Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Generierung und Abdeckung repräsentativer Pfadmengen in Straßennetzwerken

Lukas Berner

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Beginn am:	1. Dezember 2021
Beendet am:	1. Juni 2022

Kurzfassung

Für die Suche nach kürzesten Pfaden in sehr großen Graphen wurden verschiedene Beschleunigungstechniken, wie z.B. Contraction Hierarchies, Hub-Labels oder Transit Node Routing, entwickelt. Um optimale Anfragezeiten und Speicherverbrauch zu erreichen, benötigen viele Beschleunigungstechniken eine Menge *wichtiger* Knoten. In dieser Arbeit wird eine Methode zur Berechnung wichtiger Knoten eines Graphen vorgestellt. Um diese Knoten zu finden, wird auf einer *repräsentativen Pfadmenge* ein Hitting Set Problem mit einem Greedy-Algorithmus gelöst. Die repräsentative Pfadmenge, die möglichst unterschiedliche kürzeste Pfade des Graphen enthalten soll, wird mit einer well-separated pair decomposition und einem Quadtree berechnet. Das Verfahren wurde mit dem deutschen Straßennetzwerk (25M Knoten) getestet und liefert hier einige tausend wichtige Knoten, mit denen bereits etwa 99,9 % aller kürzesten Pfade im Graph abgedeckt sind.

Inhaltsverzeichnis

1 Einleitung **7**

2 Grundlagen und Definitionen **9**
2.1 Graph 9
2.2 Dijkstra-Algorithmus 10
2.3 Contraction Hierarchies 11
2.4 Quadtree 14
2.5 Well-separated pair decomposition 15

3 Berechnung repräsentativer Pfadmengen **19**
3.1 Pfadmengen und Stichproben 19
3.2 Stichproben mit WSPD 21
3.3 Berechnung mit Kartendaten 21

4 Hitting Set **23**
4.1 Grundidee 23
4.2 Meta-Graph der CH-Konstruktion 24
4.3 Histogramm berechnen 25
4.4 Getroffene Pfade finden 29
4.5 Einfache Abschätzung der unteren Schranke 29

5 Auswertung **31**
5.1 Repräsentative Pfadmengen 32
5.2 Hitting Set 38

6 Zusammenfassung und Ausblick **41**

Literaturverzeichnis **43**

1 Einleitung

Die Suche nach kürzesten Pfaden in Graphen ist ein grundlegendes Problem mit vielen Anwendungen. Der klassische Algorithmus zur Lösung dieses Problems ist der Dijkstra-Algorithmus [Dij+59], der das Problem in $O(|E| + |V| \log |V|)$ löst. In großen Graphen kann die Suche mit dem Dijkstra-Algorithmus jedoch mehrere Sekunden dauern und ist somit nicht für Echtzeitanwendungen geeignet. Um die Suche nach kürzesten Wegen auch in sehr großen Graphen, wie z.B. Straßennetzen, in wenigen Millisekunden ausführen zu können, wurden verschiedene Beschleunigungstechniken entwickelt. Diese bestehen meist aus einem Vorbereitungsschritt, in dem zusätzliche Informationen berechnet und gespeichert werden und einem modifizierten Suchalgorithmus, der diese Informationen verwendet und damit schneller Lösungen findet. Eine Übersicht findet sich in [BDG+16].

Im Vorbereitungsschritt dieser Methoden wird oft eine Ordnung der Knoten im Graph nach ihrer *Wichtigkeit* benötigt. Diese Ordnung hat großen Einfluss auf Speicherbedarf und Suchzeiten dieser Beschleunigungstechnik. Dabei wird die Wichtigkeit eines Knotens üblicherweise über sein Vorkommen in kürzesten Pfaden des Graphen definiert. Zur Berechnung dieser Ordnung gibt es verschiedene Heuristiken. Man kann diese insbesondere danach unterscheiden, ob die Ordnung lokal während der Berechnungen in der Vorbereitung festgelegt wird, oder (global) im Voraus vorgegeben ist. Ein Beispiel für eine lokale Heuristik wird etwa in [GSSD08] für Contraction Hierarchies (CH) vorgestellt. Eine globale Heuristik wird z.B. in Transit Node Routing [BFM+07] verwendet.

In dieser Arbeit wird eine weitere Methode zur Berechnung wichtiger Knoten vorgestellt. Diese wichtigen Knoten können dann beispielsweise als globale Heuristik für die Ordnung der Knoten verwendet werden.

Grundsätzlich besteht die hier vorgestellte Methode aus zwei Schritten: zuerst wird eine *repräsentative Pfadmenge* P erzeugt. Diese repräsentative Pfadmenge ist eine breit aufgestellte Sammlung unterschiedlicher kürzester Pfade. Die Berechnung von P verwendet die Einbettung der Knoten in $\mathbb{R}^2$ zur Berechnung eines Quadtrees und eine well-separated pair decomposition (WSPD) [CK95]. Im zweiten Schritt werden aus P wichtige Knoten berechnet. Die Frage nach wichtigen Knoten kann als Hitting Set Problem formuliert werden: finde eine minimale Menge $H \subset V$, sodass jeder Pfad mindestens einen Knoten aus H enthält: $\forall \pi \in P : \pi \cap H \neq \emptyset$. Dieses Problem ist NP-hart; daher wird die Lösung nur mit einem Greedy-Algorithmus approximiert. Um auch sehr große Pfadmengen verarbeiten zu können, wird der Graph einer Contraction Hierarchy verwendet, da Pfade hier mit den Abkürzungskanten der CH beschrieben werden können und somit weniger Speicherplatz benötigen.

Die hier vorgestellte Methode wurde für diese Arbeit implementiert[1] und mit dem deutschen Straßennetz (etwa 25M Knoten) getestet.

[1] github.com/bernlu/garp

Die Arbeit ist in folgender Weise gegliedert: Im nächsten Kapitel werden zunächst formale und algorithmische Grundlagen zusammengefasst. Danach folgt in Kapitel 3 eine genaue Beschreibung der Berechnung einer repräsentativen Pfadmenge. Anschließend wird der Algorithmus zur Lösung des Hitting Sets in Kapitel 4 vorgestellt. Anhand des Straßennetzes von Deutschland werden dann in Kapitel 5 die Algorithmen getestet. Schließlich fasst Kapitel 6 die Ergebnisse der Arbeit zusammen und zeigt einen Ausblick auf weitere Verwendungsmöglichkeiten.

2 Grundlagen und Definitionen

In diesem Kapitel werden Definitionen und Grundlagen für die später vorgestellten Algorithmen zusammengefasst. Zuerst folgt eine Übersicht zu Graphen und zur Suche nach kürzesten Wegen mit dem Dijkstra-Algorithmus. Anschließend werden Contraction Hierarchies, eine Methode zur schnelleren Suche nach kürzesten Wegen, vorgestellt. Zuletzt folgt noch eine Übersicht zu Quadtrees und die Berechnung der well-separated pair decomposition (WSPD).

2.1 Graph

Ein Straßennetz kann als Graph aufgefasst werden. Ein gerichteter Graph $G = (V, E)$ besteht aus einer Menge von Knoten V und gerichteten Kanten $E \subseteq V \times V$. Zu Kanten gehören Kosten. Diese werden mit der Abbildung $\mathsf{cost} : E \to \mathbb{N}$ beschrieben. Für die Verwendung mit Quadtrees gehen wir zusätzlich davon aus, dass eine Einbettung der Knoten im $\mathbb{R}^2$ existiert.

Im Folgenden werden noch einige Eigenschaften von Graphen genannt, die im Verlauf dieser Arbeit benötigt werden.

Teilgraph Eine Teilmenge $V' \subset V$ der Knoten induziert einen Teilgraphen $G' = (V', E')$, der alle Kanten zwischen Knoten aus V' enthält.

Inverser Graph Ein gerichteter Graph hat einen inversen Graph, in dem alle Kanten in die Gegenrichtung zeigen: $G^{-1} = (V, E^{-1})$ mit $E^{-1} = \{(w, v) \mid (v, w) \in E\}$

Pfade Eine Liste von Knoten, die paarweise über Kanten verbunden sind, wird Pfad oder Weg $\pi = abcde$ genannt. Alternativ kann auch eine Liste von Kanten angegeben werden: $\pi = (a, b)(b, c)(c, d)(d, e)$. Die Kosten eines Pfades sind die Summe der Kantenkosten des Pfades: $\mathsf{cost}(\pi) = \sum_{e \in \pi} \mathsf{cost}(e)$. Andere Bezeichnungen für die Kosten sind auch *Entfernung* oder *Abstand* - insbesondere wenn die Kantenkosten Entfernungen beschreiben. Auf verschiedene Darstellungen und Speichermethoden von Pfaden wird in Abschnitt 2.3.4 noch genauer eingegangen.

Kreisfreiheit Ein Kreis ist ein Pfad, der im gleichen Knoten beginnt und endet. Hat ein Graph keine Kreise, heißt er kreisfrei. Ist der Graph dazu auch gerichtet, wird er Directed Acyclic Graph (DAG) genannt.

Algorithmus 2.1 Dijkstra-Algorithmus nach [Sch11, S. 194]

```
procedure DIJKSTRA(source)
    ∀v ∈ V : l(v) ← ∞
    l(source) ← 0
    W ← V
    F ← ∅
    for i = 1 ... |V| do
        finde Knoten v ∈ W mit l(v) minimal
        W ← W \ {v}
        F ← F ∪ {k(v)}
        for v' ∈ Adj(v), v' ∈ W do
            if l(v) + cost(v, v') < l(v') then
                l(v') ← l(v) + cost(v, v')
                k(v) ← (v, v')
            end if
        end for
    end for
end procedure
```

Topologische Sortierung Eine topologische Sortierung ist eine Ordnung $<$ auf den Knoten sodass gilt: $\forall(v, w) \in E : v < w$ d. h. alle Kanten führen nur von kleineren zu größeren Knoten. Jeder kreisfreie Graph kann topologisch sortiert werden. Ein DAG ist kreisfrei, also können die Knoten topologisch geordnet werden.

2.2 Dijkstra-Algorithmus

In einem Graphen stellt sich die Frage nach dem kürzesten Weg zwischen Knoten. Um kürzeste Wege zu finden wird der Dijkstra-Algorithmus [Dij+59], oder Abwandlungen von diesem, verwendet. In diesem Abschnitt wird der Dijkstra-Algorithmus nach [Sch11, S. 193ff] und einige Optimierungen vorgestellt.

Eine einfache Form des Algorithmus (als Suche von einem Startknoten zu allen anderen Knoten) ist in Algorithmus 2.1 aufgeführt. Hier ist W die Menge der noch zu sondierenden Knoten. Schrittweise wird immer der Knoten aus W mit der kleinsten Entfernung zum Startknoten verarbeitet. Ist der Algorithmus beendet, enthält F alle Kanten, die für kürzeste Wege vom Start zu einem anderen Knoten benötigt werden. $l(v)$ ist die Entfernung zu v und $k(v)$ ist die optimale zu v führende Kante. Durch rekursives entpacken von $k(*)$ kann der Pfad zu v angegeben werden.

Verwendet man für W einen Fibonacci-Heap, ist die Laufzeit des Algorithmus $O(|E| + |V| \log |V|)$. Für die Suche eines kürzesten Weges zwischen genau zwei Punkten kann die Suche abgebrochen werden, sobald der Zielknoten aus W entfernt wird. Die Implementierung, die in den Experimenten dieser Arbeit (in Kapitel 5) verwendet wird, ist eine solche optimierte Variante und orientiert sich an den Versuchen und der Implementierung von [Bar21].

Algorithmus 2.2 Konstruktion der CH

procedure CHConstruction($G = (V, E_B)$, $<$)
  for all $v \in V$ ordered by $<$ ascending **do**
    ContractNode(v)
  end for
end procedure
procedure ContractNode(v)
  for all $(u, v) \in E_B \cup E_{CH}$ with $u > v$ **do**
    for all $(v, w) \in E \cup E_{CH}$ with $w > v$ **do**
      if uvw may be the shortest path from u to w **then**
        $E_{CH} \leftarrow E_{CH} \cup \{(u, w)\}$ with $\mathsf{cost}(u, w) = \mathsf{cost}(u, v) + \mathsf{cost}(v, w)$
      end if
    end for
  end for
end procedure

2.3 Contraction Hierarchies

Contraction Hierarchies (CH) von [GSSD08] sind eine Beschleunigungstechnik für die Berechnung kürzester Wege. Experimente mit großen Straßennetzen zeigen, dass die Suche mit CH deutlich schneller ist als eine einfache Suche mit dem Dijkstra-Algorithmus. In der Implementierung dieser Arbeit beträgt die Anfragezeit der CH-Suche für zufällige Punktpaare im deutschen Straßennetz etwa 2 ms. CH arbeiten in zwei Phasen: Zuerst wird eine neue Datenstruktur erzeugt, indem zusätzliche Abkürzungskanten E_{CH} zu dem ursprünglichen Graph $G = (V, E_B)$ hinzugefügt werden. Es entsteht ein erweiterter Graph, der CH-Graph $G_{CH} = (V, E_B \cup E_{CH})$. Abkürzungskanten sind Kanten, die kürzeste Wege im Graph nicht verändern, aber mehrere Kanten zu einer Kante zusammenfassen. Anschließend kann in der zweiten Phase mit einer modifizierten Variante des Dijkstra-Algorithmus im CH-Graph gesucht werden.

2.3.1 Konstruktion

Gegeben ist ein Graph $G = (V, E_B)$ mit Kantenkosten $\mathsf{cost} : E \to \mathbb{N}$ und zusätzlich eine beliebige Sortierung der Knoten, genannt $<$. Aus der Sortierung $<$ ergibt sich eine aufsteigende Nummerierung der Knoten, die $\mathsf{level} : V \to \mathbb{N}$ genannt wird. Der Konstruktionsalgorithmus ist in Algorithmus 2.2 beschrieben. Der CH-Graph G_{CH} wird iterativ erzeugt. Kern der Konstruktion ist die *Kontraktion* von Knoten. Alle Knoten werden nach der gegebenen Reihenfolge $<$ kontrahiert. Sind alle Knoten kontrahiert, ist $(G_{CH} = (V, E_B \cup E_{CH}), \mathsf{level})$ eine *Contraction Hierarchy*.

