuierliche Vereinfachung empirisch und mit einem Vergleich zu optimalen Lösungen bewerten.

Da wegen der veränderten Zielsetzung Einbußen bei der Berechnungszeit von kürzesten Pfaden zu erwarten sind, werden wir auch diese messen und gegen die Vorteile bei der Darstellung abwägen.

1.2. Kurzüberblick

Kapitel 1: Das momentane Einleitungskapitel.

Kapitel 2: Hier wird die algorithmische Suche nach kürzesten Pfaden und die grundlegende Datenstruktur der Contraction Hierarchies erklärt. Weiterhin wird ein Überblick über bisherige Ansätze für vereinfachte kartografische Darstellungen gegeben und der Bezug zu unserer Vorgehensweise hergestellt.

Kapitel 3: Es werden die gewünschten Eigenschaften für kontinuierliche Vereinfachungen von einzelnen Straßen formalisiert und für diese heuristische Algorithmen entwickelt.

Kapitel 4: Dort wird die Vereinfachung von Straßen mit ganzzahligen linearen Programmen modelliert, mit denen optimale Lösungen gefunden werden können.

Kapitel 5: Hier werden die konkrete Anzeige eines Graphen und ganzheitliche Aspekte der Vereinfachung besprochen.

Kapitel 6: Es erfolgt die empirische Auswertung unserer heuristisch gewonnen Vereinfachungen. Dazu wird unter anderem ein Vergleich mit optimalen Lösungen vorgenommen.

Kapitel 7: Fasst diese Arbeit zusammen und schließt mit einem Ausblick auf weitere Forschung.

2. Grundlagen

2.1. Kürzeste Pfade

Bisher hierhin haben wir eine intuitive Vorstellung von Graphen benutzt, um die Aufgabenstellung grob darzustellen. Nun definieren wir sie als mathematische Konstrukte, damit wir die gegebenen Aufgaben formal definieren, lösen und unsere Ergebnisse bewerten können.

2.1.1. Graph

Ein gerichteter Graph sei definiert als ein Tupel $G = (V, E)$ wobei V eine Menge aus Knoten und $E \subseteq V \times V$ eine Menge von Kanten ist. Im Allgemeinen können beide Menge unendlich sein, aber wir werden hier nur mit endlichen arbeiten.

Um Straßennetzwerke für Navigationsanwendungen zu modellieren, sind wir außerdem an Kosten für Pfade interessiert. Sei also die Kostenfunktion $cost : E \to \mathbb{R}$ definiert, die jeder Kante Reisekosten zuweist. Wir werden im Weiteren von der Annahme $\forall e \in E : cost(e) \geq 0$ ausgehen. In den von uns untersuchten und den allermeisten anderen Straßennetzwerken ist dies gegeben und ist auch eine notwendige Bedingung für das Einsetzen des Dijkstra-Algorithmus und viele seiner Varianten. In manchen Fällen wie z.B. dem Modellieren von Kantenkosten für Elektroautos, die bergab ihre Batterien aufladen können, kann dies durch Erhöhen der Kantengewichte kompensiert werden, wenn keine negativen Zyklen auftreten können [EFS11].

Außerdem definieren wir einen Pfad der Länge n von einem Startpunkt s zu einem Zielknoten t als eine Aneinanderreihung von Knoten $p_{s,t} = v_1 ... v_n$ mit $v_i \in V$, $v_1 = s$, $v_n = t$ und $\forall i \in \{1 \dots n-1\} : (v_i, v_{i+1}) \in E$. Damit haben Pfade alternativ eine Repräsentation als Aneinanderreihung von Kanten: $\hat{p}_{s,t} = e_1 e_2 ... e_{n-1} = (v_1, v_2)(v_2, v_3) ... (v_{n-1}, v_n)$. Die Kostenfunktion überträgt sich nun auf Pfade: $cost(p) = cost(\hat{p}) = \sum_{e \in \hat{p}} cost(e)$

2.1.2. Kürzester Pfad

Sei nun ein Graph G und eine zugehörige Kostenfunktion $cost$ gegeben, dann ist ein kürzester Pfad p^* von einem Startknoten s zu einem Zielknoten t der Pfad p, welcher $cost(p_{s,t})$ minimiert. Falls mindestens ein Pfad von s nach t existiert, so auch p^*, da keine negative

Zyklen vorkommen und Zyklen entfernt werden können um einen kürzeren Pfad zu erhalten. Die Eindeutigkeit ist nicht gegeben, da mehrere Pfade die gleichen Minimalkosten haben können. In diesem Zusammenhang definieren wir die Distanz des kürzesten Wegs zwischen zwei Knoten d: $V \times V \to \mathbb{R}^+$ als

$$d(u,v) = \begin{cases} cost(p^*_{u,v}), & \text{falls ein Pfad von s nach t existiert.} \\ \infty, & \text{sonst.} \end{cases} \tag{2.1}$$

Es sei angemerkt, dass die Bezeichnung „kürzest“ irreführend sein kann. In den untersuchten Straßennetzwerken werden die Gewichte von Kanten nicht die Luftliniendistanz zwischen ihren Endpunkten in räumlichen Maßeinheiten sein, sondern die zeitlich benötigten Kosten auf dieser Kante zu reisen.

2.1.3. Dijkstra-Algorithmus

Der Standardalgorithmus um den kürzesten Pfad in Graphen mit nicht-negativen Kantengewichte zu finden ist der Dijkstra-Algorithmus [Dij59]. Bei seiner Ausführung erstellt er nach und nach eine Knotenmenge B aller bisher erreichten Knoten mit den dazugehörigen Kosten, diese zu erreichen. Zu Beginn ist nur der Startknoten s mit Kosten 0 in dieser Menge. Aus der Menge der von B aus erreichbaren und unbearbeiteten Knoten Q wird schrittweise immer der nächste Knoten v hinzugefügt. Dieser aktive Knoten v minimiert die Distanz zu s also $v = \text{argmin}_x\{d(s,v)|v \in Q\}$. Beim Hinzufügen von v in B können seine Nachbarn in Q von s aus evtl. billiger erreichbar werden. Jeder Knoten v wird nur einmal aktiv und seine Distanz von s ist für den Rest des Algorithmus festgelegt, wir nennen den Knoten nach seiner Aktivphase *bearbeitet*. Ein Knoten v wird nur einmal bearbeitet, weil alle anderen Knoten in Q höhere Kosten haben als v und ohne negative Kantengewichte kein kürzerer Pfad über diese zu v führen könnte.

Die wesentliche Komplexität liegt darin, die Menge Q so zu verwalten, dass der nächste erreichbare Knoten effizient ermittelt werden kann. Zu diesem Zweck wird Q meist als eine Prioritätswarteschlange bzw. Heap organisiert. Bisher unerreichbare Knoten werden mit den Kosten ∞ gekennzeichnet.

Um später einen kürzesten Pfad extrahieren zu können, wird für jeden Knoten v sein Vorgänger gespeichert, d.h. der Knoten über den v seine endgültigen Kosten erhalten hat. Falls der Zielknoten erreicht wurde, kann von dort aus rückwärts ein kürzester Pfad rekonstruiert werden.

Klassischerweise werden die Knoten aus Q jeweils einmal in einem Heap gespeichert und mittels Referenzen ihre Kosten geändert und danach die Ordnung des Heaps wiederhergestellt. Unsere verwendete Implementierung und der Algorithmus 2.1 arbeitet stattdessen mit „faulen“ Aktualisierungen einer Prioritätswarteschlange. Dabei wird statt der Aktualisierung eines Knoten einfach dieser Knoten mit den neuen Kosten ein weiteres Mal als Eintrag in die

Algorithmus 2.1 Dijkstra

```
dijkstra (V, E, cost, s, t) {
        Q = PriorityQueue();
        d = initializeArrayWith ( infinity , V.size() );
        predecessor = initializeArrayWith (None, V.size() );
        d[s] = 0;
        Q.push (s, d[s]);

        while (!Q.empty()) {
                active , priority = Q.extractMin();
                // Ziel erreicht ?
                if (active == t)
                        return backtrack(predecessor, s, t);

                // ignoriere schon bearbeitete Knoten
                if ( priority > d[active ])
                        continue;

                for (Edge (active , v): outEdges(active , E)) {
                        tempDist = d[active ] + cost(active , v);
                        if (tempDist < d[v]) {
                                d[v] = tempDist;
                                predecessor[v] = active ;
                                Q.push(v, tempDist);
                        }
                }
        }
        return NULL;
}

backtrack (predecessor, s, t) {
        path = {t};
        current_predecessor = predecessor[t];
        while (current_predecessor != None) {
                path.push_front( current_predecessor );
                current_predecessor = predecessor[ current_predecessor ];
        }
        return path;
}
```

Warteschlange eingefügt. Beim Herausnehmen wird dann einfach ein Eintrag verworfen, wenn dessen Knoten schon bearbeitet worden ist. In der Praxis sind die „faulen“ Aktualisierungen nicht schlechter als die klassischen.

Die schlechteste theoretische Zeitkomplexität für eine klassische Implementierung mit binärem Heap beträgt $O((|V| + |E|) \log |V|)$. Dies rührt daher, dass jede Kante potentiell die Kosten eines Knoten herabsetzen kann. Da der Heap bis zu $|V|$ Knoten enthalten kann, sind bis zu $\log |V|$ Schritte nötig um die Heapordnung nach einer Herabsetzung wiederherzustellen.

Dasselbe gilt für das Herausnehmen eines Knoten.
Mit den „faulen“ Aktualisierungen kann im schlechtesten Fall der Heap bis zu $|V|^2$ Einträge beinhalten und hat in der Theorie somit die schlechtere Laufzeit.
Für bessere theoretische Laufzeiten kann eine Implementierung mit Fibonacci Heaps die Laufzeit auf $O(|E| + |V| \log |V|)$ reduzieren.

2.1.4. Optimierungen

Auch wenn der Dijkstra-Algorithmus auf allgemeinen Graphen den kürzesten Pfad immer findet, so lässt seine Zeitkomplexität auf fast planaren Graphen noch zu wünschen übrig. Da Navigationsanwendungen im alltäglichen Leben enorm an Bedeutung gewonnen haben und deren Datenbasis fast planare Graphen sind, besteht ein hoher Bedarf, für solche Graphen den kürzesten Pfad schneller zu finden.
Der Hauptgrund für verhältnismäßig lange Laufzeiten des Dijkstra-Algorithmus auf realen Straßengraphen ist die fehlende Berücksichtigung deren Beschaffenheit. Für allgemeine Graphen kann der Algorithmus kaum verbessert werden, da immer die Möglichkeit besteht, dass der nächste zu bearbeitende Knoten eine direkte und günstige Verbindung zum Zielknoten hat. In Straßengraphen existieren aber keine solchen „Wurmlöcher“. Der auf alle Eventualitäten vorbereitete Algorithmus baut im Straßengraph einen kreisförmigen Suchraum auf, unabhängig davon in welcher Richtung der Zielknoten legt.
Weiterhin sind reale Straßennetzwerke hierarchisch aufgebaut, sodass viele kürzeste Pfade über Autobahnen verlaufen.

Um diese Eigenschaften auszunutzen kann beispielsweise die Luftlinie mit dem Wissen um Maximalgeschwindigkeiten kombiniert werden, um solche „Wurmlöcher“ auszuschließen. Die bekannte Technik der A^*- Suche [HNR68] benutzt die Luftlinie außerdem als Heuristik um den Suchraum Richtung Zielknoten aufzubauen. Da das Reisen über die Luftlinie mit Maximalgeschwindigkeit die Kosten stets unterschätzt, ist A^* sogar immer korrekt.
Andere Algorithmen versuchen Punkte zu identifizieren, die für viele kürzeste Pfade wichtig sind wie z.B. in [Gut04]. Das derzeit schnellste Verfahren identifiziert sogenannte Transitknoten [BFSS07], die Flaschenhälse zwischen bestimmten Gebieten darstellen und deswegen für die Suche nach kürzesten Pfaden wichtig sind.
Als einfacher Ansatz kann auch die Hierarchie des Straßennetzwerkes berücksichtigen werden und z.B. vom Startpunkt die nächste Autobahnauffahrt gesucht werden und von dort aus die Ausfahrt, die dem Zielpunkt am nächsten liegt. Anders als eine solche starre, nach Straßentypen aufgebauten Hierarchie, sind Contraction Hierarches wie in [GSSD08] zuerst beschrieben sehr flexibel. Unter anderem deshalb bauen eine Reihe von Algorithmen [SFS13; DGNW13] auf ihnen auf. Wie der Titel schon angekündigt hat, wird diese Datenstruktur auch die Grundlage dieser Arbeit sein.

2.2. Contraction Hierarchies

Eine Contraction Hierarchy (CH) ist eine durch Vorverarbeitung gewonnene Datenstruktur, dies es ermöglicht, kürzeste Pfade in Straßengraphen wesentlich schneller zu finden. Darüberhinaus basieren alle Varianten nur auf dem einfachen Konzept der Knotenkontraktion.

2.2.1. Überblick

Wie alle Hierarchien ordnen CHs die Knoten des Graphen in verschiedene Ebenen der Wichtigkeit, wobei möglichst die natürliche Struktur des Straßennetzwerks bezüglich kürzesten Pfaden wiedergegeben werden sollte. Wie genau die Wichtigkeit von einzelnen Straßen oder Knoten zu bemessen ist, wurde in [AFGW10] versucht mittels einer *highway-dimension*-Metrik zu formalisieren.
Wie der Teil „Contraction“ schon andeutet, basieren CHs auf einer Kontraktionsoperation, der Knotenkontraktion. Die Knotenkontraktion ist die Umsetzung der einfachen Idee, dass ein Knoten v aus dem Graphen entfernt wird, wobei kürzeste Pfade zwischen allen anderen Knoten aber noch weiterhin gültig sein sollten. Falls also ein Knoten kontrahiert wird und manche kürzesten Pfade ausschließlich über ihn verliefen, so muss dies durch das Einfügen von Abkürzungskanten kompensiert werden.

Durch schrittweise Kontraktion aller Knoten als Vorverarbeitung eines Graphen $G = (V, E)$ erhalten wir einen überdeckenden Graphen G^+ mit einer erweiterten Knotenmenge $E^+ \supseteq E$ und eine Abbildung der Knoten auf Level $\Phi : V \to \mathbb{N}_0$. Diese Abbildung führt eine partielle Ordnung auf den Knoten ein mit der Eigenschaft, dass für alle kürzesten Pfade $p_{s,t}$ in G ein Vertreterpfad $\dot{p}_{s,t} = v_1 v_2 \dots v_k$ in G^+ existiert und außerdem ein Index $l \in [0, k]$ mit $\forall i \in [0, l] : \Phi(v_i) \leq \Phi(v_{i+1})$ und $\forall i \in [l, k] : \Phi(v_i) \geq \Phi(v_{i+1})$. Anschaulich kann $\dot{p}$ in zwei Pfade zerlegt werden, wobei im ersten das Level der Knoten steigt und im zweiten sinkt. Um einen kürzesten Pfad im Originalgraphen aus $\dot{p}$ zu rekonstruieren, können wir rekursiv seine Abkürzungskanten entpacken, bis nur noch Kanten aus E übrig sind.

2.2.2. Vorverarbeitung

Wir werden hier nicht wie in der originalen Variante die Knoten einzeln nacheinander kontrahieren, sondern in Runden gleichzeitig eine unabhängige Menge von Knoten kontrahieren und diesen die Rundennummer als Level zuweisen.

Kontraktion eines Knoten v:

Die Kontraktion eines Knoten bedeutet sein Entfernen aus dem Graph, sodass kürzeste Pfade zwischen den verblieben Knoten erhalten bleiben. Bei mehreren kürzesten Pfaden mit gleicher Länge erlauben wir, dass einer davon ausreicht.

Offensichtlich sind nur kürzesten Pfade von der Kontraktion von v beeinträchtigt, wenn sie v enthalten. Alle solchen Pfade müssen die Kanten (u, v) und (v, w) enthalten, wobei u und w Nachbarn von v sind. Es genügt also lokal den Knoten v mit seinen eingehenden und ausgehenden Kanten zu betrachten. Um festzustellen ob der Pfad $(u, v), (v, w)$ Teilpfad eines kürzesten Pfades ist, muss nur ein Dijkstra-Algorithmus gestartet werden, dem das Besuchen von v verboten ist und der herausfinden wird, ob der Knoten w von u aus mit den Kosten $cost(u, v) + cost(v, w)$ erreichbar ist. Falls er nur noch Knoten mit höherer Distanz erreichen kann, bricht er ab. So ist gewährleistet, dass sein Suchraum sehr lokal und damit nicht allzu groß ist.
Falls w so nicht erreichbar ist, muss eine Abkürzungskante (u, w) mit $cost(u, w) = cost(u, v) + cost(v, w)$ der Menge E^+ hinzugefügt werden. Um später außerdem auf abgekürzte Kanten zugreifen zu können, definieren wir die Abbildung $unpack((u, w) \rightarrow ((u, v), (v, w))$. Das Untersuchen der Pfade aller Kombinationen von Nachbarn u, w von v liefert die Menge an neuen Abkürzungskanten $E_{shortcuts}(v)$.

Kontraktion des Graphen:

Wir starten mit der Rundennummer $i = 0$ und dem Graph $G_0 = (V_0 = V, E_0 = E)$ und berechnen für jede Runde eine unabhängige Menge $I_i \subseteq V$. Der größte Teil dieser Arbeit beschäftigt sich damit, wie I_i gewählt wird.
Es werden dann alle Knoten $v \in I_i$ kontrahiert und die notwendigen Abkürzungen hinzugefügt. Das Verwenden einer unabhängigen Menge ist notwendig, damit sich die Knoten nicht gegenseitig bei ihrer Kontraktion beeinflussen.

Bei der Kontraktion eines Knoten werden auch die angrenzenden Nachbarn entfernt. Sei dazu $E_{removed}(v) = \{(v', v'') \in E_i | v = v' \vee v = v''\}$. Mit $E_{shortcuts} = \bigcup_{v \in I_i} E_{shortcuts}(v)$ und $E_{removed} = \bigcup_{v \in I_i} E_{removed}(v)$ erhalten wir dann einen Graphen $G_{i+1} = (V_{i+1}, E_{i+1})$ mit $V_{i+1} = V_i \setminus I_i$ und $E_{i+1} = (E_i \setminus E_{removed}) \cup E_{shortcuts}$.

Kontraktion nach Kantendifferenz

Um die Berechnungszeit für spätere Anfragen nach kürzesten Pfade zu reduzieren, ist die Standardstrategie die Kontraktion nach Kantendifferenz. Wir stellen eine mögliche Umsetzung vor:
In einer Runde i wird dabei zunächst eine möglichst große unabhängige Menge $I'_i \subseteq E_i$ gewählt, d.h. es gibt keine unabhängige Menge $I''_i \subseteq E_i$ für die gilt: $I'_i \subset I''_i$. Für alle $v \in I'$ wird die Kontraktion dann simuliert, um $E_{shortcuts}(v)$ zu ermitteln. Es kann dann die Kantendifferenz $\Delta_e(v) = |E_{shortcuts}(v)| - |E_{removed}(v)|$ gebildet werden. Die $v \in I'$ können dann nach $\Delta_e(v)$ sortiert werden und die Hälfte mit den niedrigeren Werten wird I bilden.

2.2.3. Korrektheit

Wir kommen nun auf die Behauptung aus 2.2.1 zurück, dass für jeden kürzesten Pfad in G ein Vertreterpfad in G^+ existiert, der sich in einen Aufwärts- und Abwärtspfad zerlegen lässt. Ein Vertreterpfad von $p_{s,t}$ in G ist ein kürzester Pfad in G^+, falls das Auspacken aller seiner Kanten wieder den Pfad $p_{s,t}$ ergibt.
Wir werden diese Behauptung über Widerspruch beweisen. Angenommen ein Pfad $p_{s,t}$ hätte nur Vertreterpfade, die sich nicht in Aufwärts- und Abwärtspfade zerlegen lassen. Dann muss in einem solchen Vertreterpfad ein *verletzender* Knoten v_l existieren, sodass Vorgänger und Nachfolger höhere Level haben, also $\Phi(v_{l-1}) > \Phi(v_l) < \Phi(v_{l+1})$. Dies bedeutet, dass v_l vor seinen Nachbarn kontrahiert wurde. Da Vertreterpfade kürzeste Pfade sind, muss aber auch der Teilpfad $v_{l-1}v_lv_{l+1}$ ein kürzester sein und deswegen muss bei der Kontraktion von v die Abkürzung (v_{l-1}, v_{l+1}) eingefügt werden. Damit ergibt sich ein Vertreterpfad, der ohne den verletzenden Knoten v_l auskommt. Da die Pfadlänge endlich ist, können wir also nach und nach alle verletzenden Knoten aussparen und so zu einem Vertreterpfad kommen, der sich in einen Aufwärts- und Abwärtspfad zerlegen lässt.

2.2.4. Anwendung

Um die Verwendung von einer CH in Algorithmen zu erläutern verwenden wir folgende Definitionen der Teilgraphen von G^+.

$$
\begin{aligned}
\text{Aufwärtsgraph: } & G^{\uparrow} = (V, E^{\uparrow}), E^{\uparrow} = \{(v,w) \in E^+ | \Phi(v) \leq \Phi(w)\} \\
\text{Abwärtssgraph: } & G^{\downarrow} = (V, E^{\downarrow}), E^{\downarrow} = \{(v,w) \in E^+ | \Phi(v) \geq \Phi(w)\} \\
\text{Aufwärtsgraph von } v\text{: } & G^{\uparrow}(v) = (V^{\uparrow}(v), E^{\uparrow}(v)) \\
& V^{\uparrow}(v) = \{v\} \cup \bigcup_{u \in V^{\uparrow}(v)} \{w | (u,w) \in E^{\uparrow}\} \\
& E^{\uparrow}(v) = \{(u,w) \in E^{\uparrow} | u, w \in V^{\uparrow}(v)\} \\
\text{Abwärtsgraph von } v\text{: } & G^{\downarrow}(v) = (V^{\downarrow}(v), E^{\downarrow}(v)) \\
& V^{\downarrow}(v) = \{v\} \cup \bigcup_{u \in V^{\downarrow}(v)} \{w | (w,u) \in E^{\downarrow}\} \\
& E^{\downarrow}(v) = \{(w,u) \in E^{\downarrow} | u, w \in V^{\downarrow}(v)\}
\end{aligned}
$$

Aufwärts- und Abwärtsgraphen sind einfach die Suchräume in G^+, wenn eine Suche auf Aufwärts- bzw. Abwärtskanten beschränkt wird. Der Aufwärts- bzw. Abwärtsgraphen eines Knoten v lässt sich somit durch eine Breitensuche ermitteln, die sich nur auf Aufwärts- bzw. Abwärtskanten beschränkt.
Durch das Einschränken von Algorithmen auf diese Suchräume lassen sich wesentlich schneller kürzeste Pfade finden.

2.2.5. Bidirektionaler Dijkstra

Diese kompliziertere Variante des Dijkstra-Algorithmus kommt für die Suche nach kürzesten Pfaden in Straßengraphen mit weniger großen Suchräumen und damit weniger Berechnungszeit aus. Für das Finden eines kürzesten Pfades von s nach t wird eine auf $E^{\uparrow}(s)$ beschränkte Vorwärtssuche und eine auf $E^{\downarrow}(t)$ beschränkte Abwärtssuche gestartet. Auch wenn beide Suchen räumlich separat verlaufen, so teilen sich beide doch eine Prioritätswarteschlange, deren Einträge die jeweilige Zugehörigkeit zur Suchrichtung speichern. Die Vorwärtssuche vergibt für die Knoten Distanzwerte $d(s, v)$ von s ausgehend während die Rückwärtssuche $d(v, t)$ zum Ziel t hin vergibt. Wir verwalten außerdem den Knoten v mit der kleinsten Summe der Vorwärts- und Rückwärtsdistanzen $d(s, v) + d(v, t)$, der bisher *bearbeitet* wurde. Dieser Knoten ist der Kandidat für den Knoten mit höchstem Level eines gesuchten Vertreterpfads von s nach t. Es ist wichtig zu beachten, dass die gesetzten Kosten der bisher *bearbeiteten* Knoten keine Distanzen von kürzesten Pfaden in G von s bzw. zu t darstellen, sondern diese evtl. überschätzen. Dies kommt daher, dass in den beiden Suchen jeweils nur aufsteigende bzw. absteigende Kanten untersucht werden. Erst wenn der nächste zu bearbeitende Knoten ein höheres $d(s, v)$ oder $d(v, t)$ aufweist, so ist gesichert, dass keine wir keinen kürzeren Vertreterpfad mehr finden als jenen der über unseren Kandidatenknoten impliziert wird. Damit können wir diesen Vertreterpfad zurückgeben und durch Entpacken zu einem kürzesten Pfad in G von s nach t kommen.

Während der Suche können von einem Knoten $v_{\geq}$ mit überschätzten Kosten Aktualisierungen zu den Kosten anderer Knoten erfolgen, die dann ebenfalls überschätzt sind. Um solche überflüssigen Aktualisierungen zu vermeiden, können von anderen bereits *bearbeiteten* und benachbarten Knoten auch Kanten entgegen der Suchrichtung betrachtet und herausgefunden werden ob über diese der Knoten $v_{\geq}$ schneller erreicht werden kann. Wenn dies der Fall ist kann $v_{\geq}$ „gestallt“ werden, d.h. es werden bei seiner Bearbeitung keine Aktualisierungen der Nachbarknoten ausgeführt. Diese einfache Optimierung führt zu kleineren Prioritätswarteschlangen und damit zu einer Zeitersparnis.

2.3. Kartografie

Nachdem wir die CHs als zugrundeliegende Datenstrukturen für die schnelle Berechnung von kürzesten Pfaden gesehen haben, wollen wir hier näher auf ihre Verwendung als Grundlage für kartografische Darstellungen eingehen.

2.3.1. Allgemein

Grundsätzlich ist die Kartografie die Wissenschaft der Erstellung von thematischen Darstellungen räumlicher Gegebenheiten in der realen Welt. Das Hauptanwendungsgebiet ist das

Erstellen von Landkarten als komprimierte Darstellungen von Teilen der Erdoberfläche.
Grundsätzlich ist das Erstellen einer solchen Karte nicht so einfach formalisierbar wie die Suche nach kürzesten Pfaden. Trotzdem besteht im Gegensatz zu anderen grafischen Darstellungen von Objekten aus der realen Welt, die rein künstlerischen Zweck haben können, für Landkarten immer die Anforderung nützlich im Sinne einer konkreten Anwendung zu sein. Je nach Anwendung kann eine solche Karte z.B. Höhenlinien enthalten oder auch Grade der Vegetation darstellen. Die meisten Karten sind topografisch, d.h. sie versuchen möglichst maßstabsgetreu die reale Geographie abzubilden. Aber es gibt auch stärker thematisierte Karten wie z.B. Metrokarten, bei denen die Maßstabstreue an zweiter Stelle steht.
Allen Karten gemein ist die Tatsache, dass sie nicht die komplette Wirklichkeit abbilden, sondern nur eine vereinfachte Darstellung sind. Die Entscheidung ob ein geografische Merkmal so wichtig anzusehen ist, dass es in einer Karte abgebildet wird oder nicht, ist die Hauptaufgabe beim Kartografieren.
Wir werden uns in dieser Arbeit auf die Darstellungen von Straßennetzwerken unter möglichst vollständiger Beibehaltung der realen Geografie einschränken.

2.3.2. Modellierung

Anders als zur Berechnung von kürzesten Pfade, reicht es nun nicht mehr aus, unsere Graphen als Mengen von Knoten und Kanten zu präsentieren. Um als visuelle Darstellung einem Menschen nützlich sein, sollte ein Graph möglichst einer Landkarte ähneln. Unser Graph ist als Extrakt[1] aus OpenStreetMap(OSM)-Daten bereits in ein Koordinatensystem bezüglich der Erdkugel eingebettet. Jeder Knoten besitzt einen Längen- und Breitengrad. Jedoch können auf typischen Anzeigegeräten nur zweidimensionalen Darstellungen angezeigt werden. Als wesentliches Problem der Kartografie wurden hierfür schon geeignete Projektionen wie z.B. die Mercator-Projektion gefunden. Im Prinzip werden wir aber den Graphen nicht für eine spezielle Projektion bearbeiten, sondern mit dem gegebenem Koordinatensystem arbeiten.

Für eine unkomplizierte Erläuterung der Algorithmen werden wir von den Koordinaten eines Knoten als $(x, y) \in \mathbb{R}$ sprechen. Der Längengrad gibt hier x und der Breitengrad y an. Trotz der Ähnlichkeit zu euklidischen Koordinaten kann hier die Distanz nicht einfach über den Satz des Pythagoras berechnet werden. Zwar gibt es exakte Formeln, um die geodätische Distanz zwischen zwei Punkten zu berechnen, aber da wir nur Distanzen in sehr lokalen Umgebungen untersuchen, werden diese nicht nötig sein. Bei nahe beieinanderliegenden Knoten fällt die Erdkrümmung nicht zu sehr ins Gewicht, sodass wir eine Distanzfunktion $dist : V \times V \to \mathbb{R}^+$ annähernd über den Satz des Pythagoras berechnen können, wenn vorher die Längengrade bezüglich des durchschnittlichen Breitengrads skaliert werden[2].

[1] Software zum Parsen von OSM-Daten: https://github.com/dbahrdt/OsmGraphCreator
[2] http://www.movable-type.co.uk/scripts/latlong.html

Die für diese Arbeit bereitgestellte Visualisierungssoftware [3] projiziert die Knoten auf einem Globus und zeichnet die Kanten als Luftlinien zwischen ihren Knoten ein. Die Knoten werden als eindimensionale Objekte nicht direkt angezeigt, sondern sind nur indirekt über anliegende Kanten zu sehen. Die Kanten werden in einer Farbe und Breite bezüglich ihres Straßentyps und unabhängig von ihrer Richtung ungerichtet eingezeichnet. Bidirektionale Straßen und Einbahnstraßen können also nicht mehr auseinandergehalten werden, wenn ihre Kanten sich die gleichen Knoten teilen.

2.3.3. Graphsimplifizierung

Es ist nun klar, wie der gegebene Ausgangsgraph G darzustellen ist. Falls jedoch ein Anzeigegerät alle Knoten und Kanten darstellen müsste, so wird dies einerseits aufwändig bezüglich von Berechnungsressourcen, die einem typischen Navigationsgerät im Auto nur begrenzt zur Verfügung stehen. Andererseits kann der Benutzer bei einer zu detailreichen Darstellung mit Informationen überfrachtet werden, sodass er länger dazu benötigt, die für ihn wichtigen zu identifizieren. Wegen letzterem sind auch Landkarten mit kleinerem Maßstab weniger genau. Im Sinne von digitalen interaktiven Karten betrachten wir solche als herausgezoomt.

Typischerweise wird in der Kartografie aus der detailreichsten Darstellung der Wirklichkeit (in unserem Fall G), der Datenbasis, eine vereinfachte Karte für eine gewisse Zoomeinstellung generiert. Dieser Prozess wird auch Generalisierung genannt. In diesem Zusammenhang fällt auf, dass die verschiedenen Level einer CH solche generalisierten Darstellungen implizieren. Naiv könnte also einfach der Graph G_i aus einer CH von G angezeigt werden.
Es wurde vielfach versucht die Vorgehensweisen beim Generalisieren abstrakt in Modelle zu fassen [BW88; MS92]. Diese Modelle haben alle das Konzept der Generalisierungsoperatoren gemein.

2.3.4. Generalisierungsoperatoren

Die Grundbausteine zum Generalisieren in der Kartografie sind die sogenannten Generalisierungsoperatoren. Diese Operatoren sind so modelliert, dass sie einer menschlicher Vorgehensweise bei der Vereinfachung von geografischen Karten möglichst nahe kommen. Es werden die wichtigsten Operatoren vorgestellt, die die Forschung bis jetzt zusammengetragen hat und die Anwendbarkeit im Kontext der CHs erläutert.

- Selektierung: Dies bezeichnet das Auswählen von spezifischen Datenpunkten, die für eine vereinfachte Darstellung beibehalten werden. Der gegenteilige Prozess ist das Eliminieren.

[3] https://github.com/invor/simplestGraphRendering

Aufgrund der Beschaffenheit einer CH muss in jeder Runde festgelegt werden, welche Knoten als Nächstes eliminiert werden.

