Hochschule für angewandte Wissenschaften München
Fakultät für Geoinformation

Bachelorarbeit

Placeholder

Verfasser: Daniel Holzner

Matrikelnummer: 26576714

Studiengang: Geoinformatik und Navigation

Betreuer: Prof. Dr. Thomas Abmayr

Abgabedatum: 18. Juli 2023

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

CHs Contraction Hierarchies

CH Contraction Hierarchie

OSM OpenStreetMap

PQ Vorrangwarteschlange (eng. Priority Queue)

1 Einleitung

1.1 Motivation und Kontext

Mit der fortschreitenden Entwicklung von Verkehrsmitteln und der dadurch zunehmenden Mobilität gewinnt die Routenplanung eine immer größere Bedeutung. Routenplanung ist ein faszinierendes und herausforderndes Gebiet, das eine wichtige Rolle in verschiedenen Bereichen spielt. Egal, ob es darum geht, den schnellsten Weg von einem Ort zum anderen zu finden, die effizienteste Route für die Zustellung von Waren zu bestimmen oder den Verkehr in einem komplexen Straßensystem zu simulieren, durch die Anwendung und Erforschung von Routenplanungsalgorithmen können effiziente und optimale Wege in komplexen Netzwerken gefunden werden, um Zeit, Ressourcen und Kosten zu sparen.

Das Problem, nach der Suche des kürzesten Weges, lässt sich auf eines der fundamentalsten Probleme aus der Graphentheorie, einem Teilgebiet der Mathematik und theoretischen Informatik zurückführen. Mit Graphen lassen sich eine Vielzahl von Problemen aus der echten Welt als mathematische Struktur, bestehend aus Knoten und Kanten, modellieren. So kann auch ein Straßennetzwerk durch Knoten, die Kreuzungen repräsentieren und Kanten, die als Straßensegmente Kreuzungen miteinander verbinden, dargestellt werden. Jeder Kante wird dabei ein Gewicht zugewiesen, das die mit dem Durchlaufen dieser Kante verbundenen Kosten widerspiegelt. Auf dem Graphen lassen sich anschließend Algorithmen ausführen, die den kürzesten Weg dist(s,t) zwischen einem Startknoten s und Zielknoten t bestimmen können, indem die Kosten des Weges minimiert werden. Einer der wohl bekanntesten Algorithmen, um diese Aufgabe zu lösen wurde von dem niederländischen Informatiker Edsger W. Dijkstra entwickelt und im Jahr 1959 veröffentlicht [1]. Der Dijkstra-Algorithmus funktioniert zwar gut auf kleinen Graphen und wird auch noch heutzutage häufig angewendet, skaliert jedoch schlecht mit immer größer werdenden Datenmengen, denn im schlechtesten Fall muss der gesamte Graph traversiert werden. Da ein Straßennetz aus mehreren Millionen Knoten und Kanten besteht, ist der Dijkstra-Algorithmus daher in Anwendungen, die in kurzer Zeit viele kürzeste Wege berechnen müssen, wie z. B. Navigationssysteme, die Routen in Echtzeit aktualisieren müssen, nicht mehr geeignet.
Um dieses Problem zu lösen, wurden im Laufe der Zeit neue Speed-Up Techniken entwickelt, die die Laufzeit der Suche verbessern. So wurde u. a. der A*-Algorithmus („A-Stern“) im Jahr 1968 von Peter Hart, Nils J. Nilsson und Bertram Raphael als eine Erweiterung des Dijkstra-Algorithmus veröffentlicht [2]. Der Algorithmus ist in der Lage durch das Einführen einer zusätzlichen Heuristik, orientierter Richtung Ziel zu suchen und damit den Suchraum deutlich

einzuschränken. Dadurch wurde die Laufzeit nochmals verbessert, war aber immer noch nicht ausreichend für sehr große Graphen.
Viele weitere Techniken basieren auf einer starken Vorverarbeitung des Graphen. So wurde 2008 von Geisberger, Sanders, Schultes, und Delling vorgestellt [3], die als Contraction Hierarchies (CHs) bezeichnet wird. Sie basiert auf einer Vorverarbeitung des Graphen, bei der ausgenutzt wird, dass Straßennetzwerke bereits eine natürliche Hierarchie besitzen. Während der Vorverarbeitung werden dem Graph zusätzliche Informationen hinzugefügt, die dann zur Laufzeit während der Suche ausgenutzt werden, was zu erheblich schnelleren Berechnung der Route führt. Da diese Technik als Sprungbrett für viele neue erweiterte Routenplanungstechniken gilt, soll sie im Rahmen dieser Arbeit genauer untersucht werden. Dazu soll ein Prototyp erstellt werden, der die Funktionsweise von CHs durch eine konkrete Implementierung demonstriert. Als Eingangsdaten werden die frei nutzbaren Geodaten des OpenStreetMap-Projekts (Open Data) verwendet, um die Ergebnisse an realen Daten zu analysieren und zu testen.