Dieser Algorithmus erzeugt für jede beliebige Knotensortierung $<$ eine Contraction Hierarchy. Allerdings hat die gewählte Reihenfolge einen großen Einfluss darauf, wie viele Abkürzungen erzeugt werden und damit auch auf die Laufzeit der Suche nach kürzesten Pfaden. Um geringe Suchzeiten zu erreichen, sollten möglichst wenige Abkürzungen erzeugt werden.

Abbildung 2.1: Eine Abkürzung für v wird hinzugefügt. Bild nach [GSSD08]

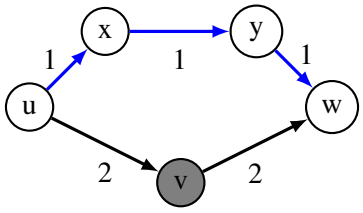


Abbildung 2.2: Da $uxyw$ kürzer als uvw ist, muss keine Abkürzung eingefügt werden. Bild nach [GSSD08]

Kontraktion Zur Kontraktion wird immer der Teilgraph von G_{CH} betrachtet, der von den Knoten $\geq v$ induziert wird, d. h. $G'_{CH} = (V', E'_B \cup E'_{CH})$ mit $V' = \{w \in V \mid w \geq v\}$. Ziel der Kontraktion ist, v aus dem Graphen zu entfernen, ohne die kürzesten Wege im Graphen zu verändern. Dazu wird für alle Paare der Nachbarknoten von v geprüft, ob der kürzeste Pfad zwischen ihnen v enthält. Falls ja, wird eine neue *Abkürzungskante* $e \in E_{CH}$ erzeugt, die das Paar direkt verbindet und die gleichen Kosten wie der kürzeste Pfad hat. Dadurch kann v entfernt werden und alle kürzesten Wege, die über v verlaufen, bleiben erhalten. Um für ein Nachbarpaar u, w zu prüfen, ob der kürzeste Weg v enthält, kann z. B. eine Dijkstra-Suche verwendet werden.

Nach der Kontraktion gilt für alle Knoten $s, t > v$, dass ein kürzester Pfad existiert, der v nicht enthält. Da die Konstruktion iterativ ist, ergibt sich auch, dass ein kürzester Pfad existiert, der nur Knoten $> v$ enthält.

2.3.2 Suche

Für alle kürzesten Wege $\pi = abc \ldots p \ldots xyz$ im CH-Graph gilt, dass es einen Knoten mit höchstem Level gibt und das Level aller Knoten von dort aus monoton sinkt: $a < b < c < p > x > y > z$. Die Suche kann also für den ersten (bzw. zweiten) Teil auf aufsteigende (bzw. absteigende) Kanten $e = (v, w)$, bei denen $v < w$ gilt, begrenzt werden. Die Suche im CH-Graph wird daher bidirektional ausgeführt. Es gibt zwei Suchgraphen: $G_\uparrow = (V, E_\uparrow)$ mit $E_\uparrow := \{(u, v) \in E | u < v\}$ und $G_\downarrow = (V, E_\downarrow)$ mit $E_\downarrow := \{(u, v) \in E | u > v\}$. Der Suchalgorithmus startet von s in $G_\uparrow$ (Vorwärtssuche), und von t in $G_\downarrow^{-1}$ (Rückwärtssuche). In den meisten Fällen wird der kürzeste Weg über einen Knoten mit höherem Level als s und t führen. In diesem Fall kann der vollständige Pfad weder von der Vorwärts- noch von der Rückwärtssuche gefunden werden, da beide nur Pfade finden, deren Knotenlevel monoton steigen. Um den kürzesten Weg zu finden, müssen Teilergebnisse beider Suchen zusammengefügt werden. In jedem Fall wird die Suche aus beiden Richtungen irgendwann einen gleichen Punkt x betrachtet haben. Über diesen führt ein Weg von s nach t: $s \ldots x \ldots t$. Dieser

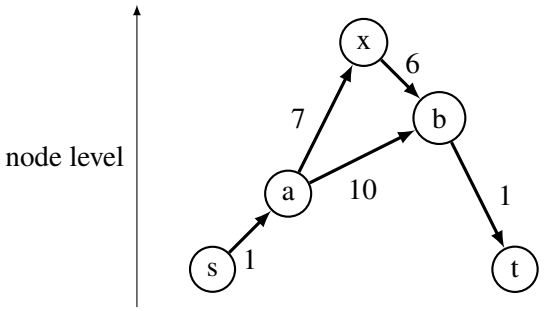


Abbildung 2.3: Die Rückwärtssuche besucht b und dann x mit Kosten 7 und die Vorwärtssuche besucht a und dann x mit Kosten 8. Damit ist ein gemeinsamer Punkt gefunden und der Kandidatweg $saxbt$ hat Länge 15. Allerdings ist der Weg $sabt$ mit Länge 12 kürzer.

Weg ist ein Kandidat für den kürzesten Weg, muss jedoch nicht der kürzeste Weg sein, wie am Beispiel Abbildung 2.3 zu sehen ist. Erst, wenn in beiden Suchrichtungen der nächste noch zu sondierende Knoten weiter entfernt ist als der Kandidatweg lang ist, ist der Kandidat sicher der kürzeste Weg. Damit kann die Suche abgebrochen werden und der Kandidat wird als kürzester Weg ausgegeben.

stall on demand *Stall on demand* ist ein Trick, mit dem die Suchläufe noch weiter optimiert werden können. Im Folgenden wird die Vorwärtssuche betrachtet; stall on demand kann analog auch auf die Rückwärtssuche angewandt werden. Stall on demand ist ein zusätzlicher Test in jedem Schritt der Suche: Wird ein Knoten in der Vorwärtssuche (in $G_\uparrow$) erreicht, muss der Weg von s zu diesem Knoten nicht der kürzeste in G sein, da einige Kanten von G in $G_\uparrow$ nicht betrachtet werden.

Angenommen, es existiert ein Knoten u, der von s über w mit Kosten 2 erreichbar ist, in $G_\uparrow$ allerdings nur über einen längeren Weg über den Knoten x (Abbildung 2.4). Dies ist der Fall, wenn $\mathsf{level}(w) > \mathsf{level}(u)$ gilt und somit $(w,u) \in E$ aber $(w,u) \notin E_\uparrow$. Dann können Pfade, die in der Suche in $G_\uparrow$ über u verlaufen, keine kürzesten Pfade sein (da schon der Teilpfad zu u nicht der kürzeste ist). Somit müssen Kanten, die von u ausgehen, nicht betrachtet werden. Um zu überprüfen, ob solch ein Nachbar w existiert, werden alle Nachbarn des aktuellen Knotens u in $G_\downarrow$ betrachtet und die Entfernung von u über den Nachbarn berechnet. Gibt es einen kürzeren Pfad, wie im Beispiel, dann werden die Nachbarn von u in $G_\uparrow$ nicht in die Liste der zu sondierenden Knoten eingefügt.

2.3.3 CH-Pfade entpacken

Der Pfad, der in der CH-Suche gefunden wird, enthält möglicherweise Abkürzungskanten. Da diese keine Bedeutung im Basis-Graphen haben, kann der Pfad wieder in einen Pfad umgewandelt werden, der keine Abkürzungen enthält. Aus der Konstruktion der Abkürzungen ergibt sich, dass jede Abkürzung genau zwei Kanten ersetzt. Im gefundenen Pfad können diese beiden Kanten also anstelle der Abkürzung eingesetzt werden. Die beiden Kanten können Abkürzungen sein, oder

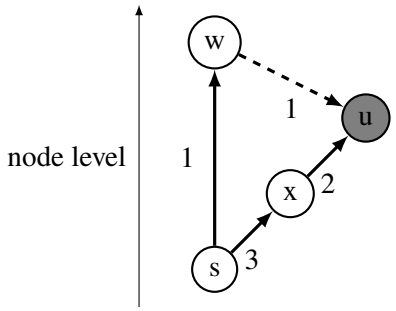


Abbildung 2.4: u ist von s in $G_\uparrow$ über x mit cost(u)= 5 erreichbar. Allerdings exisitert in $G_\downarrow$ die Kante (w, u) und über diese ist u mit cost(u)= 2 erreichbar.

Kanten im Basis-Graph. Falls eine Kante wieder eine Abkürzung ist, kann sie rekursiv weiter entpackt werden. Schließlich blieben nur Kanten aus dem Basis-Graph übrig und der CH-Pfad wurde so in einen Basis-Pfad umgewandelt.

2.3.4 Pfadrepräsentationen

Es gibt verschiedene Möglichkeiten, einen Pfad zu speichern. Im Abschnitt zur Graphdefinition wurden bereits zwei vorgestellt: In der Knotendarstellung (Kantendarstellung) werden alle Knoten (Kanten) als Liste gespeichert. Mit dem CH-Graph kann nun jeweils auch noch die CH-Darstellung verwendet werden: Anstelle aller Knoten bzw. Kanten werden nur die Knoten bzw. Kanten angegeben, die bei der Suche im CH-Graph gefunden wurden. Diese Darstellung verbraucht deutlich weniger Speicher, allerdings sind nicht mehr alle Informationen explizit gegeben. Um alle Knoten eines Pfades zu erhalten, müssen die CH-Kanten entpackt werden.

2.4 Quadtree

Ein Quadtree ist eine Datenstruktur, die eine Punktmenge $P \subset \mathbb{R}^2$ speichert und effiziente Bereichsabfragen ermöglicht. Wir beschränken uns hier auf statische Quadtrees und Punktmengen, die im Einheitsquadrat liegen (diese Bedingung ist einfach durch Skalierung der Daten erfüllt).

Besonders relevant für die Verwendung in dieser Arbeit ist der Zusammenhang zwischen einem Quadtree und einem Gitter auf dem Einheitsquadrat.

Definition 2.4.1 (Zellen und Gitter)
Ein Gitter G_d ist eine Zerlegung des Einheitsquadrats in 4^d gleich große Quadrate. Eine Zelle z mit Tiefe d ist ein solches Quadrat, das im Gitter G_d liegt. Die Seitenlänge von z ist 2^{-d}.

Ein Quadtree ist eine hierarchische Zerlegung des Einheitsquadrats in Gitter. Ein Knoten der Tiefe d im Quadtree entspricht genau einer Zelle der Tiefe d und alle Knoten der Tiefe d ergeben das Gitter G_d. Ein Knoten hat vier Kinder; diese entsprechen der Zerlegung der Zelle in die vier kleineren Zellen der Tiefe $d + 1$.

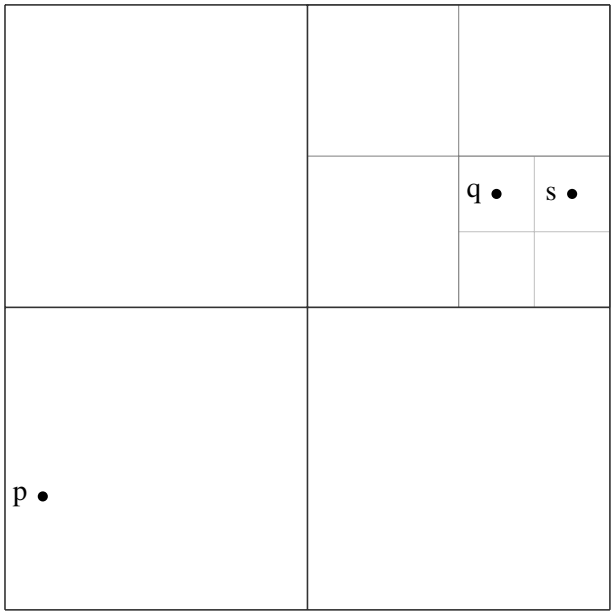


Abbildung 2.5: Ein Quadtree (der Tiefe 3). Das Einheitsquadrat ist in bis zu drei Gitter zerlegt und jede Zelle ist einem Knoten im Quadtree zugeordnet. Bild nach [Har11, Bild 2.2]

Konstruktion Die Konstruktion erfolgt rekursiv: Die Punkte einer Zelle werden entsprechend ihrer Koordinaten auf die vier Zellen der nächsten Tiefe aufgeteilt und diese Zellen als Kinder des aktuellen Knotens gespeichert. Das Vorgehen wird wiederholt, bis eine Zelle nur noch einen Punkt enthält - der entsprechende Knoten ist dann ein Blatt im Quadtree. Optional kann die Konstruktion auch bei einer beliebig gewählten maximalen Tiefe abgebrochen werden; dann können auch mehrere Punkte in einem Blatt gespeichert sein. Die Laufzeit und Größe des Quadtrees hängen insbesondere davon ab, wie nah einzelne Punkte zueinander liegen (näher zusammen $\Rightarrow$ feineres Gitter nötig, damit die Punkte in verschiedene Zellen fallen). In dieser Arbeit werden jedoch nur Quadtrees verwendet, bei denen eine (geringe) maximale Tiefe vorgegeben ist; die Konstruktionszeit ist dann praktisch nur abhängig von der gewählten maximalen Tiefe.

2.5 Well-separated pair decomposition

Die well-separated pair decomposition (WSPD) [CK95] ist eine Methode, mit der Abstände zwischen Punkten effizient beschrieben werden können. Dazu sollen Punktpaare, die sich ähnlich sind, zusammengefasst werden. Im Beispiel in Abbildung 2.6 sind die Punkte q und s von p etwa gleich weit entfernt und aus der Perspektive von p sind q und s nah beieinander. Fasst man die Punktpaare (s, p) und (q, p) zu $(\{s, q\}, p)$ zusammen, wird diese Struktur effizient beschrieben. Mit dieser Beschreibung können z.B. Informationen, die auf die einzelnen Paare zutreffen, für nur ein repräsentatives Paar berechnet und für alle Paare verwendet werden.

Die Definition und der Algorithmus zur Berechnung einer WSPD sind aus [Har11] übernommen.

Abbildung 2.6: Die Abstände der Paare (p,q) und (p,s) sind ähnlich und aus Sicht von p sind q und s nah zueinander. Bild nach [Har11, Abb. 3.1]

Definition Betrachten wir zuerst die Definition der WSPD. Sei $A \otimes B = \{\{x, y\} \mid x \in A, y \in B, x \neq y\}$ die Menge aller Punktpaare der Mengen A und B. Diese Menge wird *Pair* von A und B genannt. Wir suchen eine möglichst kleine Menge dieser Paare, die alle Punkte abdeckt.