- Simplifizierung: Bezeichnet das Reduzieren der Komplexität von geometrischen Objekten. Dies wird üblicherweise durch eine Selektierung der wichtigsten und Elimination der unwichtigen Punkte eines Objekts bewerkstelligt.
  Wie schon der Name dieser Arbeit andeutet, ist der Simplifizierungsoperator wesentlich. Dies liegt daran, dass die einzige Operation zum Aufbau der CHs die Kontraktionsoperation ist und dieser Operator ihr am nächsten kommt. Wir werden außerdem das Wort „Vereinfachung“ als Synonym für Simplifizierung verwenden.
- Amalgamierung: Bedeutet das Zusammenfügen von lokal beieinander liegenden geometrischen Strukturen zu einer einzelnen Struktur.
  Amalgamierung wird stattfinden durch das Einfügen von Abkürzungen, die bisherige Kanten ersetzen. Darüber hinaus werden auf höheren Leveln der CH mehrere Knoten, die zusammen z.B. eine Ortschaft darstellen, auf wenige zentrale Verkehrsknoten innerhalb der Ortschaft reduziert werden.
- Kollabieren: Bezeichnet das Reduzieren der geometrischen Dimension eines Objektes.
  Da hier keine Flächen oder Ähnliches zu vereinfachen sind, findet das Kollabieren keine Anwendung.
- Verschiebung: Das räumliche Versetzen von Punkte, um z.B. topologische Unstimmigkeiten zu bereinigen, die durch andere Operatoren entstanden sind. In diesem Zusammenhang werden bei einigen Generalisierungsalgorithmen sogenannte Steinerpunkte eingefügt.
  Da eine CH das Neueinfügen oder Verschieben von Knoten nicht zulässt, fällt dieser Operator aus.
- Glättung: Eine Simplifizierung, die z.B. darauf achtet, dass Kurven nicht schärfer werden.
  Unsere Algorithmen werden dies nicht explizit berücksichtigen.
- Verfeinerung: Das Bevorzugen von wichtigen Merkmalen einer Struktur gegenüber unwichtigen bei der Simplifizierung.
  Wir werden Fehlermaße definieren, um innerhalb einer Straße die wichtigen von den unwichtigen Merkmalen zu trennen.
- Übertreibung: Die zusätzliche Hervorhebung von kritischen Strukturen.
  Dies findet hier keine Anwendung.
- Verbessern: Falls z.B. zu viele Datenpunkte für einen Bereich der Karte gespeichert sind und diese alle angezeigt werden, kann dies bis zur Unlesbarkeit der Karte führen. Es ist daher oft angebracht, zusätzlich zum Selektieren und Eliminieren in einer Nachbearbeitung noch einige Merkmale bei der Darstellung hinzuzufügen bzw. wegzulassen.
  Wir werden dies nach Konstruktion einer CH auf die hinzugekommen Abkürzungen anwenden.

Im Allgemeinen ist die Generalisierung nicht rein subtraktiv bezüglich der detailreichsten Darstellung. Am deutlichsten wird dies beim Übertreibungsoperator.
Wenn wir wie oben vorgeschlagen einfach ein G_i anzeigen, wäre dies bezüglich der Anzahl der Knoten rein subtraktiv. Jedoch kann es passieren, dass mit höherem Zoomlevel i die Kantenmenge sogar größer wird, also gilt $|E_i| \nleq |E_{i-1}|$. Dies liegt daran, dass die Zahl der hinzukommenden Abkürzungen größer sein kann als die Zahl der gelöschten Kanten. Für normale Straßengraphen ist dies auf hohen Leveln praktisch immer der Fall. Die naive Vorgehensweise G_i als vermeintliche Vereinfachung darzustellen kann also sogar zur Anzeige eines komplizierteren Graphen führen.
Wie werden in Kapitel 5 genauer darauf eingehen, welcher Graph genau angezeigt werden soll. Es wird darauf hinauslaufen, dass wir einzelne Abkürzungskanten entpacken, d.h. also statt einer Kante e mit $unpack(e) = (e_1, e_2)$ die Kanten e_1 und e_2 anzeigen werden. Auf den ersten Blick und auch rein theoretisch, könnte dies sogar zu einer noch größeren Menge von anzuzeigenden Kanten führen. Es stellt sich jedoch heraus, dass in der Praxis auf hohen Leveln viele Abkürzungskanten aus den gleichen Kanten entstehen und so auf einem Zoomlevel i wesentlich weniger Kanten anzuzeigen sind als $|E_0|$.

Für niedrige Level ist die direkte Anzeige von G_i jedoch eine gute Arbeitshypothese, um die Algorithmen für lokale Vereinfachungen zu erläutern.

2.3.5. Kontinuität der Vereinfachung

Um unterschiedlich detailgetreue Darstellungen bei interaktiven Karten zu realisieren, verwenden die meisten Online-Dienste nicht kontinuierliche Darstellungen. Dabei wird aus der detailreichsten Darstellung eine Reihe von vereinfachten Darstellungen für eine begrenzte Anzahl an Zoomleveln auf dem Server vorberechnet. Abhängig vom gewählten Zoomlevel eines Nutzers wird bei einer Anfrage eine entsprechende vorberechnete Darstellung vom Server zurückgegeben. Falls nun z.B. näher heran gezoomt wird, und die bisherige Darstellung nicht mehr ausreichend detailliert ist, so muss eine genauere Darstellung geladen werden, um die bisherige zu ersetzen wie in Abbildung 2.1 zu sehen ist. Diese Ersetzung hat Potenzial den Nutzer zu irritieren, da abrupt eine stark veränderte Darstellung angezeigt wird.

Dieser Effekt kann abgemildert werden, wenn für mehr Zoomlevel Darstellungen vorberechnet werden. Mehr Darstellungen zu berechnen, zu speichern und zum Benutzer zu transferieren führt aber zu höheren technischen Kosten.
Zusätzlich muss eine separate Datenstruktur für das Suchen von kürzesten Pfaden gespeichert werden. Durch diese Separierung kann es passieren, dass eine berechnete Route abseits des Straßennetzes zu verlaufen scheint, wenn wie in Abbildung 2.2 die entsprechenden Straßen aufgrund zu geringer Detailauflösung nicht angezeigt werden.

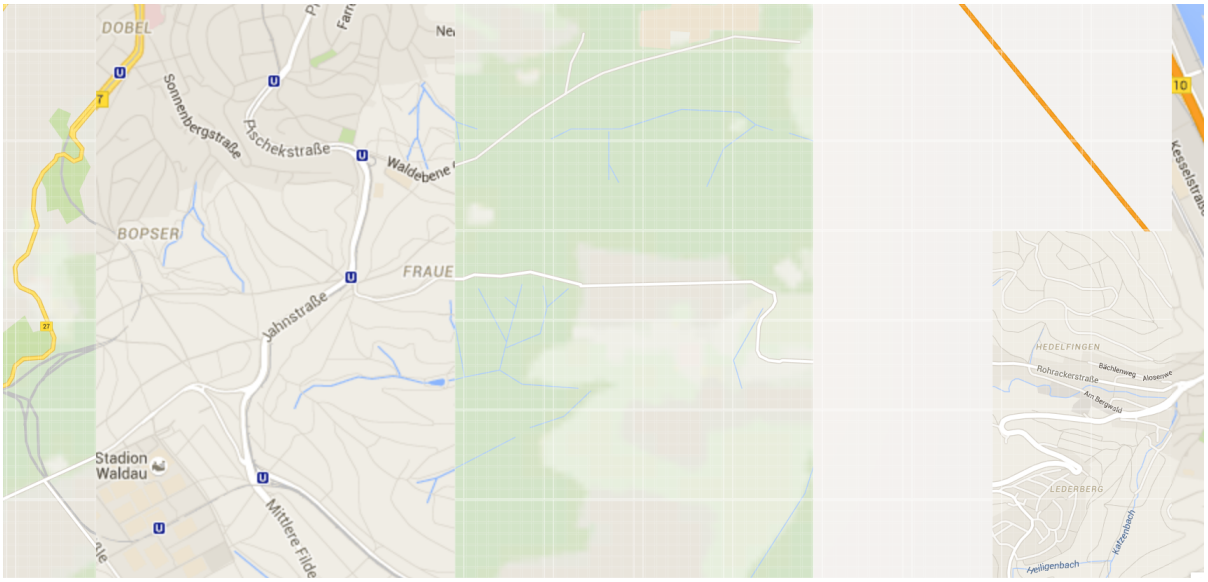


Abbildung 2.1.: Ein Screenshot aus Google Maps während des Zoomvorgangs. Es ist deutlich zu sehen, wie sich die gesamte Darstellung aus Feldern vorberechneter, nicht kontinuierlicher Vereinfachungen zusammensetzt, die in diesem Moment geladen werden müssen.

(a) Anzeige eines kürzesten Pfades **(b)** Karte ohne kürzesten Pfad

Abbildung 2.2.: In diesem Screenshot aus OSRM (OpenStreetMap Routing) ist die Anzeige einer Route zu sehen, obwohl die dazugehörigen Straßen nicht Teil der gezeigten Karte sind. (Quelle: [Sch15])

Um technische Ressourcen zu sparen, den Zoomvorgang natürlicher zu gestalten und die Diskrepanz beim Anzeigen von kürzesten Pfaden zu vermeiden, kann stattdessen eine kontinuierliche Simplifizierung vorberechnet werden. Eine kontinuierliche Simplifizierung ist eine Datenstruktur aus der sich für beliebig fein gewählte Zoomeinstellungen schnell passende Darstellungen berechnen lassen, die ineinander überführbar sind. Falls gezoomt wird, so muss die angezeigte Darstellung nicht verworfen, sondern kann mit geringen Aufwand modifiziert werden, sodass sie der neuen Zoomeinstellung entspricht.
Da für eine CH ein G_i leicht in G_{i+1} überführt werden kann, können wir CHs als kontinuierliche Simplifizierungen nutzen.
Dieser Ansatz wurde bereits mit einer CH verfolgt [Sch15], die für die Berechnung von kürzesten Pfaden optimiert war. Wir wollen nun Strategien untersuchen um CHs von vorneherein aufzubauen, damit sich aus diesen möglichst akkurate Vereinfachungen gewinnen lassen.

3. Lokale Vereinfachung

Wie in Abschnitt 2.3.4 angesprochen wurde, gehen wir der Einfachheit halber zunächst davon aus, dass direkt ein G_i als vereinfachte Darstellung von G angezeigt wird. Außerdem nehmen wir an, dass in jeder Kontraktionsrunde genau ein Knoten kontrahiert wird.

Die Aufgabe besteht nun darin eine CH zu erstellen, sodass für alle Level i der jeweilige Graph G_i eine möglichst wiedererkennbare Darstellung von $G = G_0$ liefert. Da eine CH nur über die Kontraktionsreihenfolge der Knoten bestimmt ist, müssen wir also diese festlegen. Generischer Code zum Aufbau von CHs war für diese Arbeit schon bereitgestellt. [1]
Praktisch ist uns ein G_i gegeben und es ist die Entscheidung zu treffen, welcher Knoten als nächstes zu kontrahieren ist. Graphelemente werden sich deswegen immer auf ein G_i beziehen, solange nicht explizit ein anderer Graph genannt wird.
Es ist naheliegend einen Knoten v mit einer kleinen Anzahl an Nachbarn, also geringen Grad $deg(v)$ zu kontrahieren, da hier nicht viele Abkürzungen entstehen können.

Für $deg(v) = 0$ ist die Kontraktion trivial und sollte möglichst schnell geschehen.
Für $deg(v) = 1$ kommen ebenfalls keine Kanten hinzu, sodass eine schnelle Kontraktion fast immer sinnvoll ist. Außerdem stellt v das Ende einer Sackgasse dar, worauf in Abschnitt 5.4 noch weiter eingegangen wird. Da nach der Kontraktion von v sein Nachbar von Grad eins ist, kann dies fortgesetzt werden, sodass Sackgassen in höheren G_i rückstandslos entfernt werden können.
Falls $deg(v) = 2$, so können bei der Kontraktion nicht mehr Kanten hinzugefügt werden als entfernt werden. Aus solchen Knoten bestehende Pfade stellen in G_0 Straßenverläufe ohne Verzweigungen dar, die möglichst ganzheitlich vereinfacht werden sollten. Falls inmitten einer langen Straße jedoch eine Ausfahrt an einem Knoten v_{aus} zu einer kleinen Sackgasse existiert, so sollte die Straße dennoch als Gesamtes vereinfacht werden. Wenn wir zunächst diese Sackgasse entfernen, gilt für ein höheres G_i: $deg(v_{aus}) \leq 2$. In diesem G_i ist die lange Straße dann leichter als Ganzes erfassbar.
Wenn $deg(v) \geq 3$, so müssen in den meisten Fälle eine Menge von Abkürzungen eingefügt werden, die visuell verstörend sind. Allgemein sollte die Kontraktion solcher Knoten möglichst lange hinausgezögert werden. Da dies trotzdem nicht völlig zu vermeiden ist, wird in den Abschnitten 5.2 und 5.3 darauf eingegangen wie sich der Effekt außerhalb der Festlegung einer Kontraktionsreihenfolge abmildern lässt.

[1] https://theogit.fmi.uni-stuttgart.de/nusserae/chconstructor

3.1. Ketten

Verbesserungspotenzial für Vereinfachungen weisen vor allem Grad-2-Knoten auf. Sie sind einerseits nicht trivial und andererseits sind ihre Auswirkungen auf die spätere Anzeige während des Kontraktionsprozesses noch überschaubar. Praktisch unterteilt ein Grad-2-Knoten die Straße zwischen seinen Nachbarn in zwei Straßenabschnitte, sodass nach seiner Kontraktion diese durch einen Straßenabschnitt approximiert werden. Innerhalb einer langen Straße, die sich durch ein Pfad von Grad-2-Knoten manifestiert, stellt sich also die Frage in welcher Reihenfolge die Knoten kontrahiert werden sollten, um die Gesamtform der Straße möglichst lange zu erhalten. Start- und Endpunkt eines solchen Pfades haben außerdem typischerweise Nachbarn mit einem Grad ungleich 2, wobei die verbindenden Kanten ebenfalls noch Abschnitte der repräsentierten Straße darstellen und deswegen für die Vereinfachung in Betracht gezogen werden sollten.
Weiterhin können wir an Straßentypen erkennen, ob Kanten zur gleichen Straße gehören. Aus den OSM-Daten ist uns für jede Kante aus G zusätzlich der Straßentyp gegeben. Nach einer Vorverarbeitung ist formal eine Abbildung $st : E \to \mathbb{N}$ gegeben, in der kleinere Zahlen wichtigere Straßentypen bezeichnen. Wir erweitern diese Abbildung auf $st^+ :\to \mathbb{N}$ indem für eine Abkürzungskante e, die $(e_1 e_2)$ abkürzt gilt: $st^+(e) = \max(st^+(e_1), st^+(e_2))$.

Die Datenstruktur um Straßen zu verwalten nennen wir Kette. Die Menge aller Ketten sei C und sei ebenfalls die Abbildung $st : C \to \mathbb{N}$ definiert. Um Einbahnstraßen bzw. einzelne Fahrbahnen und bidirektionale Straßen auseinanderhalten zu können, unterscheiden wir außerdem noch zwischen uni- und bidirektionalen Ketten. Jeder Kette c ist außerdem ein Pfad $p = v_A v_1 v_2 \dots v_n v_E$ mit $\forall i \in [1, n] : deg(v_i) = 2$ und $\forall e \in \hat{p} : st^+(e) = st(c)$ zugewiesen. Falls zusätzlich ein Pfad $p_r = v_E v_n \dots v_2 v_1 v_A$ existiert, ist die Kette bidirektional, ansonsten unidirektional. Wir wollen außerdem die Vereinfachung einzelner Ketten separat berechnen, aber ihre Knoten in den gleichen Runden kontrahieren können. Dazu dürfen Knoten v_A und v_E nicht im Kontext lokaler Vereinfachung kontrahiert werden und die Pfade zweier Ketten dürfen außer ihren v_A und v_E keine gemeinsamen Knoten haben.

Wir detektieren die Ketten in G_i mit dem skizzierten Markierungsalgorithmus:
Falls ein Knoten v mit $deg(v) <= 2$ noch nicht markiert ist, bildet er die Keimzelle einer neuen Kette. Der Straßentyp der Kette sei der niedrigste, also wichtigste Straßentyp von allen angrenzenden Kanten von v. Falls ein Nachbarknoten über Kanten dieses Straßentyps bidirektional erreichbar ist, so ist die Kette bidirektional. Bezüglich des Straßentyps und gegebenenfalls der Bidirektionaliät wird der Pfad der Kette ausgehend von v auf weitere Knoten mit Grad 2 expandiert, die bis jetzt unmarkiert waren. Die Knoten, die zum stoppen der Expansion geführt haben, werden als v_A und v_E gewählt.

Für unsere Analysen werden wir weiterhin annehmen, dass der Pfad einer Kette bei Kontraktionen konsistent gehalten werden kann, d.h. bei einer Kontraktion wird immer eine Abkürzung zwischen den Nachbarknoten benötigt.
Das allgemeine Problem solche Pfade zu vereinfachen, ist in der Literatur vor allem als Vereinfachung von Polygonzügen bzw. polygonalen Pfaden [GNS07] bekannt. Um nicht von der

üblichen Terminologie abzuweichen, werden wir im Kontext der visuellen Vereinfachung Polygonzug als Synonym für Pfad verwenden.

Falls einzelne Abschnitte der repräsentierten Straßen extrem überflüssig sind, so ist es theoretisch denkbar, dass keine Abkürzungen eingefügt werden und Lücken entstehen. Um gegenüber diesen sehr seltenen Fällen robust zu sein, werden wir Ketten nach der Detektion lediglich mit deren Straßentyp und deren Uni- oder Bidirektionalität als Liste von Knoten speichern und Kanten zwischen ihnen implizit annehmen. Schlimmstenfalls müssen wir vereinzelt eine lokal degenerierte Vereinfachung hinnehmen.

Bevor wir uns der konkreten Vereinfachung von Polygonzügen zuwenden, wollen wir noch kurz eine Alternative zu den gerade definierten Ketten vorstellen.

3.1.1. OSM-Ways

Ursprünglich war geplant, die Straßen direkt aus den OSM-Daten herauszulesen. Dazu muss man Folgendes über den Aufbau der OSM-Daten wissen: Die OSM-Daten sind als eine Menge von Nodes und Ways gegeben. Jede Node ist eine Kombination aus Längen- und Breitengrad. Jeder Way ist eine Liste aus Knoten und zusätzlichen Informationen.
Der für dieses Projekt bereitgestellte OsmGraphCreator aus Fußnote 1 filtert alle für das Straßennetz relevanten Nodes und Ways heraus und liefert unseren Basisgraphen $G(V, E)$. Die benötigten Nodes werden dabei in Knoten der Menge V umgewandelt und die Ways in einzelne Kanten zerlegt, die E bilden.
Da über die OSM-Daten also schon über die Ways ein logischer Zusammenhang der Kanten bereitgestellt wurde, schienen die Ways als Datenstruktur dafür prädestiniert zu sein, um die lokal zu vereinfachenden Straßen des Graphen zu verwalten. In seiner originalen Form verwarf der OsmGraphCreator die Way-Informationen, konnte aber leicht dahingehend modifiziert werden.
Es kristallisierten sich jedoch einige Probleme mit diesem Ansatz heraus:

1. Zunächst ist festzustellen, dass Ways keine vollständige Straßen darstellen, sondern nur Abschnitte von Straßen wie in Abbildung 3.1 gezeigt ist. Dies ist dem Umstand geschuldet, dass es Straßenabschnitte gibt, die sich in für uns unwichtigen Eigenschaften unterscheiden, z.B. der Straßensteigung.
   Eine Möglichkeit wäre, Ways an ihren Enden zusammenzufügen, um damit ganze Straßen bzw. Fahrbahnen zu regenerieren. Zwar fasst die Relation *Route* aus den OSM-Daten mehrere Ways zusammen, jedoch sind die wenigsten Ways Element einer *Route*.

2. Insbesondere haben die Ways ihre Nodes nicht für sich allein „gepachtet“, d.h. es kann ein anderer Way auf eine Node mitten im ersten Way führen. Damit stellen die Ways keine Pfade von Grad-2-Knoten dar und Vereinfachungen ihrer Knoten können somit weitreichende Veränderungen auf anliegende Straßen ausüben.
   Um Algorithmen zur Polygonzugsvereinfachung anzuwenden, hätten diese Ways auf Knoten mit Grad größer 2 untersucht und aufgeteilt werden müssen.

3. Da im Vordergrund der Arbeit eine kontinuierliche Graphvereinfachung stand, war das Einlesen der Ways insofern problematisch, als dass es nur einmal, nämlich am Anfang geschehen konnte. Längere Pfade von Grad-2-Knoten, die erst durch Entfernung von Sackgassen entstehen, hätten dann nicht ganzheitlich erfasst werden können.

Bei der Abschätzung des Aufwands diesen Problemen abzuhelfen und des dadurch gewonnenen zweifelhaften Mehrnutzen, wurde dieser Ansatz verworfen. Da Ketten robuster als Ways und jederzeit detektierbar sind, wurden diese als repräsentative Datenstruktur für Straßen gewählt.

Es ist jedoch beachtenswert, dass in den OSM-Daten noch weitere Informationen stecken, z.B. die Anzahl der Spuren einer Straße, welche für äußerst penible kartografischen Betrachtungen einen Nutzen haben könnten.

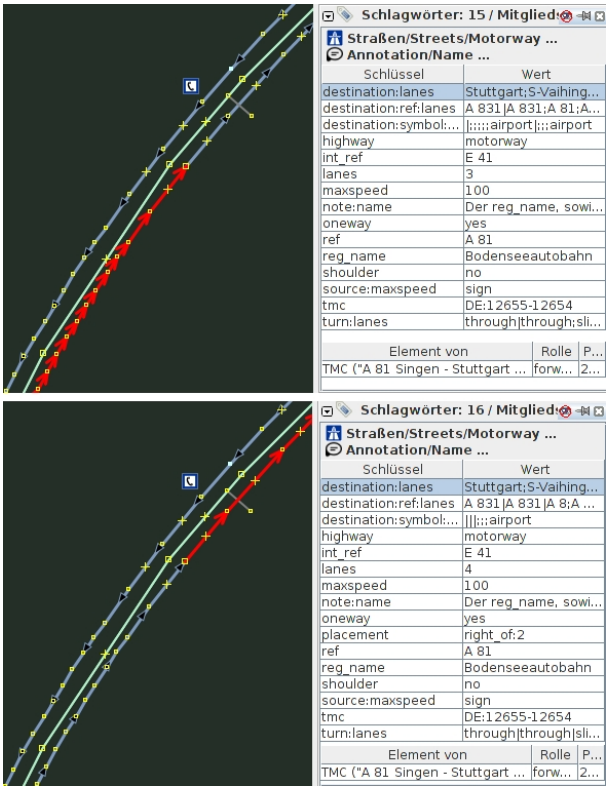


Abbildung 3.1.: Ausschnitt aus JOSM (JavaOpenStreetMap-Editor): Zwei Ways (rot markiert), die leicht unterschiedliche Tags haben.

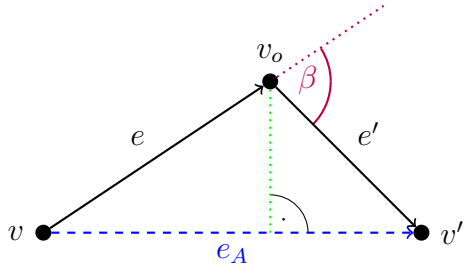


Abbildung 3.2.: Verschiedene Fehlermaße für die Kante e_A, die den Knoten v_o abkürzt.

3.2. Fehlermaße

Um abschätzen zu können, welche der möglichen Vereinfachungen kartografisch am günstigsten ist, benötigen unsere Algorithmen ein Fehlermaß als Heuristik. Es läuft auf ein Fehlermaß für Abkürzungen hinaus, von denen bestimmt werden muss, wie gut sie die abgekürzten Kanten kartografisch wiedergeben. Bei der Vereinfachung von Ketten bedeutet die Kontraktion eines Knoten in der Regel das Entstehen genau einer Abkürzung[2]. Folglich kann der Fehler einer Abkürzung auch dem abgekürzten Knoten zugeordnet werden. Die größten Fehler treten typischerweise bei den markantesten Knoten des Polygonzugs auf.

Da die Vereinfachung in der Kartografie im Allgemeinen und im Speziellen bei Polygonzügen einen subjektiven Aspekt hat, gibt es kein einziges, eindeutiges Fehlermaß. Eine Vielzahl von Fehlermaßen wurde in [McM86] untersucht und in [Jen89] wird eine gute Übersicht zu diesem Thema gegeben.
Wir werden hier einige Fehlermaße aus der Literatur definieren und unsere Algorithmen zur Polygonzugsvereinfachung so generisch gestalten, dass sie mit jedem dieser Fehlermaße parametrisiert werden können.

Sei $e_A = (v, v')$ eine Abkürzung, die einen Pfad $((v, v_o)(v_o, v')) = (ee')$ ersetzt, d.h. $unpack(e_A) = (e, e')$ wie in Abbildung 3.2.

Zu den verwendeten Fehlermaßen für eine Abkürzung e_A gehören:

1. **Lotabstandsfehler:** Dies ist ein einfaches und wohl in der Literatur das am häufigsten verwendete Fehlermaß. Der Fehler ist hier die Länge des Lots von e_A zu dem abgekürzten Knoten v_o. Man beachte, dass ein Lot nicht auf der Abkürzungskante stehen muss, es kann auch auf deren Verlängerung stehen.

[2]Das Entstehen von Lücken sei ausgeschlossen und für bidirektionale Ketten würden zwei Abkürzungen hinzugefügt, die in der Anzeige aber zusammenfallen.

2. **Flächenfehler:** Dieser Fehler ist die Fläche, die vom Dreieck v, v_o, v' eingeschlossen wird. Sie kann mit einer einfachen Determinantenformel[3] berechnet werden:

$$A = \text{abs}\left(\frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}\right)$$

   Hier wird also die Fläche berechnet, die nach der Vereinfachung durch die Abkürzung auf der anderen Seite der Straße liegt.

3. **Knickfehler:** Die Kanten e und e' stellen immer einen Knick dar, der von der Abkürzung e_A geglättet wird. Hierbei soll der Beitrag des Knicks zur Gesamterscheinung eines Polygonzugs bewertet werden. Solche Knicke wurden in [BLR00] im Zusammenhang mit der kontinuierlichen Vereinfachung von Polygonen für schematische Darstellungen untersucht. Die Beispielobjekte in Abbildung 3.3 vermitteln eine intuitive Vorstellung von verschiedenen Knicken. Während der Knick in (a) als irrelevant abgetan werden kann, so sind die beiden Knicke in (b) und (c) deutlich auffälliger. Zweifellos ist der Knick bei (d) jedoch wesentlich für die Gesamterscheinung des Polygons.

   Diese intuitiven Einschätzungen lassen sich mithilfe geometrischer Eigenschaften der Knicke erklären. Der Knick bei (b) hat den gleichen Drehwinkel wie der bei (a), aber seine Schenkel sind länger. Die Schenkel des Knicks bei (c) sind gleich lang wie bei (a), aber der Winkel ist größer. Der Knick bei (d) ist am signifikantesten, da er die längsten Schenkel mit dem höchsten Drehwinkel kombiniert.

   Sei $\beta(e, e')$ der Drehwinkel, dann gilt für den Knickfehler $\mathcal{K}$:

$$\mathcal{K}(e, e') = \frac{\beta(e, e') \cdot l(e) \cdot l(e')}{l(e) + l(e')}$$

Die Kanten e und e' können durchaus selbst Abkürzungen darstellen. Damit können der Lotabstands- und Flächenfehler erweitert werden, da sie sich an dem abgekürzten Knoten v_o orientieren. Offensichtlich kürzt e_A auch alle Knoten ab, die e oder e' abkürzen. Eine so rekursiv definierte Menge der abgekürzten Knoten einer Kante e ist endlich, da die Rekursionstiefe mit der Anzahl der Level einer CH begrenzt ist. Der erweiterte Lotabstands- bzw. Flächenfehler sei als das Maximum über die Fehler aller abgekürzten Knoten definiert.

[3] Für eine Herleitung siehe https://people.richland.edu/james/lecture/m116/matrices/applications.html

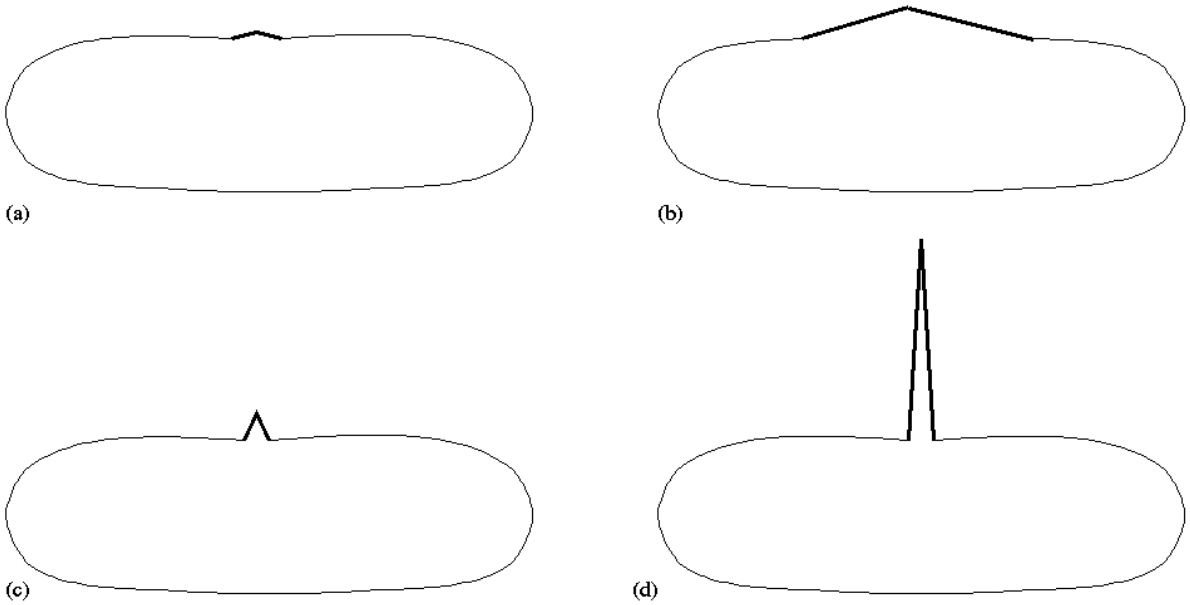


Abbildung 3.3.: Verschiedene Knicke mit unterschiedlichem Beitrag zur Gesamterscheinung eines Polygons, Abbildung aus [BLR00]

3.3. Einzelne Polygonzüge

In der Forschung zur Generalisierung von kartografischen Daten gab es nirgendwo mehr Beiträge als im Bereich der Vereinfachung von einzelnen Polygonzügen. Auf den ersten Blick bietet uns daher die Literatur eine Fülle von Algorithmen zur Lösung unseres Problems an. Typischerweise haben alle diese Algorithmen teils spezielle Randbedingungen unter denen sie eingesetzt werden, erstellen Vereinfachungen nach speziellen Fehlermaßen und sind unterschiedlich auf Berechnungsressourcen optimiert.

Als Erstes werden wir untersuchen, welche der Algorithmen für unser Problem zweckdienlich sind. Allgemein kommen nur Algorithmen in Frage, die sich bei der Vereinfachung auf die Knoten des Polygonzugs und direkte Verbindungen zwischen ihnen einschränken. Verfahren wie in [GHMS93], die neue Knoten einfügen, wie die sogenannten Steiner-Punkte, können nicht eingesetzt werden, da die Kontraktionsoperation keine neue Knoten einführen kann. Auch Algorithmen wie z.B. in [SR03] beschrieben, die versuchen mit Kurven den Polygonzugverlauf zwischen zwei beibehaltenen Knoten nachzubilden, scheiden also aus.

Wenden wir uns zunächst den schnellen Algorithmen zu, d.h. denen mit geringer Zeitkomplexität.
Der schnellste und wohl einfachste Simplifizierungsalgorithmus ist der sogenannte k-Punkt-Algorithmus (Nth-point-algorithm). Er entfernt einfach jeden k-ten Knoten eines Pfades. Zu Vergleichszwecken wurde hier ein 2-Punkt-Algorithmus implementiert, der jeden zweiten Knoten entfernt, bzw. kontrahiert. Nach der Kontraktion dieser Knoten wird mit dem verbliebenen

Pfad so weiterfahren, bis alle Knoten kontrahiert sind. Offensichtlich werden kartografische Unstimmigkeiten bei der Vereinfachung nicht berücksichtigt.
Bemerkenswerterweise kommt die bereits gegebene Standardkontraktionsstrategie nach der Kantendifferenz diesem Verfahren sehr nahe. Dort wird eine unabhängige Menge gierig gewählt, sodass jeder zweite oder dritte Knoten eines Polygonzugs in der Menge ist. Da Knoten eines Polygonzugs niedrige Kantendifferenz haben, werden sie in der Regel auch sofort kontrahiert.