Definition 2.5.1 (Pair Decomposition)
Eine Pair Decomposition einer Punktmenge $P \in \mathbb{R}^2$ ist

$W = \{\{A_1, B_1\}, \cdots, \{A_s, B_s\}\}$

sodass gilt

1. $\forall i : A_i, B_i \subset P$

2. $\forall i : A_i \cap B_i = \emptyset$

3. $\cup_{i=1}^{s} A_i \otimes B_i = P \otimes P$

Eine pair decomposition ist also eine Zerlegung von P in Paare von Teilmengen, sodass jedes Punktpaar in mindestens einem Paar (und üblicherweise in genau einem Paar) enthalten ist.

Definition 2.5.2
Ein Paar Q, R ist $(1/\epsilon)$-separiert, wenn

$\max(\mathsf{diam}(Q), \mathsf{diam}(R)) \leq \epsilon * \mathbf{d}(Q, R)$

mit $\mathbf{d}(Q, R) = min_{q \in Q, r \in R} |q - r|$

Ein Paar ist $(1/\epsilon)$-separiert, wenn die Punkte der Mengen jeweils *nah zusammen* und die Mengen voneinander *weit entfernt* sind. Beide Definitionen zusammen ergeben dann die WSPD.

Definition 2.5.3 (Well-separated Pair Decomposition)
Eine well-separated Pair Decomposition einer Punktmenge P mit Parameter $1/\epsilon$ ist eine pair decomposition

$W = \{\{A_1, B_1\}, \cdots, \{A_s, B_s\}\}$

sodass gilt

$\forall i : A_i, B_i$ *sind* $(1/\epsilon)$*-separiert.*

Algorithmus 2.3 Konstruktion der WSPD. Algorithmus aus [Har11]

```
procedure AlgWSPD(u, v)
    if u = v and Δ(u) = 0 then                          // Δ(u) = diameter of u
        return                                          // do not pair a leaf with itself
    end if
    if Δ(u) < Δ(v) then
        exchange u and v
    end if
    if Δ(u) ≤ ϵ ∗ d(u, v) then
        return {{u, v}}
    end if
    return ⋃_{i=1}^{4} AlgWSPD(u_i, v)                  // u_1, ..., u_4 children of u
end procedure
```

Berechnung Eine WSPD kann mithilfe eines Quadtrees berechnet werden wie in Algorithmus 2.3 beschrieben. Die Zellen, die den Knoten des Quadtrees entsprechen, werden als Punktmengen für die WSPD verwendet. Die WSPD ist also eine Sammlung von Zellpaaren. Wir definieren für die Berechnung des WSPD den Durchmesser einer Zelle als die Kantenlänge 2^{-k}, falls mehr als ein Punkt in der Zelle enthalten ist, und sonst 0. Der Abstand zwischen zwei Zellen ist der kleinste Abstand der Zellenecken.

Beginnend mit der Wurzel wird der Baum rekursiv durchlaufen. In jedem Schritt wird geprüft, ob das aktuelle Paar (u, v) schon $(1/\epsilon)$-separiert ist. Falls ja, wird dieses Paar zur Lösung hinzugefügt und die Rekursion beendet. Falls nein, wird der größere Teilbaum (u) in seine vier Kinder $(u.a, u.b, u.c, u.d)$ zerlegt und der Algorithmus rekursiv auf den vier Paaren $(u.a, v)$, $(u.b, v)$, $(u.c, v)$, $(u.d, v)$ ausgeführt. An dieser rekursiven Form ist leicht zu sehen, dass die Berechnung einfach parallel ausgeführt werden kann.

Da jedes Blatt eines Quadtrees nur einen Punkt enthält und somit Durchmesser 0 hat, wird in jeder Rekursion spätestens mit den Blättern ein Paar gefunden. Somit erzeugt der Algorithmus eine Pair Decomposition und jedes Paar ist $(1/\epsilon)$-separiert $\Rightarrow$ der Algorithmus berechnet eine WSPD. Dies gilt jedoch nur für vollständige Quadtrees: Falls ein Blatt mehr als einen Punkt enthält (wie im Fall eines begrenzten Quadtrees), könnten einige Punktpaare nicht im Ergebnis enthalten sein. Insbesondere sind alle Paare aus Punkten, die im gleichen Blatt gespeichert sind, nicht im Ergebnis enthalten. In dieser Arbeit werden jedoch auch solche unvollständigen Ergebnisse des Algorithmus WSPD genannt.

Zusätzlich ist diese WSPD minimal im Bezug auf die Anzahl der Paare, denn aus der Konstruktion ergibt sich, dass für jedes Paar gilt: Ersetzt man eine Zelle in einem Paar durch die Elternzelle, ist dieses Paar noch nicht $(1/\epsilon)$-separiert. Es können daher keine Paare zusammengefasst werden - die WSPD ist also minimal groß. Außerdem ist jedes Punktpaar maximal einmal in der WSPD vorhanden, da die rekursive Berechnung abgebrochen wird, wenn ein Zellpaar in die WSPD eingefügt wird.

3 Berechnung repräsentativer Pfadmengen

Wie in der Einleitung beschrieben, ist eines der beiden Ziele dieser Arbeit die Generierung einer repräsentativen Pfadmenge mithilfe einer well-separated pair decomposition. Dieses Verfahren wird in diesem Kapitel vorgestellt. Die Grundlagen des Quadtrees und WSPD wurden bereits im vorherigen Kapitel zusammengefasst; hier folgt nun die Anwendung auf Kartendaten und eine genauere Motivation des Vorgehens.

Zunächst wird formal definiert, wie eine repräsentative Stichprobe aussehen könnte. Danach folgt die hier verwendete Methode, die eine repräsentative Stichprobe generieren soll. Anschließend gehen wir noch auf einige Details der Implementierung ein.

3.1 Pfadmengen und Stichproben

Da es in großen Graphen zu viele Punktpaare und damit zu viele kürzeste Wege gibt, um alle Wege betrachten zu können, müssen wir uns mit einer Stichprobe zufriedengeben. Es stellt sich die Frage, wie eine *repräsentative* Stichprobe aussehen soll.

Betrachten wir dazu zuerst eine einfache Methode, eine Stichprobe zu erzeugen und überlegen, welche Probleme sich daraus ergeben: Um diese Stichprobe zu erzeugen, wählt man zufällige Punktpaare und berechnet jeweils den kürzesten Weg zwischen ihnen. Aus der Verteilung der Punkte auf der Karte ergibt sich, dass die so erzeugten Pfade sich oft recht ähnlich sind: Man kann z. B. damit rechnen, dass es mehrere Punktpaare geben wird, die jeweils einen Pfad von Stuttgart nach Berlin erzeugen, da es in beiden Städten sehr viele Punkte gibt. Dagegen wird es (wenn die Stichprobe nicht sehr groß ist) wahrscheinlich überhaupt kein Punktpaar geben, das die Verbindung zwischen zwei kleinen Dörfern im Schwarzwald beschreibt, da beide Dörfer jeweils aus nur wenigen Punkten bestehen.

Nun kann man sich die Frage stellen, ob eine solche Pfadmenge *repräsentativ* für alle Pfade im Graph ist. Bezogen auf die Häufigkeit der Pfade ist die Stichprobe sicherlich korrekt. Allerdings könnte man die Häufigkeiten, mit denen *ähnliche* Pfade (also z. B. alle Pfade, die zwischen Stuttgart und Berlin verlaufen) auftreten, auch effizienter beschreiben, indem man jedem Pfad in der Stichprobe ein Gewicht gibt und dieses dann bei der Berechnung des Hitting Sets berücksichtigt. Speichert man nun also nur noch einen solchen Pfad von Stuttgart nach Berlin mit großem Gewicht, verbleibt deutlich mehr (Speicher-) Platz in der Stichprobe, um auch die weniger häufigen lokalen Verbindungen mit aufzunehmen. Diesen Gedanken wollen wir weiter verfolgen. Es stellt sich die Frage, wann Pfade *ähnlich* sind, wie man möglichst viele nicht-*ähnliche* Pfade findet, und wie man für diese effizient ein Gewicht berechnen kann.

Lokale Veränderungen Eine Beobachtung zu kürzesten Wegen in Straßennetzen ist, dass sich kürzeste Wege zwischen Punktpaaren, die nah zusammen liegen (die Startpunkte und Zielpunkte beider Paare liegen jeweils nah zueinander) einen ähnlichen Verlauf haben: abgesehen von einem Teilpfad am Anfang und Ende verlaufen beide über die gleichen großen Straßen und Autobahnen. Besonders für Paare mit großem Abstand gilt, dass sich der Weg kaum verändert, wenn man einen Punkt lokal verschiebt - der Großteil des Weges bleibt gleich. Mit dieser Überlegung definieren wir *ähnliche* Pfade: Wir nennen solche Wege, die sich durch lokale Verschiebungen kaum verändern, δ-ähnlich:

Definition 3.1.1 (δ-ähnliche Pfade)
Gegeben zwei Pfade π_1 und π_2. Sei M der längste Pfad, so dass gilt:

M ist Teilweg von π_1

M ist Teilweg von π_2

M ist der längste Weg dieser Art.

Zwei Wege sind δ-ähnlich, wenn gilt: $\max(\frac{\text{cost}(M)}{\text{cost}(\pi_1)}, \frac{\text{cost}(M)}{\text{cost}(\pi_2)}) \geq \delta$

Aus dieser Definition ergeben sich Pfadmengen, für die gilt, dass sich alle Pfade in der Menge ähnlich sind. Für unsere Stichprobe reicht uns aus einer solchen Pfadmenge ein Pfad, zusammen mit der Anzahl der ähnlichen Pfade als Gewicht. Definieren wir also eine komprimierte Stichprobe:

Definition 3.1.2 (δ-komprimierte Stichprobe)
Eine δ-komprimierte Stichprobe ist eine gewichtete *Menge von Pfaden. Für alle Paare von Pfaden gilt: die Pfade sind nicht δ-ähnlich.*

Diese komprimierte Stichprobe benötigt deutlich weniger Speicherplatz und kann somit bei gleichem Speicherverbrauch mehr lokale Wege enthalten als eine einfache Stichprobe.

Eine δ-komprimierte Stichprobe kann aus einer Stichprobe erzeugt werden, indem für jedes Paar von δ-ähnlichen Pfaden einer entfernt und der andere entsprechend stärker gewichtet wird, bis keine ähnlichen Pfade mehr enthalten sind. Dieses Vorgehen ist jedoch praktisch nicht umsetzbar: Selbst wenn man die einfache Stichprobe nach Bedarf nach und nach erzeugt (um den Speicherplatz gering zu halten), müsste man für jedes Punktpaar den Pfad berechnen und dann mit allen bereits in der komprimierten Stichprobe enthaltenen Pfaden auf Ähnlichkeit überprüfen. Es werden dabei sehr viele kürzeste Pfade berechnet, die in der komprimierten Stichprobe nur als höheres Gewicht eines anderen Pfades vorkommen. Mit dieser Methode würde also eine Anzahl an kürzesten Pfaden berechnet werden, die der Summe der Gewichte in der komprimierten Stichprobe entspricht. Ziel wäre jedoch, die Summe der Gewichte (und damit die Anzahl der mit der komprimierten Stichprobe repräsentierten Pfade) möglichst groß, im besten Fall etwa bis $O(|V|^2)$ zu bringen; in dieser Größenordnung ist die Berechnung aller Pfade praktisch nicht umsetzbar.

Es stellt sich daher die Frage, wie man eine komprimierte Stichprobe erzeugt (oder zumindest etwas Ähnliches), ohne vorher eine deutlich größere einfache Stichprobe berechnen zu müssen. Hier kommt die WSPD zur Hilfe.

3.2 Stichproben mit WSPD

Aus der Definition der WSPD wissen wir, dass Zellpaare der WSPD im Verhältnis zu ihrer Größe weit voneinander entfernt sind. Daraus ergibt sich, dass kürzeste Pfade zwischen Punktpaaren aus beiden Zellen zueinander ähnlich sein sollten, da die Start- und Zielpunkte nur lokal (im Verhältnis zur Länge des Pfades) verschoben werden. Gleichzeitig sollten sich die Pfade, die sich aus verschiedenen Zellpaaren ergeben, nicht ähnlich sein, da die Start- und Zielpunkte dann eben nicht nur lokal, sondern zumindest bis in eine andere Zelle verschoben werden. Da diese Zerlegung nur geometrisch definiert ist, wird es besonders an den Rändern benachbarter Zellen natürlich auch Punktpaare geben, die sich ähnlich sind. Zugleich ist die Zerlegung der Pfade in Klassen über die Definition der δ-ähnlichen Pfade auch nicht eindeutig; irgendwie muss die Zuordnung jedoch entschieden werden und in dieser Lösung erfolgt diese über die geometrische Aufteilung des Quadtrees. Wegen der geometrischen Zerlegung der Punktmenge im Quadtree, die nur indirekt mit der Struktur des Graphen zusammenhängt[1], kann hier auch nicht formal gezeigt werden, dass sich aus der WSPD eine komprimierte Stichprobe ergibt, sondern nur mit der Idee und den Versuchsergebnissen argumentiert werden. In der Auswertung werden wir auch Beispiele sehen, die diesen Unterschied zwischen geometrischer Zerlegung und Straßennetz zeigen.

Wählt man nun für jedes Zellpaar der WSPD einen Pfad aus und gewichtet ihn mit der Anzahl der Punktpaare zwischen den Zellen, erhält man eine gewichtete Pfadmenge, die zumindest der Idee der komprimierten Stichprobe entspricht: Die WSPD enthält viele kleine Zellpaare aus Zellen mit großer Tiefe - hier erhalten wir viele lokale Pfade mit geringem Gewicht. Außerdem enthält sie wenige größere Paare, aus denen wir lange Pfade mit hohem Gewicht erhalten. Allerdings ist diese Pfadmenge keine echte Stichprobe im Sinne der gleichmäßigen zufälligen Auswahl von Pfaden mehr und wir können auch nicht ausschließen, dass sich zwei Pfade ähnlich sind. Daher nennen wir diese so erzeugte Pfadmenge eine repräsentative Pfadmenge. Die so erzeugten repräsentativen Pfadmengen werden in der Auswertung in Kapitel 5 genauer betrachtet. Zuvor folgen noch einige Aspekte der Implementierung und Details zur Wahl der Parameter.

3.3 Berechnung mit Kartendaten

Betrachten wir nun noch die praktische Implementierung in einem Straßennetz. Um auf dem Straßennetz eine WSPD zu berechnen, verwenden wir die Koordinaten der Knoten im Graph. Die Koordinaten werden mit der Mercator-Projektion und einer Skalierung in den Wertebereich $[0, 1]$ in das Einheitsquadrat übertragen.

Anschließend wird ein Quadtree mit einer maximalen Tiefe d_{max} erzeugt, wobei d_{max} dann so gewählt werden sollte, dass die Zellen im kleinsten Gitter $G_{d_{max}}$ noch immer eine sinnvolle Größe haben und einen kleinen lokalen Bereich umfassen. Würde man kein d_{max} festlegen, erhalten wir mit der WSPD schließlich auch alle Pfade zwischen benachbarten Punkten (da jedes Blatt des Quadtrees nur einen Punkt enthält und diese Blätter dann well-separated Pairs bilden); dann wäre die erzeugte Pfadmenge viel zu groß um damit weiterarbeiten zu können. Zu Beachten ist außerdem, dass mit der Wahl eines d_{max} die erzeugte WSPD nicht mehr alle $O(|V|^2)$ Punktpaare abdeckt;

[1] Der Zusammenhang hat insbesondere damit zu tun, wie stark sich der Graph des Straßennetzes von einem vollständigen euklidischen Graph unterscheidet

der Anteil der abgedeckten Paare sinkt mit kleinerer maximalen Tiefe des Quadtrees. Die Punkte, die nicht abgedeckt werden, sind insbesondere alle Punktpaare, die sich im Quadtree in einem Blatt befinden, also sehr lokale Pfade. Eine genauere Auswertung der Parameter am Beispiel des deutschen Straßennetzes folgt in Kapitel 5.

Mit dem Quadtree wird dann die WSPD berechnet. Hier kann auch die Separationskonstante ϵ der WSPD gewählt werden. d_{max} und ϵ beeinflussen, wie viele Punktpaare in der WSPD repräsentiert sind und wie viele Zellpaare gefunden werden und damit auch, wie viele Pfade schließlich in der repräsentativen Pfadmenge enthalten sind.

Für jedes Zellpaar des WSPD wird der kürzeste Pfad berechnet, indem eine Dijkstra-Suche von allen Punkten der einen Zelle zu allen Punkten der anderen Zelle gleichzeitig ausgeführt wird. Der erste gefundene Weg ist der kürzeste Weg zwischen den Zellen und wird in die repräsentative Pfadmenge mit dem Gewicht entsprechend der Größe der Zellen aufgenommen.

Um die kürzesten Wege schneller berechnen zu können wird eine CH verwendet. Außerdem werden die Pfade in der repräsentativen Pfadmenge in CH-Kantendarstellung (wie in Abschnitt 2.3.4 beschrieben) gespeichert, da die Kantendarstellung für die Berechnung des Hitting Sets nützlich ist. CH-Pfade benötigen weniger Speicherplatz und ermöglichen somit eine größere Pfadmenge.

4 Hitting Set

Der zweite Teil dieser Arbeit ist die Berechnung *wichtiger* Punkte basierend auf einer repräsentativen Pfadmenge. Wie bereits in der Einleitung beschrieben definieren wir *Wichtigkeit* als *Häufigkeit eines Punktes in der Pfadmenge*, basierend auf der Intuition, dass Punkte, die in vielen Pfaden enthalten sind, wichtig sind. Damit ergibt sich dann das Hitting Set Problem:

Definition 4.0.1 (Hitting Set Problem)
Gegeben ein Graph $G = (V, E)$ und eine Pfadmenge P von Pfaden aus G.

Wähle die kleinste Menge von Knoten $H \subset V$, sodass jeder Pfad mindestens einen Knoten aus H enthält: $\forall \pi \in P : \pi \cap H \neq \emptyset$.

Das Problem ist NP-hart und da wir das Problem auf sehr großen Eingaben (etwa 10^7 Pfade) lösen wollen, werden wir nur eine Approximation mit einem Greedy-Algorithmus berechnen. Wir gehen hier davon aus, dass die Pfadmenge gewichtet ist und in CH-Kantendarstellung vorliegt, wie z. B. von der Berechnung im letzten Kapitel erzeugt.

Nachfolgend wird zuerst die grundlegende Idee des Algorithmus vorgestellt. Danach folgt eine genaue Betrachtung der Struktur des CH-Graphen, die benötigt wird, um das Hitting Set effizient zu berechnen. Anschließend folgen noch einige Tricks, mit denen die Laufzeit reduziert werden kann. Im letzten Abschnitt wird noch eine Methode zur schnellen Abschätzung einer unteren Grenze für die Größe des Hitting Sets vorgestellt.

4.1 Grundidee

Die Idee des Greedy-Algorithmus ist in Algorithmus 4.1 beschrieben: Solange noch Pfade übrig sind, wird der häufigste Knoten zum Hitting Set hinzugefügt und die von diesem Knoten getroffenen Pfade aus der Eingabe entfernt.

Grundsätzlich müssen hier zwei Schritte ausgeführt werden:

Zum einen wird ein Histogramm berechnet. Dabei ist zu beachten, dass die Pfade in CH-Kantendarstellung vorliegen und wegen des hohen Speicherbedarfs nicht in einfache Knotenlisten umgewandelt werden können. Naiv könnte man die Pfade nacheinander betrachten, jeden Pfad in eine Knotenliste entpacken, die Häufigkeiten zählen und die Knotenliste dann wieder löschen. Dabei werden CH-Kanten, die in mehreren Pfaden vorkommen, jedoch mehrfach entpackt. Eine deutlich effizientere Methode ist dagegen, zuerst die Häufigkeiten der CH-Kanten zu zählen und diese dann auf die abgekürzten Knoten zu übertragen - und somit jede CH-Kante nur einmal zu entpacken, unabhängig davon, in wie vielen Pfaden sie vorkommt. Eine genauere Beschreibung dieses Schrittes folgt im Abschnitt 4.3.

Algorithmus 4.1 Idee des Hitting Set Algorithmus

```
procedure GreedyHittingSet(G = (V, E), P)
    H ← ∅                                                    // Hitting Set
    while P ≠ ∅ do
        C ← gewichtete Häufigkeiten aller Knoten in P
        v ← argmax_k C(k)
        H ← H ∪ {v}
        P ← {π ∈ P | v ∉ π}
    end while
    return H
end procedure
```

Zum anderen müssen im zweiten Schritt für einen Knoten alle Pfaden gefunden werden, die ihn enthalten. Auch hier besteht wieder das Problem, dass eine naive Lösung, die alle Pfade nacheinander betrachtet, sehr langsam ist. Da die Zuordnung von Knoten zu Pfaden sich in den Iterationen des Greedy-Algorithmus nicht ändert, kann die Abbildung von Knoten auf Pfade einmalig berechnet und dann wiederverwendet werden. Im einfachsten Fall benötigt man dafür allerdings so viel Speicherplatz wie die Pfadmenge in der Knotendarstellung benötigen würde (für jeden Knoten in jedem Pfad wird ein Zeiger auf den entsprechenden Pfad gespeichert). Diese Variante ist also auch nicht umsetzbar; eine effizientere Methode basierend auf der Struktur des CH-Graphen wird in Abschnitt 4.4 vorgestellt.

4.2 Meta-Graph der CH-Konstruktion

Um die beiden Schritte des Greedy-Algorithmus effizient umzusetzen, benötigen wir eine weitere Eigenschaft der Struktur des CH-Graphen. Wir definieren hier den *Metagraph* des CH-Graphs. Die Metaknoten des Metagraphen sind die Kanten und Knoten des CH-Graphs. Die Metakanten beschreiben, welche Kante des CH-Graphen welche andere Kante abkürzt.

Zuerst einige Bezeichnungen: Kanten, die im Basis-Graph enthalten sind, heißen Basis-Kanten E_B. Die Abkürzungskanten, die in der CH-Konstruktion neu erzeugt werden, heißen CH-Kanten E_{CH}. Beide Kantenmengen zusammen sind $E = E_{CH} \cup E_B$. Der CH-Graph, der aus der Konstruktion entsteht, ist dann $G_{CH} = (V, E)$

Jede CH-Kante $e = (u, w) \in E_{CH}$ ersetzt genau zwei Kanten, denn sie ist eine Abkürzung, die genau einen Knoten umgeht (folgt aus der Konstruktion der CH: bei der Kontraktion wird immer genau ein Knoten und somit die zwei Kanten zu den Nachbarn entfernt). Wir nennen diese ersetzten Kanten die Kinder von e. Es gilt also für jede CH-Kante: es gibt genau zwei Kinder und für jede Basis-Kante: es gibt genau 0 Kinder. Alle Eltern-Kind-Beziehungen sind mit $C_{CH} = \{(p, c) \mid c \text{ ist Kind von } p\} \subseteq E_{CH} \times E$ beschrieben.

Diese Kantenbeziehungen erzeugen einen Graph $G_E = (E, C_{CH})$. Aus der Konstruktion des CH-Graphen ergibt sich, dass G_E kreisfrei ist: Wird in der CH-Konstruktion ein Knoten kontrahiert, werden Elternkanten (Abkürzungskanten) erzeugt. Alle Elternkanten sind neue Kanten und die Kinder dieser Kanten werden zum Zeitpunkt der Erzeugung festgelegt. Um einen Kreis zu erhalten,

müssten die Kinder einer bereits vorhandenen Kante verändert werden; dies ist jedoch nie der Fall. Daher erzeugen die Abkürzungskanten keine Kreise; die Basiskanten haben keine Kinder und können somit auch keine Kreise enthalten.

Legt man nun noch die Knoten, die zu einer Basiskante gehören, als Kinder dieser Basiskante fest, erhält man $C_B = \{(e, v) \mid e = (v, *) \vee e = (*, v)\} \subset E_B \times V$. Daraus ergibt sich der *Metagraph* des CH-Graphen: $G_M = (E \cup V, C_{CH} \cup C_B)$. Dieser Graph ist immer noch kreisfrei, da die neu eingefügten Knoten V keine ausgehenden Kanten in G_M haben. G_M ist somit ein DAG. Ein Beispiel ist in Abbildung 4.1 gegeben. Die Kanten des Metagraphen G_M (also $C_{CH} \cup C_B$) werden hier immer *Meta*kanten genannt. Der Metagraph ist implizit bereits im CH-Graph gespeichert, da dieser die Kindkanten der CH-Kanten kennen muss, um Pfade entpacken zu können. Zur Berechnung des Hitting Sets ist jedoch eine Variante, die auch die Elternkanten explizit speichert (in der Implementierung zu dieser Arbeit z. B. durch zusätzliche Pointer zu den Eltern), nützlich, da auch diese Information benötigt wird.

Abdeckung von Kanten und Knoten Aus G_M lassen sich Abdeckungen von Kanten und Knoten ablesen. Diese Information wird für die Hitting Set Berechnung benötigt. Für die beiden hier beschriebenen Operationen ist je ein Beispiel in Abbildung 4.2 zu finden.

Betrachtet man den Pfad $\pi = e$, der von einer einzigen Kante $e \in E$ beschrieben wird und schreibt ihn in Knotendarstellung auf, erhält man eine Liste aus mindestens zwei, potenziell aber deutlich mehr Knoten (falls e eine CH-Kante mit vielen rekursiven Kindern ist). Diese Knoten sind die von *e abgedeckten* Knoten. Für eine Kante $e \in E$ können aus G_M alle Knoten abgelesen werden, die von e abgedeckt werden: Die Blätter V', die von e in G_M erreichbar sind, sind genau die Knoten, die durch e abgedeckt werden.

Anders herum können auch alle Kanten angegeben werden, die einen Knoten $v \in V$ abdecken: Dies sind die Kanten E', die im inversen Metagraph G_M^{-1} von v aus erreichbar sind. V' und E' können jeweils mit einer einfachen Breitensuche o.Ä. gefunden werden.

Wenn man die Abdeckung mehrerer Kanten (bzw. Knoten) finden möchte, kann die Suche auch von mehreren Kanten (bzw. Knoten) aus gleichzeitig gestartet werden. Dann erhält man die Knoten (bzw. Kanten), die von mindestens einer Kante (bzw. Knoten) abgedeckt werden. In diesem Fall sollte die Suche nach der topologischen Sortierung des Metagraphen priorisiert werden, damit jeder Metaknoten maximal einmal bearbeitet wird.

4.3 Histogramm berechnen

Betrachten wir nun die Berechnung des Histogramms. Da die Pfade in der CH-Kantendarstellung vorliegen, müssen diese entpackt werden, um alle Knoten zu zählen. Wie bereits beschrieben ist die naive Methode, Pfade nacheinander zu entpacken, ineffizient, da die gleiche CH-Kante mehrfach entpackt wird, falls sie in mehreren Pfaden vorkommt. Deutlich effizienter ist das Entpacken aller Pfade gleichzeitig.