Da der Douglas-Peucker-Algorithmus [DP73] laut [Jen89] vielen Kartographen als der akkurateste Vereinfachungsalgorithmus gilt, aber anderen zu langsam ist, wurden in [HS92] Anstrengungen unternommen, für einen Polygonzug mit n Knoten die schlechtestmögliche Laufzeit auf $O(n \log n)$ zu drücken. Andere Verfahren wie z.B. in [AHMW05] versuchen auf Kosten der Darstellung eine nahezu lineare Laufzeit zu erreichen.

Solche schnellen Algorithmen sind vor allem bei zeitgleicher Verarbeitung und Anzeige von Kartendaten interessant. Da sich die gegebene Aufgabenstellung jedoch innerhalb der Vorverarbeitung befindet, hat die Zeitkomplexität nicht höchste Priorität. Des weiteren hat die Praxis gezeigt, dass die quadratische Zeitkomplexität des schlechtestmöglichen Falls wenig ins Gewicht fällt. Bei unserer Berechnung der CH benötigen die wiederholten Berechnungen des Dijkstra-Algorithmus und die Verwaltung der Kantendatenstruktur des Gesamtgraphen am längsten. Die Verwaltung der Kantendatenstruktur fordert nach jeder Runde das Sortieren der Liste von eingehenden Kanten des gesamten Graphen, benötigt also $O(|E_i| \log |E_i|)$.

Viele Algorithmen stellen zudem Bedingungen an den eingegebenen Polygonzug, die zwar meistens, aber nicht immer in einem Straßengraph gegeben sind. Beispielsweise garantiert der Umwegealgorithmus aus [EM01] die Selbstüberschneidungsfreiheit der Vereinfachung, fordert diese jedoch auch schon von der Eingabe. Eine raffinierte Methode aus [Saa99], den Douglas-Peucker-Algorithmus topologisch konsistent vereinfachen zu lassen, teilt unter anderen Schwächen auch die Forderung nach Selbstüberschneidungsfreiheit der Eingabe. Andere Algorithmen zur Vermeidung von Selbstüberschneidungen aus [DVS95] fordern zusätzlich eine x-Monotonie der Eingabe. Ein Polygonzug ist genau dann x-monoton, wenn es einen Winkel gibt, sodass alle Geraden, die bezüglich dieses Winkels in der Ebene liegen, den Polygonzug jeweils maximal einmal schneiden.

Diese Algorithmen müssen aber allein wegen dieser Bedingungen nicht ausgeschlossen werden. Vor dem Anwenden eines solchen Algorithmus müsste ein Test auf Selbstüberschneidungsfreiheit mit einem Sweep-Line Algorithmus in $O(n \log n)$ oder auf x-Monotonie in $O(n)$ [PS81] durchgeführt werden. Für den seltenen Fall, dass eine Bedingung nicht erfüllt ist, könnte ein robusterer Algorithmus eingesetzt werden.

Leider sind diese Algorithmen nicht zur kontinuierlichen Vereinfachung entworfen worden. Wie die allermeisten Algorithmen aus der Literatur sind sie entweder auf Lösung des min-#- oder des min-ϵ-Problems ausgerichtet. Diese Algorithmen können für eine gegebene Fehlertoleranz ϵ oder eine Anzahl an Knoten, die die Vereinfachung maximal haben darf, nach gewissen Kriterien eine optimale Lösung finden. Jedoch kann bei einer solchen Lösung das Auslassen

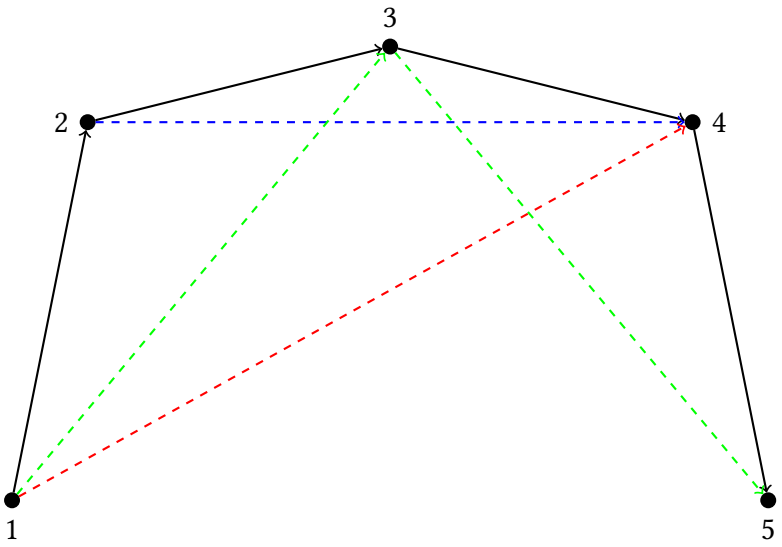


Abbildung 3.4.: Gegebener Polygonzug (schwarz), mit Abkürzungen (rot, blau, grün)

eines weiteren Knoten zu massivem Absinken der Qualität der Vereinfachung führen, wenn jeder einzelne Knoten wichtig ist.
Dies ist in Abbildung 3.4 dargestellt. Zu Vereinfachen ist der schwarze Polygonzug $(v_1, v_2, v_3, v_4, v_5)$. Für ein min-$\epsilon$-Problem bezüglich dem Lotabstandsfehler, bei der das Verwenden von vier Knoten erlaubt ist, würde die Vereinfachung mit der blauen Abkürzung gewählt. Um Kontinuität zu gewährleisten muss, diese dann aber auch weiter zu einer Vereinfachung mit nur drei Knoten überführt werden. Dabei bleiben nur die Optionen entweder den Knoten v_2 oder v_4 zu wählen und damit z.B. die rote Abkürzung zu wählen. Es fällt auf, dass unter den Vereinfachungen mit drei Knoten die grüne den ursprünglichen Polygonzug wesentlich besser approximiert als die rote. Ausgehend von der besten Vereinfachung für vier Knoten könnten wir diese aber nicht mehr erreichen.

Da der Benutzer auf einen beliebigen Level zoomen kann, betrachtet er sozusagen einen Schnappschuss, bei dem wir solche massiven Einbrüche in der Qualität vermeiden wollen. Offensichtlich können wir auch nicht für jede Anzahl an Knoten ein min-ϵ Problem lösen lassen, weil die Vereinfachungen ineinander überführbar sein müssen. Rein formal suchen wir für die kontinuierliche Vereinfachung eines Polygonzug p mit n Knoten eine Folge von Polygonzügen p_1, p_2, p_{n-2}, mit $p_1 = p$. Es muss außerdem gelten:

$$\forall i \in \{1, ..n-1\} \exists k : p_i = v_1 \dots v_{k-1} v_k v_{k+1} \dots v_i \Rightarrow p_{i+1} = v_1 \dots v_{k-1} v_{k+1} \dots v_i$$

Theoretisch könnten wir alle potenziellen Kontraktionsreihenfolgen für den gegebenen Polygonzug untersuchen und ein Gesamtgewicht berechnen. Bedauerlicherweise ist der Raum für alle diese Kontraktionsreihenfolgen exponentiell groß in n, sodass dies keine praktikable Vorgehensweise ist.

Da sich die Gesamterscheinung des Polygonzugs für den Benutzer beim Zoomen von einem Level zum nächsten möglichst wenig ändern soll, bieten sich Greedy-Algorithmen für dieses Problem an.
Ausgehend von den Anwendungen des Heraus- und Hineinzoomens werden im Folgenden dafür zwei solche Vorgehensweisen beschrieben. Hohe Level in der CH werden hier als oben (top) und niedrige Level als unten (down) verstanden.

3.3.1. Bottom-UP

Ein menschlicher Kartograph würde bei diesem Ansatz zunächst den herangezoomten Polygonzug in allen Details sehen. Wenn er nun schrittweise herauszoomt und die Detaildichte verringern möchte, so muss er nacheinander Punkte des Polygonzugs löschen und für jeden gelöschten eine Luftlinienstrecke zwischen den Nachbarpunkten einzeichnen, damit die Verbundenheit der Knoten weiterhin repräsentiert bleibt. Damit sich die Vereinfachung mit dem ursprünglichen Polygonzug möglichst leicht identifizieren lässt, sollte sich insbesondere die wesentliche Erscheinung nicht verändern.
Um algorithmisch festzustellen, wie viel ein Knoten zur Gesamterscheinung seines Polygonzugs beiträgt haben wir in Abschnitt 3.2 unterschiedliche Fehlermaße zusammengetragen. Wie in [BLR00] demonstriert wurde, eignet sich hierfür besonders der Knickfehler. Aber auch die anderen Fehlermaße sind denkbar.
Für die kontinuierliche Vereinfachung des Polygonzugs wird also immer der Knoten mit dem geringsten Fehler gelöscht. Dieses Vorgehen ist sehr lokal und da sich die Fehlermaße nur an einem Knoten und seinen Nachbarn orientieren, müssen nach Herauslöschen eines solchen Knoten auch nur deren Fehlermaße aktualisiert werden.

An dieser Stelle ist zu erwähnen, dass der Algorithmus gedächtnislos ist, d.h. er orientiert sich nicht mehr an dem originalen Graphen sondern am zuletzt errechneten Vereinfachungspolygonzug. Allgemein wurden gedächtnislose Ansätze z.B. auch in [LT98] zur Vereinfachung von Polygonen angewendet
Dies ist durchaus sinnvoll, da es für den Benutzer angenehm ist, wenn sich die Darstellung eines Polygonzugs beim Herauszoomen möglichst wenig ändert. Dagegen wird die kartografische Genauigkeit als Priorität an die zweite Stelle gedrängt.
Falls der Fehler nur auf dem Drehwinkel basieren würde, können unpassende Vereinfachungen wie in Abbildung 3.5 auftreten. Hier wäre der Drehwinkel bei v_3 (bzw. v_4) am kleinsten und würde deswegen zur Einzeichnung der blauen Abkürzung führen. Danach wäre der Drehwinkel bei v_4 am kleinsten und die rote Abkürzung würde eingezeichnet. Der visuell charakteristische und weitläufige Umweg über die Knoten v_3 und v_4 wird dabei vernachlässigt. Die aus den grünen Abkürzungen bestehende Vereinfachung mit weniger Knoten approximiert den Gesamtverlauf des Polygonzugs wesentlich besser. Dies demonstriert deutlich, dass ein kurzsichtiges Fehlermaß mit dieser Vorgehensweise zu einer Eliminierung bzw. Verflachung von wichtigen Teilen des Polygonzugs führen kann. Insbesondere hilft es dann nichts, wenn an den kritischen Stellen die Knotendichte höher ist.

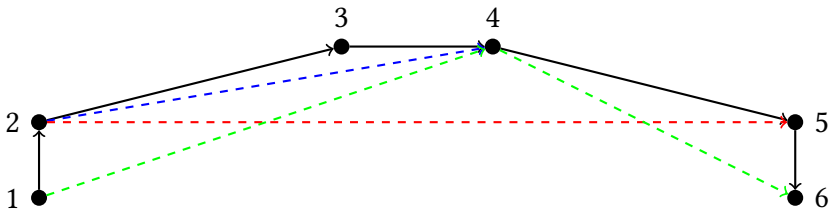


Abbildung 3.5.: Gegebener Polygonzug (schwarz), mit Abkürzungen

Dieses Problem relativiert sich für den Knickfehler, da dieser sich an der Länge der Schenkel orientiert und damit verhindert, dass der Polygonzug so unregelmäßig wie in Abbildung 3.5 ausgedünnt wird.

Es bietet sich an, die Knoten bezüglich ihrer Fehler in einer Min-Heap-Datenstruktur zu verwalten. Damit kosten das Herausnehmen des Knoten mit minimalem Fehler und die darauffolgenden Aktualisierungen jeweils nur $O(\log n)$ Zeit. Da schließlich alle Knoten des Polygonzugs bis auf die Endpunkte gelöscht werden müssen, beträgt die Laufzeit insgesamt $O(n \log n)$.

3.3.2. Top-Down

Diese Vorgehensweise arbeitet in der umgekehrten Richtung wie der vorige Ansatz. Wieder auf einen menschlichen Kartographen bezogen, würde dem eine stark vereinfachte Darstellung gegeben. Außerdem ist ihm als Wissensbasis die detailreichste Darstellung bzw. die geografische Datenbasis vollständig bekannt. Seine Aufgabe bestünde nun darin, schrittweise Knoten hinzuzufügen, damit die gegebene Darstellung sich topografisch möglichst schnell an die reale Geografie annähert.
Nach diesem Prinzip arbeitet auch der bekannte Douglas-Peucker-Algorithmus [DP73], welcher zunächst von der gröbsten Vereinfachung eines Polygonzugs, einer einzelnen Kante zwischen Start- und Endknoten, ausgeht. Falls der größte Lotabstandsfehler von allen Zwischenknoten zu dieser Kante eine Fehlerschwelle überschreitet, wird der entsprechende Zwischenknoten hinzugenommen und die alte Kante durch zwei neue ersetzt. Dieses Vorgehen wird rekursiv auf den zwei entstehenden Kanten fortgesetzt.

Wir können den Douglas-Peucker-Algorithmus nicht in seiner Originalform übernehmen, da wir nicht an der Lösung eines min-ϵ-Problems interessiert sind, sondern vielmehr an der Reihenfolge, in der die Knoten den Polygonzug verfeinern. Wir werden deswegen immer mit dem Knoten verfeinern, der im gesamten Polygonzug den höchsten Fehler aufweist. Eine ähnliche Variante wurde auch schon in [BB, S. 234] beschrieben.
Im Vergleich zur rekursiven Variante ändert sich an der Laufzeit theoretisch wenig, da prinzipiell so viele Aktualisierungsoperationen wie bei der normalen Variante durchgeführt werden müssen. Einzig die Datenstruktur, z.B. ein Max-Heap, zum Ermitteln des nächsten Knotens kann jetzt nicht aufgeteilt werden, sodass diese Ermittlung bzw. eine Aktualisierungsoperation für alle Schritte nun näher an $O(\log n)$ liegt.

Die Kontraktionsreihenfolge der Knoten wird die umgekehrte Bearbeitungsreihenfolge des Top-Down Algorithmus sein. Da mit den wichtigsten Knoten des Polygonzugs zuerst verfeinert wird, müssen diese als letzte kontrahiert werden. Dieser Punkt war für den Bottom-Up Algorithmus nicht zu beachten, weil der Kontraktionsprozess ebenfalls bottom-up ausgeführt wird.
Weiterhin ist auch der Top-Down Algorithmus mit den verschiedenen Fehlermaßen konfigurierbar und nicht wie der klassische Douglas-Peucker-Algorithmus auf den Lotabstandsfehler fixiert.

Auch wenn der Top-Down Ansatz aus der Sicht des Heranzoomens „greedy“ ist, so kann durch Umkehrung der Knotenreihenfolge bei der Kontraktion dieser Ansatz im Vergleich zu Bottom-Up als vorausplanend angesehen werden. Ein weiterer Vorteil des Top-Down Ansatzes ist die Berücksichtigung des zu Beginn gegebenen Polygonzugs über alle Vereinfachungsschritte hinweg, im Gegensatz zur Gedächtnislosigkeit von Bottom-Up.

3.3.3. Grenzwechsel verhindern

Wir wollen weiterhin verhindern, dass Punkte beim Vereinfachen von Straßen danach auf der anderen Straßenseite liegen. Da dieses Problem insbesondere bei der Vereinfachung von Grenzen auftritt, wird dies auch Grenzwechsel genannt. Leider bringt schon eine klare Definition von Grenzwechseln Probleme mit sich, die in Kapitel 6.1 ausführlich erläutert werden. Aufgrund der dort erörterten Kosten, die allein zum Messen von Grenzwechseln benötigt werden, greifen wir hier auf eine einfache Heuristik zurück, die in den allermeisten Fällen die Intuition widerspiegelt.

Es ist wichtig zu beachten, dass bei fortschreitender Vereinfachung Grenzwechsel irgendwann nicht mehr vermieden werden können. Dies ist ein weiterer Grund, weswegen Algorithmen wie aus [Saa99] nicht verwendet werden können, da sie dazu konstruiert wurden, Grenzwechsel völlig zu vermeiden.

Als einfache Heuristik berechnen wir für jede Kante $e = (v, v')$ des originalen Polygonzugs eine Box B_e in Form eines achsenorientierten[4] Fensters. Die Ecken von B_e sind genau die Start- und Endknoten v und v'. Abbildung 3.6 zeigt exemplarisch für die Kanten (v_1, v_2) und (v_2, v_3) die zugehörigen grünen Boxen. Die Boxen werden im Weiteren alle Knoten, die für Grenzwechsel in Frage kommen, verwalten.

Um geographisch nahe Knoten zu finden, reichen die Nachbarschaftsbeziehungen aber nicht mehr aus. Naiv müssten bei jedem Fenster für jeden Knoten im Graphen anhand seiner Koordinaten überprüft werden, ob er ins Fenster fällt. Indem wir in einem Vorverarbeitungsschritt ein Raster über den gesamten Graphen legen, können wir die Suche beschleunigen. Für eine Rasterzelle werden alle hineinfallenden Knoten in einer Liste abgespeichert. Das Raster stellt

[4]Wir nehmen hier an, dass x und y Koordinaten in der euklidischen Ebene darstellen.

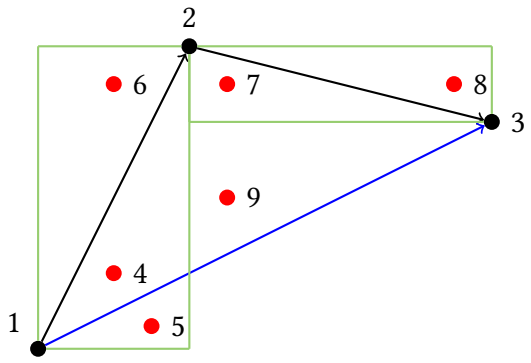


Abbildung 3.6.: Zwei Kanten mit achsenorientierten Boxen um Grenzwechsel aufzuspüren

damit eine grobe Unterteilung des Gesamtgraphen in geographische Gebiete dar. Um geographisch nächste Knoten eines Knoten v zu bestimmen müssen, kann die Suche auf Listen der benachbarten Rasterzellen eingeschränkt werden Für die Rasterzellen muss eine feste Größe gewählt werden. Alternativ wären flexiblere Datenstrukturen wie Bereichsbäume einsetzbar, die aber mehr Speicherplatz benötigen würden.

Zunächst ermitteln wir also mit Hilfe der Rasterdatenstruktur alle Knoten die in die Boxen fallen. Für B_{v_1,v_2} ist dies die Menge $\{v_4, v_5, v_6\}$ und für B_{v_2,v_3} die Menge $\{v_7, v_8, v_9\}$. Für alle Knoten, die in dieser Box liegen, berechnen wir mit einem Orientierungstest auf welcher Seite der Kante die Knoten liegen. Um spätere Berechnungen zu sparen, speichern wir die Knoten mit der zugehörigen Seiteninformation als Liste in der Box ab.
Um nun zu untersuchen, wie viele Grenzwechsel beim kontrahieren des Knoten v_2 stattfinden würden, müssen die resultierende Abkürzung (v_1, v_3) und alle Boxen auf dem Pfad von v_1 zu v_3 betrachtet werden. Für jeden Knoten in einer der Boxen wird mittels eines Orientierungstest ermittelt auf welcher Seite der Abkürzung er liegt. Wenn er auf der anderen Seite als der vorberechneten liegt so zählt dies als Grenzwechsel. Beispielsweise würden die Knoten v_5, v_6 und v_8 die gleiche Orientierung haben, während für v_4 und v_7 ein Grenzwechsel erkannt würde.
Für jeden Knoten kann also die Zahl der induzierten Grenzwechsel berechnet und als Entscheidungskriterium für die nächste Kontraktion hinzugezogen werden.
Es sei betont, dass die Boxen nur einmal für die Kanten des ursprünglichen Polygonzugs berechnet werden und nicht für neu eingefügte Abkürzungen. Implementierungstechnisch wird zunächst für den gesamten Polygonzug eine Liste der Boxen errechnet. Die Knoten besitzen Verweise auf die anliegenden Boxen. Falls ein Knoten entfernt wird, so werden die Verweise auf seine anliegenden Boxen seinen Nachbarknoten zugeschoben, da diese Boxen nun für sie relevant sind.

Natürlich ist diese Vorgehensweise nur eine Heuristik, die einige Problematiken aufweist: Es muss darauf geachtet werden, dass für eine Box $B(v, v')$ die Knoten v und v' nicht selbst in $B(v, v')$ fallen und aufgrund numerischer Ungenauigkeit einer Seite zugewiesen werden.
Schwerer wiegen nicht entdeckte Grenzwechsel. Im Beispiel fällt der Knoten v_9 in keine Box.

Hier wird also ein Grenzwechsel übersehen.
Für komplexere Polygonzüge kann es sogar passieren, dass einzelne Knoten in mehreren Boxen auftreten und doppelt oder sogar einmal als grenzwechselnd und einmal nicht gezählt wird. Wie in Kapitel 6.1 diskutiert wird, ist es insbesondere im Falle selbst überschneidender Polygonzüge schwierig Grenzwechsel überhaupt vernünftig zu definieren und auch dementsprechend schwer zu detektieren. Glücklicherweise stellen reale Straßen in den allermeisten Fällen einfache Polygonzüge dar, für die diese Heuristik effektiv ist.

Um im Bottom-Up oder Top-Down Ansatz den nächsten Knoten auszuwählen, konnte bisher einfach jener mit dem kleinsten bzw. größten Fehler genommen werden. Die Hinzunahme der Anzahl von induzierten Grenzwechsel macht die Auswahl in zweierlei Hinsicht komplizierter. Einerseits sollte bei beiden Ansätzen die Anzahl der induzierten Grenzwechsel des nächsten Knoten minimiert werden. Anders als z.B. für den Lotabstandsfehler sind beim Top-Down-Ansatz die markantesten Knoten nicht durch einen hohen Fehler, also eine große Anzahl an Grenzwechseln erkennbar.
Zweitens müssen zusammen aus dem gegebenen Fehlermaß und der Anzahl der induzierten Grenzwechsel eine Ordnungsrelation für die Knoten definiert werden.
Eine Option wäre das Bilden eines Prioritätswertes als gewichtete Summe aus Fehler und Anzahl der induzierten Grenzwechsel. Das Bestimmen einer Gewichtung für die Summanden wäre jedoch nicht trivial. Für jede Kombination von Fehlermaß und Anzahl der Grenzwechsel müsste eine eigene Gewichtung erstellt werden, da die Fehlermaße in völlig unterschiedlichen Einheiten wie Längen oder Flächen messen. Von uns handverlesene Gewichte wären immer stark von Subjektivität geprägt und würden eine objektive Überprüfung der Algorithmen erschweren. Für ein wirklich zuverlässiges Wählen der Gewichte müssten in einer empirischen Studie Vereinfachungen von Menschen bewertet werden. Die daraus gewonnenen Trainingsdaten könnten für ein maschinelles Lernen der Gewichtswerte eingesetzt werden.
Aufgrund der Undurchsichtigkeit des Algorithmus durch eine gewichtete Summe, wurde stattdessen der Anzahl von Grenzwechsel absolute Priorität über dem verwendeten Fehlermaß eingeräumt. Dies bedeutet, dass zuerst die Anzahl der Grenzwechsel als Kriterium betrachtet wird und nur falls dabei keine Entscheidung getroffen werden kann, wird das Fehlermaß hinzugezogen. Diese Vorgehensweise kommt auch den oben beschriebenen Algorithmen aus der Literatur möglichst nahe, da diese um jeden Preis bezüglich eines Fehlermaßes versuchen, Grenzwechsel zu vermeiden.

Die konkrete Umsetzung dieser Entscheidungsfindung ist im Bottom-Up Ansatz naheliegend: Von allen Knoten, die die niedrigste Anzahl an Grenzwechsel bezüglich des originalen Polygonzugs durch ihre Abkürzung hervorrufen, wird jener ausgewählt der von diesen das geringste Fehlermaß aufweist. Der Ansatz ist also im Bezug auf Grenzwechsel nicht gedächtnislos.

Für den Top-Down Ansatz könnte naiv durchaus ähnlich vorgegangen werden, indem immer die Anzahl der induzierten Grenzwechsel des nächsten Knoten minimiert würden. Um jedoch hinsichtlich der Grenzwechsel sinnvoller vorzugehen, muss berücksichtigt werden, dass der ausgewählte Knoten zur Vereinfachung hinzugefügt wird. Damit bietet sich vielmehr die Möglichkeit Grenzwechsel der bisherigen, gröberen Vereinfachung zu korrigieren als neue

zu induzieren. Für jeden Knoten wird dazu seine Hinzufügung simuliert und die Anzahl der Grenzwechsel davor und danach als gw_t bzw. gw_{t+1} berechnet. Wir bilden nun die Grenzwechseldifferenz $\Delta gw = gw_t - gw_{t+1}$. Eine positives Δgw impliziert also eine Verbesserung der Vereinfachung bezüglich Grenzwechsel.
Zur Bestimmung des nächsten hinzuzufügenden Knotens werden also alle Knoten mit höchsten Δgw betrachtet und unter diesen jener mit dem geringsten Fehlermaß ausgewählt.

In beiden Ansätzen muss die Anzahl der induzierten Grenzwechsel bzw. Δgw für die einzelnen Knoten gespeichert und genauso oft wie das Fehlermaß aktualisiert werden. Für normale Straßen sind die allermeisten Boxen wenn überhaupt mit wenigen Knoten befüllt, sodass kaum höhere Berechnungszeit benötigt wird. Falls jedoch die Zahl der Knoten in den Boxen ($|V_{inBox}|$) beträchtlich höher wäre, könnte auch die Laufzeit theoretisch wesentlich höher sein. Da für jeden Punkt im schlechtesten Fall bei $O(n^2)$ Aktualisierungsoperationen ein Orientierungstest durchgeführt werden muss, liegt die Laufzeit bei $|V_{inBox}| \cdot O(n^2)$.

3.4. Parallele Polygonzüge

Wichtige Straßen wie Autobahnen sind als zwei weitgehend parallel verlaufende Richtungsfahrbahnen modelliert. Falls wir die Fahrbahnen unabhängig voneinander vereinfachen, kann es leicht passieren, dass in einem vereinfachten Graphen die Fahrbahnen nicht mehr als zueinander passende Gegenfahrbahnen erkennbar sind. Solche Situationen sind in Abbildung 3.7 dargestellt. Damit die Erkennbarkeit als eine Straße (a) erhalten bleibt, darf also nicht eine Fahrbahn ohne die andere (b) bzw. nicht auf völlig verschiedenen „Höhen“ (c) vereinfacht werden. Wir werden diese intuitive Vorstellung von „Höhe“ noch formalisieren.

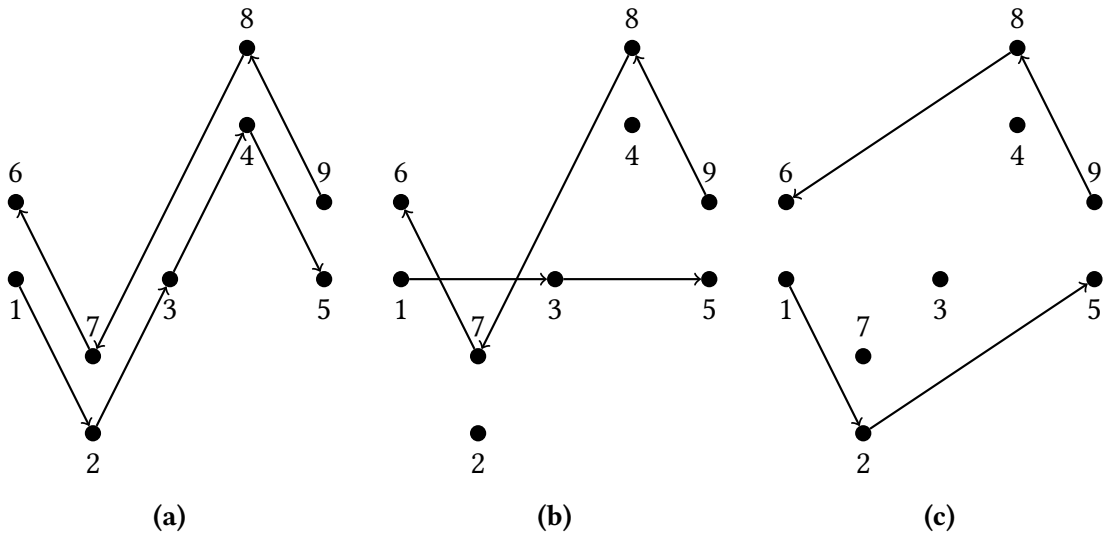


Abbildung 3.7.: Verschiedene Vereinfachungen zweier Richtungsfahrbahnen einer Straße

Um die Vereinfachung der einen Fahrbahn an die andere zu koppeln, sind Datenstrukturen erforderlich, die eine Beziehung zwischen den Knoten oder Kanten der beiden Fahrbahnen herstellen. Aus implementierungstechnischen Gründen basieren die Datenstrukturen wie auch schon die Ketten auf den Knoten.
Problematisch ist die Unregelmäßigkeit der OSM-Daten: Die Fahrbahnen können durch eine unterschiedliche Anzahl an Knoten bzw. Kanten modelliert und zudem leicht versetzt sein. Falls die Fahrbahnen absolut regelmäßig wären, könnten gegenüberliegende Knoten bzw. Kanten einfach einander zugeordnet werden. Angesichts der genannten Unregelmäßigkeiten wurden robustere Ansätze gewählt, um die Fahrbahnen algorithmisch zu verknüpfen.

3.4.1. Zusammenfassen von Fahrbahnen

Zunächst müssen wir die Paare von Fahrbahnen identifizieren, die zusammen eine Straße modellieren. Bis jetzt haben wir alle Ketten berechnet und es ist klar, dass für einzelne Fahrbahnen einer Straße nur unidirektionale Ketten in Frage kommen. Des weiteren werden in den OSM-Daten nur Straßen von wichtigen Straßentypen wie Autobahnen als zwei Fahrbahnen modelliert. Mit dieser zusätzlichen Information müssen wir z.B. keine gerichteten Ketten mit Wohngebietsstraßentyp (residential) untersuchen, da diese einzelne Einbahnstraßen darstellen. Damit zwei Fahrbahnen gemeinsam eine solche Straße bilden, müssen ihre Ketten außerdem dem gleichen Straßentyp zugeordnet sein.

Sei C nun eine Menge aller Ketten, die nach obigen Kriterien als zueinander passende Gegenfahrbahnen in Frage kommen. Zunächst einmal sollten der Startknoten[5] der einen Kette nahe dem Endknoten der anderen Kette liegen und umgekehrt, wenn die beiden Ketten Gegenfahrbahnen zueinander modellieren.
Wir können einer Kette einen Punkt im vierdimensionalen Raum mit $(x_{v_A}, y_{v_A}, x_{v_E}, y_{v_E})$ zuordnen. Um eine Partnerkette zu finden, sollte deren Punkt möglichst nahe an $(x_{v_E}, y_{v_E}, x_{v_A}, y_{v_A})$ liegen, also dem Punkt mit vertauschten Anfangs- und Endknoten. Um diesbezüglich nicht alle Ketten $c \in C$ mit einem Aufwand von $O(|C|^2)$ miteinander vergleichen zu müssen, verwenden wir ähnlich wie in 3.3.3 ein Raster, das hier vierdimensional ist.
Konkret suchen wir für ein $c \in C$ das $c' \in C$, welches den Abstand

$$\sqrt{(x_{v_A} - x_{v'_E})^2 + (y_{v_A} - y_{v'_E})^2 + (x_{v_E} - x_{v'_A})^2 + (y_{v_E} - y_{v'_A})^2}$$

minimiert. Da c durchaus eine einzelne Einbahnstraße darstellen kann, wird c' nur als Partnerkandidat für c gespeichert, wenn der Abstand einen Schwellwert unterschreitet. Das Suchen eines Partnerkandidaten wird für alle $c \in C$ durchgeführt.
Für alle $c \in C$ mit Partnerkandidat c' wird nun überprüft, ob c' als Partnerkandidat wiederum c selbst gespeichert hat. Falls dies zutrifft, so entfernen wir c und c' aus der Menge C und

[5] Genauer gesagt v_A für ihren Pfad $p = v_A v_1 \dots v_n v_E$ und v_E entsprechend als Endknoten.

verknüpfen sie zu einem Tupel (c, c'), welches wir in der Menge C_t verwalten. Alle übriggeblieben $c \in C$ werden einzeln vereinfacht.
Auch hier hätte statt des Rasters ein Bereichsbaum verwendet werden können, der flexibler konstruiert werden kann, aber wegen der vier Dimensionen höheren Speicherplatzbedarf und längere Anfragezeiten hat. Sei k die Anzahl der gefundenen Punkte im Anfragerechteck, so gilt für die Komplexität des Speicherbedarfs $O((|C|(\log |C|)^3)$ und der Anfragezeit mit der Technik des Fractional Cascading $O((\log |C|)^3 + k)$.

3.4.2. Knotenkopplung

Wir werden uns nun der Aufgabe widmen, für ein Tupel (c, c') die zugeordneten Pfade gemeinsam zu vereinfachen, indem wir die Kontraktion ihrer Knoten koppeln. Um auf gleicher „Höhe“ zu koppeln, werden wir für eine einfachere Analyse die Reihenfolge der Knoten in einem Pfad invertieren und danach zwei nebeneinander verlaufende Pfade betrachten. Da die Analyse außerdem unabhängig vom restlichen Graph ist und wir umständliche Notation wie Multiindizes vermeiden wollen, werden wir für den Rest diese Kapitels eine überdeckende Definition von der Knotenmenge V erlauben.

Formal seien unsere Pfade bzw. Polygonzüge:

$$p = u_1 u_2 u_3 \dots u_{n_p}$$

$$q = v_1 v_2 v_3 \dots v_{n_q}$$

mit den jeweiligen Knotenmengen U und V.

Metriken

Da wir bis jetzt nur eine visuelle Vorstellung bezüglich der Parallelität von zwei Polygonzügen haben, müssen wir eine geeignete Formalisierung finden. Wie unsere bisherigen Fehlermaßen werden, wir diese einerseits für eine Heuristik und andererseits für eine später Messung brauchen. Glücklicherweise wurden Ähnlichkeitsmetriken zwischen Polygonzügen bereits intensiv erforscht. Im Folgenden werden die wichtigsten davon dargestellt und auf Eignung für unsere Problemstellung untersucht.

Hausdorff Diese Metrik ist sogar für beliebige Mengen $A, B \in \mathbb{R}^2$ definiert:

$$\delta_H = \max\left(\sup_{a \in A} \inf_{b \in B} dist(a, b), \sup_{b \in B} \inf_{a \in A} dist(a, b)\right) \tag{3.1}$$

Polygonzüge stellen somit einen diskreten Spezialfall dar.
Anschaulich gesehen ist δ_H die längste Distanz, die eine Reise von der einen Menge in die andere benötigt, wenn der Startpunkt möglichst ungünstig gewählt worden ist.

Somit stellt δ_H ein Maß für die Nähe zwischen beiden Polygonzügen dar. Eingangs stand die Vorstellung, das zudem Knoten auf gleicher „Höhe“ in Beziehung zu einander gesetzt werden, jedoch berücksichtigt δ_H dies überhaupt nicht.

Um dieses Problem zu beheben, betrachten wir die nächste Metrik:

Fréchet Die Fréchet-Distanz bezieht sich auf kontinuierliche Kurven, die eine Verallgemeinerung von Polygonzügen darstellen. Eine Kurve B sei eine kontinuierliche Abbildung aus dem Einheitsintervall $[0, 1]$ in den euklidischen Raum $\mathbb{R}$. Eine Reparametrisierung β von $[0, 1]$ ist eine kontinuierliche, monoton steigende Surjektion $\beta : [0, 1] \to [0, 1]$

Dann ist die Fréchet-Distanz zwischen zwei Kurven B_1 und B_2 mit den zugehörigen Reparametrisierungen β_1 und β_2:

$$\delta_F = \inf_{\beta_1,\beta_2} \max_{t\in[0,1]}(dist(B_1(\beta_1(t)), dist(B_2(\beta_2(t))))) \tag{3.2}$$

Die populärste Anschauung ist, dass beide Kurven gleichzeitig von einem Hund und seinem Besitzer abgelaufen werden, die durch eine Leine verbunden sind. Der Parameter t kann hierbei als Zeit interpretiert werden und $B_1(\beta_1(t))$ sowie $B_2(\beta_2(t))$ als Positionen des Hundes bzw. des Besitzers zum Zeitpunkt t. Da β_1 und β_2 monoton steigend sind, dürfen Hund und Besitzer außerdem nie rückwärts laufen. Die Fréchet-Distanz δ_F ist die kürzeste mögliche Leinenlänge um die Kurven vollständig abzuschreiten.
Damit ermöglicht die Fréchet-Distanz eine formale Umsetzung der Kopplung von Wegabschnitten auf gleicher „Höhe“.

Da wir nur Polygonzüge betrachten und insbesondere bei der Kontraktion nur Knoten handhaben können, schränken wir uns auf die diskrete Fréchet-Distanz δ_{dF} ein. Diese ist in [EM94] mit sogenannten Couplings definiert. Ein Coupling L der Polygonzüge p und q ist eine Sequenz von Tupeln:

$$((u_{a_1}, v_{b_1}), (u_{a_2}, v_{b_2}), \dots (u_{a_f}, v_{b_f}))$$

sodass gilt $a_1 = 1, b_1 = 1, a_f = n_p, b_f = n_q$ und

$$\forall i \in \{1 \dots a_f\} : (a_{i+1} = a_i + 1 \vee a_{i+1} = a_i) \wedge (b_{i+1} = b_i + 1 \vee b_{i+1} = b_i)$$

Die Länge von L ist definiert als die größte Distanz innerhalb eines Tupels:

$$|L| = \max_{i\in\{1,..f\}} dist(u_i, v_i)$$

Die diskrete Fréchet-Distanz ist dann:

$$\delta_{dF}(p, q) = \min \{|L| \text{ wobei } L \text{ ein Coupling zwischen } p \text{ und } q \text{ ist}\}$$

Anschaulich ist die Zeit jetzt diskret, wobei Hund und Hundebesitzer in jedem Zeitschritt von einem Knoten zum nächsten springen können. Insbesondere können beide gleichzeitig springen.

Im Weiteren wird eine effiziente und robuste Vorgehensweise beschrieben, die Knoten zu koppeln, welche sich an die diskrete Fréchet-Distanz anlehnt.

Zickzackordnung

Da Hund und Herrchen wegen der Leine nicht unabhängig laufen, müssen beide Pfade zeitlich ineinander verzahnt werden. Dazu erstellen wir einen Zickzackpfad, für den gelten soll, dass die Ordnungen der beiden Einzelpfade noch erhalten bleiben müssen. Anschaulich gibt der Zickzackpfad vor in welcher zeitlichen Reihenfolge entweder der Hund oder sein Herrchen einen Schritt vorwärts machen müssen.

Definition 3.4.1
Zickzackpfad
Seien 2 Pfade p, q vorhanden mit den Knotenmengen U, V und

$$p = u_1u_2u_3 \dots u_{n_p}, n_p = |U|, u_i \in U \wedge \forall i, j \in \{1 \dots n_p\} : i \neq j \Rightarrow u_i \neq u_j$$

$$q = v_1v_2v_3 \dots v_{n_q}, n_q = |V|, v_i \in V \wedge \forall i, j \in \{1 \dots n_p\} : i \neq j \Rightarrow v_i \neq v_j$$

So ist ein Pfad z mit $W = U \dot\cup V$

$$z = w_1w_2w_3 \dots w_{n_p+n_q} \textit{ mit } w_i \in W \wedge \forall i, j \in \{1, n_p + n_q\} : i \neq j \Rightarrow w_i \neq w_j$$

und außerdem

$$\forall u_i, u_j \in p \, \forall w_k, w_l \in W : u_i = w_k \wedge u_j = w_l \Rightarrow (i < j \Leftrightarrow k < l)$$

$$\forall v_i, v_j \in q \, \forall w_k, w_l \in W : v_i = w_k \wedge v_j = w_l \Rightarrow (i < j \Leftrightarrow k < l)$$

ein Zickzackpfad.

Ein solcher Zickzackpfad ist Abbildung 3.8 dargestellt.

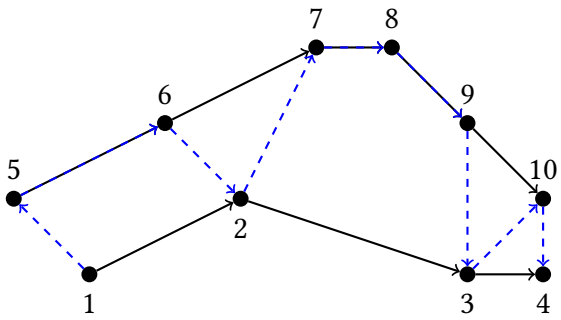


Abbildung 3.8.: Ein möglicher Zickzackpfad (blaue Kanten)

Natürlich gibt es viele mögliche Zickzackpfade, wir wollen jedoch möglichst einen, der mit einer kleinen Leinenlänge auskommt. Der Algorithmus 3.1 in C++-ähnlichem Pseudocode erstellt einen Zickzackpfad wie aus Abbildung 3.8.

Anschaulich gesehen, lässt der Algorithmus Hund und Herrchen stückweise auf ihren jeweiligen Pfaden voranschreiten. Nachdem einer von beiden einen Schritt vorangegangen ist (der Aktive), soll nun entschieden werden, wann der andere einen Schritt nachrückt. Dazu wird geprüft, ob der andere sofort nachrücken soll oder das Nachrücken mit kürzerer Leine möglich ist, wenn der Aktive noch einen Schritt vorangeht.
Dieser Algorithmus versucht also mittels lokaler Sicht mit möglichst kurzer Leine auszukommen. Er ist jedoch nicht optimal, da es Strecken gibt, bei denen er zur kurzsichtig ist.

Wir haben mit dem gefundenen Zickzackpfad nun eine Datenstruktur, die die beiden Pfaden verknüpft, jedoch müssen wir für einen später folgenden Vereinfachungsalgorithmus nun explizit die Knoten der beiden Pfade verkoppeln.

Algorithmus 3.1 Zickzack

```
List p, q;
List zickzack = emptyList();
// wind up, Annahme: p.size() >= 1
activeNode = p.pop_front();
acitvePath = p;
zickzack.push_back(activeNode);
while (!p.empty() && !q.empty) {
        if(activePath == p) {
                if (Distanz(activeNode, q.front())
                        < Distanz(p.front(), q.front()))
                {
                        activeNode = q.pop_front();
                        activePath = q;
                } else {
                        activeNode = p.pop_front();
                }
        } else if(activePath == q) {
                if (Distanz(activeNode, p.front())
                        < Distanz(q.front(), p.front()))
                {
                        activeNode = p.pop_front();
                        activePath = p;
                } else {
                        activeNode = q.pop_front();
                }
        }
        zickzack.push_back(activeNode);
}
// sobald p oder q leer ist, kommt der Rest ans Ende von zickzack
zickzack.splice(w.end(), p);
zickzack.splice(w.end(), q);
```

Der Zickzackpfad ist eine Datenstruktur, die uns nun ermöglicht im Rahmen der Gesamtpfade für jeden Knoten eine Menge an potenziellen Partnerknoten auf dem anderen Pfad zuzuweisen. Wir können nun einfach von einem Knoten w_i im Zickzackpfad den nächsten Knoten w_j finden, der auf dem anderen Pfad liegt, d.h. also:

$$j = \text{argmin}_x\{i < x \wedge ((w_i \in U \wedge w_x \in V) \vee (w_i \in V \wedge w_x \in U))\}$$

Ebenso den letzten vorherigen Knoten w_k auf dem anderen Pfad:

$$k = \text{argmax}_x\{x < i \wedge ((w_i \in U \wedge w_x \in V) \vee (w_i \in V \wedge w_x \in U))\}$$

Für die Knoten am Anfang oder am Ende des Zickzackpfades kann teilweise nur ein nächster bzw. letzter Knoten gefunden werden.

Die Grundidee ist nun, die Kontraktion eines Knoten mit der Kontraktion von Knoten auf dem anderen Pfad zu verknüpfen. Jedoch wollen wir nicht, dass die Kontraktion eines Knotens auf einem Pfad die Kontraktion von zwei Knoten auf dem anderen Pfad nach sich zieht, da der Vereinfachungsprozess möglichst gleichmäßig ablaufen soll. Auch rekursiv die nachgezogenen Knoten weitere Knoten nach sich ziehen zu lassen, ist keine Option, da sich dann nur der Vereinfachungsprozess lokal im Pfad ausbreiten würde.

Also darf jeder Knoten höchstens einen Knoten haben, dessen Kontraktion er nach sich zieht. Wir bezeichnen den nachziehenden Knoten als Anhänger und setzen diese Beziehung in der Anhängerabbildung a um.

Definition 3.4.2
Seien p, q, w, U, V und W wie in obigen Definitionen. Dann sei a eine Abbildung:

$$a : U \to V \cup V \to U$$

Die Abbildung a weist also jedem Knoten des Pfades p einen Knoten des Pfades q zu und umgekehrt. Das durch eine konkrete Abbildung $a(w) = w'$ implizierte Tupel (w, w') nennen wir Querpfeil.

Wie oben beschrieben, können wir für w_i den nächsten Knoten w_j und letzten vorherigen Knoten w_k auf der anderen Seite zu ermitteln. Damit haben wir nun zwei Kandidaten für $a(w_i)$. Naiv können wir einfach den Kandidaten nehmen, der näher an w_i liegt. Dies wird in den allermeisten Fällen gut funktionieren.
Jedoch müssen wir damit rechnen, dass es auch bei weitgehend parallelen Pfaden lokal zu Situationen wie in Abbildung 3.9 kommt. In diesem Fall schneiden sich die Querpfeile von a hinsichtlich der Ordnung der Knoten in den Pfaden.

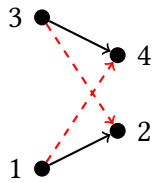


Abbildung 3.9.: Pfadordungsüberschneidung

Dies ist sehr problematisch, da ein Abbildungspfeil bei einem Ablaufen der Pfade von Hund und Hundebesitzer zwei Knoten verknüpfen soll, die gleichzeitig besucht werden. Intuitiv kann man sich den Querpfeil als gestraffte Leine zu diesem Zeitpunkt vorstellen. Wenn sich aber zwei Querpfeile hinsichtlich der Ordnung in den Pfaden überkreuzen, so wäre dies nur mit Zurücksetzen des Hundes oder des Hundebesitzers zu bewerkstelligen.
Damit wir im Folgenden direkt für solche Querpfeile (w, w') von U nach V oder umgekehrt die U- bzw. V-Komponente referenzieren können, sei:

$$u((w, w')) = \begin{cases} w, & \text{falls } w \in U. \\ w', & \text{sonst.} \end{cases}$$

und analog $v((w, w'))$.

Um den Makel dieser Überschneidungen auszuschließen, definieren wir die Abbildung α:

Definition 3.4.3
$\alpha : U \to V \cup V \to U$

sodass gilt: $\forall (w_1, \alpha(w_1)), (w_2, \alpha(w_2)) \in \alpha$ *mit*
$u_i = u((w_1, \alpha(w_1))), u_j = u((w_2, \alpha(w_2))), v_k = v((w_1, \alpha(w_1))), v_l = u((w_2, \alpha(w_2))) :$
$(i \geq j \wedge k \geq l) \vee (i \leq j \wedge k \leq l)$

Die Abbildung α bildet also wie a Knoten aus den Mengen U und V in die jeweils andere Menge ab. Die komplizierten Bedingungen sind anschaulich gesehen die Anforderung, dass sich keine zwei Querpfeile überschneiden, wenn man die Knoten von p und q bezüglich der Pfadordnung aufreiht und die Querpfeile einzeichnet, wie in Abbildung 3.10 illustriert ist.

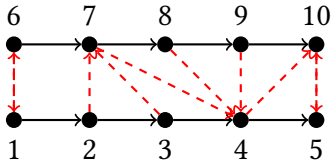
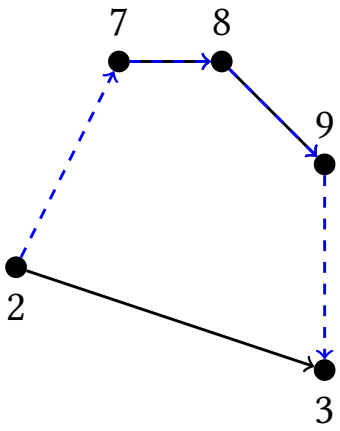
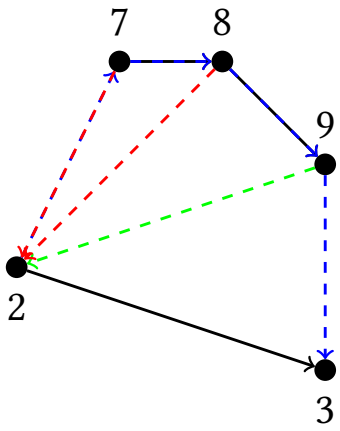


Abbildung 3.10.: Zwei Pfade nebeneinander aufgereiht bezüglich der Pfadordnung, keine zwei Querpfeile dürfen sich hier überschneiden.



(a) Der einzelne Segmentpfad $v_7v_8v_9$ aus dem Zickzackpfad von Abbildung 3.8 mit unterteilenden Kanten (v_2, v_7) und (v_9, v_3)



(b) Grüner Pivotquerpfeil und folgende Zuweisung der Knoten v_7 und v_8.

Abbildung 3.11.: Segmentpfad

Um ein α zu erhalten, ist es ausreichend im Zickzackpfad jeweils alle hintereinander auftretenden Knoten aus U bzw. V zu betrachten.
Die zwischen p und q verlaufenden Kanten des Zickzackpfades unterteilen ihn in Segmentpfade wie in Abbildung 3.11a. Für alle Knoten eines Segmentpfades legen die angrenzenden unterteilenden Kanten den letzten und nächsten Knoten auf der anderen Seite fest. Diese sind auch die Kandidaten für die Querpfeile. Damit können sich von verschiedenen Segmentpfaden ausgehende Querpfeile hinsichtlich der Pfadordnung nicht überschneiden. Das endgültige Auswählen eines Kandidaten können wir deswegen lokal in den Segmentpfaden vornehmen.

Wenn für einen Knoten in einem Segmentpfad ein Querpfeil festgelegt wird, verlieren entweder alle Knoten im Segmentpfad vor oder nach ihm einen Kandidaten, da ein solcher Kandidat einen überschneidenden Querpfeil zu dem gerade festgelegten ergeben würde.
Konkret werden zunächst für einem Segmentpfad alle potenziellen Querpfeile auf ihre Länge untersucht. Der kürzeste wird eingezeichnet und allen Knoten des Segmentpfades, die einen Kandidaten verlieren, wird der verbleibende Kandidat zugewiesen. Auf die mit zwei Kandidaten verbliebenen Knoten wird rekursiv die gleiche Vorgehensweise angewandt, bis alle Knoten des Segmentpfades zugewiesen sind. Der Querpfeil mit der kürzesten Distanz dient somit als Pivotelement wie auch in Abbildung 3.11b zu sehen ist.

Dieser Algorithmus berechnet uns ein mögliches α, bei dem die Längen der Querpfeile tendenziell kurz sind. Mit Hinblick auf die diskrete Fréchet-Distanz ist außerdem eine minimale Länge des längsten Querpfeils wünschenswert.
Sei dazu $|\alpha|$ die Länge des längsten Querpfeils eines α. Dann bezeichne α^* ein α, für das $|\alpha|$ minimal ist. Offensichtlich ist $|\alpha^*|$ eindeutig. Wir werden nun zeigen das $|\alpha^*| = \delta_{dF}$ gilt.

„Perfekte“ Kopplung

Neben der Definition von δ_{dF} (siehe Abschnitt 3.4.2) wurde in [EM94] außerdem ein einfacher Algorithmus vorgestellt um $\delta_{dF}(p, q)$ in $O(n_p \cdot n_q)$ Zeit zu berechnen. Mit dynamischer Programmierung berechnet er ein Coupling L mit $|L| = \delta_{dF}$.
Da L jeden Knoten in mindestens einem Tupel enthalten muss und die Tupel sich bezüglich der Pfadordnung nicht überschneiden, können wir über diesen Algorithmus auch eine Abbildung α gewinnen.

Weiterhin gilt deswegen $\delta_{dF} \geq |\alpha^*|$. Umgekehrt können wir den Querpfeilen eines $|\alpha^*|$ geeignete Tupel zuordnen, sodass diese zu einem Coupling L^* zusammengesetzt werden können, für das gilt $|\alpha^*| = |L^*|$. Mit der Definition von δ_{dF} folgt $\delta_{dF} \leq |L^*| = |\alpha^*|$ und damit $|\alpha^*| = \delta_{dF}$.

Eine weitere Möglichkeit die Knoten der Pfade p und q zu koppeln, bietet die Constrained-Delaunay-Triangulierung.

Constrained-Delaunay-Triangulierung

In der Regel werden sich für reale Straßen die beiden Fahrbahnen nicht schneiden. Für diesen einfachen Fall fällt auf, dass das Einfügen der Querpfeile die Fläche zwischen den Pfaden trianguliert. Auch wenn die beiden Pfade streng genommen keine Fläche einschließen, so implizieren sie visuell einen Korridor. Um formal von einer Fläche sprechen zu können, fügen wir jeweils eine Einschlusskante zwischen den Startknoten und eine zwischen den Endknoten der Pfade ein[6].

Die Idee einer Triangulierung wird auch in [BJW+08] in einem ähnlichen Kontext erwähnt. Es ist zu beobachten, dass lange, dünne Dreiecke auf große δ_{d_F} hinweisen. Die Triangulierung durch Querpfeile ist natürlich nicht die einzig mögliche. Um hohe δ_{d_F} zu vermeiden, sollte die Triangulierung also aus Dreiecken bestehen, die möglichst gleichseitig sind. Die Delaunay-Triangulierung [Del34] erfüllt diese Forderung am besten. In einer Delaunay-Triangulierung enthält der Umkreis eines Dreiecks nur dessen Eckpunkte und keine weiteren sonst. Diese Eigenschaft wird auch als „delaunay“ bezeichnet. Die Delaunay-Triangulierung setzt jedoch voraus, dass die Kanten zwischen den Knoten beliebig gesetzt werden können. Um eine für uns sinnvolle Triangulierung wie in Abbildung 3.12 zu erhalten, müssen wir aber zumindest verlangen, dass die Kanten der Pfade Teil der Triangulierung sein müssen. Sobald solche festen Kanten gefordert sind, ist eine Delaunay-Triangulierung nicht mehr zwingend möglich. Wir fordern deswegen von den Dreiecken in der Triangulierung eine schwächere Eigenschaft: Beschränkt-delaunay [7]. Man stelle sich die geforderten Kanten, die Beschränkungen, als Sichtblockaden vor. Dann ist ein Dreieck genau dann beschränkt-delaunay wenn in seinem Umkreis

[6] Nur temporär für die Analyse und die folgende Triangulierung.
[7] Siehe auch http://doc.cgal.org/latest/Triangulation_2/

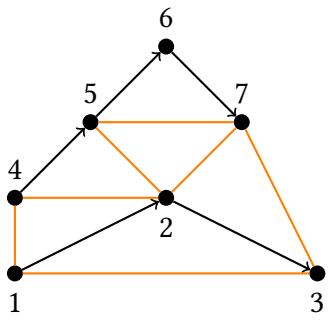


Abbildung 3.12.: Triangulierung

keine weiteren Knoten liegen, die vom Umkreismittelpunkt aus sichtbar sind. Die Bibliothek CGAL bietet für genau dieses Problem eine Constrained-Delaunay-Triangulierung an.

Die Idee ist nun, aus dieser Triangulierung eine Anhängerabbildung ähnlich zu α zu finden. Für jeden Knoten kommen als Anhänger die Knoten auf dem anderen Pfad in Frage, die über eine Kante der Triangulierung mit ihm verbunden sind. Bei mehreren solchen Kandidaten bietet es sich wieder an, den geographisch nächsten als Anhänger auszuwählen. Ähnlich zu α ist diese Abbildung deshalb, weil sie partiell sein kann. Ein Knoten muss in der Triangulierung nämlich nicht zwingend mit einem auf dem gegenüberliegenden Pfad verbunden sein.

Einerseits ergibt es durchaus Sinn, Knoten wie v_6 in Abbildung 3.12 einzeln zu kontrahieren weil sie nur einen Knick in einer der Pfade ohne Gegenstück im anderen darstellen und die Pfade aneinander angeglichen würden. Andererseits kann diese Vorgehensweise die extremsten Knoten ohne Anhänger lassen, die gleichzeitig die Knoten sind, welche vom Top-Down-Ansatz als wichtigste angesehen werden.

Um unerwünschte, aber theoretisch denkbare Triangulierungen auszuschließen fordern wir, dass auch die Einschlusskanten als Beschränkungskanten gelten müssen. Damit bilden die Beschränkungskanten ein Polygon.
Da die Triangulierung nur auf einem planaren Graphen ein für uns sinnvolles Ergebnis liefert, muss das Polygon davor auf Überschneidung getestet werden. Im Falle einer Überschneidung kann dieser Ansatz nicht verwendet werden.
Weiterhin dürfen nur Triangulierungskanten, die innerhalb des eingeschlossenen Polygons liegen als Kandidaten für Querpfeile in Frage kommen.

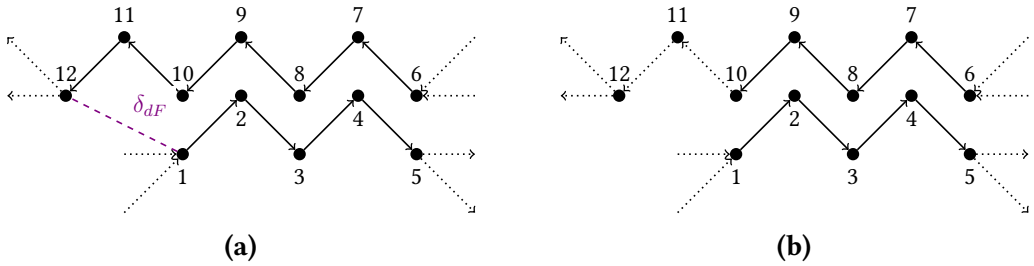


Abbildung 3.13.: Stutzen eines Pfadtupels

3.4.3. Zurechtstutzen

Es hat sich herausgestellt, dass die Pfade gefundener Tupel (c, c') oft wie in Abbildung 3.13 (a) aussehen. Die Pfade starten bzw. enden typischerweise in Grad-3-Knoten wie hier in v_1, v_5, v_6 und v_{12}. Für den Pfad $p = v_1v_2v_3v_4v_5$ und den invertierten Pfad $q = v_{12}v_{11}v_{10}v_9v_8v_7v_6$ gilt dann $\delta_{dF} = dist(v_1, v_{12})$ obwohl die wesentlichen Teile der beiden Fahrbahnen viel enger beisammen liegen.
Da besonders die Knotenkopplung mit dynamischen Programmieren stark von δ_{df} abhängt, können durch ein „Zurechtstutzen“ der Pfade bessere Ergebnisse erzielt werden. Konkret würden wir den Pfad $v_{12}v_{11}v_{10}v_9v_8v_7v_6$ zu $v_{10}v_9v_8v_7v_6$ stutzen und wie in Abbildung 3.13 (b) ein wesentliches kleineres δ_{dF} erhalten.
Wir werden dies umsetzen, indem wir für ein gegebenes Pfadpaar wiederholt $\delta_{dF}(p, q)$ berechnen und den äußersten Knoten abschneiden, wenn $\delta_{dF}(p, q)$ benötigt wird. Nach dem Stutzen ist sichergestellt, dass $\delta_{dF}(p, q)$ nicht an den Endknoten benötigt wird.
Die wiederholte Berechnung kann zwar algorithmisch teuer sein, war für die Praxis aber effizient genug und führte zu einer Verbesserung der Vereinfachungen.

Da für eine objektive Bewertung von Vereinfachungen ebenfalls die gestutzten Pfadpaare wesentlich sind, werden wir bei der Messung auch zurechtstutzen und danach δ_{dF} messen.

3.4.4. Verwendung

Zur Vereinfachung der Pfade eines Tupels (c, c') mit dem Bottom-Up- oder Top-Down-Algorithmus werden die Knoten von beiden Pfaden in einer Min- bzw. Max-Heap-Datenstruktur verwaltet. Die Priorität eines Knoten mit Anhänger berechnet sich aus den Mittelwerten der Fehler und den Anzahlen der induzierten Grenzwechsel des Knoten und seines Anhängers. Wenn ein Knoten wegen höchster Priorität bearbeitet wird, dann wird direkt auch sein Anhänger bearbeitet.
Die Identifikation von Anhängern findet außerdem nur einmal vor Anwendung des Bottom-Up- bzw. Top-Down-Algorithmus statt, sodass Knoten ihre Anhänger währenddessen verlieren können und dann einzeln kontrahiert werden müssen.

4. ILP-Ansätze

Da die verwendeten Algorithmen in Kapitel 3 erstens kontinuierliche Vereinfachungen liefern müssen und zweitens aus Effizienzgründen Heuristiken verwenden, ist offensichtlich, dass diese nicht zwangsläufig optimale Lösungen für einzelne Vereinfachungsstufen produzieren. Wir werden ganzzahlige lineare Programme (ILPs) erstellen, um optimalen Lösungen zu finden. Hauptsächlich werden wir min-ϵ-Probleme lösen, d.h. für eine gegebene Fehlertoleranz eine Vereinfachung mit möglichst wenig Knoten finden. Weiterhin soll Überschneidungsfreiheit der Vereinfachung sichergestellt werden.

Offensichtlich ist das Vereinfachen von einzelnen Fahrbahnen in Form von Polygonzügen nur ein Unterproblem des Vereinfachens von zwei zueinander gehörigen Fahrbahnen. Deswegen wird im Folgenden zunächst das einfachere Problem gelöst und danach das schwierigere, welches auf dem einfacheren aufbaut.

4.1. Einzelne Polygonzüge

4.1.1. Eingabe

Die Eingabedaten sind:

1. Ein Polygonzug bzw. Pfad $p = v_1 v_2 \dots v_n$.
2. Ein Parameter ϵ, der den Grad der Vereinfachung steuert. Je kleiner ϵ, desto weniger Abweichung vom ursprünglichen Polygonzug ist für eine Vereinfachung erlaubt.

Wir fordern außerdem

- $n > 2$, d.h. der gegebene Polygonzug muss eine Mindestlänge haben, ab der eine Vereinfachung sinnvoll wird.
- p darf nicht selbstüberschneidend sein, da wir von unseren Vereinfachungen fordern nicht selbstüberschneidend zu sein und die Eingabe eine Vereinfachung für $\epsilon = 0$ darstellen muss.

4.1.2. Notation

Da wir auch dieses Problem lokal behandeln, sei $V = \{v_1, v_2, ...v_n\}$ und die Menge der anfänglichen Kanten $\mathcal{E} = \{(v_1, v_2), (v_2, v_3), \dots (v_{n-1}, v_n)\}$.

Um alle möglichen Vereinfachungen zu analysieren, werden wir alle zulässigen Kanten, also auch mögliche Abkürzungen betrachten. Dazu sei unsere Menge aller zulässiger Kanten E wobei $\mathcal{E} \subseteq E \subseteq V \times V$.
Beim Aufbauen des Gleichungssystem für das ILP werden wir außerdem Knoten bzw. Kanten mit Variablen assoziieren. Die Variable eines Knoten bzw. einer Kante ist eine Variable x mit der Knoten- bzw. Kantenbezeichnung als Subskript, z.B: $x_{(v_i,v_j)}$ für die Kante (v_i, v_j).

4.1.3. Optimalitätskriterien

Falls eine Vereinfachung eine Abkürzungskante wählt, so wenden wir den erweiterten Lotabstandsfehler an: Der Fehler einer Abkürzungskante ist das Maximum über den Lotabstand der Kante zu allen Knoten, die durch sie abgekürzt worden sind.
Da eine Vereinfachung einen Fehler von ϵ nicht überschreiten darf, sind nur Kanten mit Fehler kleiner ϵ zulässig.

(4.1) $(v_i, v_j) \in E \Leftrightarrow i < j \wedge \forall k \in [i, j] : dist_\perp((v_i, v_j), v_k) \leq \epsilon$

Damit ist klar, dass die Menge der ursprünglichen Kanten $\mathcal{E}$ zulässig ist.

Weiterhin muss sichergestellt werden, dass die Vereinfachung keine Lücken beinhaltet. Deswegen ist der Ansatz von [CvDH14] nicht anwendbar, da dort kein Einfügen von Kanten erlaubt ist.
Wie schon in Kapitel 3 nehmen wir an, dass unser Polygonzug eine Straße in einem größeren Straßennetzwerk darstellt und deshalb v_1 und v_n in jeder Vereinfachung beibehalten werden müssen.