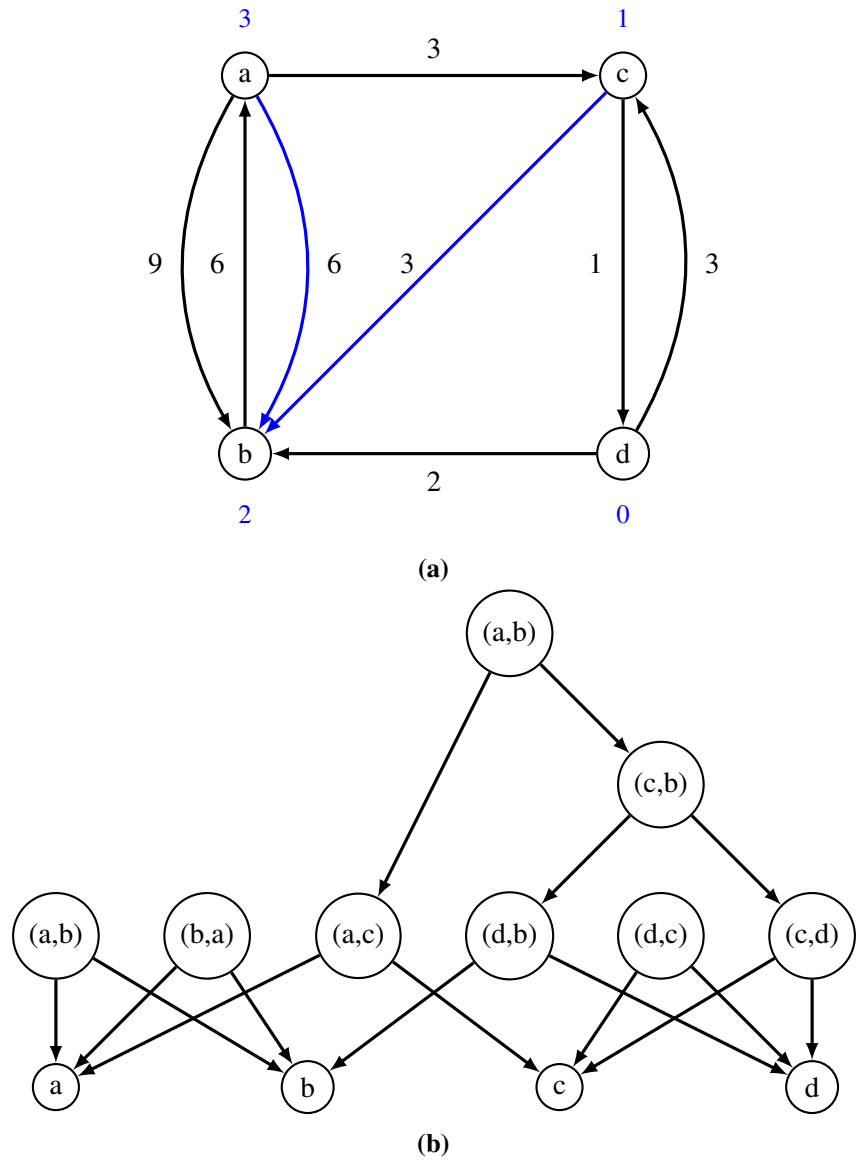
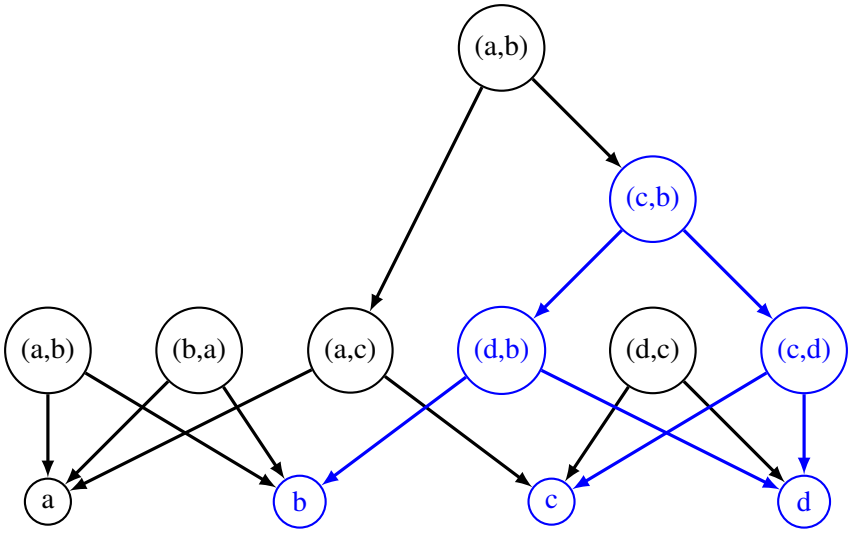
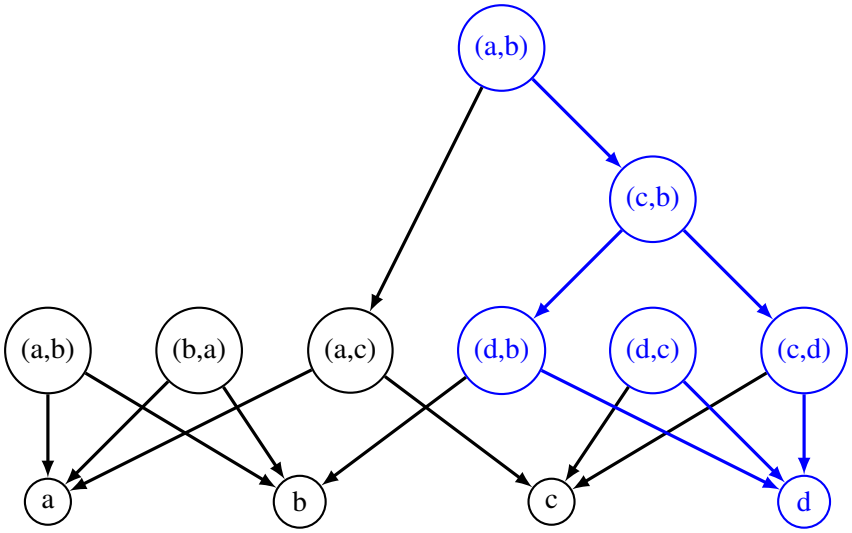


Abbildung 4.1: Oben: Ein Graph (schwarze Kanten) und der CH-Graph (blaue Kanten: CH-Kanten; Level der Knoten in blau). Unten: Der Metagraph.

Häufigkeiten zählen Dazu werden zuerst die Häufigkeiten der CH-Kanten in den Pfaden gezählt. Anschließend werden die CH-Kanten entpackt und ihre Häufigkeiten auf die Kinderkanten übertragen. Diese Operation ist also eine Suche nach den abgedeckten Knoten in G_M, allerdings wird beim Durchsuchen von G_M immer auch die Häufigkeit der Elternkante auf die Kinder übertragen. Hat ein Knoten mehrere Eltern mit einer Häufigkeit, werden diese addiert. Irgendwann werden in der Suche Basiskanten erreicht; hier werden die Kantenhäufigkeiten dann auf die Knoten übertragen. Um zu verhindern, dass dabei Knoten doppelt gezählt werden (jeder Knoten des Pfades hat eine eingehende und eine ausgehende Kante, die jeweils den Knoten zählen. Ausnahme: Start- und Zielknoten), wird die Häufigkeit einer Kante hier nur auf ihren Zielknoten übertragen. Anschließend werden die Startknoten aller Pfade je einmal gezählt. Somit wird jeder Knoten eines Pfades genau



(a) Die Kante (c, b) deckt die Knoten b, c, d ab.



(b) Der Knoten d wird von den Kanten $(d, b), (d, c), (c, d), (c, b), (a, b)$ abgedeckt.

Abbildung 4.2: Kanten- und Knotenabdeckung im Metagraph

einmal gezählt, wie in Abbildung 4.3 zu sehen ist. Wichtig ist die Reihenfolge, in der die Häufigkeiten der Kanten auf ihre Kindkanten übertragen werden: Ist sichergestellt, dass die Werte der Eltern bereits vollständig sind, wenn sie auf die Kinder übertragen werden, dann muss jede Kante insgesamt nur einmal bearbeitet werden. Da G_M ein DAG ist, existiert eine topologische Sortierung und somit eine Reihenfolge, die diese Bedingung erfüllt.

Vollständige und explorative Suche Grundsätzlich gibt es zwei Möglichkeiten, diese Kantenzählung zu implementieren. Zum einen können einfach alle Kanten nacheinander in topologischer Reihenfolge bearbeitet werden, also eine vollständige Suche durchgeführt werden. Die Laufzeit dieser Methode ist unabhängig von der Anzahl der Pfade, die betrachtet werden. Auch im Fall eines einzigen Pfades, der aus nur zwei Knoten besteht, werden immer alle Kanten genau einmal

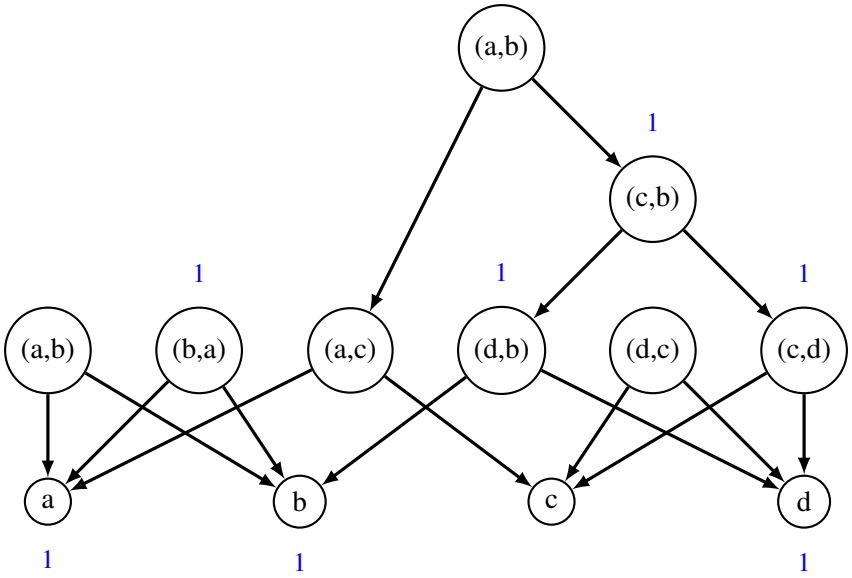


Abbildung 4.3: Übertragung des Pfades von c nach a, also $cdba = (c,b)(b,a)$ auf die Knoten: die Gewichte werden von (c,b) und (b,a) auf Kindkanten übertragen und dann auf die Zielknoten der Basiskanten. Zusätzlich muss c gezählt werden, da c der Startknoten des Pfades ist.

betrachtet. Vorteil dieser Methode ist, dass sie einfach ist, und ohne weitere Berechnungen zur Reihenfolge auskommt, wenn man die Kanten im CH-Graph in topologischer Reihenfolge bezüglich G_M speichert. Dann muss einfach die Kantenliste abgearbeitet werden.

Alternativ bearbeitet man nur die Kanten, die in den Pfaden vorkommen. In einen Heap, der topologisch sortiert ist, werden zuerst die Kanten der CH-Pfade eingetragen. Dann werden bei Bearbeitung einer Kante je die beiden Kinder zum Heap hinzugefügt und die Häufigkeit der Kante auf ihre Kinder übertragen. Im Vergleich mit der ersten sollte diese zweite Methode für kleine Eingaben schneller sein, da sie nur die relevanten Kanten betrachtet, und für große langsamer, da die Verwendung des Heaps aufwändiger ist als die einfache Iteration einer Liste.

Da die Eingabegröße im Verlauf der Hitting Set Berechnung immer kleiner wird, sollte eine Kombination beider Methoden, bei denen Anfangs die vollständige und später die explorative Methode verwendet wird, optimal sein. In unserer Implementierung legen wir einen Grenzwert für die Größe der Eingabe fest und verwenden in der Auswertung praktisch nur die explorative Variante, da diese selbst mit den größten getesteten repräsentativen Pfadmengen schneller als die vollständige Suche ist. Eine bessere Implementierung wäre jedoch eine adaptive Auswahl, die regelmäßig beide Varianten parallel ausführt und so ihre aktuellen Laufzeiten vergleicht. Da die Berechnung des Histogramms wie hier vorgestellt nicht parallel ausgeführt wird, ist ein solcher Vergleich auch ohne nennenswerte Kosten bezüglich der Laufzeit auf modernen Mehrkernprozessoren möglich.

Speedup mit Update statt Neuberechnung Beide Methoden berechnen ein Histogramm für eine Eingabe von Pfaden, $C(*)$. Diese Pfade können die "übrigen"(noch nicht getroffenen) Pfade sein - dann erhält man ein Histogramm für diese übrigen Pfade. Alternativ kann jedoch auch ein Histogramm für die Pfade berechnet werden, die gerade im vorherigen Schritt durch den neu gefundenen häufigsten Knoten v getroffen wurden.

Betrachtet man einen Schritt des Hitting Set Algorithmus: Seien P_i alle Pfade im Schritt i, R_i die Pfade, die den häufigsten Knoten in $C(P_i)$ enthalten und somit aus der Eingabe entfernt werden und $P_{i+1} = P_i \setminus R_i$ die übrigen Pfade und zugleich die Eingabe für den nächsten Schritt. Es gilt $C(P_{i+1}) = C(P_i) - C(R_i)$. R_i, P_i und $C(P_i)$ sind bekannt; gesucht ist $C(P_{i+1})$. Dazu muss nun nur $C(P_{i+1})$ *oder* $C(R_i)$ berechnet werden. Die Differenz lässt sich beim Kantenzählen sehr einfach implementieren - das alte Histogramm wird beibehalten und in dem Schritt, in dem die Häufigkeiten einer Basiskante auf den zugehörigen Knoten übertragen werden, werden diese Häufigkeiten vom alten Wert abgezogen. Je nachdem, ob P_{i+1} oder R_i größer ist, ist die Berechnung des entsprechenden Histogramms schneller und sollte verwendet werden.

4.4 Getroffene Pfade finden

Das Problem, das hier gelöst werden muss, lässt sich wie folgt formulieren: Gegeben ein Knoten v (üblicherweise der Häufigste aus dem Histogramm). Welche Pfade enthalten diesen Knoten? Dabei liegen die Pfade noch immer in CH-Kantendarstellung vor. Um dieses Problem platzsparend zu lösen, verwenden wir wieder die Abdeckungen in G_M.

In Abschnitt 4.2 wurde bereits gezeigt, dass alle Kanten, die einen Knoten abdecken (also als Pfad diesen Knoten enthalten), in G_M^{-1} von diesem Knoten aus erreichbar sind. Damit können also alle Pfade gefunden werden, die einen Knoten enthalten. Um alle Pfade zu finden, die eine dieser abdeckenden Kanten enthalten, wird in einem Vorbereitungsschritt zunächst an jeder Kante aller Pfade gespeichert, welche Pfade sie enthalten. Der benötigte Speicherplatz entspricht der Anzahl der Kanten in der CH-Kantendarstellung der Pfade, denn nur an diesen Kanten wird die Information zum entsprechenden Pfad gespeichert. Diese Vorbereitung hat lineare Laufzeit in der Größe der CH-Kantendarstellung.

Um nun alle Pfade für einen Knoten v zu finden, werden alle von v aus in G_M^{-1} erreichbaren Kanten betrachtet und alle Pfade, die an einer dieser Kanten gespeichert sind, ausgegeben.