4.1.4. Modellierung

Um das Enthaltensein von Knoten oder Kanten als Teil einer Vereinfachung zu modellieren, werden Variablen benötigt, die nur ganzzahlige Werte annehmen können. Fließkommazahlen sind hier nicht sinnvoll, da ein Knoten bzw. eine Kante entweder Teil der Vereinfachung ist oder nicht. Insbesondere bietet sich die Verwendung von binären Variablen an, die nur die Werte 0 und 1 annehmen können. Eine 1 bedeutet, dass der zugehörige Knoten bzw. die Kante aktiv ist, also Teil der Vereinfachung. Somit wird ein ganzzahliges lineares Programm statt eines reinen linearen Programms (LP) benötigt. Das optimale Lösen eines LP wäre in Polynomialzeit möglich, während dies für ein ILP **NP**-schwer ist.

Die weitere Umsetzung dieses Problems in ein ILP ist jedoch keinesfalls eindeutig oder offensichtlich. Beispielsweise können entweder die Knoten oder die Kanten als binäre Variablen benutzt werden. Auf den ersten Blick scheint es vorteilhaft zu sein, Variablen mit Knoten zu assoziieren, da es davon nur $n = |V|$ gibt. Falls ϵ groß genug ist, so können vom Knoten v_i aber $|V| - i$ Kanten ausgehen. Und damit würde gelten:

$$|E| = \sum_{i=1}^{|V|} |V| - i = \sum_{i=1}^{|V|-1} i = (|V| - 1) \cdot (|V| - 2) \in O(|V|^2) \tag{4.2}$$

Wir würden also mehr Gleichungen brauchen und es würde sich außerdem unvorteilhaft für darauf aufbauende Problemstellungen eignen. In frühere Arbeiten zu diesem Thema [Mil] haben sich außerdem in jeder Hinsicht Kantenvariablen den Knotenvariablen in der Praxis überlegen gezeigt.
Auch hinsichtlich der zu maximierenden Zielfunktion und denen als Parameter gegebenen Restriktionen sind verschiedene Alternativen denkbar.

Ursprünglich war geplant, das Abstandsmaß ϵ zu minimieren, wobei eine Höchstanzahl an Knoten bzw. Kanten für die Vereinfachung vorgeschrieben sein sollte. Weil die Anzahl der Knoten einer Vereinfachung unter allen Umständen genau eins größer ist als die Anzahl der Kanten, ist die Minimierung über die Anzahl der Knoten außerdem austauschbar mit der Minimierung über die Anzahl der Kanten.
Der Vorteil dieser Variante war, dass für einen Polygonzug und eine durch Heuristiken bereits berechnete Vereinfachung die Prüfung auf deren Qualität leicht gewesen wäre. Dazu hätte die Anzahl der besuchten Knoten bzw. verwendeten Kanten der zu überprüfenden Vereinfachung als Parameter dem ILP übergeben und danach die ϵ-Fehler der beiden Vereinfachungen verglichen werden müssen, um ein feingranulares Maß für die Güte der zu überprüfenden Vereinfachung zu erhalten.

Bei der jetzigen Variante, die für ein vorgeschriebenes ϵ die Anzahl der verwendeten Knoten minimiert, besteht das Problem, dass wir bei einem Vergleich der gegebenen Vereinfachung und der optimalen Vereinfachung durch das ILP nur die Anzahl der Knoten vergleichen können. Da die Anzahlen der verwendeten Knoten aber nur natürliche Zahlen sein können und nicht wie ϵ Fließkommazahlen, wird der Vergleich gröber.
Konkret berechnen wir also den ϵ-Fehler der gegebenen Vereinfachung und übergeben diesen als Parameter an unser ILP. Um die gegebene Lösung als nicht optimal zu identifizieren, muss es nun eine Vereinfachung geben, die für gegebenen ϵ-Fehler mit mindestens einem ganzen Knoten weniger auskommt. Falls es so eine Vereinfachung gibt, findet sie das ILP. Falls es solch eine Vereinfachung nicht gibt, können gegebene Vereinfachungen bezüglich dieses Tests als optimal durchgehen, auch wenn mit der gleichen Anzahl an Knoten eine Vereinfachung mit geringerem ϵ-Fehler existieren würde, welche die erste Variante erkennen könnte.
Der große Vorteil der jetzigen Variante ist jedoch, das wegen Bedingung(4.1) viele mögliche Kanten bereits von vorne herein ausgeschlossen werden können und dementsprechend weniger Variablen benötigt werden. Typische Straßen sind eher gestreckt, d.h. Straßenabschnitte bzw.

Kanten konzentrieren sich im Allgemeinen nicht auf die gleiche Umgebung. Da ϵ eine lokale Umgebung definiert, führt dies in der Praxis zu wesentlich weniger als $O(n^2)$ potenziellen Kanten.
In der ersten Variante werden im Gegensatz dazu genau so viele potenzielle Kanten bzw. Variablen benötigt wie in Gleichung (4.2) ausgerechnet.
Die Bedeutung der Anzahl an potenziellen Kanten wird in den darauf aufbauenden Problemstellungen wie z.B. der Vermeidung von Überschneidungen potenziert, da dort eine Bedingung bzw. eine Gleichung für jedes Paar an potenziellen Kanten benötigt werden kann.

4.1.5. Aufbau des ILP

Seien alle ausgehenden bzw. eingehenden Kanten eines Knoten v_i:

$$E_{v_i\_} = \{e \in E : \text{ mit } e = (v, v') \wedge v = v_i\}$$

$$E_{\_v_i} = \{e \in E : \text{ mit } e = (v, v') \wedge v' = v_i\}$$

Dann ist das ILP:

$$\begin{aligned} \text{minimiere} & \sum_{e \in E} x_e \\ \text{so dass gilt} : & \sum_{e \in E_{v_1\_}} x_e = 1 \\ & \sum_{e \in E_{\_v_n}} x_e = 1 \\ \forall v_i \in V \backslash \{v_1, v_n\} : & \sum_{e \in E_{v_i\_}} x_e = \sum_{e' \in E_{\_v_i}} x_{e'} \end{aligned} \tag{4.3}$$

Da wir Kanten als Variablen benutzen, minimieren wir in der ersten Zeile die Anzahl der Kanten statt der Anzahl der Knoten. In der zweiten und dritten Zeile setzen wir den Ausgangsgrad des ersten Knoten bzw. den Eingangsgrad des letzten Knotens auf 1. Damit stellen wir sicher, dass diese Teile der Vereinfachung sind. In der letzten Zeile wird der Ausgangs- und Eingangsgrad aller anderen Knoten gleichgesetzt. Wegen oberen Randbedingungen kann für eine Lösung jeder dieser Knoten entweder genau eine eingehende und eine ausgehende aktive Kante haben oder gar keine. Im ersteren Fall bezeichnen wir den Knoten als indirekt aktiv.
Die Anzahl der erstellten Gleichungen ist linear zur Anzahl der Knoten.

Bis jetzt könnten wir die optimale Lösung von diesem Problem sogar effizient ohne eine ILP finden. Ein Dijkstra-Algorithmus der sich im Graph (V, E) mit einheitliche Kantenkosten den kürzesten Pfad p_{v_1,v_n} sucht, würde damit ebenfalls die optimale Lösung errechnen. Jedoch kommen im Folgenden weitere Bedingungen hinzu, die ein Dijkstra-Algorithmus nicht berücksichtigen kann und für deren Unverletzlichkeit wir ein ILP brauchen.

4.1.6. Verhindern von Überschneidungen

Insbesondere fordern wir von einer Vereinfachung, dass sie sich nicht selbst überschneidet. Dafür ist es nötig, alle Paare der potentiellen Kanten auf Überschneidung zu untersuchen. Dies kann naiv mit $O(|E|^2)$ Vergleichen getestet werden. Es gibt schnellere Alternativen, die mit einer Sweepline arbeiten, wie solche von CGAL oder den Bentley-Ottman-Algorithmus. Für k Überschneidungen braucht dieser nur $O(|E| + k) \cdot \log |E|)$ Zeit. Komplexitätstechnisch ist dies jedoch bei weitem nicht der Flaschenhals, da das Lösen des resultierenden Gleichungssystems ein **NP**-schweres Problem ist.

Für alle gefundenen Überschneidungen wird festgelegt, dass höchstens eine von beiden Kanten aktiv sein darf. Sei ein Tupel (e, e') die Überschneidung der Kanten e und e'. Sei ferner I_E die Menge aller Überschneidungen. Dann werden dem ILP folgende Gleichungen hinzugefügt:

$$\forall (e, e') \in I_E : x_e + x_{e'} \leq 1 \tag{4.4}$$

4.2. Parallele Polygonzüge

Wie bereits in Abschnitt 3.4.2 erläutert, ist die diskrete Fréchet-Distanz δ_{dF} ein gutes Maß für die Parallelität zweier Polygonzüge. Im Weiteren wird sie als Anforderung an die Vereinfachung von zwei Polygonzügen umgesetzt. Zunächst stellen wir ein ILP auf, das testet, ob die diskrete Fréchet-Distanz zweier Polygonzüge einen Wert unterschreitet.

4.2.1. Test auf δ_{dF}

Seien zwei Polygonzüge:

$$p = u_1 u_2 u_3 \dots u_{p_n} = w_{p_1} w_{p_2} w_{p_3} \dots w_{p_n}$$

$$q = v_1 v_2 v_3 \dots v_{q_n} = w_{q_1} w_{q_2} w_{q_3} \dots w_{q_n}$$

und sei W die Menge aller Knoten.

Zu testen ist nun, ob $\delta_{dF}(p, q)$ einen gegebenen Wert η unterschreitet. Dies ist konzeptuell sehr ähnlich zu der Frage, ob eine Vereinfachung existiert, sodass ein gewisses ϵ unterschritten wird.

Es gibt schnellere Algorithmen, um sogar δ_{dF} zu ermitteln, wie wir in Abschnitt 3.4.2 gesehen haben. Jedoch muss ohnehin für eine Vereinfachung zweier Polygonzüge eine Formulierung von δ_{dF} in Form von Gleichungen gefunden werden, da innerhalb dieses ILP ein Algorithmus nicht anwendbar wäre. Die gegebene Problemstellung bietet Gelegenheit die Modellierung

von δ_{dF} zunächst in leichter verständlichem Kontext zu demonstrieren, bevor sie in den komplizierteren eingebettet wird. Leichter verständlich ist sie deshalb, weil die relevanten Knoten schon zu Beginn feststehen.

Da wie in Abschnitt 3.4.2 erläutert, δ_{dF} äquivalent mit $|\alpha^*|$ ist, bietet sich die Verwendung von Querpfeilen an. Diese können als binäre Variablen modelliert werden. Sei F die Menge aller potentiellen Querpfeile, d.h. alle Querpfeile (w, w'), für die gilt $dist(w, w') < \eta$.
Wegen $|\alpha^*|$ = δ_{dF} kann genau dann ein α gefunden werden, wenn $\delta_{dF} \leq \eta$.

Es sei an dieser Stelle darauf hingewiesen, dass prinzipiell $|F| \in O(|W|^2)$ ist, wie es ähnlich schon für $|E|$ galt. Für ein nicht allzu großes η können jedoch analog zu den Kanten viele Querpfeile ausgeschlossen werden, da auch η Lokalität einfordert.

Damit die aktiven Querpfeile eine Abbildung α darstellen, muss außerdem sichergestellt werden, dass von jedem Konten genau ein Querpfeil ausgeht. Weiterhin muss gelten, dass sich die gewählten Querpfeile hinsichtlich der Ordnung in den Pfaden nicht überkreuzen dürfen. Hier kann ähnlich vorgegangen werden wie beim Ermitteln der Überschneidungen. Sogar ein Sweep-Line-Algorithmus kann eingesetzt werden, wenn die Pfadknoten hinsichtlich ihrer Reihenfolge nebeneinander aufgereiht werden und diese neuen Koordinaten statt den geographischen Koordinaten als Eingabe übergeben werden. Wir bezeichnen einen Querpfeil hier mit f, eine Überschneidung hinsichtlich Pfadordnung stellt ein Tupel (f, f') dar und die Menge aller dieser Tupel ist I_F.

Da es hier nur um die Frage geht, ob eine Abbildung α mit der Einschränkung auf die potentiellen Querpfeile existiert, kann das Maximierungs- bzw. Minimierungskriterium vernachlässigt werden. Interessant ist nur, ob das ILP eine Lösung besitzt.

Seien nun analog zu den potentiellen Kanten:

$$F_{w_{i\_}} = \{f \in F : \text{ mit } f = (w, w') \wedge w = w_i\}$$

dann ergeben sich die Gleichungen:

$$\begin{aligned} &\text{minimiere } 0 \\ &\text{so dass gilt :} \\ &\forall w_i : \sum_{f \in F_{w_{i\_}}} x_f = 1 \\ &\forall (f, f') \in I_F : x_f + x_{f'} \leq 1 \end{aligned} \tag{4.5}$$

Bemerkung 4.2.1 (Berechnung von δ_{dF})
Es sei angemerkt, dass dieser Test auch in einen Algorithmus zu Berechnung von δ_{d_F} modifiziert werden kann. Dazu werden alle möglichen Querpfeile als Variablen modelliert, d.h. keiner wird wegen zu großer Länge ausgeschlossen. Zusätzlich werde δ_{d_F} direkt als Variable eingeführt und diese minimiert. Damit δ_{dF} eine Obergrenze für die Länge der Querpfeile darstellt, muss für jeden eine weitere Gleichung erstellt werden. Hinzu kämen also folgende Gleichungen:

$$\begin{aligned} &\textit{minimiere } \delta_{d_F} \\ &\forall f \in F : |f| \cdot x_f \leq \delta_{d_F} \end{aligned} \tag{4.6}$$

4.2.2. Vereinfachung mit δ_{dF}

Es sind nun zwei Polygonzüge zu vereinfachen, sodass die gemeinsame Vereinfachung einen gegebenen ϵ-Fehler nicht überschreitet, sich Kanten nicht überschneiden und die diskrete Fréchet-Distanz δ_{dF} das Maß η unterschreitet. Dazu werden die bisher behandelten Lösungen für die Teilprobleme miteinander kombiniert.

Wir können die Vereinfachung eines Polygonzugs bezüglich eines gegebenen ϵ-Fehlers leicht auf zwei Polygonzüge ausweiten. Dazu stellen wir die ILPs für die getrennten Vereinfachungen auf und fassen deren disjunkte Variablenmengen und Gleichungen zusammen. Die Minimierungsfunktion muss sich nun über alle Kantenvariablen erstrecken.

Das Verhindern von Überschneidungen bezog sich ursprünglich nur auf das Vermeiden der Selbstüberschneidung eines einzelnen vereinfachten Polygonzugs. Damit die Polygonzüge sich nicht gegenseitig schneiden, müssen jetzt außerdem alle potenziellen Kanten des einen Polygonzugs mit allen des anderen auf Überschneidung getestet werden. Damit außerdem die Eingabe weiterhin als mögliche Vereinfachung gelten darf, muss diese schon vorab auf Überschneidungen zwischen p und q getestet werden.

Von vorherigem Entscheidungstest ist zumindest kein Minimierungskriterium zu übernehmen. Trotzdem ist die Anforderung bezüglich δ_{dF} der kritische Punkt. Das Problem besteht darin, dass eine Abbildung α nur für die in der Vereinfachung verwendeten Knoten definiert sein soll und nicht für alle Knoten der Eingabe. Hinzu kommt, dass die verwendeten Knoten nicht direkt als Variablen, sondern nur indirekt über ihren Ein- und Ausgangsgrad modelliert wurden. Die Idee ist nun, die Grade einzelner Knoten bezüglich der Kanten und der Querpfeile zu koppeln: Damit von jedem verwendeten Knoten ein Querpfeil ausgeht, müssen die Ausgangsgrade bezüglich Kanten und Querpfeilen gleich sein. Ausnahmen bilden die Endpunkte der Polygonzüge, von denen keine Kante ausgeht und deswegen der Ausgangsgrad der Querpfeile jeweils gleich eins sein muss.

Der Bildraum von α darf nur verwendete Knoten umfassen. Da α keine injektive Abbildung sein muss, ist jedoch für einen verwendeten Knoten eine beliebig hohe Anzahl an eingehenden

Querpfeilen zulässig. Intuitiv ist die Forderung an jeden Knoten klar: Falls der Grad der eingehenden Kanten mindestens [1] eins ist, so sind beliebig viele eingehende Querpfeilen zulässig, ansonsten keiner. Die Ausnahmen bilden hier die Start- und Endknoten der Polygonzüge. Diese sind immer Teil der Vereinfachung und dürfen somit einen beliebig hohen Eingangsgrad an Querpfeilen haben.

Leider lässt sich obige Forderung nicht direkt in eine Ungleichung für das ILP umsetzen. Stattdessen gewichten wir den Beitrag der eingehenden Kanten in solchen Ungleichungen höher, z.B. mit dem Faktor $|W|$, da wir nicht ∞ einsetzen können. Falls also ein Knoten eine aktive eingehende Kante besitzt, so wird dieser als Ziel von bis zu $|W|$ Querpfeilen zulässig. $|W|$ ist als Faktor völlig ausreichend, da maximal $|W|$ Knoten vorhanden sind, von denen Querpfeile ausgehen können.

Sei nun:

$$F_{\_w_i} = \{f \in F : \text{ mit } f = (w, w') \wedge w' = w_i\}$$

$$\begin{aligned}
&\text{minimiere} \sum_{e \in E} x_e \\
&\text{so dass gilt :} \\
&\sum_{e \in E_{w_{p_1-}}} x_e = 1, \sum_{e \in E_{_w_{p_n}}} x_e = 1 \\
&\sum_{e \in E_{w_{q_1-}}} x_e = 1, \sum_{e \in E_{_w_{q_n}}} x_e = 1 \\
\forall w_i \in W \backslash \{w_{p_1}, w_{p_n}, w_{q_1}, w_{q_n}\} &: \sum_{e \in E_{w_{i-}}} x_e = \sum_{e' \in E_{_w_i}} x_{e'} \\
&\sum_{f \in F_{w_{p_n-}}} x_f = 1, \sum_{f \in F_{w_{q_n-}}} x_f = 1 \\
\forall w_i \in W \backslash \{w_{p_n}, w_{q_n}\} &: \sum_{e \in E_{w_{i-}}} x_e = \sum_{f \in F_{w_{i-}}} x_f \\
\forall w_i \in W \backslash \{w_{p_1}, w_{p_n}, w_{q_1}, w_{q_n}\} &: \sum_{e \in E_{_w_i}} |W| \cdot x_e >= \sum_{f \in F_{_w_i}} x_f \\
\forall (e, e') \in I_E &: x_e + x_{e'} \leq 1 \\
\forall (f, f') \in I_F &: x_f + x_{f'} \leq 1
\end{aligned} \tag{4.7}$$

Es sei darauf hingewiesen, dass das Wählen der Knoten als Variablen statt der Kanten hier eine einfachere Modellierung für die Querpfeile ermöglicht hätte. Dafür wären aber die anderen Anforderungen wie das Überschneiden verkompliziert worden und die sonstigen bisher genannten Nachteile aufgetreten.

[1] Genau gleich eins für eine Lösung.

Wir werden in den weiteren Abschnitten skizzieren wie entsprechende Tests bzw. Vereinfachungen mit der genaueren Fréchet-Distanz δ_F modelliert werden können.

4.2.3. Test auf δ_F

Analog zur diskreten Fréchet-Distanz δ_{dF} wird zunächst ein Entscheidungsalgorithmus konstruiert, um zu testen, ob die Frechet-Distanz δ_F zweier Polygonzüge ein gegebenes η unterschreitet.
Um diese Problem anders zu lösen, gaben Alt und Godau [AG95] einen solchen Entscheidungsalgorithmus in $O(|p| \cdot |q|)$ an, der mit sogenannten Free-Space-Diagrammen arbeitet. Wiederrum gilt, dass ein solcher Algorithmus bei der Vereinfachung von zwei Polygonzügen innerhalb eines ILP nicht anwendbar ist.

Da nun in der Analogie mit Hund und Hundebesitzer beide Teilnehmer ihre Postionen auf beliebigen Punkten der Kanten einnehmen dürfen, erscheint es zunächst so, als ob man zwischen allen Punkten paarweise eine Beziehung herstellen müsste. Dies umzusetzen wäre aber unmöglich, da es unendlich viele Paare gäbe und dies zu einer unendlichen Anzahl an Gleichungen im ILP führen würde.
Glücklicherweise können wir beobachten, dass die Knoten immer noch kritische Wegpunkte beim Ablaufen der Pfade für die Teilnehmer darstellen:

Lemma 4.2.1
Situationen in denen eine maximale Leinenlänge notwendig ist, lassen sich auf solche zurückführen, in denen einer der Teilnehmer auf einem Knoten steht.

Beweis:

Angenommen die maximale Leinenlänge zwischen den Teilnehmern wäre nur dann nötig, während beide auf Punkten von Kanten stehen, also explizit nicht auf Knoten.
Fallunterscheidung:

1. Die Kanten sind parallel:

   a) Die Richtung der Kanten ist gleich, d.h. als Vektoren in der euklidischen Ebene aufgefasst, würden sie sich beide durch einen positiven Faktor in den jeweils anderen skalieren lassen. Dann können beide Teilnehmer mit der gleichen Geschwindigkeit weitergehen. Hierbei bleibt die erforderliche Leinenlänge konstant. Irgendwann muss einer von beiden auf einen Knoten treffen. Also ist zwischen einem Knoten und einem Punkt einer Kante auch die maximale Leinenlänge nötig.

   b) Andernfalls kann nicht ohne eine längere Leine weitergegangen werden, was im Widerspruch zur Annahme steht.

2. Die Kanten sind nicht parallel:

   a) Falls die Kanten in Vorwärtsrichtung auseinander streben, so ist auch hier keine Weitergehen mit dieser Leinenlänge möglich.

   b) Falls die Kanten nicht in Vorwärtsrichtung zueinander streben, werden sie zwangsläufig in Rückwärtsrichtung auseinander streben. Damit hätten die Teilnehmer die jetzigen Positionen nur mit einer längeren Leine erreichen können.

□

Also können wir uns darauf beschränken, modifizierte Querpfeile zu betrachten, die zumindest ihren Ursprung in der Menge der Knoten haben. Der Bildraum der neuen Querpfeile wird aber dennoch die Menge der Punkte auf dem jeweils gegenüberliegenden Polygonzug sein.
Um mit endlich vielen Querpfeilen für einen Knoten auszukommen, werden wir nur für jede Kante des gegenüberliegenden Polygonzugs einen Querpfeil verwalten. Damit sichergestellt ist, dass ein Querpfeil die kürzestmögliche Leine zwischen einem Knoten und einer Kante darstellt, muss der zum Knoten nächste Punkt auf der Kante gefunden werden. Dieser ist damit entweder ein Lotfußpunkt oder einer der beiden Knoten der Kante und wird zum Zielpunkt des Querpfeils. Analog zu oben lassen wir nur Querpfeile zu, deren Länge η unterschreitet. Sei G die Menge aller dieser Querpfeile.

Damit die aktiven Querpfeile eine Abbildung analog zu einem α ergeben, muss noch für jeweils zwei Querpfeile untersucht werden, ob sich diese hinsichtlich der Ablaufreihenfolge in den Polygonzügen unterscheiden. Dies ist nicht mehr so einfach wie vorher, da nun alle Punkte auf den Kanten hinzugekommen sind. Dennoch ist die Reihenfolge bzw. Ordnung der Punkte auf den Pfaden klar, sodass für zwei Querpfeile eine Überschneidung feststellbar ist. Sei die Menge aller überschneidenden Paare I_G. Wir fordern nun, dass für jeden Knoten ein Querpfeil aktiv ist und dass diese sich hinsichtlich der Ordnung in den Pfaden nicht überschneiden.

Sei eine Abbildung nun so gewählt, dass die Länge des längsten aktiven Querpfeils minimal ist. Diese Länge sei l. Es verbleibt zu beweisen, dass $l = \delta_F$ gilt. Dazu ist einerseits zu zeigen dass $l \leq \delta_F$ ist und andererseits als Leinenlänge für ein Ablaufen der Pfade ausreicht, d.h. $l \geq \delta_F$.

Lemma 4.2.2
$l \leq \delta_F$

Beweis:

Die Forderungen an eine Abbildung, die l minimiert sind schwächer als die Forderung an ein kontinuierliches Ablaufen der Pfade ohne Zurücksetzen. Die Forderungen an eine solche Abbildung entsprechen einem Ablaufen der Pfade ohne Zurücksetzen, bei dem die Leine nur benötigt wird, wenn einer der Teilnehmer einen Knoten passiert.

□

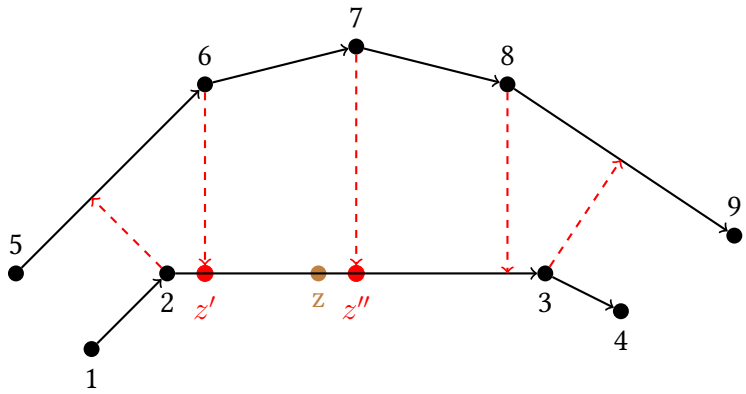


Abbildung 4.1.: Ein Punkt z befindet sich im implizierten Viereck $z'z''v_7v_6$

Wir haben in Lemma 4.2.1 zwar gezeigt, dass die kritischen Wegpunkte die Knoten sind und sich Situationen, in denen eine maximale Leinenlänge notwendig ist, immer auf solche zurückführen lassen, an denen mindestens ein Knoten beteiligt ist. Jedoch ist nicht klar, dass an solchen Stellen ein Querpfeil aktiv ist, da von jedem Knoten nur ein aktiver ausgehen muss. Dennoch reicht l aus:

Lemma 4.2.3
$l \geq \delta_F$

Beweis:

Betrachte dazu einen beliebigen Punkt z auf einem der Polygonzüge wie z.B. in Abbildung 4.1. Dieser Punkt liegt auf einer Kante e_z, deren Start- und Endknoten jeweils einen Querpfeil auf den gegenüberliegenden Polygonzug besitzen. Die Zielpunkte dieser Querpfeile grenzen somit auf dem gegenüberliegenden Polygonzug einen zu e_z korrespondierenden Bereich ein, der im gleichen Zeitintervall abgelaufen wird.

Sei V_B die Menge der Knoten in diesem Bereich, die auch leer sein kann. Von allen $v \in V_B$ muss jeweils ein aktiver Querpfeil ausgehen. Diese Querpfeile müssen ihre Zielpunkte auf der Kante e_z haben, sonst würde es zur Überschneidung hinsichtlich der Pfadordnung kommen. Damit befindet sich z in einem Viereck, das durch die benachbarten Querpfeile impliziert wird. Die Seite S_z des Vierecks auf der z liegt ist ein Kantensegment der Kante e_z. Aufgrund der Konstruktion der Querpfeile werden also S_z und die gegenüberliegende Seite des Vierecks im gleichen Zeitraum abgelaufen. Alle Situationen innerhalb des Vierecks, bei denen eine maximale Leinenlänge notwendig wird, lassen sich wie im Beweis von Lemma 4.2.1 zu einem der benachbarten Querpfeile verschieben.

Da l die maximale Länge aller aktiven Querpfeile ist, reicht l also für alle Situationen aus und damit $l \geq \delta_F$.

□

Aus den Lemmata 4.2.2 und 4.2.3 folgt:

Theorem 4.2.1
$l = \delta_F$

Wir können nun ein ILP für den Test auf δ_F erstellen. Sei nun Z die Menge aller Punkte auf den Polygonzügen und analog zu den unmodifizierten Querpfeilen aus F:

$$G_{w_{i\_}} = \{g \in G : \text{ mit } g = (w, z) \wedge w = w_i\}$$

dann ergeben sich die Gleichungen:

$$\begin{aligned} &\text{minimiere } 0 \\ &\text{so dass gilt :} \\ &\forall w_i : \sum_{g \in G_{w_{i\_}}} x_g = 1 \\ &\forall (g, g') \in I_G : x_g + x_{g'} \leq 1 \end{aligned} \tag{4.8}$$

Bemerkung 4.2.2 (Berechnung von δ_F)
Analog zum Berechnen von δ_{d_F} (s. Bemerkung 4.2.1) kann auch dieser Test in einen Berechnungsalgorithmus für δ_F umgewandelt werden.

4.2.4. Vereinfachung mit δ_F

Die weitere Vorgehensweise ähnelt stark der Vereinfachung mit der diskreten Fréchet-Distanz δ_F. Die potentiellen Kanten und ihre Überschneidungen werden genauso wie dort behandelt. Der wesentliche Unterschied ist, dass der Bildraum der Querpfeile größer geworden ist, sich nämlich von den Knoten des Polygonzugs auf alle Punkte des Polygons ausgeweitet hat. Da der Urbildraum nach wie vor die Menge der Knoten ist, werden weiterhin für jeden Knoten die Ausgangsgrade bezüglich der potentiellen Kanten und der potentiellen Querpfeile miteinander verknüpft.

Um sicherzustellen, dass die Ziele der Querpfeile aktiv sind, ist es jetzt sogar ein Stück einfacher geworden: Ein Punkt ist dann in der Vereinfachung, wenn die Kante aktiv ist, auf der er sich befindet. Da die Kanten als Variablen modelliert sind, sind komplizierte Gleichungen über den Eingangsgrad des Knoten nicht notwendig.

Wenn also eine Kante aktiv ist, dann sollen beliebig viele Querpfeile zugelassen werden, die auf Punkte dieser Kanten zeigen. Hier stellt sich wieder das Problem, dass diese Forderung nicht ideal als Gleichung umgesetzt werden kann und stattdessen wie oben der Gewichtungsfaktor $|W|$ hinzugezogen werden muss. Diesmal wird die Aktivität der Kante mit diesem Faktor gewichtet.

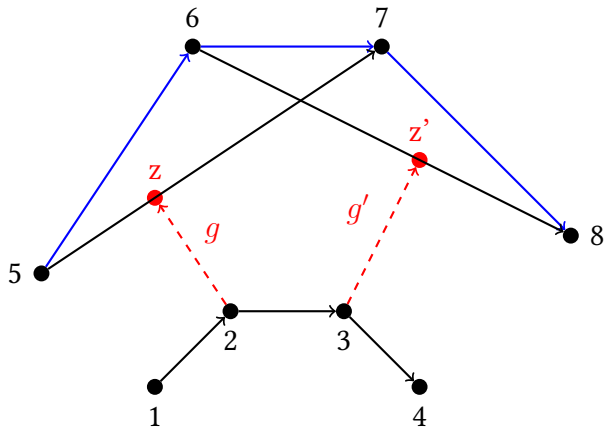


Abbildung 4.2.: Zwei Querpfeile g und g', die nicht gleichzeitig aktiv sein können

Der originale Pfad (blau) kann nicht gleichzeitig durch die Zielkanten der beiden Querpfeile abgekürzt werden.

Um Überschneidungen der Querpfeile hinsichtlich der Pfadordnung festzustellen, ist es zunächst wichtig zu bemerken, dass viele Paare von Querpfeilen nicht gleichzeitig aktiv sein können. Beispielsweise seien g und g' zwei Querpfeile wie in Abbildung 4.2, die beide von Knoten des Polygonzugs p ausgehen und in Punkte z, z' auf potenziellen Kanten des Polygonzug q enden. Befinden sich z und z' auf den Kanten $e = (v_i, v_j)$ bzw. $e' = (v_k, v_l)$ und es gelte $i < k < j$, so können nicht g und g' gleichzeitig aktiv sein, da es schon die Kanten e und e' nicht sein können.

Seien nun die Querpfeile g und g' von den verschiedenen Polygonzügen ausgehend und $g = (v_i, z)$ bzw. $g' = (u_j, z')$ mit $z' \in e' = (v_k, v_l)$. Falls $k < i < l$ so kann unmöglich gleichzeitig die Kante e' aktiv sein und der Knoten v_i indirekt aktiv sein, da er durch die Kante e' abgekürzt würde. Falls er abgekürzt wurde, besitzt er keine aktive ausgehende Kante und damit kann auch g nicht aktiv sein.

Außerdem ist festzustellen, dass keine Pfadordnung zwischen zwei beliebigen Punkten auf den möglichen Polygonzügen definiert ist. Dennoch müssen Überschneidungen von Querpfeilen bezüglich den Pfadordnungen zweier konkreter Vereinfachungspolygonzüge verhindert werden. Wir müssen dies jedoch nicht für alle Paare von Querpfeilen tun, da einige Paare schon aufgrund der Forderung an die Aktivität der Kanten und indirekten Aktivität der Knoten unmöglich sind. Für mögliche Paare bedingt die Aktivität beider Querpfeile die Aktivität von so vielen Knoten und Kanten, dass das Paar für alle dann noch möglichen Kombinationen von Vereinfachungspolygonzügen entweder immer oder nie eine Überschneidung bezüglich deren Pfadordnungen darstellt. Dies kann mit einer ausführlichen Fallunterscheidung gezeigt werden, auf die wir hier verzichten.

5. Ganzheitliche Darstellung

Bis hierhin sind wir davon ausgegangen, dass ein Anzeigegerät für eine vereinfachte Darstellung direkt einen Graphen G_i angezeigt. Wir werden in diesem Kapitel zuerst auf die Kontinuität solcher G_i eingehen. Danach wird sich zeigen, dass die Kontraktion von Knoten mit Mindestgrad drei eine modifizierte Anzeige nötig macht. Im Zusammenhang mit der Anzeige und der Behandlung von Sackgassen werden wir auf ganzheitliche Aspekte für eine Vereinfachung eingehen.

5.1. Kontinuierliche Darstellung

Im Unterschied zu anderen Varianten für den Aufbau von CHs, liefert die Kontraktion in Runden nicht direkt G_i, die sich für eine kontinuierliche Darstellung eignen. Bei Varianten für die jeder Knoten ein eigenes Level hat, könnte für eine beliebige Anzahl n_z an Knoten mit $0 \leq n_z \leq |V_0|$ ein G_i mit $|V_i| = n_z$ angezeigt werden.
Die Kontraktion in Runden ist aber nicht wesentlich einschränkend weil die Kontraktion der Knoten in unabhängigen Mengen stattfand. Da sich deren Kontraktionen nicht gegenseitig beeinflussen, können wir jedem Knoten einer Runde im Nachhinein noch ein eindeutiges Unterlevel zuweisen. Wir können also so rechnen, als ob jeder Knoten ein eigenes Level hat. Damit können wir im Weiteren davon ausgehen, dass wir für ein n_z immer eine Zwischenstufe G'_i mit $|V'_i| = n_z$ aus G^+ generieren können.

5.2. Ausblenden von Abkürzungen

Auch wenn im Vordergrund dieser Arbeit die Kontraktionsreihenfolge stand, um für kartografische Zwecke geeignete Graphen zu gewinnen, so wurden weitere Ansätze verfolgt um dieses Ziel zu erreichen.
Wie in Kapitel 3 erwähnt, bringt die Kontraktion von Knoten mit Mindestgrad 3 oft visuell verstörende Abkürzungskanten hervor, weswegen wir diese Kontraktionen möglichst lange hinausschieben wollen. Typischerweise ist bei der Kontraktion eines Straßengraphen irgendwann einmal der Punkt gekommen, an dem Knoten mit Mindestgrad 3 kontrahiert werden müssen. Deren Kontraktion ist aus kartografischer Sicht insbesondere dann problematisch, wenn die neu eingefügten Abkürzungen den Charakter des realen Straßennetzes untergraben.

Abbildung 5.1.: Graph vor der Kontraktion von v_2 (a) und danach (b). Die Abkürzungen spiegeln das Straßennetz nicht wieder.

Beispielsweise können Abkürzungen den Eindruck vermitteln, dass das Straßennetz viel dichter ist, als es eigentlich der Fall ist. Die Abbildung 5.1 zeigt eine solche Situation.
Damit der verbesserte Dijkstra-Algorithmus aus 2.2.5 ein richtiges Ergebnis liefert, müssen die blauen Abkürzungen eingefügt werden. Aber sie vermitteln hier den Eindruck, dass eine Vielzahl von Straßen verlaufen und dass zum einzigen Zweck, von dem Knoten v_1 auf der linken Seite alle Knoten auf der rechten Seite schnellstmöglich zu erreichen. Typischerweise stellen die Knoten auf der rechten Seite eine stark zusammenhängende Menge dar, zwischen denen schnell gereist werden kann. Ein Umweg über einige dieser Knoten stellt also keinen wesentlichen zusätzlichen Zeitaufwand gegenüber dem kürzesten Pfad dar. Straßenbautechnisch ist dies offensichtlich auch der Grund, nur eine Straße zu bauen, wobei hohe finanzielle Kosten gespart werden können und nur minimal erhöhte Reisezeiten in Kauf genommen werden.

Um das Anzeigen einer solchen Vielzahl von nahezu redundanten Abkürzungen zu vermeiden, können wir schon bei der Konstruktion des CH-Graphen G^+ eingreifen. Aus obigen Überlegungen geht hervor, dass eine Abkürzung umso unnötiger ist, je weniger teuer ein Umweg ist. Für das spätere Ausblenden von diesen relativ unwichtigen Abkürzungen definieren wir die Abbildung $vis : E^+ \to \{0, 1\}$ wobei $\forall e \in E : vis(e) = 1$. Ein Anzeigeprogramm kann dann einfach nur die Kanten e mit $vis(e) = 1$ anzeigen.

Beim Konstruieren von E_{i+1} wie in Abschnitt 2.2.2 beschrieben, rechnen wir, als ob wir jeden Knoten in einer einzelnen Runde kontrahiert hätten. Dazu sei zunächst $E_{temp} = (E_i \setminus E_{removed})$ und die Menge der davon sichtbaren Kanten $E_{temp\_vis} = \{e \in E_{temp} | e \in vis(e) = 1\}$. Das Hinzufügen von allen $e \in E_{shortcuts}$ zu E_{temp} um schließlich E_{i+1} zu erhalten, geschieht schrittweise.
Bevor wir ein solches $e = (u, w)$ in E_{temp} aufnehmen, testen wir ob ein Dijkstra-Algorithmus auf dem aktuellen E_{temp_vis} einen Umweg, also einen Pfad $p_{u,w}$ mit $cost(p) \leq r \cdot cost(e)$ für einen Faktor $r \geq 1$ findet. Falls ein solcher Umweg gefunden wird, setzen wir $vis(e) = 0$ ansonsten $vis(e) = 1$.

Wenn ein Umweg gefunden wird, so verläuft dieser meist über eine stellvertretende Abkürzung e_s, die kurz zuvor bearbeitet wurde, also $e_s \in E_{shortcuts} \wedge vis(e_s) = 1$. Jedoch geben nicht alle $e \in E_{shortcuts}$ visuell gleich gute Stellvertreter ab. In der Abbildung 5.1 ist die mittlere Abkürzung (v_1, v_4) die angemessenste. Falls die oberste Abkürzung (v_1, v_3) als Stellvertreter gewählt würde, könnte es leicht passieren, dass auch die unterste (v_1, v_5) gewählt werden

müsste, wenn sich der Umweg $((v_1, v_3), (v_3, v_5))$ als zu groß herausstellen würde. Wie schon bei der Polygonzugvereinfachung können wir ein Fehlermaß als Entscheidungskriterium hinzuziehen, z.B. den Lotabstand. Falls dieser klein ist, können wir ebenso den erlaubten Suchradius für den Umwege suchenden Dijkstra-Algorithmus verringern, also r kleiner wählen. Damit wird eher kein Umweg für diese Abkürzungen gefunden und sie verbleiben eher sichtbar. Da die Reihenfolge der Einfügungen wichtig ist, sortieren wir alle $e \in E_{shortcut}$ aufsteigend nach Fehlermaß. Damit bekommen passende Abkürzungen mit geringem Fehler wie (v_1, v_4) als erstes die Gelegenheit Stellvertreter für die nachfolgenden zu werden.

Das Problem dieses Ausblendens ist jedoch, dass es zu den besprochenen Inkonsistenzen aus Abbildung 2.2 führen kann, da Anzeige und die Suche nach einem kürzesten Pfad getrennt würden. Aus diesem Grund sehen wir das Ausblenden als eine immer mögliche Nachbearbeitung an und betrachten einen konsistenten Ansatz.

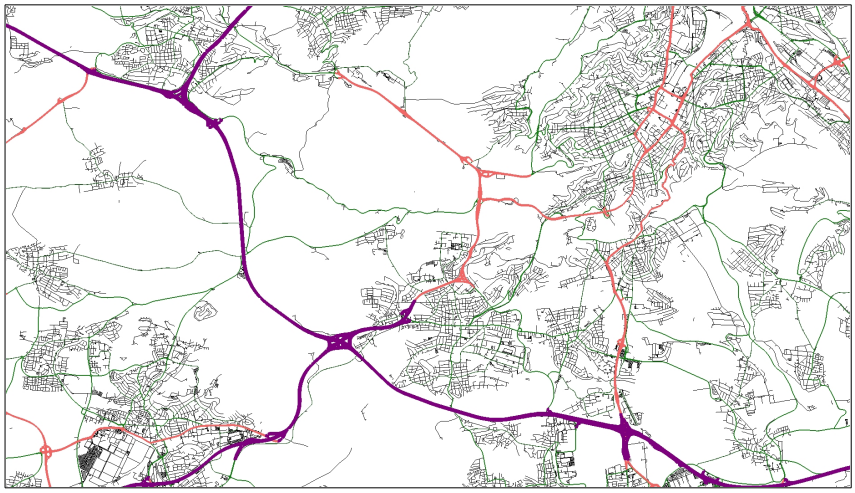
5.3. Entpacken

Wir demonstrieren nun, dass die direkte Anzeige von einem G'_i für die übersichtlichsten Darstellungen nicht ausreichend ist. In Abbildung 5.2 sind verschiedene Graphen als Darstellung eines Teil des Straßennetzwerks von Stuttgart verwendet worden. Alle basieren auf dem gleichen G^+, das durch die Standardkontraktionsstrategie nach Kantendifferenz entstanden ist. In (a) ist der vollständige Graph G_0 abgebildet, der vor allem viele Straßen mit niedrigem Tempolimit (schwarz) anzeigt, die es aber schwierig machen, die Verläufe von wichtigeren Straßen wie z.B. den Autobahnen (lila) zu erkennen.
In (b) ist der Graph G'_i mit einem Promille der ursprünglichen Knoten verwendet. Praktisch ist der Straßengraph nicht wiederzuerkennen und als Vereinfachte Darstellung untauglich.
Wie man deutlich sehen kann, haben die wenigen verbliebenen Knoten in (b) viele und lange Kanten, die in G_0 nicht zu sehen sind und deswegen Abkürzungen sein müssen. So viele Abkürzungen können generell nur bei der Kontraktion von Knoten mit Mindestgrad drei entstehen. Es wurden im Rahmen dieser Arbeit einige Kontraktionstrategien speziell für Knoten mit Mindestgrad drei ausprobiert, aber es war bestenfalls möglich die Zahl solcher Abkürzungen nur gering zu reduzieren und dabei Nachteile einzuhandeln, die in Abschnitt 5.5 erläutert werden. In der Praxis liegt dies daran, dass in späteren Kontraktionsrunden tendenziell mehr Abkürzungen pro Knotenkontraktion hinzukommen.

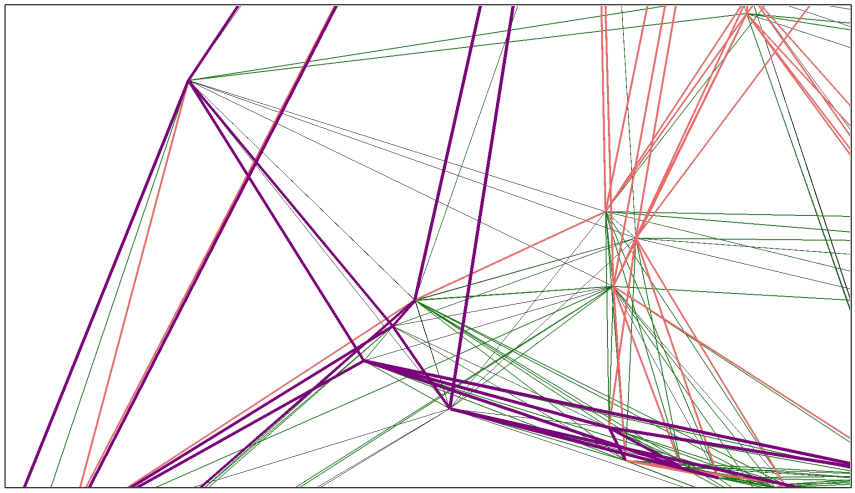
Um trotzdem eine verwendbare Darstellung zu erhalten, können die Abkürzungen von G_i entpackt werden. Ausgehend von G_i können statt einer Abkürzung e mit $unpack(e) = (e_1, e_2)$ die Kanten e_1 und e_2 angezeigt werden. Falls e_1 bzw. e_2 auch Abkürzungen sind, so können auch diese rekursiv weiter entpackt werden. Formal erhalten wir von einem Graphen $G_a = (V_a, E_a)$ durch Entpacken einer Abkürzung $e \in E_a$ einen Graphen $G_b = (V_b, E_b)$ mit $V_b = V_a \cup \{v | e_1 = (u, v)\}$ und $E_b = E_a \setminus \{e\} \cup \{e_1, e_2\}$.

Der dargestellte Graph $\hat{G}'_i$ in Abbildung 5.2 (c) basiert auf dem G'_i in (b), wobei dessen Abkürzungen vollständig entpackt wurden, d.h. alle dargestellten Kanten sind auch in (a) zu finden. Diese Darstellung ist schon überraschend gut, da viele unwichtige Straßen aus (a) verschwunden sind und nicht mehr den Blick auf die wichtigeren verstellen.

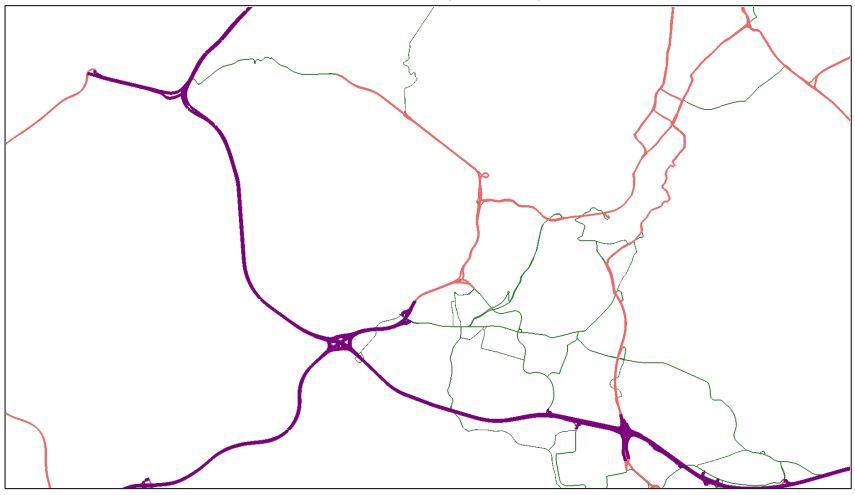
Für eine vereinfachte Darstellung wollen wir aber selbstverständlich nicht alle Abkürzungen entpacken, da dies zu unnötig hochaufgelösten Straßen führt und zur Anzeigezeit algorithmisch zu teuer ist.
In [Sch15] wurden bereits Kriterien festgelegt, wann eine Abkürzung entpackt werden sollte. Dort wurden Abkürzungen mit einer geografischen Distanz unter einem Schwellwert nie und über einem anderen immer entpackt. Für Distanzen zwischen den Schwellwerten wurde nur bei zu kleinen eingeschlossenen Winkel zwischen den Kanten e_1 und e_2 entpackt. Diese Kriterien wurden im Kontext einer performanten Server-Client Anwendung gewählt weil sie sich zur Anzeigezeit schnell berechnen lassen bzw. keine zusätzliche Informationen erfordern.



(a) Anzeige des Graphen G_0



(b) Anzeige des Graphen G_i' mit $|V_i'| = 0.001 \cdot |G_0|$



(c) Anzeige des Graphen $\hat{G}_i'$ mit $|V_i'| = 0.001 \cdot |G_0|$
und vollständig entpackten Abkürzungen

Abbildung 5.2.: Verschiedene Darstellungen eines Ausschnitt aus dem Straßennetzwerk von Stuttgart

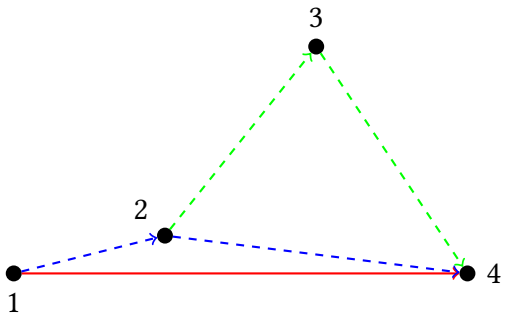


Abbildung 5.3.: Verdeckung des Knoten v_3

Diese Kriterien sind aber zu kurzsichtig, um zuverlässig den Fehler zum Ursprungsgraphen zu minimieren. Da dabei nur die Gegebenheiten einer Entpackoperation betrachtet werden, kann es schnell zum Abbruch der Rekursion kommen. Dabei kann es leicht passieren, das größere Fehler in tieferen potenziellen Rekursionsstufen verdeckt werden. Beispielsweise wird in Abbildung 5.3 der Knoten v_3 verdeckt. Bei der Entscheidung ob die Kante (v_1, v_4) zu (v_1, v_2) und (v_2, v_4) entpackt werden sollte, wird der ebenfalls abgekürzte Knoten v_3 nicht betrachtet, obwohl dieser wichtiger für den Originalpfad als v_2 ist.

Die Entscheidung eine Abkürzung zu entpacken, sollte also immer ihre vollständige rekursive Entpackung berücksichtigen.
In genannter performanter Anwendung wäre ein zur Laufzeit erfolgendes vollständiges Entpacken nicht tragbar, sodass hier für jede Kante stattdessen schon dieser Fehler vorberechnet und gespeichert werden könnte.

5.3.1. Detailgrad

Im konkreten Anwendungsfall legt der Benutzer typischerweise eine Zoomeinstellung fest, für die eine entsprechend detaillierte Darstellung angezeigt wird. Um in weiteren die ganzheitliche Betrachtungen erläutern zu können, definieren wir einen Detailgrad als Tupel (n_z, ϵ_z) und ein im Kontext gegebenes Fehlermaß. Diese führt zu einem Anzeigegraphen, indem wir für eine CH das G'_i mit $|V'_i| = n_z$ konstruieren und dann Kanten mit einem Fehlerwert über ϵ_z entpackt werden.

Die bereitgestellte Visualisierungssoftware wurde modifiziert, um aus einer CH einen Graphen für ein (n_z, ϵ_z) generieren und anzeigen zu können.

5.4. Sackgassen

Zunächst ist die Erkennung von Sackgassen in einem Straßengraph nur ein spezielles Unterproblem der Erkennung von Ketten aus 3.1 und kann dementsprechend gelöst werden. Zu klären sind die Fragen, in welcher Reihenfolge die Knoten innerhalb einer Sackgasse kontrahiert und wie Sackgassen im Kontext des Gesamtgraphen behandelt werden sollten.

In Abschnitt 3 wurde schon erklärt, dass Sackgassen durch sukzessives Kontrahieren ihres Endknotens ohne Hinzufügung von Abkürzungen entfernt werden können. Für alle von uns vorgestellten lokalen Fehlermaße bezüglich der visuellen Vereinfachung wäre dies optimal. Jedoch wäre dies für die Suche nach kürzesten Pfaden am allerschlechtesten, da der Suchraum dann nicht durch das Abkürzen von Knoten reduziert werden kann.
Die Vereinfachung bei Sackgassen zeigt also deutlich im Kleinen, dass sich die verschiedenen Optimierungsziele widersprechen.
Diese einfache Kontraktionsreihenfolge ist wegen ihrer unmittelbar ersichtlichen Auswirkungen uninteressant für eine empirische Auswertung. Stattdessen werden wir auf die Sackgassen den gleichen Algorithmus anwenden wie auf die anderen Ketten.

Offensichtlich verlaufen kürzeste Pfade nur über die Kanten einer Sackgasse, wenn sich Start- oder Endknoten darin befinden. Dies bedeutet, dass die Kontraktion aller Knoten einer Sackgasse also keine Auswirkung auf den übrigen Graph hat. Für die Suche nach kürzesten Pfaden sollte die Sackgasse als ganzes also möglichst schnell entfernt werden [1]
Weiterhin ermöglicht das Entfernen von Sackgassen das Entstehen von weiteren Sackgassen in höheren G_i's, die dann ebenfalls entfernt werden können. Das Entfernen der Sackgassen kann solange fortgesetzt werden, bis nur noch Knoten mit Grad größer eins übrig sind.

Es bietet sich an, Sackgassen zu entfernen, weil diese einfache erkennbare Strukturen sind und ihr Entfernen strikt lokale Auswirkungen hat. Eine ähnliche Struktur stellen Ketten dar, für die Start- und Endknoten identisch sind. Anschaulich stellen sie also Rundkurse dar, die an einem Knoten mit dem übrigen Graphen verknüpft sind. Bezüglich kürzester Pfade gilt für Rundkurse dasselbe wie bei Sackgassen. Folglich können wir obige Überlegungen auch auf Rundkurse anwenden bis der Graph keine mehr enthält.

Auch wenn eine solche sukzessive Entfernung für die Suche nach kürzesten Pfaden und unsere lokalen Fehlermaße optimal ist, so indiziert sie schon ein Problem bei der Gesamtdarstellung des Graphen.

[1] Nach der trivialen Kontraktion von Grad-0-Knoten.

5.5. Gleichmäßige Kontraktion

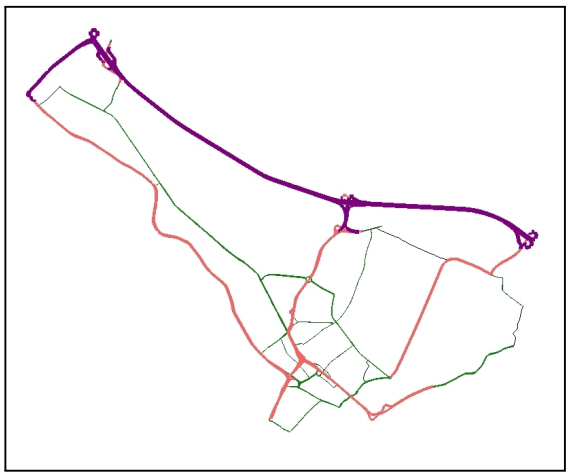
Falls Teile des Graphen eine Baumstruktur aufweisen, wie z.B. das Straßennetz in einem nur einseitig erreichbaren Bergtal, wird mit obiger Vorgehensweise dieser gesamte Teilgraph entfernt. Damit kann es passieren, dass großflächige Bereiche der Karte nicht angezeigt während andere detailliert dargestellt werden.
Ein ähnliches Problem besitzt schon die Kontraktionsstrategie der Kantendifferenz im Hinblick auf die Optimierung der Berechnungszeit für kürzeste Pfade. Wenn stur hintereinander immer der Knoten mit der geringsten Kantendifferenz kontrahiert wird, kommen die gleichen Probleme auf, die wir im ersten Ansatz für Sackgassen gesehen haben. Die gleichzeitige Kontraktion von Knoten in unabhängigen Mengen ist ein Schritt zur Lösung dieses Problems.
Eine allgemeine Problematik resultiert aus der Tatsache, dass Knoten am Rand eines Straßennetzwerksgraphen tendenziell eine geringere Kantendifferenz aufweisen als solche im Zentrum. Da Knoten am Rand früher kontrahiert werden, schrumpfen Graphen bei ihrer Vereinfachung zusammen.

In Abbildung 5.4 ist dieser Effekt demonstriert. Für (a) wurde nach Kantendifferenz kontrahiert. Um den Schrumpfeffekt hervorzuheben wurden in (b) zuerst Sackgassen und Rundkurse entfernt, dann Ketten vereinfacht und schließlich mit kleineren unabhängigen Mengen als in (a) nach Kantendifferenz kontrahiert. Außerdem wurde der Straßengraph von Bremen gewählt, da solche kleinen Graphen verhältnismäßig viel Rand besitzen und der Effekt stärker wird.
Man kann sehen, dass der Graph von (b) von der geographischen Ausdehnung her nur ein kleiner Teil von dem in (a) darstellt.
Auch in [Sch15] wurde dieser Effekt bei der Standardkontraktionsstrategie nach Kantendifferenz für sehr kleine n_z beobachtet.

Es sind jedoch Anwendungsfälle ohne die Randproblematik denkbar. Beispielsweise würde eine Modellierung der globalen Schifffahrtsrouten in einem Graphen ohne Rand resultieren.

Da die hier untersuchten Straßennetzwerke nur Ausschnitte von größeren sein werden, also beispielsweise das deutsche als Teil des europäischen, müssen an Grenzen die Straßen abrupt enden. Insbesondere wenn diese Straßen Autobahnen sind, kann die Entscheidung, diese als wichtig einzustufen, zwischen den Extremen wanken:
Wenn vor allem Straßen angezeigt werden sollen, auf denen kürzeste Pfade innerhalb des Graphen verlaufen, so können halbe dargestellte Autobahnen wie alle anderen Sackgassen früh entfernt werden. Falls ein Benutzer grob das Netzwerk der Autobahnen kennt, kann andererseits selbst die Anzeige einer halben Autobahn ihm helfen, sich schneller zu orientieren.

Um ganzheitlich zu vereinfachen, können praktisch je nach Vorliebe beliebige Kriterien zur Bestimmung der Kontraktionsreihenfolge umgesetzt werden. Beispielsweise kann mithilfe von Voronoi-Diagrammen geografisch gleichmäßig kontrahiert werden [GSSD08]. Eine primitive Strategie, um sich den Rand zu erhalten wurde in [Sch15] vorgeschlagen: Einzelne Knoten von Autobahnen am Rand werden fixiert, sodass sie als Allerletzte kontrahiert werden.



(a) Kontraktion nach Kantendifferenz

(b) Kontraktion durch Sackgassenentfernung, Kettenvereinfachung und Kantendifferenz mit kleinen unabhängigen Mengen

Abbildung 5.4.: Zwei Darstellungen von Bremen
Beiden dargestellten Graphen liegt jeweils ein Detailgrad ($n_z = 0.0002 \cdot |V|, \epsilon_z = 0$) zugrunde. $\epsilon_z = 0$ bedeutet, dass immer alle Abkürzungen entpackt werden.

Für eine ganzheitliche Vereinfachung wurden in dieser Arbeit mit einer Reihe von Maßen für die visuelle Wichtigkeit von Knoten mit Mindestgrad 3 experimentiert. Beispielsweise wurde ähnlich zur Kantendifferenz die Kontraktion von Knoten simuliert und dann berechnet, ob die hinzukommenden Abkürzungen große geografische Distanzen mit relativ kleinen Kosten überwinden. Falls dies zutraf, wurde der Knoten als wichtiger eingestuft und später kontrahiert.

Aufgrund der sehr unterschiedlich erzeugten Darstellungen und den erwähnten subjektiven Aspekten, scheint eine objektive Messung von ganzheitlichen Qualitäten jedoch unmöglich, weswegen wir uns auf die lokalen konzentrieren werden.

6. Auswertung

Wir werden in diesem Kapitel die Qualität unserer entwickelten Algorithmen zur Vereinfachung messen. Dazu werden wir zuerst Messmethoden vorstellen, die topologische Inkonsistenzen besser erfassen können als die Heuristik in Abschnitt 3.3.3. Danach wird der Aufbau für die empirische Qualitätsmessung von Kontraktionsreihenfolgen beschrieben und begründet. Es folgen schließlich die Ergebnisse und deren Interpretation.

6.1. Grenzwechseltest

Wir betrachten eine Abkürzung (v_1, v_n) und ihre vollständige Entpackung, den Pfad p, aus dem sie also hervorgegangen ist. Offensichtlich ist es unter kartografischen Gesichtspunkten wichtig, dass geografische Punkte[1] nach der Vereinfachung durch die Abkürzung auf der gleichen Seite des Pfades liegen wie davor. Leider lässt sich dieses intuitive Verständnis nicht direkt in formale Anforderungen formulieren.
Falls wir unendlich Ressourcen zur Verfügung stehen hätten, könnten wir den Graph ganzheitlich betrachten und feststellen, wann immer ein geografischer Punkt nach einer Vereinfachung in eine andere Facette des Graphen wechselt. Unsere Algorithmen werden aus Effizienzgründen nur mit einer lokalen Umgebung arbeiten.

Sei jetzt also wie oben nur der Pfad p gegeben und sei zunächst der Einfachheit halber angenommen, dass p sich nicht selbst schneidet und sich außerdem kein Knoten von p „hinter“ den Start- oder Endknoten befindet, d.h. alle Lotfußpunkte der Knoten von p zu (v_1, v_n) liegen auch auf (v_1, v_n). Dieser einfache Fall wird auch in der Literatur betrachtet, wenn Punkte zu detektieren sind, die die Seiten eines Polygonzugs wechseln. Da dies in der Kartografie besonders bei Grenzen Anwendung findet, werden solche Seitenwechsel als Grenzwechsel bezeichnet. Ein solcher einfacher Pfad induziert zunächst keine neuen Facetten in der euklidischen Ebene, die wir als rechte oder linke Seite verwenden könnten.

[1] Wir betrachten hier Knoten, die nicht in p sind, nur als geografische Punkte, da für den Grenzwechseltest ihre Kanten irrelevant sind.

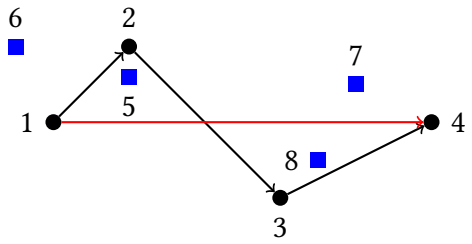


Abbildung 6.1.: Grenzwechsel von Punkten

Zwei der blauen Quadrate werden nach Vereinfachung durch die rote Abkürzung die Straßenseite wechseln.

Wenn wir jedoch zusätzlich zu p die Abkürzung (v_1, v_n) einzeichnen, so entstehen neue Facetten wie in Abbildung 6.1 dargestellt. Die in den Facetten befindlichen Punkte v_5 und v_8 befinden sich nach der Vereinfachung nach visuellen Kriterien auf der „anderen“ Seite der Straße. Es ist natürlich im formalen und topologischen Sinn problematisch von der „anderen“ Seite zu sprechen, da p und seine Vereinfachung (v_1, v_n) einzeln keine rechten oder linken Facetten definieren, zwischen denen ein Punkt wechseln kann. Da jedoch die Vereinfachung vor allem visuellen Anforderungen genügen muss, ist es angemessen die Knoten in den Facetten als grenzwechselnd aufzufassen.

Um eine Mitgliedschaft eines Punktes s in diesen Facetten zu testen, bietet sich ein simpler Raycasting Algorithmus nach [Hai94] an:

6.1.1. Raycasting

Hierbei wird wie in Abbildung 6.2 ein Punkt r erstellt, der einfach nur außerhalb des Graphen liegen muss, sodass er garantiert nicht in eine Facette fallen kann. Die Punkte s und r definieren die Strecke $\overline{s,r}$, den „Ray“, also den Strahl. Nun werden alle Kanten von p und (v_1, v_n) auf Schnitt mit $\overline{s,r}$ getestet. Falls nun ein Punkt innerhalb einer Grenzwechselfacette liegt, so ist die Anzahl der Schnitte gerade, andernfalls ungerade.

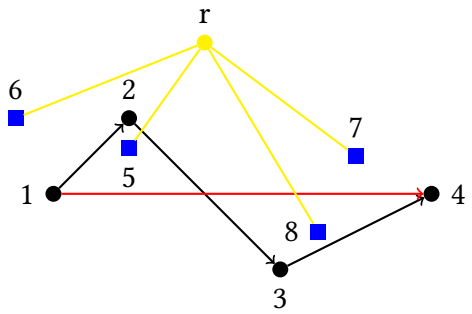


Abbildung 6.2.: Raycasting Test auf den Graph von Abbildung 6.1

Um nun die Anzahl der Grenzwechsel für (v_1, v_n) zu zählen, wird folgendermaßen vorgegangen:
Zunächst einmal können für einen Grenzwechsel nur Punkte in Frage kommen, die in einem achsenorientierten Fenster liegen, das den Pfad p vollkommen enthält. Sei die Menge dieser Punkte $\mathcal{S}$. Dann müssen alle $s \in \mathcal{S}$ mit allen Kanten in p geschnitten werden. Dies benötigt $O(|\mathcal{S}| \cdot n)$ Zeit.