4.5 Einfache Abschätzung der unteren Schranke

Für die Auswertung im nächsten Kapitel ist noch eine Abschätzung der unteren Schranke interessant. Wir wollen also abschätzen, wie groß das Hitting Set für eine bestimmte Eingabe mindestens ist. Eine recht einfache Methode ist, wiederholt einen Pfad auszuwählen und dieser mit allen Pfaden, die er schneidet, aus der Eingabe zu entfernen, bis keine Pfade mehr übrig sind. Die Anzahl der Schritte ist dann eine untere Schranke für die Größe des Hitting Sets: für jeden Pfad und seine Schnittpartner wird mindestens ein Knoten im Hitting Set benötigt (genau ein Knoten, falls sich alle diese Pfade in einem Punkt schneiden). Die Qualität der Abschätzung wird verbessert, wenn Pfade mit möglichst wenigen Schnitten priorisiert werden - allerdings ist diese Priorisierung sehr

aufwändig, da sie nach jedem Schritt neu berechnet werden muss. Daher wird in der Auswertung nur die einfachere Priorisierung nach Länge der Pfade verwendet. Diese muss nur einmal berechnet werden.

Schnittpfade finden Um alle Pfade zu finden, die einen Pfad π schneiden, wird ähnlich vorgegangen wie schon bei der Berechnung der Pfade, die einen Knoten enthalten. Betrachtet man wieder G_M, kann man alle Knoten, die π abdeckt, wie in Abschnitt 4.2 beschrieben finden. Dann kann die zusätzliche Information über Pfade an den Kanten wie in Abschnitt 4.4 verwendet werden, um von den abgedeckten Knoten aus in G_M^{-1} alle Pfade zu finden, die π schneiden.

5 Auswertung

Die Methoden der beiden vorherigen Kapitel wurden am Beispiel des deutschen Straßennetzes getestet. Wir betrachten die Ergebnisse und einige Messwerte und vergleichen diese mit unseren Erwartungen aus den vorherigen Kapiteln.

Datengrundlage der Auswertung ist das deutsche Straßennetzwerk aus OpenStreetMap [Ope17]. Das Straßennetzwerk wird mit dem OSMGraphCreator [Bah15] in einen Graph umgewandelt und aus dem Graph anschließend mit dem CHConstructor [NB17] ein CH-Graph berechnet. Für die Berechnung des Quadtrees werden die Koordinaten aller Knoten zuerst mit der Mercatorprojektion auf eine Ebene projiziert und anschließend in den Wertebereich $[0, 1] \times [0, 1]$ skaliert. CH-Graph und Quadtree werden dann für die Berechnung der repräsentativen Pfadmenge und des Hitting Sets verwendet. Die Implementierung ist auf Github verfügbar: https://github.com/bernlu/garp. Einige Auswertungen in diesem Kapitel beinhalten auch Messungen zu Laufzeiten; diese Ergebnisse wurden mit einem Intel i5-9500 und 64 GB Arbeitsspeicher gemessen.

Die Ergebnisse und Laufzeiten ausgewählter Parameter sind in Tabelle 5.1 zusammengefasst. Da das Testsystem nur Hitting Sets mit maximal etwa 60M Pfaden berechnen kann, sind hier Parameterpaare gewählt worden, bei denen ungefähr 60M Pfade erzeugt werden. Die Laufzeit in der Tabelle ist dabei nur die Zeit, die zur Berechnung der WSPD und des Hitting Sets benötigt wurde. Die Zeit zur Berechnung der kürzesten Pfade zwischen den WSPD-Paaren (mit einer CH-Suche) ist nicht angegeben und liegt bei etwa 2ms pro Pfad (parallel berechnet etwa 5.5h für 60M Pfade).

In den nächsten Abschnitten wird genauer auf die Einträge der Tabelle eingegangen. Zunächst wird die Wahl der Parameter für die repräsentative Pfadmenge beschrieben, dann genauer auf verschiedene Eigenschaften der repräsentativen Pfadmenge eingegangen und zuletzt das Hitting Set genauer betrachtet.

Parameter			Covering		Lower		Pfadabdeckung		
d	ϵ	| P |	Error	| HS |	Bound	Laufzeit	99 %	99.9 %	99.99 %
8	0.06	$63 * 10^6$	2.838 %	2505	957	11 min	379	837	1315
9	0.15	$49 * 10^6$	0.185 %	22978	10747	12 min	829	3444	8062
10	0.25	$59 * 10^6$	0.024 %	104322	60479	42 min	962	5595	18926
11	0.45	$60 * 10^6$	0.004 %	382127	248725	106 min	1051	6719	29394
12	0.9	$59 * 10^6$	0.001 %	1190699	844573	320 min	1136	7572	37624

Tabelle 5.1: Für eine Auswahl von Parameterpaaren: Größe der repräsentativen Pfadmenge *P* und Anteil der Punktpaare, die nicht repräsentiert sind (Covering Error), Größe des Hitting Sets (HS) und Abschätzung der minimalen Größe des Hitting Sets (Lower Bound), Berechnungszeit (ohne kürzeste Wege), und Größe eines Hitting Sets, das 99 %, 99.9 %, bzw. 99.99 % aller gewichteten Pfade in P abdeckt (Pfadabdeckung).

d	Zellgröße [m]	Punkte pro Blatt
5	27500 × 20000	41906,9
6	13750 × 10000	11224,8
7	6875 × 5000	2924,4
8	3438 × 2500	757,7
9	1720 × 1250	209,3
10	860 × 625	66,6
11	430 × 313	24,5
12	215 × 156	10,0
13	107 × 78	4,4

Tabelle 5.2: Zellgrößen und durchschnittliche Anzahl Punkte in einem Blatt des Quadtrees für verschiedene maximale Tiefen d im deutschen Straßennetz.

5.1 Repräsentative Pfadmengen

Zur Berechnung der repräsentativen Pfadmengen müssen zwei Parameter gewählt werden: die maximale Tiefe $d \in \mathbb{N}$ des Quadtrees und die Separationskonstante $\epsilon \in (0, 1)$. In der nachfolgenden Analyse wollen wir uns möglichst viele relevante Werte von ϵ ansehen; für d können wir jedoch vor der Auswertung der Pfadmengen schon eine Auswahl treffen.

Die Wahl von d legt fest, wie groß die kleinsten Zellen des Quadtrees sind und wir sollten d so wählen, dass die kleinsten Zellen eine lokale Punktmenge umfassen. Deutschland ist etwa 880km × 640km groß und eine Zelle auf Tiefe k ist somit etwa $2^{-k} * 880\text{km} \times 2^{-k} * 640\text{km}$ groß. In Tabelle 5.2 finden sich die gerundeten Größen der kleinsten Zellen für verschiedene Werte von d und außerdem die durchschnittliche Anzahl der Punkte, die in einem Blatt des Quadtrees gespeichert sind. Für die folgende Analyse werden wir hauptsächlich $d \in [8, 12]$ betrachten, da die kleinsten Zellen hier wenige Kilometer bis hunderte Meter groß sind. Diese kleinsten Zellen begrenzen die Länge kürzester Wege nach unten auf einen lokalen Bereich, der etwa einem Dorf oder Stadtteil entspricht.

5.1.1 Größe der repräsentativen Pfadmenge

Eine erste offensichtliche Auswertung ist die Größe der repräsentativen Pfadmenge, die der Anzahl der Zellpaare in der WSPD entspricht.

Die Größe der repräsentativen Pfadmenge beschreibt dabei indirekt, wie grob die kürzesten Pfade des Graphen zusammengefasst sind: In einer kleinen repräsentativen Pfadmenge repräsentiert jeder Pfad der Pfadmenge einen größeren Anteil der kürzesten Pfade im Graph. Dann ist die repräsentative Pfadmenge also eine eher grobe, stark komprimierte Zusammenfassung des Graphen. Wird die repräsentative Pfadmenge größer, wird der Graph dagegen genauer abgebildet.

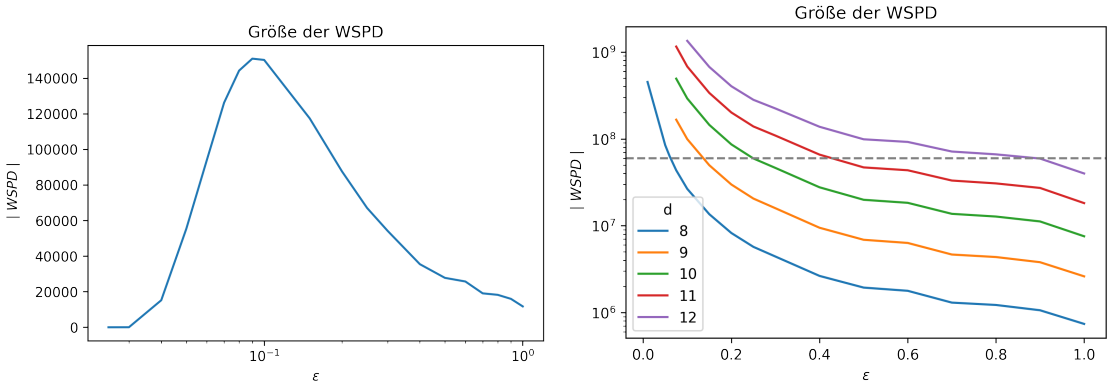


Abbildung 5.1: Größe der WSPD (und damit der Pfadmenge) für verschiedene Parameter d und ϵ. *Links*: $d = 5$ und *Rechts*: $d \in [8, 12]$. Die gestrichelte Linie zeigt die hier gesuchte WSPD-Größe von 60M.

Zu beachten ist, dass man die repräsentative Pfadmenge natürlich nicht beliebig groß erzeugen sollte: Eine größere Pfadmenge benötigt im nächsten Schritt bei der Berechnung des Hitting Sets mehr Rechenzeit und verbraucht generell auch mehr Speicherplatz. In der hier verwendeten Implementierung können auf der Testhardware nur etwa 60M Pfade verarbeitet werden; diesen Wert gilt es bei der Wahl der Parameter zu berücksichtigen.

Aus der Konstruktion erwarten wir bezüglich ϵ folgendes Verhalten, das auch durch die Ergebnisse in Abbildung 5.1 bestätigt wird: Für sehr große ϵ ist die WSPD eher klein, da viele Zellpaare die Separationsbedingung schon auf kleiner Tiefe erfüllen und die Konstruktion der WSPD somit nicht so tief im Quadtree suchen muss. Wählt man ϵ dagegen sehr klein, wird die Anzahl der Zellpaare in der WSPD auch wieder klein, da es kaum noch Zellpaare gibt, die weit genug auseinander liegen, um die Bedingung noch zu erfüllen. Die Zellpaare, die dann noch in der WSPD enthalten sind, sind tief im Quadtree und liegen weit auseinander. Im Extremfall enthält die WSPD nur noch Zellen, die nur einen Punkt enthalten, da diese per Definition einen Durchmesser von 0 haben und somit immer weit genug voneinander entfernt sind. Bei mittleren ϵ-Werten wird die WSPD dagegen sehr groß: Viele Zellpaare, die tief im Quadtree liegen, erfüllen die Separationsbedingung; die Zellpaare weiter oben jedoch nicht. Somit enthält die WSPD sehr viele Zellpaare. Auf unserem Testsystem wird die WSPD dabei für $d \in [8, 12]$ mit bestimmten ϵ-Werten so groß, dass der Speicher überläuft und keine Lösung gefunden wird. Daher findet sich in Abbildung 5.1 zusätzlich noch eine Messreihe für $d = 5$, in der wir den Gesamtverlauf sehen können.

Um eine bestimmte Größe der WSPD zu erreichen (etwa die in dieser Auswertung gesuchten ~60M), können für ein festes d zwei verschiedene ϵ gewählt werden: entweder ein recht großer oder ein recht kleiner Wert. Welche der beiden Optionen besser ist, müssen wir uns anhand anderer Eigenschaften überlegen.

5.1.2 Verteilung der Punktpaare

Neben der Größe der WSPD kann auch die Verteilung der Punktpaare auf die Zelltiefen innerhalb der WSPD betrachtet werden. Dazu zählen wir für jedes Zellpaar die Anzahl der Punktpaare, die im Zellpaar enthalten sind und kategorisieren die Ergebnisse nach der Tiefe der Zellpaare. Für ein

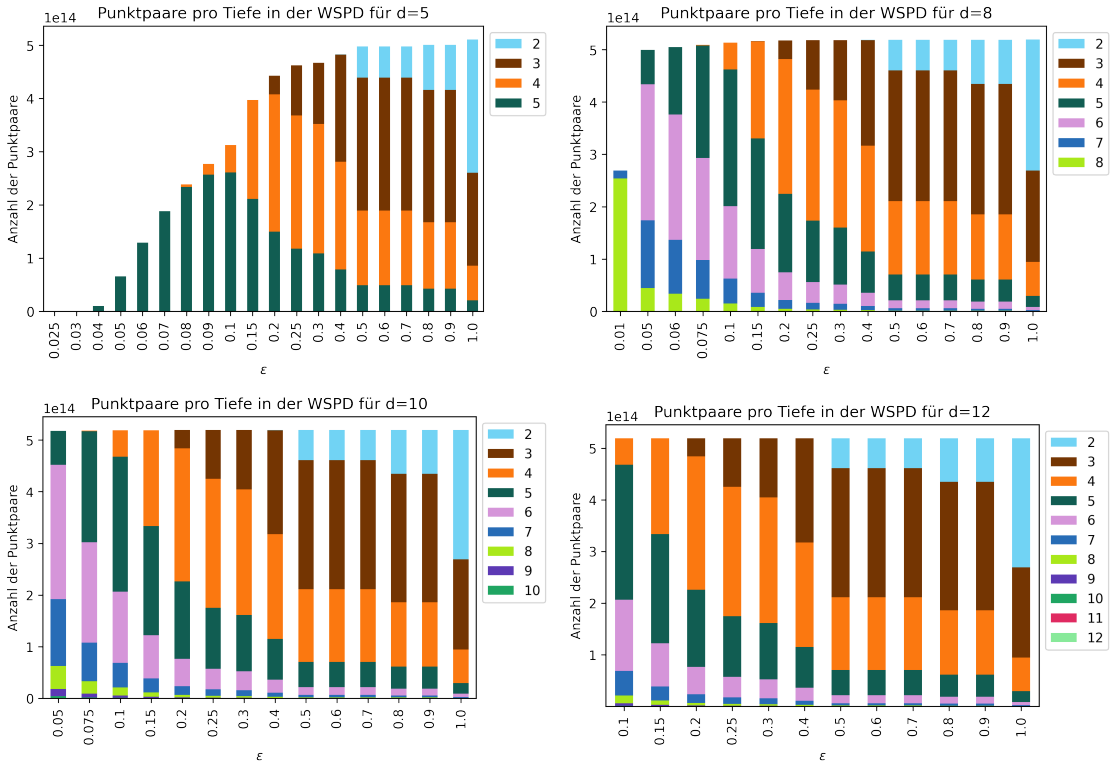


Abbildung 5.2: Verteilung der Punktpaare nach Zelltiefe für verschiedene ϵ.

festes d und verschiedene ϵ zeichnen wir dann ein Histogramm (Abbildung 5.2). Beispielsweise gibt es für das Parameterpaar $d = 8, \epsilon = 0.01$ etwa $2.5 * 10^{14}$ Punktpaare, die in Zellpaaren der Tiefe 8 liegen. Auch hier sehen wir die vollständige Entwicklung nur für $d = 5$; bei den anderen Beispielen fehlen Werte für verschiedene kleine ϵ, da unser Testsystem diese nicht berechnen kann.