Es sei angemerkt, dass die in Frage kommenden Punkte sogar noch weiter eingegrenzt werden können [Saa99]. Um einen Grenzwechsel verursachen zu können, muss ein $s \in \mathcal{S}$ in der konvexen Hülle von p liegen. Sei k die Anzahl der Knoten auf dem Rand der konvexen Hülle, so erfordert deren Berechnung nach Chans Algorithmus $O(n \log k)$ Zeit. Weiterhin müsste für alle $s \in \mathcal{S}$ das Enthaltensein in der konvexen Hülle geprüft werden, z.B. auch durch ein Raycasting Algorithmus. Alle nicht enthaltenen Knoten würden aus der Menge $\mathcal{S}$ entfernt. Diese Vorverarbeitung kann sich lohnen, wenn ein hoher Anteil der Knoten im Fenster nicht in der konvexen Hülle ist und k im Vergleich zu n klein ist.

Leider ergeben sich einige auf den ersten Blick nicht offensichtliche Schwierigkeiten mit dem Raycasting. Es kann passieren, dass $\overline{s, r}$ ein Knoten des Pfades p enthält. Dies könnte natürlich nur für den unwahrscheinlichen Fall dreier kollinearer Punkte entstehen. Solche Probleme können durch Einbeziehung der Nachbarknoten bezüglich des Pfades gelöst werden, jedoch sind Situationen vorstellbar, in denen auch solche Ansätze beliebig weit ausgehebelt werden. Beispielsweise könnte $\overline{s, r}$ eine oder sogar mehrere Kanten von p vollständig enthalten. Dies würde aber mindestens 4 kollineare Punkte erfordern.

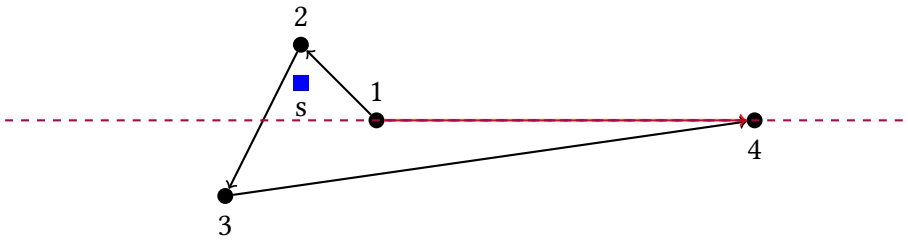


Abbildung 6.3.: Grenzwechsel „hinter“ Abkürzung

Es ist unklar ob s hier einen Grenzwechsel erfährt

Außerdem ist dieser Algorithmus nicht besonders robust, da er nur für den beschriebenen einfachen Fall zuverlässig ist. Wenn sich nun einige Knoten von p „hinter“ Start- oder Endknoten befinden, dann werden eingeschlossene Punkte in den anliegenden Facetten als Grenzwechsel detektiert so wie in Abbildung 6.3 zu sehen ist. Bei einigen dieser Punkte könnte argumentiert werden, dass sie nach der Vereinfachung immer noch auf der „gleichen“ Seite liegen wenn man (v_1, v_n) zu einer unendlichen Geraden verlängert. Aber auch aus menschlicher Sicht ist ohne Wissen um das anschließende Verkehrsnetz die Beziehung des Punkts zu einer Seite des Pfades bzw. seiner Vereinfachung nicht vollkommen eindeutig. Deswegen ist es durchaus vertretbar hier im Zweifel einen Grenzwechsel zu zählen.

(a) **(b)**

Abbildung 6.4.: Fragliche Grenzwechsel bei Selbstüberschneidung

Problematischer sind Überschneidungen im Pfad. Es kann beispielsweise zu Situationen wie in Abbildung 6.4a kommen.
Hierbei befindet sich der Punkt s beim Ablaufen des Pfades p von v_1 nach v_n immer auf der rechten Seite. Beim Ablaufen der Abkürzung (v_1, v_n) jedoch auf der linken. Trotzdem wird s nicht als Grenzwechsel erkannt, da ein ausgehender Strahl immer eine gerade Anzahl von Kanten schneidet.

Angesichts dieser Unzulänglichkeiten wurde das Erkennen von Grenzwechseln anders umgesetzt:

6.1.2. Eingeschlossene Facetten

Wenn alle Punkte in eingeschlossenen Facetten als grenzwechselnd gezählt werden, können Richtig-Negative wie in Abbildung 6.3 ausgeschlossen werden. Natürlich provoziert eine solche Vorgehensweise förmlich das Auftreten von Falsch-Negativen, also das Feststellen von Grenzwechseln bei Punkten, die eigentlich auf der gleichen Seite bleiben.

In Abbildung 6.4b würde der Punkt s als grenzwechselnd erkannt werden, da er sich innerhalb einer beschränkten Facette befindet. Jedoch könnte argumentiert werden, dass er sich beim Ablaufen des Pfades oder der Abkürzung s immer links befindet und ergo kein Grenzwechsel stattfindet.

Obige Argumentation ist nur ein Beispiel unter vielen in der Kartografie, in der die Bewertung einer Vereinfachung stark subjektiv sein kann.
Allgemein stellen die Vereinfachungen von selbstüberschneidenden Pfaden durch eine einzelne Kante immer drastische Einschritte in die Topologie eines Graphen dar. Von dieser Sichtweise ausgehend detektiert diese Vorgehensweise nicht nur alle Punkte, die einen Grenzwechsel erfahren sondern auch alle, die topologisch beeinträchtigt werden.

Um den Graphen in eingeschlossene Facetten aufzuteilen, bieten sich Triangulierungen an, wie in Abbildung 6.5 illustriert ist. Für eine konkrete Umsetzung wird zunächst eine beschränkte Triangulierung der CGAL-Bibliothek angewendet wie in Abschnitt 3.4.2. Dabei werden die Kanten von p und (v_1, v_n) als Beschränkungskanten festgelegt. Falls sich zwei Beschränkungskanten überschneiden, so werden diese jeweils in zwei Beschränkungskanten aufgeteilt und der

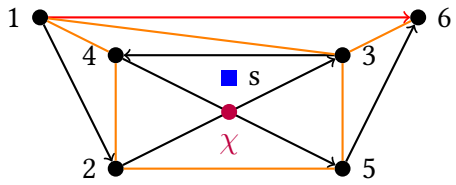


Abbildung 6.5.: Triangulierung des Grapen aus Abbildung 6.4a

Schnittpunkt (in Abbildung 6.5 χ) wird dem zu triangulierenden Graphen hinzugefügt. Nach Erstellung der Triangulierung T sind alle bisherigen Facetten in Dreiecksfacetten aufgeteilt worden. Es verbleibt eine unbeschränkte Facette, die als einzige keine Dreiecksfacette ist. Diese Facette wird mit dem Attribut *außen* gekennzeichnet. Alle Facetten, die an diese über eine Nichteinschränkungskante benachbart sind, werden ebenfalls als *außen* gekennzeichnet. Rekursiv setzt sich dieser Prozess mit den benachbarten Facetten fort. Es ist leicht zu sehen, dass danach alle Dreieckfacetten, die aus den ursprünglichen beschränkten Facetten entstanden sind, nicht als *außen* markiert worden sind. Konkret wären im Abbildung 6.5 nur die unbeschränkte Facette und (v_2, v_5, χ) als *außen* markiert worden.

Die Triangulierung T hatte nicht nur den Nutzen algorithmisch innen und außen zu markieren, sondern wird uns auch helfen die zu untersuchenden Punkte in den Facetten zu lokalisieren. Naiv könnten für jeden Punkt nacheinander alle Facetten angeschaut werden bis jene gefunden wird, die ihn enthält, um dann zu entscheiden ob er sich innen oder außen befindet.

CGAL bietet für Triangulierungen zusätzlich eine Hierarchie an, die es ermöglicht, Punktlokalisierungsanfragen in logarithmischer Zeit zu beantworten. Die klassische Triangulierungshierarchie ist die Kirkpatrickhierarchie [Kir83]. Eine Kirkpatrickhierarchie für eine Triangulierung T mit n Knoten ist eine Folge von Triangulierungen $T_0, T_1, \ldots T_h(n)$, wobei $T_0 = T$ und $h(n)$ die Höhe der Hierarchie darstellt. Dabei ist $h(n) \in O(\log n)$. Außerdem ist jede Facette bzw. Region R von T_i mit einer Region R' von T_{i+1} verbunden falls $R \cap R' \neq \emptyset$. Die Knotenmenge von T_{i+1} ist immer eine echte Teilmenge der Knotenmenge von T_i. Ähnlich zur CH wird eine unabhängige Menge von Knoten entfernt um von T_i nach T_{i+1} zu gelangen. Es ist möglich die Triangulierungshierarchie in $O(n)$ Platz zu speichern. Bei Punktlokalisierungsanfragen wird der Punkt s zunächst in $T_h(n)$ lokalisiert. Da s sich nur in verknüpften Regionen der niedrigeren Level der Kirkpatrickhierarchie befinden kann, wird die Suchzeit stark reduziert und ist letztendlich in $O(\log n)$ möglich. Außerdem kann die Kirkpatrickhierarchie in $O(n)$ konstruiert werden.

CGAL verwendet eine ähnliche Triangulierungshierarchie [2], die ebenfalls eine Folge von Triangulierungen speichert, deren Knotenmengen Teilmenge voneinander sind. Die Punktlokalisierung geschieht hier auch vom höchsten Level ins niedrigste. Im Unterschied zur Kirkpatrickhierarchie werden hier die Verknüpfungen zwischen den Knoten gespeichert.

[2] Siehe http://doc.cgal.org/latest/Triangulation_2/

Dort wird zuerst im höchsten Level der geografisch nächste Knoten zu s gesucht. Im Level darunter wird ausgehend von dem gefundenen Knoten wiederum der nächste Knoten zu s gesucht. Sobald im untersten Level der Hierarchie der nächste Nachbarknoten gefunden wurde, kann die angrenzende Facette, die s enthält zurückgegeben werden.

Die theoretischen Schranken dieser Triangulierungshierarchie sind nicht so gut wie die der Kirkpatrickhierarchie, sie hat sich aber auf praktischen Daten bewährt [Dev98].

Wie schon beim Raycasting Algorithmus müssen nur die Punkte lokalisiert werden, die in einem Fenster sind, das auch den Pfad enthält. Ausgehend von der Anzahl der Punkte in diesem Fenster kann die Zeitkomplexität der Lokalisierungsanfragen noch weiter optimiert werden. Für sehr wenige Punkte könnte die Konstruktion einer Hierarchie in der Praxis mehr Kosten verursachen als die Ersparnis beim Lokalisieren gegenüber dem naiven Einsatz einbringt. Umgekehrt wäre es für sehr viele zu lokalisierende Punkte sogar besser die Triangulierung weiter zu augmentieren. Es wurde gezeigt, dass die vorgestellten Hierarchien am besten funktionieren wenn sie auf einer Delaunay-Triangulierung aufbauen [Dev98]. Intuitiv kann man sich vorstellen, dass eine beliebige Triangulierung aus aufeinandergestapelten, langen und dünnen Dreiecken wie in Abbildung 6.6 bestehen kann und dann kein Dreieck mehr als zwei benachbarte Dreiecksfacetten besitzt, in denen die Punktlokalisierung fortgesetzt werden kann. Die Delaunay-Triangulierung hingegen versucht solche Unregelmäßigkeiten zu vermeiden und ist deswegen bei der Punktlokalisierung weniger eingeschränkt. Allerdings ist eine Delaunay-Triangulierung mit einem Aufwand von $O(n \log n)$ verbunden [Kir83]. In diesem speziellen Fall käme diese aber nicht direkt zur Anwendung, da hier eine Constrained-Delaunay-Triangulierung zu konstruieren wäre.

Um in keinem der Fälle eine schlechte Performanz zu haben, wurde schließlich eine in Linearzeit erstellbare Triangulierung verwendet, die zwar die Beschränkungskanten berücksichtigt, aber nicht die Form ihrer Dreiecke optimiert. Diese wurde dann mit der Triangulierungshierarchie von CGAL kombiniert.

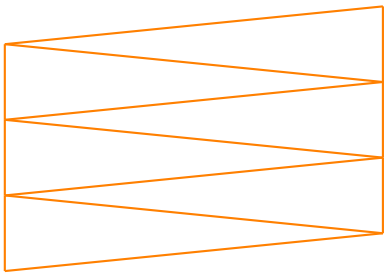


Abbildung 6.6.: Ungünstige Triangulierung für Punktlokalisierung

6.2. Messaufbau

Um unsere Algorithmen zu bewerten, soll nun ein empirischer Vergleich zwischen verschiedenen CHs von einem Graphen G vorgenommen werden.
Wir werden dazu für jede CH einige Kennzahlen berechnen, die ihre Qualitäten bezüglich unseren Anforderungen widerspiegeln.

Die Beschleunigung für das Suchen von kürzesten Pfaden kann relativ einfach verglichen werden, da für jede CH die gleiche Menge an Knoten V zugrunde liegt und hier die benötigte Zeit als einziger Wert gemessen werden kann.

Es ist wesentlich schwieriger, die Qualität einer CH als kontinuierliche Graphsimplifizierung zu messen. Praktisch werden wir Stichproben bzw. Schnappschüsse mit einigen Detailgraden machen. Um anwendungsnahe Messungen anzustellen, müssen wir dabei Kanten entpacken, wie in Abschnitt 5.3 erläutert worden ist. Wir müssen darauf achten, dass das Entpacken unsere Messungen möglichst wenig beeinflusst.
Wie wir in Abschnitt 5.5 gesehen haben, können außerdem verschiedene CHs sehr unterschiedliche Darstellungen liefern. Es wurde beobachtet, dass sich das Schrumpfen von Graphen im Allgemeinen günstig auf die lokal ermittelbaren Fehler auswirkt. Dies liegt daran, dass nach dem vollständigen Entfernen geographisch beschränkter Teilgraphen dort keine Kanten mehr bestehen, die einen lokalen Fehler induzieren könnten. Da wir die Untauglichkeit geschrumpfter Graphen als ganzheitliche Darstellungen festgestellt haben, scheinen möglichst ganzheitliche und zugleich lokal fehlerfreie Vereinfachungen unvereinbar.

6.2.1. Kontraktionsstrategien

Wir stellen hier kombinierte Kontraktionsstrategien für lokale und ganzheitliche Vereinfachungen vor und erläutern, welche sich für die Messungen am besten eignet.

Dynamisch

Zuerst wurde ein dynamischer Ansatz implementiert, bei dem in derselben Kontraktionsrunde Knoten aus Ketten und solche mit einem Mindestgrad von 3 kontrahiert wurden. Die Intention war, dass sich kleine Teilgraphen ähnlich wie Sackgassen auflösen und mit wiederholten Kettendetektionen im Abstand einiger Runden größere Ketten gefunden werden. Diese konnten dann ganzheitlicher vereinfacht werden als die zuletzt gefundenen.
Die unterschiedliche Behandlung der verschiedenen Knoten und deren wechselseitige Beeinflussung führte jedoch zu einem stark nichtdeterministischen Charakter der Gesamtkontraktionsreihenfolge. Falls ein Algorithmus zum Vereinfachen der Kette ausgetauscht wurde, konnte dies indirekt so starken Einfluss auf das Wählen der Kontraktionsreihenfolge von Knoten mit Mindestgrad 3 haben, dass die entstehenden CHs zu verschieden für einen aussagekräftigen Vergleich waren.

Das erstmalige Abweichen zwischen zwei Kontraktionsreihenfolgen hat überdies einen kaskadierenden Effekt: Beispielsweise kann das Auslassen einer Kontraktion in einer Runde zu einer völlig anderen unabhängigen Menge in der nächsten Runde führen.
Leider interessieren uns vor allem die hohen Level mit nur noch sehr wenigen Knoten, bei denen sich dieser Effekt am stärksten auswirkt.

Um für Messungen zumindest ähnliche Graphen zu erhalten, wird der gesamte Kontraktionsprozess stattdessen in Phasen unterteilt.

Phasen

Anders als obige dynamische Kontraktionsstrategie ist die Phasenstrategie statischer.

In der ersten Phase werden wir wie in Abschnitt 5.4 beschrieben, zunächst alle Sackgassen und Rundkurse entfernen [3]. Dies schließt mehrmaliges Detektieren von neu entstandenen Sackgassen bzw. Rundkursen ein. Zum Zwecke besser vergleichbarer Ergebnisse werden wir versuchen, die oben beschriebenen kaskadieren Effekte so gering zu halten wie möglich, indem wir sorgfältig, d.h. eventuell sogar nur einen Knoten pro Runde, kontrahieren. Da in jeder Runde die Datenstruktur der Kanten neu sortiert werden muss, bezahlen wir dies mit Effizienz. In Anwendungen außerhalb solcher Messungen kann dies natürlich eingespart werden.
Diese Phase versucht ein Stück weit die Idee der dynamischen Strategie nachzubilden, um anschließend Straßen ganzheitlicher erfassen zu können.

Am Anfang der zweiten Phase sind nun keine Sackgassen oder Rundkurse mehr vorhanden. Es werden einmalig alle Ketten detektiert und optional parallele Ketten zusammengefasst. Dann werden alle Knoten in diesen Ketten bis auf ihre Endpunkte nach einer der vorgestellten Vereinfachungsalgorithmen kontrahiert.

In der dritten Phase sind fast nur noch Knoten mit Mindestgrad drei übrig. Diese sind meist so zentral, dass sie sich nicht ohne schwerwiegende lokale Fehler kontrahieren lassen. Wir gehen davon aus, dass diese Fehler durch Entpacken bereinigt werden.

Um sich ein Stück weit gegen Kaskaden abzusichern und um eine gleichmäßige, ganzheitliche Vereinfachung zu erzwingen, wurden Straßentypen als ein stabiles Kriterium herangezogen. In jeder Runde werden zunächst durch eine Medianberechnung die Hälfte der Knoten mit niedrigeren, also wichtigeren Straßentypen[4] ermittelt. Aus diesen Knoten wird gierig eine unabhängige Menge gesucht. Für alle Knoten in der unabhängigen Menge wird die Kontraktion simuliert und die Kantendifferenz ermittelt. Mit einer erneuten Medianberechnung wird wieder die Hälfte mit den niedrigeren Kantendifferenzen für die tatsächliche Kontraktion in dieser Runde ausgewählt.

[3] Wir behandeln in dieser Phase außerdem die trivialen Grad-0-Knoten
[4] Der Straßentyp eines Knoten sei der wichtigste Straßentyp aller angrenzenden Kanten in G_0

Es sei an dieser Stelle noch einmal betont, dass diese Kontraktionsstrategie für die Vereinfachung von Straßen fest darauf vertraut, dass später Kanten entpackt werden. Da auch die längste Straße sehr früh zu einer einziger Kante zusammengefasst wird, muss diese praktisch immer entpackt werden. Für eine Umsetzung in der Praxis sollte zweifellos die dynamische Strategie gewählt werden, um solche voraussehbaren Entpackungsoperationen zur Anzeigezeit zu reduzieren.

Um unsere Algorithmen zur Vereinfachung von Straßen mit der Standardkontraktionsstrategie nach Kantendifferenz zu vergleichen, werden wir stellvertretend den 2-Punkt-Algorithmus messen, der Polygonzüge effektiv genauso vereinfacht(siehe Abschnitt 3.3).

6.2.2. Beschleunigung

Um die Beschleunigung einer CH zur Berechnung von kürzesten Pfaden zu messen, erstellen wir für den Graphen G eine Liste von Anfragen mit zufällig gewählten Start- und Zielknoten. Wir berechnen dann sequentiell mit dem verbesserten Dijkstra-Algorithmus aus Abschnitt 2.2.5 und dieser CH die kürzesten Pfade und messen die benötigte Zeit. Um die Zeiten verschiedener CHs vergleichen zu können, benutzen wir immer die gleiche Anfrageliste.

6.2.3. Detailgrad

Für die Durchführung unserer empirischen Auswertungen ergibt sich mit dem Entpacken ein Dilemma: Einerseits wollen wir anwendungsnah die konkret angezeigten Darstellungen untersuchen, für welche das Entpacken nach einem Fehlermaß notwendig wird. Andererseits wollen wir die Tauglichkeit unserer Algorithmen zur Minimierung genau dieser Fehler bewerten.

Wir werden deswegen die Fehlerbereinigung dem Entpacken überlassen und messen, wie gut eine Kontraktionsreihenfolge diesen Prozess unterstützt. Um einheitliche Messungen zu erhalten, werden wir immer mit dem gleichen Fehlermaß entpacken, auch wenn zu erwarten ist, dass Ansätze mit der entsprechenden Heuristik bevorteilt werden. Wir wählen den Lotabstandsfehler, da er von allen Fehlermaßen in der Literatur am weitesten verbreitet ist. Um Verdeckungseffekte zu vermeiden, entpacken wir nach dem erweiterten Lotabstandsfehler.

Für einen Detailgrad (n_z, ϵ_z) wird zunächst $G'_i(V'_i, E'_i)$ mit $|V'_i| = n_z$ konstruiert und durch Entpacken der Graph $G_{n_z,\epsilon_z} = (V_{n_z,\epsilon_z}, E_{n_z,\epsilon_z})$. Die Größen der Kantenmengen vor und nach dem Entpacken, also $|E'_i|$ und $|E_{n_z,\epsilon_z}|$, geben Aufschluss darüber, wie gut eine CH die Fehlerbereinigung unterstützt, d.h. mit wie wenig zusätzlichen Kanten Fehler über dem Schwellwert ϵ_z vermieden werden können. Wir werden deswegen den Entpackquotienten $\frac{|E_{n_z,\epsilon_z}|}{|E'_i|}$ berechnen.

6.2.4. Ketten

Da sich der allergrößte Teil der vorgestellten Algorithmen auf die Vereinfachung von Straßen in Form von Ketten bezog, beschränken wir unsere weiteren empirischen Messungen auf diese. Folglich ist wieder ein Detektieren von Ketten angebracht. Doch hier stellt sich zusätzlich die Frage, auf welchem Graphen sie überhaupt detektiert werden sollten. Einerseits liegt für Messungen G_0 als unverfälschte Repräsentation unserer gegebenen Daten auf der Hand. Andererseits können Straßen dort nicht ganzheitlich erfasst werden, weswegen bei der Kontraktion zuerst die Sackgassen entfernt wurden. Außerdem müssen diese Ketten dann in G_{n_z,ϵ_z} wiedergefunden werden. Dieser Ansatz wurde implementiert aber wieder verworfen, da sich manche Ketten nur teilweise oder gar nicht wiedererkennen ließen und unklar war, wie solche Ketten in ein Gesamtfehlermaß einfließen sollten.
Deswegen wird die Kettenerkennung einmalig auf G_{n_z,ϵ_z} durchgeführt. Sei C die Menge dieser Ketten. Wie schon beim Kontraktionsprozess werden parallele Fahrbahnen zusammengefasst, aus C entfernt und als Tupel in C_t gespeichert. Um alle Ketten referenzieren zu können, sei außerdem $C_{all} = C \cup \{c_1, c_2 | (c_1, c_2) \in C_t\}$.

6.2.5. Grenzwechsel

Sei P die Menge der zugeordneten Pfaden von Ketten $c \in C_{all}$. Für alle $p \in P$ betrachten wir alle Kanten $e \in \hat{p}$. Für jedes e zählen wir wie in Abschnitt 6.1.2 beschrieben, die Anzahl der Grenzwechsel, falls e eine Abkürzung ist. Die Gesamtanzahl aller Grenzwechsel sei $\sharp gw$.

6.2.6. Optimalität

Wir bewerten nun die gefundenen Ketten als Vereinfachungen, indem wir sie mit optimalen Lösungen aus Kapitel 4 vergleichen.

Einzelne Ketten

Sei P wie oben, dann werden wir für alle $p \in P$ mit dem ILP aus Abschnitt 4.1 optimale Lösungen berechnen. Dazu ermitteln wir die vollständige entpackte Form $\hat{p}'$ von $\hat{p}$. Die Eingabe des ILP ist nun p' als zu vereinfachenden Polygonzug und der maximale erweiterte Lotabstandsfehler von allen Kanten aus $\hat{p}$ als ϵ. Für die optimale Vereinfachung mit n_{opt} Knoten wird gelten: $n_{opt} \leq |p|$. Die Knotendifferenz $\Delta n_p = |p| - n_{opt}$ ist das Maß wie nahe p an der optimalen Vereinfachung ist. Um ein globales Gesamtmaß zu erhalten, bilden wir einen gewichteten Durchschnitt mit geographischen Distanzen bezüglich der vollständig entpackten Pfade.

Sei dazu

$$dist(p) = \sum_{e \in \hat{p}'} dist(e) \text{ und } dist(P) = \sum_{p \in P} dist(p)$$

und

$$\Delta n_P = \frac{\sum_{p \in P} \Delta n_p \cdot dist(p)}{dist(P)}$$

Es ist außerdem wichtig, dass die Pfade $\hat{p}$ und $\hat{p}'$ vor Eingabe für das ILP auf Selbstüberschneidung getestet werden müssen. Wir zählen diese Fälle und geben ihren prozentualen Anteil an $|P|$ mit $\times_P$ an, aber lassen sie für die Berechnungen außen vor.
Da das Lösen eines ILP **NP**-schwer ist, teilen wir außerdem Pfade $\hat{p}$ auf, für die $|\hat{p}'|$ zu groß ist um den Test in akzeptabler Zeit durchführen zu können. Es bezeichne $\not{P}$ die Menge der Pfade, die so aus P entsteht. Sie enthält statt langer Pfade $p = v_1 \dots v_n$ die Pfade $p_1 = v_1 \dots v_k$ und $p_2 = v_k \dots v_n$. Statt für P führen wir obige Berechnungen für $\not{P}$ durch.

Parallele Ketten

Für alle Tupel $(c_1, c_2) \in C_t$ definieren wir eine entsprechende Menge P_t aus Pfadtupeln (p_1, p_2). Als zusätzliche Optimierung werden hier außerdem p_1 und p_2 gestutzt wie in Abschnitt 3.4.3 beschrieben. Dann werden die Pfade $\hat{p}_1$ und $\hat{p}_2$ zu $\hat{p}'_1$ bzw. $\hat{p}'_2$ entpackt. Der maximale erweiterte Lotabstandsfehler von $\hat{p}_1$ und $\hat{p}_2$ sei $\epsilon_{p_1 p_2}$. Um weiter verfahren zu können, invertieren wir p_2 und p'_2 zu $\overline{p_2}$ bzw. $\overline{p'_2}$. Dann berechnen wir die diskrete Fréchet-Distanz $\delta_{dF}(p_1, \overline{p_2})$ nach [EM94]. Um vor allem bezüglich δ_{dF} die Qualität zu messen, relaxieren wir unsere Forderung nach einem geringem Lotabstandsfehler mit einem Faktor $r = 1.2$. Dem ILP aus 4.2.2 übergeben wir also p'_1, $\overline{p'_2}$ als Polygonzüge, $r \cdot \epsilon_{p_1 p_2}$ als ϵ und $\delta_{dF}(p_1, \overline{p_2})$ als η. Wir bilden nun $\Delta n_{p_1, p_2} = |p_1| + |p_2| - n_{opt}$ und ähnlich zu oben:

$$dist(P_t) = \sum_{(p_1, p_2) \in P_t} dist(p_1) + dist(p_2)$$

$$\Delta n_{P_t} = \frac{\sum_{(p_1, p_2) \in P_t} \Delta n_{p_1, p_2} \cdot (dist(p_1) + dist(p_2))}{dist(P_t)}$$

Da das Entpacken nicht δ_{dF} optimiert, können wir dieses ebenso Messen:

$$\delta_{dF}(P_t) = \frac{\sum_{(p_1, p_2) \in P_t} \delta_{dF}(p_1, \overline{p_2}) \cdot (dist(p_1) + dist(p_2))}{dist(P_t)}$$

Im Unterschied zu Abschnitt 6.2.6 müssen die Pfade p_1 und p_2 bzw. p'_1 und p'_2 zusammen auf Selbstüberschneidung getestet werden. Das Aufteilen betrifft hier die Pfadpaare und wird so durchgeführt, dass δ_{dF} wenig beeinträchtigt wird. Sei deren Menge mit $\not{P}_t$ bezeichnet. Wie oben müssen wir die Berechnungen wegen des ILP auf $\not{P}_t$ statt auf P_t ausführen. Es wird außerdem aufschlussreich sein, für das unveränderte P_t den Term $\delta_{dF}(P_t)$ zu berechnen.

6.2.7. Technische Daten

Der erstellte Programmcode für die Kontraktionsstrategien und die Messung der CHs wurde in C++ geschrieben und mit Optimierungslevel -o 3 kompiliert. Es wurden dabei die Bibliotheken CGAL, Boost und GLPK eingebunden.

Die Messungen wurden mit einem Kern eines Intel Xeon E3-1225 v3 Prozessors und 16GB Arbeitsspeicher ausgeführt.
Alle Messungen basieren auf einem Graphen mit 23401239 Knoten und 47390841 Kanten, der aus den OSM-Daten für Deutschland gewonnen wurde.

6.3. Ergebnisse

Die Kürzel für die verwendeten Algorithmen bei der statischen Strategie sind hier dargestellt:

- Der allgemeine Ansatz zur Polygonzugvereinfachung:

  2P: 2-Punkt-Algorithmus

  BU: Bottom-Up

  TD: Top-Down

- Fehlermaß (nur für TD und BU):

  K: Knick

  A: Fläche

  L: Lotabstand

- Verknüpfung von parallelen Ketten (optional):

  CD: Constrained-Delaunay Triangulierung

  ZO: Zickzickordnung

  P: „perfekt“ über δ_{dF}

- Die Heuristik für Grenzwechselvermeidung (optional):

  g: Zugeschalten

				t (s)	$\frac{\lvert E_{n_z,\epsilon_z}\rvert}{\lvert E_i'\rvert}$	$\sharp gw$	$\Delta n\mathbb{P}$	$\times\mathbb{P}(\%)$	$\Delta n\mathbb{P}_t$	$\times\mathbb{P}_t(\%)$	$\delta_{dF}(\mathbb{P}_t)$	$\delta_{dF}(P_t)$
BU	K	-	-	550	10.98	34263	0.116	0.038	1.694	21.743	0.141	0.324
BU	K	CD	-	572	11.18	35323	0.265	0.036	5.846	10.479	0.094	0.241
BU	K	-	g	569	10.83	27720	0.127	0.034	2.124	20.102	0.140	0.323
BU	L	-	-	548	11.02	34249	0.115	0.040	1.708	21.787	0.142	0.323
BU	L	ZO	-	493	11.18	35487	0.240	0.040	5.008	10.667	0.101	0.252
TD	A	-	-	523	10.96	34029	0.105	0.044	1.304	14.315	0.136	0.319
TD	L	-	-	476	11.01	34124	0.104	0.043	1.308	14.224	0.136	0.319
TD	L	-	g	488	10.90	26800	0.137	0.040	2.699	17.389	0.152	0.360
TD	L	CD	-	479	10.96	35340	0.224	0.048	4.870	8.838	0.095	0.243
TD	L	P	-	489	11.09	35340	0.229	0.045	5.431	11.443	0.104	0.260
TD	L	ZO	-	472	10.99	35303	0.226	0.042	4.910	8.955	0.097	0.247
TD	L	ZO	g	608	11.02	31034	0.288	0.037	7.236	8.001	0.098	0.251
2P	-	-	-	548	12.29	36444	0.566	0.038	5.139	23.635	0.156	0.394

Tabelle 6.1.: Messungen für Detailgrad ($n_z = 0.0005 \cdot |V_0|$ und $\epsilon_z = 0.02$ km) mit Lotabstandsfehler. Es gilt außerdem $73700 \leq |E_i'| \leq 76100, 69800 \leq dist(\mathbb{P}) \leq 78000$ (km), $26000 \leq dist(\mathbb{P}_t) \leq 34000$ (km) und $8.773 \cdot 10^7 \leq |E^+| \leq 8.779 \cdot 10^7$

Wir werden unsere Interpretation auf die Tabelle 6.1 beziehen. Eine visuelle Darstellung eines kleineren Graphen mit dem gleichen Detailgrad ist im Anhang in Abbildung A.1 gezeigt. Dort werden auch weitere Messergebnisse aufgeführt.

6.3.1. Interpretation

Erwartungshaltung

Die Erwartungshaltung war, dass manche Algorithmen wegen der Ähnlichkeit zu unseren Messmethoden besser abschneiden. Da die Kanten bezüglich Fehler zum ursprünglichen Graphen entpackt wurden, sollte der Bottom-Up-Ansatz wegen seiner Gedächtnislosigkeit schlechter abschneiden. Weil der Fehler sich auf den Lotabstand bezog, sollten Algorithmen mit dieser Heuristik profitieren. Für parallele Ketten sollten die dafür entwickelten Algorithmen näher an den optimalen Lösungen sein. Weiterhin war zu erwarten, dass das teurere und bezüglich der maximalen Leinenlänge perfekte Knotenkopplungsverfahren besser abschneidet als die Zickzackordnung. Bei den benötigten Zeiten für die Berechnung von kürzesten Pfaden sollte außerdem der 2-Punkt-Algorithmus wegen seiner Gleichmäßigkeit am besten abschneiden.

Beschleunigung

Obwohl für jede Messung [5] die gleiche Liste aus 500000 Anfragen verwendet wurden, sind die benötigten Zeiten relativ breit gestreut. Praktisch liegen alle Messungen im Bereich von 450 bis 610 Sekunden. Es kommt hinzu, dass zwischen den Ansätzen zur Polygonzugvereinfachung durch Kaskadeneffekte die Knoten mit Mindestgrad 3 anders kontrahiert werden, die für eine schnelle Suche wichtiger sind.
Die Messergebnisse deuten darauf hin, dass die Kontraktionsreihenfolge der Knoten in den Polygonzügen für schnelle Suchen praktisch vernachlässigbar ist.

Da die statische Strategie mit Berücksichtigung der Straßentypen außerdem eine ganzheitliche Vereinfachung anstrebt, wurde hier noch mit einer durch Kantendifferenz aufgebauten CH verglichen. Es war erstaunlich, dass die Suchen auf dieser CH stets 900-1100 Sekunden benötigten.
Dieses Ergebnis ist damit erklärbar, dass die CH nach Kantendifferenz nur $|E^+| = 82848064$ Kanten hatte, also wesentlich weniger als $8.773 \cdot 10^7$ wie die statisch erstellten CHs. Die Entfernung der Sackgassen und die Berücksichtigung von Straßentypen ist also auch für die Beschleunigung eine bessere Heuristik als eine Kontraktion nach reiner Kantendifferenz.

Es scheint zunächst so, als ob schnelles Suchen und die ganzheitliche Darstellung auf der einen Seite gegen mehr Speicherplatz auf der anderen Seite abzuwägen sind. Mehr Abkürzungskanten müssen aber nicht zwingend zu schnelleren Suchen führen bzw. können diese sogar verlangsamen. Für eine Darstellung ist es außerdem zur Anzeigezeit ein Mehraufwand, diese zusätzlichen Abkürzungskanten zu entpacken.
Um endgültig zwischen Darstellung und Beschleunigung abwägen zu können, müssten zuerst einmal eindeutig optimale Kontraktionsstrategien für die Beschleunigung gefunden werden, an denen im Moment noch geforscht wird [FS15].

[5] Der Einfachheit halber sind diese Messwerte auch in den Tabellen für die unterschiedlichen Detailgrade aufgeführt obwohl sie natürlich davon unbeeinflusst sind.

Entpackfaktor

Für den Entpackquotienten $\frac{|E_{n_z,\epsilon_z}|}{|E'_i|}$ können wir deutlich sehen, dass der naive 2-Punkt-Algorithmus deutlich mehr Entpackungen benötigt als alle unsere Ansätze. Damit haben wir für die angezeigten Darstellungen von Straßen einen zweifellosen Qualitätsgewinn.
Erwartungsgemäß entpacken von unseren Ansätzen die einfacheren Varianten tendenziell etwas weniger Kanten als die mit zusätzlichen Heuristiken.

Grenzwechsel

Um Grenzwechsel geringfügig gegenüber dem 2-Punkt-Algorithmus zu verringern, reichen schon alle unsere Ansätze aus. Mit der Heuristik zur Grenzwechselvermeidung können diese mit minimal höherem $\frac{|E_{n_z,\epsilon_z}|}{|E'_i|}$ sogar deutlich reduziert werden. Da sich einerseits Grenzwechsel unter Umständen nicht vermeiden lassen und wir im Zweifelsfall eher zu viele als zu wenige messen, ist es nicht verwunderlich, dass $\sharp gw$ immer noch relativ groß ist.

Einzelne Ketten

In Tabelle 6.2 ist zu sehen, dass für die Vereinfachung von einzelnen Ketten unsere Ansätze allesamt wesentlich näher am Optimum sind als der naive 2-Punkt-Algorithmus. Erwartungsgemäß schneiden die Top-Down Varianten besser ab als die Bottom-Up Varianten. Die unterschiedlichen Fehlerheuristiken wirken sich nur marginal aus.

				$\frac{\lvert E_{n_z,\epsilon_z}\rvert}{\lvert E'_i\rvert}$	$\Delta n_{\not P}$	$\times_{\not P}(\%)$
BU	K	-	-	10.98	0.116	0.038
BU	L	-	-	11.02	0.115	0.040
TD	A	-	-	10.96	0.105	0.044
TD	L	-	-	11.01	0.104	0.043
2P	-	-	-	12.29	0.566	0.038

Tabelle 6.2.: Ausgewählte Messwerte für einzelne Ketten aus Tabelle 6.1

				$\frac{\lvert E_{n_Z,\epsilon_Z}\rvert}{\lvert E_i'\rvert}$	$\Delta n_{\not P_t}$	$\times_{\not P_t}(\%)$	$\delta_{dF}(\not P_t)$	$\delta_{dF}(P_t)$
BU	K	-	-	10.98	1.694	21.743	0.141	0.324
BU	K	CD	-	11.18	5.846	10.479	0.094	0.241
BU	L	-	-	11.02	1.708	21.787	0.142	0.323
BU	L	ZO	-	11.18	5.008	10.667	0.101	0.252
TD	L	-	-	11.01	1.308	14.224	0.136	0.319
TD	L	CD	-	10.96	4.870	8.838	0.095	0.243
TD	L	P	-	11.09	5.431	11.443	0.104	0.260
TD	L	ZO	-	10.99	4.910	8.955	0.097	0.247
TD	L	ZO	g	11.02	7.236	8.001	0.098	0.251
2P	-	-	-	12.29	5.139	23.635	0.156	0.394

Tabelle 6.3.: Ausgewählte Messwerte für parallele Ketten aus Tabelle 6.1

Parallele Ketten

Bei den Vereinfachungen von parallelen Ketten (Tabelle 6.3) ist überraschend, dass die dafür ausgelegten Zusatzoptimierungen größere $\Delta n_{\not P_t}$ ergeben. Kleinere Werte für $\delta_{dF}(\not P_t)$, $\delta_{dF}(P_t)$ und $\times_{\not P_t}$ zeigen jedoch, dass Überschneidungen reduziert und die Parallelität mit den Zusatzoptimierungen beibehalten werden kann. Am 2-Punkt-Algorithmus ist zu sehen, dass diese Werte auch wesentlich höher sein können, obwohl dort ähnlich stark von der optimalen Anzahl der verwendeten Knoten abgewichen wird.
Wegen der Werte für $\Delta n_{\not P_t}$ wurden sowohl das heuristische als auch das optimale Vereinfachen von parallelen Polygonzügen intensiv überprüft.

Ein Problem scheint darin zu liegen, dass für die gleiche Straße in zwei verschiedenen Messläufen unterschiedliche optimale Lösungen berechnet werden. Die Parameter für deren Berechnung hängen stark von der zu untersuchenden Vereinfachung ab.
Die Relaxierung der ϵ-Bedingung mit r war ein Versuch, die Messung stärker auf δ_{dF} zu lenken, hat aber auch für verschiedenes Wählen von r nichts wesentlich geändert.
Das Zurechtstutzen der Pfadpaare war eine weitere Optimierung für eine bessere Performanz der Knotenkopplungen. Im Zusammenhang des Zurechtstutzens fällt außerdem auf, dass $\delta_{dF}(\not P_t)$ in der Regel halb so klein wie $\delta_{dF}(P_t)$ ist. Eigentlich sollte das Aufteilen der Pfade zu ungefährer Gleichheit der beiden Werte führen. Anfangs stand sogar die Befürchtung, dass δ_{dF} durch ungünstiges Aufteilen steigen würde.

Es ist zu vermuten, dass die Pfade einiger Paare weitläufig parallel, aber an kurzen Stellen verhältnismäßig weit auseinander sind. Dann ist $\delta_{dF}(P_t)$ wegen dieser Stellen hoch und die Aufteilung zerlegt diese Pfadpaare in die einzelnen parallelen Abschnitte, sodass $\delta_{dF}(\not P_t)$ gering ist. Da der „perfekte“ Algorithmus zur Knotenkopplung nur die maximal nötige Distanz (siehe Abschnitt 3.4.2) zwischen zwei gekoppelten Knoten minimiert, können Knoten in den weitläufig parallelen Abschnitte ungünstig gekoppelt sein. Das erklärt außerdem plausibel, warum die

ZO- und CD-Varianten im Vergleich besser abschneiden. Diese orientieren sich stärker an der lokalen Umgebung und sind somit robuster.
Solche Einzelfälle sind dann in der Regel sehr weit vom Optimum entfernt, da unsere Ansätze auf annähernd gleichmäßig parallele Straßen ausgelegt sind. Auch wenn der Großteil der anderen Kettenpaare sehr nahe am Optimum liegt, können sich die obigen Einzelfälle wegen der Durchschnittsbildung stark auf die Gesamtbilanz auswirken.
Um dies zu verhindern, könnte z.B. der Median statt des Durchschnitts als Messwert gebildet werden.

Die wahrscheinlichste Ursache der vergleichsweise hohen Δn_{P_t} für unsere Zusatzoptimierungen liegt im Entpacken der Kanten, welches auf Reduzieren von ϵ und nicht auf δ_{dF} ausgelegt ist, und deswegen den einfacheren Varianten entgegenkommt. Alternativ könnten einzelne Kanten entpackt werden, bis das Pfadpaar, welches die Kante enthält, ein festes δ_{dF} unterschreitet.

Solche Anpassungen würden aber den ohnehin schon komplizierten Messprozess noch undurchschaubarer machen und damit die Aussagekraft der Messwerte relativieren.

Trotzdem haben wir schon mit den Varianten ohne Zusatzoptimierungen eine deutlich und strikt bessere Vereinfachung von parallelen Straßen als mit dem 2-Punkt-Algorithmus erreicht. Auf den einfachen Varianten aufbauend, bringt das Zuschalten der Optimierungen noch einmal weniger Überschneidungen und mehr Parallelität für einen nur gering höheren, globalen Entpackquotienten. Die Grenzwechselheuristik drückt die Anzahl der Überschneidungen noch weiter.

6.3.2. Vorverarbeitungszeit

Wir werden hier nicht umfangreich die benötigten Zeiten zur Konstruktion der CHs vorstellen, da dies kein primäres Optimierungsziel war und in Abschnitt 6.2.1 höhere Zeiten in Kauf genommen wurden, um genauer vergleichen zu können.
Unabhängig davon hat sich gezeigt, dass die simulierten Kontraktionen zur Bestimmung der Kantendifferenz wesentlich mehr Berechnungszeit benötigten als die anderen Schritte. Falls die Kantendifferenz durch Simulation berechnet werden sollte und noch sehr viele Knoten (≥ 500000) übrig waren, wurde stattdessen das Produkt des Eingangs- und des Ausgangsgrades als Heuristik verwendet, um die Konstruktion für den Deutschlandgraphen in einigen Stunden durchführen zu können.

Die Algorithmen zur Vereinfachung von Polygonzügen und ihre Zusatzoptimierungen konnten in weniger als 15 Minuten durchgeführt werden, was außerdem für die statische Variante nur selten nötig war. Dies liegt daran, dass die Algorithmen für die in der Praxis zu verwendende dynamische Variante konzipiert sind, bei der sie öfter ausgeführt werden müssen.
Es ist trotzdem davon auszugehen, dass für reale Straßengraphen lokale Vereinfachungsalgorithmen mit quadratischer Zeitkomplexität hinnehmbar sind, wenn ohnehin schon Kontraktionen in jeder Runde simuliert werden.

Des weiteren hat sich die Anzahl der Kandidaten für Grenzwechsel bei der Heuristik und beim Messen als relativ klein herausgestellt, sodass die hohen theoretischen Laufzeiten nicht auftraten. Für eine praktische Anwendung wäre deswegen die Umsetzung unserer Messmethode als Heuristik eine naheliegende Verbesserung.

7. Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden wir zunächst zusammenfassen, welche Herausforderungen während dieser Arbeit gegeben waren, wie sie angegangen wurden und welche Lösungen bzw. neue Erkenntnisse gefunden wurden.
Danach werden wir einen Ausblick geben, an welchen Stellen noch Verbesserungspotenzial für die vorgestellten Lösungsansätze besteht und wie praktische Anwendungen auf den gefundenen Ergebnissen aufbauen können.

Zusammenfassung

Wir haben untersucht, wie sich die Datenstruktur der CHs konstruieren lässt, sodass sie in Navigationssystemen eine qualitativ hohe und kontinuierlich vereinfachte Darstellung des Straßengraphen ermöglicht.

Wir definierten formale Fehlerkriterien, um die visuelle Qualität von lokalen Vereinfachungen algorithmisch zu behandeln und zu bewerten. Für diese Kriterien entwickelten wir heuristische Algorithmen zur konkreten Konstruktion von CHs. Um deren Qualität als Vereinfachung zu messen, wurden Messmethoden erstellt, unter anderem auch ganzzahlige lineare Programme, die für kleine Eingaben optimale Lösungen zum Vergleichen liefern.

Beim Untersuchen von Strategien, den Graphen ganzheitlich zu vereinfachen, wurde zum Einen erkannt, dass für eine sinnvolle visuelle Darstellung eines vereinfachten Graphen eine Nachbearbeitung unumgänglich ist. Zum Zweiten lässt sich ein Minimieren von lokalen Fehlern erkaufen, wenn dafür ganzheitliche Qualitäten des Graphen vernachlässigt werden. Da diese ganzheitlichen Qualitäten außerdem einige zwiespältige Aspekte haben, stellte sich eine universelle objektive Messung als unmöglich heraus.
Um trotzdem die Strategien für lokale Vereinfachungen bewerten zu können, wurden diese als austauschbare Module mit einer möglichst neutralen Strategie zur ganzheitlichen Vereinfachung kombiniert.

Die empirischen Messungen auf dem Deutschlandgraphen ergaben, dass für unsere auf Darstellung optimierten CHs im Vergleich zu einer herkömmlichen teilweise sogar kleinere Berechnungszeiten für kürzeste Pfade erreicht wurden.
Vor allem konnten aber für die heuristischen Vereinfachungen einzelner und paralleler Fahrbahnen deutliche Verbesserungen bei der Topologiekonsistenz und für die Formerhaltung

sogar deren Nähe zum Optimum nachgewiesen werden. Bei Zusatzoptimierungen für das Vereinfachen von parallelen Fahrbahnen konnten nur Teilerfolge nachgewiesen werden, was eventuell den nur umständlich möglichen Messungen zuzuschreiben ist.

Ausblick

Neben minimalen Verbesserungen für unsere Implementierung, verspricht vor allem das Verwenden von etwas teureren Algorithmen zur Polygonzugsvereinfachung noch Potenzial. Beispielsweise könnten unsere verwendeten Messalgorithmen für Grenzwechsel schon als Heuristik eingebunden werden und dann deren Qualität mit noch genaueren und teureren Algorithmen gemessen werden.

Für objektive Messungen musste hier auf die Kontraktionsstrategie in Phasen zurückgegriffen werden, aber die dynamische sollte eindeutig weniger Entpackoperationen zur Folge haben, sodass diese für eine praktische Umsetzung gewählt werden sollte. Für ganzheitliche Vereinfachungen sollte außerdem konzeptuell die Frage geklärt werden, inwiefern oder für welche Anwendung zur Hälfte dargestellte Autobahnen sinnvoll sind. Abhängig davon bietet die dynamische Strategie für eine Umsetzung ohne wissenschaftlichen Anspruch ein Vielzahl von Stellschrauben um gefühlt ganzheitliche Darstellungen zu erhalten.

Wir haben außerdem gezeigt, wie unabhängig von der Kontraktionsstrategie eine CH um Zusatzinformationen für bessere Darstellungen augmentiert werden kann. Insbesondere bei der Modellierung von thematischen Karten durch CHs mit Einbeziehung von spezielleren OSM-Daten könnten ähnliche Optimierungen vorgenommen werden.

In Hinsicht auf die ILPs sollte es außerdem interessant sein, die Vereinfachung mit der Fréchet-Distanz umzusetzen und gegebenenfalls sogar mit dem Vermeiden von Grenzwechseln zu erweitern. Weiterhin könnten optimale kontinuierliche Vereinfachungen als ILPs modelliert werden, wobei vor allem die in der Praxis benötigten Berechnungszeiten evaluiert werden müssten.

A. Anhang

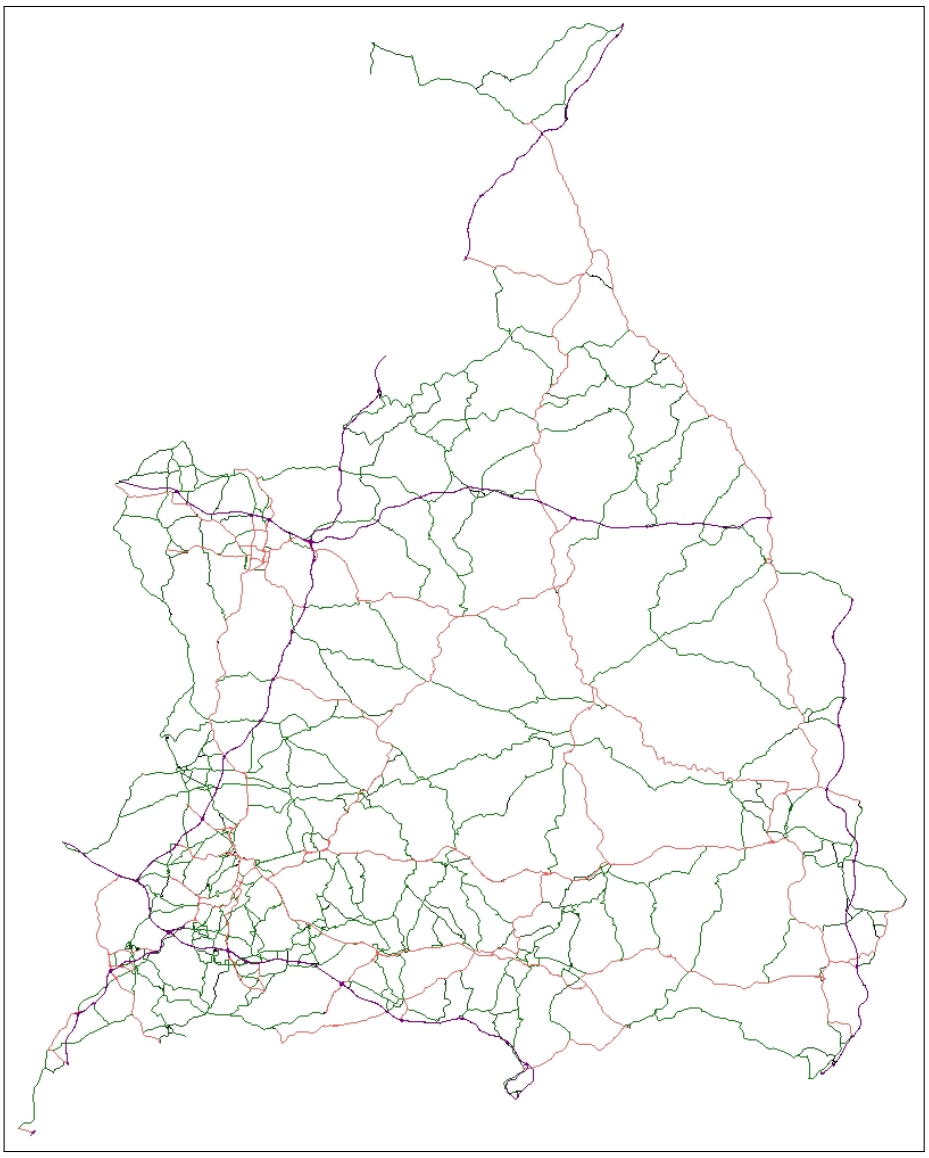


Abbildung A.1.: Regierungsbezirk Stuttgart für den Detailgrad ($n_z = 0.0005 \cdot |V_0|$, $\epsilon_z = 0.02$ km) mit Lotabstandsfehler. Der Graph wurde mit der statischen Variante und den Optionen (TD L - -) erstellt.

				t (s)	$\frac{\lvert E_{n_z,\epsilon_z}\rvert}{\lvert E_i'\rvert}$	$\sharp gw$	$\Delta n\not{P}$	$\times\not{P}(\%)$	$\Delta n\not{P}_t$	$\times\not{P}_t(\%)$	$\delta_{dF}(\not{P}_t)$	$\delta_{dF}(P_t)$
BU	A	CD	-	521	10.89	8255	0.461	0.021	8.078	1.740	0.084	0.204
BU	L	-	-	552	10.45	7889	0.198	0.020	3.603	2.007	0.109	0.247
BU	L	ZO	-	459	10.53	7973	0.307	0.022	7.080	1.541	0.085	0.203
TD	A	-	-	458	10.40	8063	0.211	0.021	3.383	1.591	0.107	0.245
TD	L	-	-	463	10.43	8154	0.212	0.020	3.389	1.618	0.107	0.245
TD	L	-	g	509	10.46	5572	0.237	0.019	4.491	1.473	0.118	0.277
TD	L	P	-	494	10.63	8297	0.328	0.021	8.282	1.508	0.089	0.209
TD	L	ZO	-	492	10.56	8328	0.326	0.020	7.690	1.318	0.082	0.200
2P	-	-	-	507	11.79	8400	0.911	0.018	8.068	1.822	0.129	0.314

Tabelle A.1.: Messungen für Detailgrad ($n_z = 0.002 \cdot |V_0|$ und $\epsilon_z = 0.01$ km) mit Lotabstandsfehler. Es gilt außerdem $17000 \leq |E_i'| \leq 18000$, $114678 \leq dist(\not{P}) \leq 125000$ (km), $43400 \leq dist(\not{P}_t) \leq 47000$ (km) und $8.773 \cdot 10^7 \leq |E^+| \leq 8.779 \cdot 10^7$

				t (s)	$\frac{\lvert E_{n_z,\epsilon_z}\rvert}{\lvert E_i'\rvert}$	$\sharp gw$	$\Delta n\not{P}$	$\times\not{P}(\%)$	$\Delta n\not{P}_t$	$\times\not{P}_t(\%)$	$\delta_{dF}(\not{P}_t)$	$\delta_{dF}(P_t)$
BU	K	-	-	541	25.30	24742	0.030	0.000	0.338	58.824	0.206	0.504
BU	L	-	-	539	29.25	31145	0.031	0.000	0.400	58.977	0.211	0.496
BU	L	ZO	-	512	24.60	11858	0.243	0.000	3.535	19.635	0.097	0.367
TD	A	-	-	564	16.58	16626	0.022	0.072	0.274	40.171	0.173	0.488
TD	L	-	-	571	24.80	22943	0.020	0.025	0.274	34.930	0.180	0.507
TD	L	-	g	522	33.14	21977	0.099	0.020	1.664	49.858	0.189	0.514
TD	L	CD	-	546	35.94	26622	0.205	0.064	3.541	16.353	0.129	0.391
TD	L	P	-	523	34.63	29826	0.216	0.019	3.986	26.127	0.133	0.398
TD	L	ZO	-	549	32.91	31253	0.215	0.037	3.974	15.162	0.123	0.381
2P	-	-	-	531	36.52	21156	0.174	0.028	2.472	61.529	0.207	0.555

Tabelle A.2.: Messungen für Detailgrad ($n_z = 0.000002 \cdot |V_0|$, $\epsilon_z = 0.05$ km) mit Lotabstandsfehler. Es gilt außerdem $590 \leq |E_i'| \leq 1060$, $7800 \leq dist(\not{P}) \leq 13000$ (km), $1900 \leq dist(\not{P}_t) \leq 7500$ (km) und $8.773 \cdot 10^7 \leq |E^+| \leq 8.779 \cdot 10^7$

Literaturverzeichnis

[AFGW10] I. Abraham, A. Fiat, A. V. Goldberg und R. F. Werneck. „Highway dimension, shortest paths, and provably efficient algorithms“. In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial und Applied Mathematics. 2010, S. 782–793 (Zitiert auf S. 17).

[AG95] H. Alt und M. Godau. „Computing the Fréchet distance between two polygonal curves“. In: *International Journal of Computational Geometry & Applications* 5.01n02 (1995), S. 75–91 (Zitiert auf S. 61).

[AHMW05] P. K. Agarwal, S. Har-Peled, N. H. Mustafa und Y. Wang. „Near-linear time approximation algorithms for curve simplification“. In: *Algorithmica* 42.3-4 (2005), S. 203–219 (Zitiert auf S. 34).

[AS01] M. Agrawala und C. Stolte. „Rendering effective route maps: improving usability through generalization“. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, S. 241–249 (Zitiert auf S. 11).

[BB] D. H. Ballard und C. M. Brown. „Computer vision, 1982“. In: *Prenice-Hall, Englewood Cliffs, NJ* () (Zitiert auf S. 37).

[BFSS07] H. Bast, S. Funke, P. Sanders und D. Schultes. „Fast routing in road networks with transit nodes“. In: *Science* 316.5824 (2007), S. 566–566 (Zitiert auf S. 16).

[BJW+08] S. Bereg, M. Jiang, W. Wang, B. Yang und B. Zhu. „Simplifying 3D polygonal chains under the discrete Fréchet distance“. In: *LATIN 2008: Theoretical Informatics*. Springer, 2008, S. 630–641 (Zitiert auf S. 50).

[BLR00] T. Barkowsky, L. J. Latecki und K.-F. Richter. „Schematizing maps: Simplification of geographic shape by discrete curve evolution“. In: *Spatial Cognition II*. Springer, 2000, S. 41–53 (Zitiert auf S. 32, 33, 36).

[BW88] K. E. Brassel und R. Weibel. „A review and conceptual framework of automated map generalization“. In: *International Journal of Geographical Information System* 2.3 (1988), S. 229–244 (Zitiert auf S. 22).

[CvDH14] M. Chimani, T. C. van Dijk und J.-H. Haunert. „How to eat a graph: computing selection sequences for the continuous generalization of road networks“. In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2014, S. 243–252 (Zitiert auf S. 12, 54).

[Del34] B. Delaunay. „Sur la sphere vide“. In: *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7.793-800 (1934), S. 1–2 (Zitiert auf S. 50).

[Dev98] O. Devillers. „Improved incremental randomized Delaunay triangulation“. In: *Proceedings of the fourteenth annual symposium on Computational geometry*. ACM. 1998, S. 106–115 (Zitiert auf S. 82).

[DGNW13] D. Delling, A. V. Goldberg, A. Nowatzyk und R. F. Werneck. „Phast: Hardware-accelerated shortest path trees“. In: *Journal of Parallel and Distributed Computing* 73.7 (2013), S. 940–952 (Zitiert auf S. 16).

[Dij59] E. W. Dijkstra. „A note on two problems in connexion with graphs“. In: *Numerische mathematik* 1.1 (1959), S. 269–271 (Zitiert auf S. 14).

[DP73] D. H. Douglas und T. K. Peucker. „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), S. 112–122 (Zitiert auf S. 34, 37).

[DVS95] M. De Berg, M. Van Kreveld und S. Schirra. *A new approach to subdivision simplification*. Citeseer, 1995 (Zitiert auf S. 34).

[EFS11] J. Eisner, S. Funke und S. Storandt. „Optimal Route Planning for Electric Vehicles in Large Networks.“ In: *AAAI*. 2011 (Zitiert auf S. 13).

[EM01] R. Estkowski und J. S. Mitchell. „Simplifying a polygonal subdivision while keeping it simple“. In: *Proceedings of the seventeenth annual symposium on Computational geometry*. ACM. 2001, S. 40–49 (Zitiert auf S. 34).

[EM94] T. Eiter und H. Mannila. *Computing discrete Fréchet distance*. Techn. Ber. Citeseer, 1994 (Zitiert auf S. 44, 50, 87).

[FMM+] S. Funke, T. Mendel, A. Miller, S. Storandt und M. Wiebe. *Map Simplification with Topology Constraints: Exactly and in Practice*. Unpublished manuscript (Zitiert auf S. 12).

[FS15] S. Funke und S. Storandt. „Provable Efficiency of Contraction Hierarchies with Randomized Preprocessing“. In: *Algorithms and Computation*. Springer, 2015, S. 479–490 (Zitiert auf S. 90).

[GHMS93] L. J. Guibas, J. E. Hershberger, J. S. Mitchell und J. S. Snoeyink. „Approximating polygons and subdivisions with minimum-link paths“. In: *International Journal of Computational Geometry & Applications* 3.04 (1993), S. 383–415 (Zitiert auf S. 33).

[GNS07] J. Gudmundsson, G. Narasimhan und M. Smid. „Distance-preserving approximations of polygonal paths“. In: *Computational Geometry* 36.3 (2007), S. 183–196 (Zitiert auf S. 28).

[GSSD08] R. Geisberger, P. Sanders, D. Schultes und D. Delling. „Contraction hierarchies: Faster and simpler hierarchical routing in road networks“. In: *Experimental Algorithms*. Springer, 2008, S. 319–333 (Zitiert auf S. 16, 74).

[Gut04] R. J. Gutman. „Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks.“ In: *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*. 2004, S. 100–111 (Zitiert auf S. 16).

[Hai94] E. Haines. „Point in polygon strategies“. In: *Graphics gems IV* 994 (1994), S. 24–26 (Zitiert auf S. 78).

[HNR68] P. E. Hart, N. J. Nilsson und B. Raphael. „A formal basis for the heuristic determination of minimum cost paths“. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), S. 100–107 (Zitiert auf S. 16).

[HS92] J. E. Hershberger und J. Snoeyink. *Speeding up the Douglas-Peucker line-simplification algorithm*. University of British Columbia, Department of Computer Science, 1992 (Zitiert auf S. 34).

[Jen89] G. F. Jenks. „Geographic logic in line generalization“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 26.1 (1989), S. 27–42 (Zitiert auf S. 31, 34).

[Kir83] D. Kirkpatrick. „Optimal search in planar subdivisions“. In: *SIAM Journal on Computing* 12.1 (1983), S. 28–35 (Zitiert auf S. 81, 82).

[LT98] P. Lindstrom und G. Turk. „Fast and memory efficient polygonal simplification“. In: *Visualization'98. Proceedings*. IEEE. 1998, S. 279–286 (Zitiert auf S. 36).

[McM86] R. B. McMaster. „A statistical analysis of mathematical measures for linear simplification“. In: *The American Cartographer* 13.2 (1986), S. 103–116 (Zitiert auf S. 31).

[Mil] A. Miller. *Optimale Vereinfachung von polygonalen Ebenenunterteilungen unter Topologieeinschränkungen*. Bachelorarbeit: Universität Stuttgart (Zitiert auf S. 12, 55).

[MKVV06] N. Mustafa, S. Krishnan, G. Varadhan und S. Venkatasubramanian. „Dynamic simplification and visualization of large maps“. In: *International Journal of Geographical Information Science* 20.3 (2006), S. 273–302 (Zitiert auf S. 11).

[MS92] R. B. McMaster und K. S. Shea. „Generalization in digital cartography“. In: Association of American Geographers Washington, DC. 1992 (Zitiert auf S. 22).

[Pao11] L. Paoletti. „Leonard Euler's solution to the Königsberg bridge problem“. In: *URL http://www. maa. org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem.(Cited on pages 9 and 10)* (2011) (Zitiert auf S. 9).

[PS81] F. P. Preparata und K. J. Supowit. „Testing a simple polygon for monotonicity“. In: *Information Processing Letters* 12.4 (1981), S. 161–164 (Zitiert auf S. 34).

[Saa99] A. Saalfeld. „Topologically consistent line simplification with the Douglas-Peucker algorithm“. In: *Cartography and Geographic Information Science* 26.1 (1999), S. 7–18 (Zitiert auf S. 34, 38, 79).

[Sch15] N. Schnelle. „Unified routing and map rendering“. In: (2015) (Zitiert auf S. 9–11, 25, 26, 70, 74).

[SFS13] N. Schnelle, S. Funke und S. Storandt. „Dorc: Distributed online route computation-higher throughput, more privacy“. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE. 2013, S. 344–347 (Zitiert auf S. 16).

[SR03] A. Safonova und J. Rossignac. „Compressed piecewise-circular approximations of 3d curves“. In: *Computer-Aided Design* 35.6 (2003), S. 533–547 (Zitiert auf S. 33).

[Van01] M. Van Kreveld. „Smooth generalization for continuous zooming“. In: *Proc. 20th Intl. Geographic Conference*. 2001, S. 2180–2185 (Zitiert auf S. 11).

Alle URLs wurden zuletzt am 24. 01. 2016 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