Zuerst fällt auf, dass die Gesamtzahl der Punktpaare, die hier gezeigt werden, mit kleinerem ϵ sinkt. Auf diese Beobachtung wird im nächsten Abschnitt (Abschnitt 5.1.3) genauer eingegangen. Wir sehen, dass die Punktpaare mit kleinerem ϵ wie erwartet in tiefere Zellen verschoben werden (mit jedem Schritt zu kleineren ϵ werden einige Punktpaare von jeder Tiefe in die nächst tiefere verschoben). Der Anteil der kleinen Zellen steigt mit kleinerem ϵ und große Zellen kommen nur in WSPDs vor, die mit großen ϵ berechnet wurden. Am Beispiel $d = 5$ sehen wir auch, dass mit sehr kleinem ϵ auch die Anzahl der Punktpaare, die in den kleinsten Zellen liegen (hier also Tiefe 5), wieder kleiner wird, da selbst diese Zellpaare die Separationsbedingung nicht mehr erfüllen. Für große ϵ verhält sich die Verteilung der Punktpaare für alle d ähnlich, da die meisten Punktpaare in großen Zellen enthalten sind und die maximale Tiefe der verschiedenen Quadtrees hier keinen Einfluss auf die WSPD hat. Erst mit kleineren ϵ (etwa ≤ 0.2) ist ein Unterschied zu sehen: hier werden auch Zellen verwendet, die tief im Quadtree liegen und in den flacheren Bäumen nicht vorhanden sind.

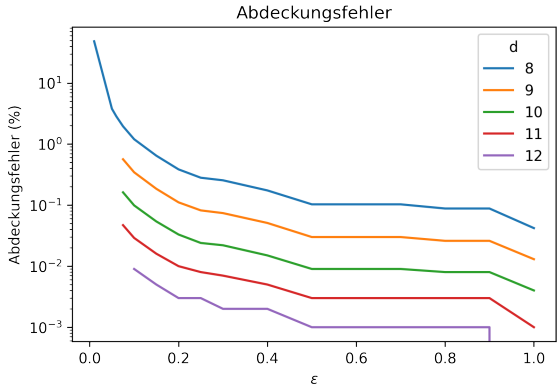


Abbildung 5.3: Abdeckungsfehler abhängig von d und ϵ.

5.1.3 Abdeckungsfehler

Eine weitere Eigenschaft der WSPD, die uns bei der Entscheidung über die Wahl der Parameter hilft, ist der Abdeckungsfehler der WSPD. Wir wissen bereits, dass die WSPD nicht alle Punktpaare enthält, da der Quadtree begrenzt ist (da alle Punktpaare innerhalb eines Blatts nie in der WSPD auftauchen können). Abgesehen von diesen Punkten gibt es je nach Wahl von ϵ jedoch noch mehr Punktpaare, die nicht abgedeckt sind: wenn ϵ klein ist, sind (wie im vorherigen Abschnitt bereits beobachtet) immer mehr Punktpaare nicht mehr in der WSPD enthalten. Wir nennen den Anteil der Punktpaare, die in einer WSPD *nicht* enthalten sind, den *Abdeckungsfehler* und sehen diesen in Abbildung 5.3. Der Abdeckungsfehler steigt mit kleinerem ϵ und kann einen Wert, der von d abhängt, nicht unterschreiten (dies sind gerade die Punktpaare, die sich in einem gemeinsamen Blatt befinden). Mit steigendem Abdeckungsfehler wird der Graph immer schlechter zusammengefasst; Ziel bei der Wahl der Parameter ist also, den Abdeckungsfehler klein zu halten.

Kombinieren wir diese Beobachtungen nun mit der vorherigen zur Größe der WSPD. Es gibt zwei ϵ-Werte, die gleich große repräsentative Pfadmengen erzeugen. Die Pfadmenge, die sich aus dem größeren der beiden Werte ergibt, hat einen kleineren Abdeckungsfehler. Demnach würde sich also die Wahl des größeren ϵ anbieten. Allerdings sollte durch ein kleineres ϵ die WSPD *ähnlichere* Pfade enthalten - je nach Verwendungszweck könnte also auch das kleinere ϵ die besseren Ergebnisse liefern, solange der Abdeckungsfehler nicht zu groß wird. Wir werden uns für die Auswertung des Hitting Sets jedoch auf das größere ϵ beschränken.

5.1.4 Geometrische Abweichung

Die dritte Eigenschaft der repräsentativen Pfadmenge, die wir uns ansehen können, ist die *geometrische Abweichung*.

Aus der geometrischen Zerlegung der Punkte im Quadtree entsteht, wie schon in Kapitel 3 angesprochen, ein Unterschied zwischen der Struktur des Graphen und dem Quadtree (und somit der WSPD). Wir nennen diesen Unterschied die *geometrischen Abweichung*. Die Annahme, dass für alle Pfade im Graph gilt, dass sich alle durch lokales verschieben der Start- und Zielknoten erzeugten Pfade ähnlich sind (Definition 3.1.1), gilt in einem Straßennetz offensichtlich nicht. Ein einfaches Gegenbeispiel ist eine Autobahnbrücke. Hier könnten Knoten sogar die gleichen

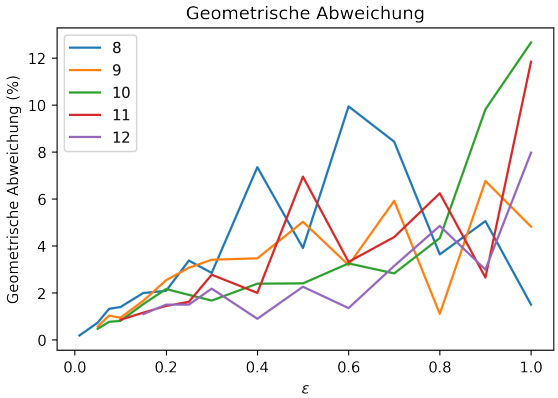


Abbildung 5.4: Geometrische Abweichung der WSPD abhängig von d und ϵ.

Koordinaten haben, aber es gibt keine direkte Verbindung zwischen ihnen. In Abbildung 5.5 ist zusätzlich noch ein anderer Fall der geometrischen Abweichung zu sehen: zwei annähernd parallel verlaufende Autobahnen können dafür sorgen, dass der kürzeste Weg entlang ihrer Strecke je nach genauer Position der Start- und Zielknoten auf der einen oder anderen Autobahn verläuft. In beiden Fällen gibt es also Knoten, bei denen kleine lokale Verschiebungen der Start- oder Zielknoten die kürzesten Wege stark beeinflussen.

Da wir alle Punktpaare in einem Zellpaar (gewichtet) zusammenfassen, übertragen wir den Begriff der geometrischen Abweichung auf das Zellpaar, das ein solches geometrisch falsches Punktpaar enthält. Insgesamt ist dann der Anteil aller Punktpaare, die von der WSPD abgedeckt werden und in einem Zellpaar mit geometrischer Abweichung liegen, die geometrische Abweichung der WSPD.

Die exakte Berechnung der geometrischen Abweichung ist in unserer Implementierung praktisch unmöglich, da man hier $O(|V|^2)$ kürzeste Wege berechnen müsste. Für eine praktisch durchführbare Berechnung müssen wir uns also mit einer Abschätzung zufriedengeben. Eine mögliche Abschätzung kann etwa so aussehen: Wir wählen für jedes Zellpaar zufällig je drei Punkte beider Zellen, berechnen daraus neun kürzeste Wege und überprüfen, ob sie mindestens einen gemeinsamen Punkt enthalten[1]. Eine vollständige Überprüfung aller Punktpaare wäre natürlich genauer, ist allerdings praktisch nicht umsetzbar. Da auch diese reduzierte Operation auf allen Zellpaaren noch zu aufwändig ist, wird nur 1 % der Zellpaare getestet.

Die Ergebnisse dieser Auswertung sind in Abbildung 5.4 zu sehen. Leider liefert uns die Auswertung keine eindeutigen Ergebnisse. Generell wäre zu erwarten, dass die geometrische Abweichung mit kleinerem ϵ sinken sollte: die Abstände der Zellen zueinander werden größer, und somit steigt die Länge der Pfade und die Wahrscheinlichkeit, dass sich die Pfade schneiden (da überregionale Straßen verwendet werden, von denen es weniger gibt). Zugleich werden die Abstände der Punkte in den Zellen kleiner und auch dadurch sollte die Wahrscheinlichkeit, dass sich die Pfade schneiden, steigen. Wir sehen dieses Verhalten in Abbildung 5.4 jedoch nicht eindeutig.

[1] Nach der Definition für ähnliche Pfade überprüfen wir also, ob die Punkte überhaupt (also für ein beliebiges δ) ähnlich sind

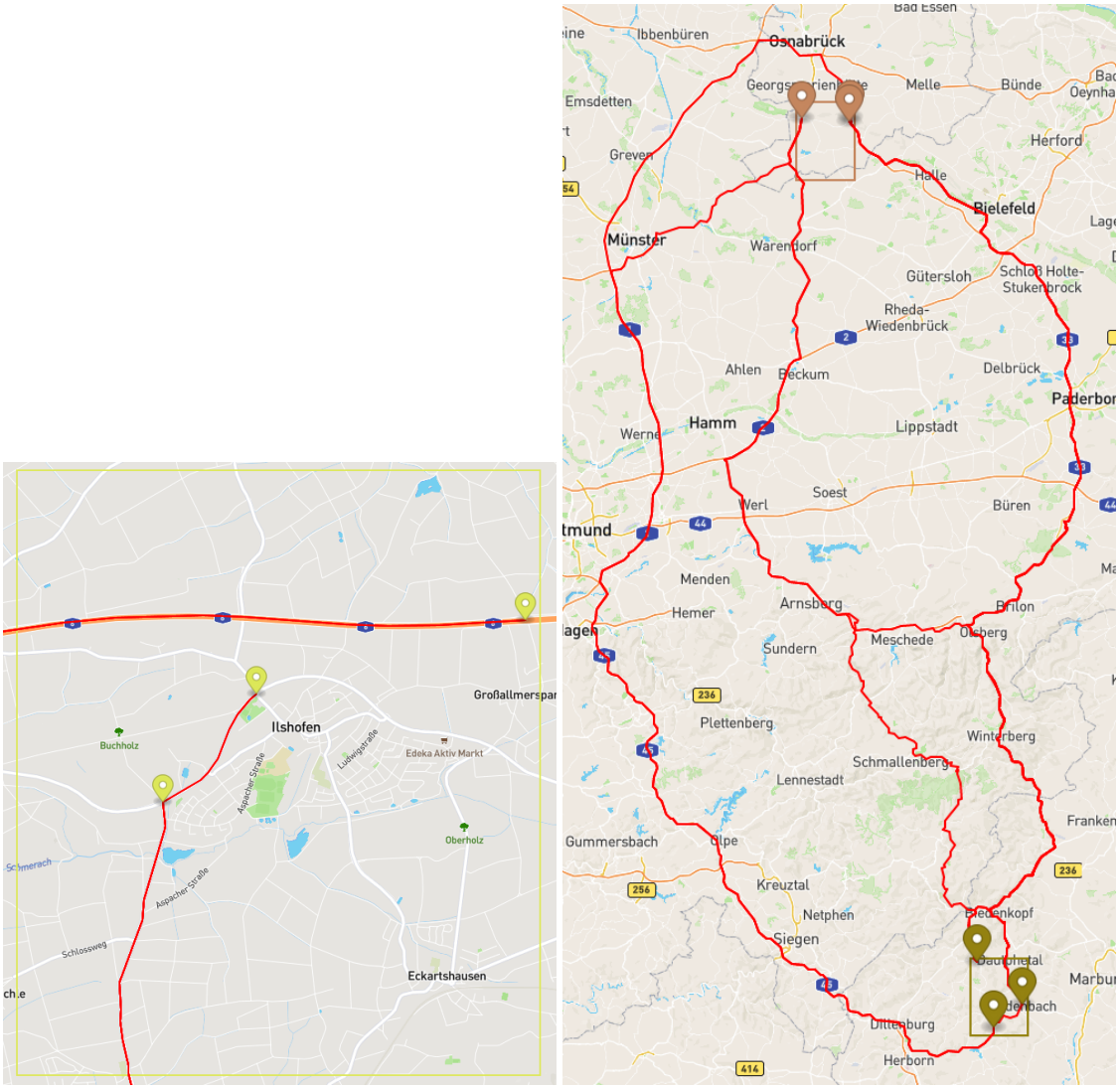


Abbildung 5.5: Zwei Beispiele für geometrische Fehler. *Links*: Die Knoten auf der Autobahn und abseits der Autobahn liegen zwar geometrisch nah zusammen, im Graph allerdings nicht, da es in dieser Zelle keine Autobahnausfahrt gibt. *Rechts*: Es gibt mehrere Routen über verschiedene Autobahnen, die ähnlich lang sind und je nach genauer Position der Start- und Zielknoten kürzer sind.

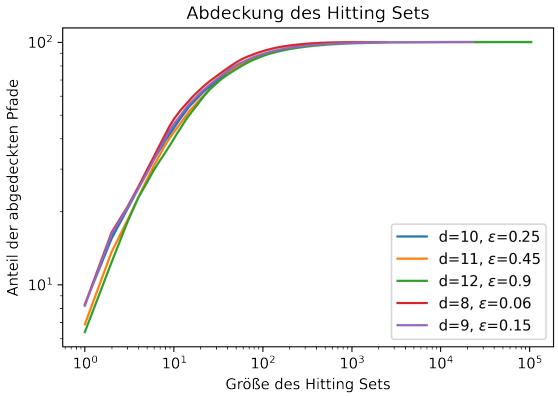


Abbildung 5.6: Anteil der Pfade, die im Verlauf der Hitting Set Berechnung abgedeckt werden.

Weiterhin wäre zu erwarten, dass die geometrische Abweichung für große ϵ, bei denen die Zellen der WSPD meist groß sind, recht unabhängig von d ist. Der Quadtree der verschiedenen d unterscheidet sich erst ab der Tiefe, an der einer der Quadtrees durch d begrenzt und im anderen die Zellen noch weiter geteilt werden. Besteht die WSPD hauptsächlich aus großen Zellen (in unserem Beispiel $k \leq 8$), dann sind die Zellen der WSPD gleich, unabhängig von d. Dies wäre hier etwa für $\epsilon \geq 0.6$ der Fall. Eindeutig kann man jedoch auch diese These mit dieser Auswertung nicht belegen.

5.2 Hitting Set

Aus den repräsentativen Pfadmengen werden mit dem Greedy Hitting Set Algorithmus wichtige Punkte gefunden. Für die Auswertung des Hitting Set Algorithmus wurden fünf Parameterpaare gewählt, die jeweils etwa 60M Pfade erzeugen und damit noch auf der Testhardware verarbeitet werden können. Diese sind mit den Ergebnissen in Tabelle 5.1 aufgelistet und werden nun noch genauer beschrieben.

Untere Schranke In der Tabelle ist neben der Größe des Hitting Sets auch noch eine Abschätzung der unteren Schranke für die Größe angegeben. Da es keine nützliche allgemeine untere Schranke für das Hitting Set Problem gibt, wird hier eine Abschätzung für die spezifische Probleminstanz angegeben. Diese Abschätzung wird, wie in Abschnitt 4.5 beschrieben, durch iteratives entfernen eines Pfades mit allen Pfaden, die ihn schneiden, berechnet. Wir sehen, dass die Lösung des Greedy-Algorithmus besonders für große Lösungen recht nah an der unteren Schranke liegt.

5.2.1 Pfadabdeckung durch unvollständige Hitting Sets

Im Verlauf der Berechnung steigt mit jeder Iteration der gewichtete Anteil der Pfade, die vom aktuellen (unvollständigen) Hitting Set abgedeckt werden. In Abbildung 5.6 ist der Verlauf zu sehen: Anfangs werden sehr schnell die meisten Pfade abgedeckt; anschließend werden sehr viele weitere Knoten gefunden, die jeweils nur eine kleine Anzahl neuer Pfade abdeckt. In Tabelle 5.3 sehen wir genauer, dass nur einige hundert bis zehntausend Knoten benötigt werden, um bereits 90 % bis

Parameter			Pfadabdeckung					
d	ϵ	| HS |	90%	95%	99%	99.9%	99.99%	99.999%
8	0.06	2505	84	145	379	837	1315	1735
9	0.15	22978	106	211	829	3444	8062	13114
10	0.25	104322	111	228	962	5595	18926	40643
11	0.45	382127	121	244	1051	6719	29394	87323
12	0.9	1190699	125	259	1136	7572	37624	104322

Tabelle 5.3: Für eine Auswahl von Parameterpaaren: Größe des vollständigen Hitting Sets (HS) und Größe eines Hitting Sets, das 90 %, 95 %, 99 %, 99.9 %, 99.99 % bzw. 99.999 % aller gewichteten Pfade in P abdeckt (Pfadabdeckung).

99.999 % der Pfade im Graphen abzudecken. Abbildung 5.7 zeigt beispielhaft die Knotenmenge, die für $d = 8$ und $\epsilon = 0.06$ bereits 99 % der Pfade abdeckt. Wie erwartet liegen die meisten Knoten hier auf Autobahnen oder anderen überregionalen Straßen.

Die Größen der vollständigen Hitting Sets unterscheiden sich zwar je nach Parameterwahl deutlich voneinander, aber trotzdem sind die meisten Pfade bei allen Parametern schon früh, und ungefähr gleichzeitig, abgedeckt. Der Unterschied liegt hauptsächlich darin, wie viele disjunkte Pfade (bzw. Pfade mit sehr wenigen Überschneidungen) gegen Ende der Berechnung übrig sind, die dann jeweils einen eigenen Knoten im Hitting Set erhalten. Das Verhalten lässt sich mit der Wahl von ϵ erklären. Wie bereits beschrieben, werden die Zellen, die die Separationsbedingung erfüllen, mit kleinerem ϵ kleiner. Der Abstand zwischen den Zellen und damit auch die Länge der Pfade wächst daher mit kleinerem ϵ. Längere Pfade haben eine größere Wahrscheinlichkeit, andere lange Pfade zu schneiden (besonders in Straßennetzen, da hier zusätzlich Fernstraßen hinzukommen) und somit wird das Hitting Set kleiner.

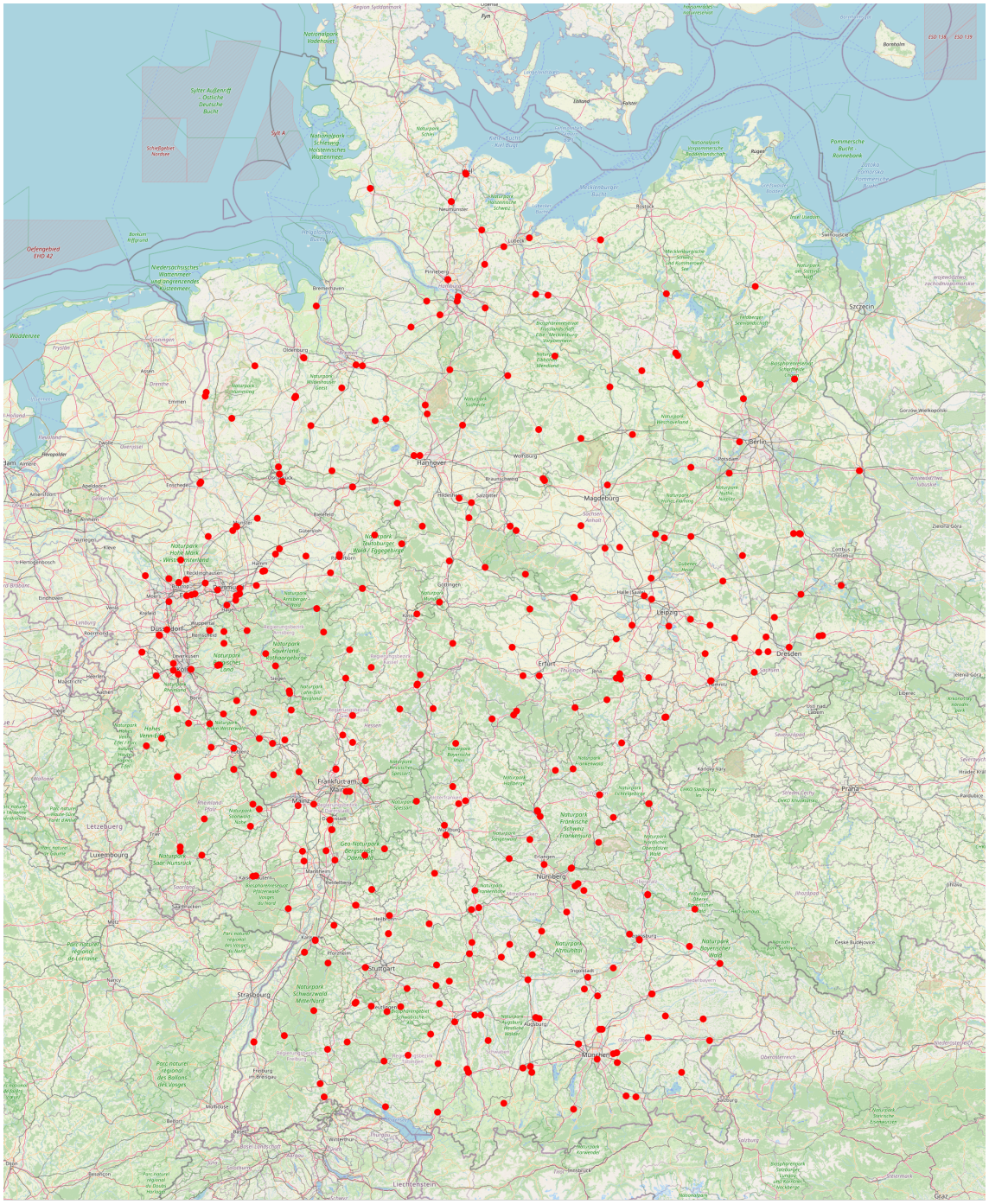


Abbildung 5.7: Knotenmenge, die 99 % aller Pfade in der repräsentativen Pfadmenge mit $d = 8, \epsilon = 0.06$ abdeckt. Oft liegen zwei Punkte sehr nah zusammen, da Autobahnen in OpenStreetMap mit zwei Kanten (eine pro Richtung) beschrieben werden und auf beiden Richtungen ein wichtiger Punkt liegt.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Methode zur Berechnung *wichtiger* Knoten in Graphen vorgestellt. Diese liefert am Beispiel des deutschen Straßennetzes gute Ergebnisse. Der Algorithmus zur Berechnung des Hitting Sets kann Probleme im Bereich von 10^7 Pfaden schnell berechnen und ist hier nur durch den Speicherbedarf limitiert. Die Berechnung der repräsentativen Pfadmenge mit einer WSPD scheint vielversprechend; in der recht einfachen Version mit einem Quadtree erhalten wir bereits Pfadmengen, die unsere Vorgaben weitgehend erfüllen. Diese Berechnung lässt sich gut parallelisieren und kann somit auch auf deutlich größeren Problemen noch Ergebnisse liefern. Für das deutsche Straßennetz erhalten wir eine sehr kleine Menge von einigen tausend Knoten, die bereits 99,9 % aller kürzesten Pfade im Graph abdecken.

Ausblick

Aus der geometrischen Zerlegung der Punktmenge mit dem Quadtree ergeben sich noch offene Fragen: Die Zerlegung geht implizit davon aus, dass Knoten, die geometrisch nah zusammen liegen, immer auch ähnliche kürzeste Pfade zu anderen Knoten haben. Einige Beispiele zeigen, dass diese Annahme nicht immer gilt. Es stellt sich also die Frage, wie häufig diese geometrischen Abweichungen auftauchen und wie stark sie sich auf die Ergebnisse auswirken. Erste Ergebnisse in dieser Arbeit scheinen darauf hinzudeuten, dass das Problem keinen allzu großen Einfluss hat, aber eindeutige Ergebnisse sind noch offen.

Außerdem stellt sich natürlich die Frage nach der Qualität der Lösung in der weiteren Verwendung. Offensichtlich ist hier die Verwendung in Beschleunigungstechniken, die eine Menge wichtiger Knoten verwenden. Besonders beim Transit Node Routing würde sich eine Verwendung anbieten. Versuchsreihen zum Vergleich der hier vorgestellten Methode mit anderen Heuristiken sind ein naheliegendes nächstes Projekt.

Möchte man Varianten der hier vorgestellten Methode betrachten, dann stellt sich die Frage, ob sich durch die Verwendung einer anderen Baumstruktur statt des Quadtrees zur Berechnung der WSPD bessere repräsentative Pfadmengen ergeben. Gibt es eine Baumstruktur, die die Struktur des Graphen besser beschreibt oder keinen Umweg über die geometrische Zerlegung machen muss? Auch wäre ein Baum, der nicht auf der Einbettung des Graphen in $\mathbb{R}^2$ basiert, für die Anwendung auf andere Graphtypen nützlich.

Literaturverzeichnis

[Bah15] D. Bahrdt. *OSMGraphCreator*. https://github.com/fmi-alg/OsmGraphCreator. 2015 (zitiert auf S. 31).

[Bar21] F. Barth. *Dijkstra Performance Study*. https://github.com/Lesstat/dijkstra-performance-study. 2021 (zitiert auf S. 10).

[BDG+16] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, R. F. Werneck. „Route planning in transportation networks“. In: *Algorithm engineering*. Springer, 2016, S. 19–80 (zitiert auf S. 7).

[BFM+07] H. Bast, S. Funke, D. Matijevic, P. Sanders, D. Schultes. „In Transit to Constant Time Shortest-Path Queries in Road Networks“. In: *2007 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. 2007, S. 46–59. DOI: 10.1137/1.9781611972870.5 (zitiert auf S. 7).

[CK95] P. B. Callahan, S. R. Kosaraju. „A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields“. In: *Journal of the ACM (JACM)* 42.1 (1995), S. 67–90 (zitiert auf S. 7, 15).

[Dij+59] E. W. Dijkstra et al. „A note on two problems in connexion with graphs“. In: *Numerische mathematik* 1.1 (1959), S. 269–271 (zitiert auf S. 7, 10).

[GSSD08] R. Geisberger, P. Sanders, D. Schultes, D. Delling. „Contraction hierarchies: Faster and simpler hierarchical routing in road networks“. In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2008, S. 319–333 (zitiert auf S. 7, 11, 12).

[Har11] S. Har-Peled. *Geometric approximation algorithms*. 173. American Mathematical Soc., 2011 (zitiert auf S. 15–17).

[NB17] A. Nusser, S. Bühler. *chconstructor*. https://theogit.fmi.uni-stuttgart.de/nusserae/chconstructor. 2017 (zitiert auf S. 31).

[Ope17] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017 (zitiert auf S. 31).

[Sch11] U. Schöning. *Algorithmik*. Spektrum Akademischer Verlag, 2011 (zitiert auf S. 10).

Alle URLs wurden zuletzt am 18. 05. 2022 geprüft.

Erklärung

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

Ort, Datum, Unterschrift