1.2 Stand der Forschung

Die Berechnung von kürzesten Wegen in dynamischen gewichteten Graphen ist in den letzten Jahrzehnten intensiv untersucht worden und es entstanden viele neue Techniken zur Lösung verschiedener Varianten des Problems. Das kürzeste-Wege-Problem ist so relevant, dass regelmäßig Wettbewerbe stattfinden, bei denen die aktuell besten Routenplanungsalgorithmen auf speziellen Eingabedaten ermittelt werden. So wurde z. B. 2006 die neunte DIMACS Implementation Challenge [4] ausgerichtet, in der die „State of the Art“-Techniken vorgestellt wurden. Eine ausführliche Übersicht über verschiedene Algorithmen zur Routenplanung in Straßennetzwerken wurde von Delling et al. [5] veröffentlicht, ist aber durch die signifikante Weiterentwicklungen der letzten Jahre nicht mehr topaktuell. Es sind neue Algorithmen entstanden, die Suchanfragen auf Straßennetzwerken in der Größenordnung von Kontinenten in wenigen hundert Nanosekunden beantworten können oder aktuelle Verkehrsinformationen mit in die Suche einfließen lassen [6]. Durch den aktuellen Stand der Technik können Methoden des maschinellen Lernens verwendet werden, um je nach Situation den Verkehrsfluss in Echtzeit vorherzusagen und mit in der Routenplanung zu berücksichtigen [7].
Diese Entwicklungen zeigen, dass die Routenplanung in Straßennetzwerken ein aktiver Forschungsbereich ist, der sich ständig weiterentwickelt, um den Bedürfnissen der Nutzer und modernen Anwendungen gerecht zu werden.

1.3 Beitrag der Arbeit

Im Rahmen dieser Arbeit wird die Implementierung der CHs-Technik in der Programmiersprache *Rust* untersucht. Rust ist eine moderne, systemsprachenorientierte Programmiersprache, die auf Performance, Sicherheit und Nebenläufigkeit abzielt [8]. Die Wahl von Rust als Implementierungssprache bietet die Möglichkeit, die Vorteile dieser Sprache in Bezug auf Geschwindigkeit, Speichersicherheit und Thread-Sicherheit in der Routenplanung zu erforschen.
Das Hauptziel dieser Arbeit ist, die Implementierung von CHs detailliert zu beschreiben und anschließend eine umfassende Leistungsanalyse durchzuführen, indem die Ergebnisse mit herkömmlichen Algorithmen wie dem Dijkstra-Algorithmus und dem A*-Algorithmus verglichen werden. Die Evaluierung erfolgt anhand verschiedener Metriken wie Laufzeit, Vorverarbeitungszeit und Speicherbedarf. Durch diesen Vergleich wird ein tieferes Verständnis für die Leistungsfähigkeit von CHs gewonnen und die verbundenen Vor- und Nachteile im Vergleich zu herkömmlichen Algorithmen ermittelt.
Die Ergebnisse dieser Arbeit können wichtige Erkenntnisse liefern, die zur Weiterentwicklung und Optimierung von Routenplanungssystemen beitragen. Darüber hinaus bietet die Implementierung in Rust einen wertvollen Beitrag zur wachsenden Gemeinschaft von Rust-Entwicklern und demonstriert die Anwendung der Sprache in einem relevanten Anwendungsfall.

1.4 Struktur der Arbeit

Die folgenden Abschnitte der Arbeit werden die theoretischen Grundlagen von Routenplanungsalgorithmen, insbesondere des Dijkstra-Algorithmus, des A*-Algorithmus und der CH-Technik, erläutern. Anschließend wird die Implementierung der CH-Technik in Rust beschrieben und detailliert analysiert. Abschließend werden die Ergebnisse der Leistungsanalyse präsentiert und diskutiert, gefolgt von einem Fazit, das die Erkenntnisse dieser Arbeit zusammenfasst und mögliche Ansätze für zukünftige Forschungen aufzeigt.

2 Theoretische und technische Grundlagen

2.1 Datenstrukturen

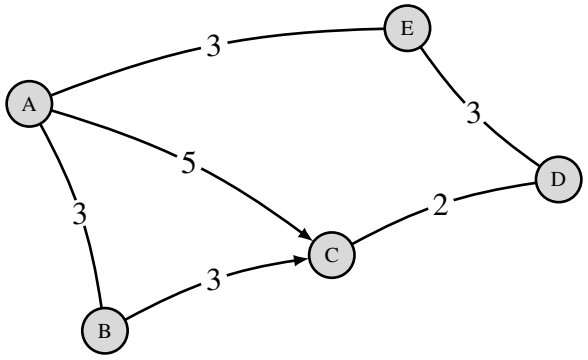
2.1.1 Graph

Um das kürzeste-Wege-Problem zu lösen, muss zunächst das reale Straßennetz in eine abstrakte Form gebracht werden. Hierzu wird das Straßennetz als Graph modelliert. Ein Graph $G = (V, E)$ besteht aus einer Menge von Knoten V und einer Menge von Kanten E. Jede Kante $e = (u, v) \in E$ verbindet zwei Knoten $u, v \in V$. Knoten bilden Kreuzungen ab und Kanten Straßensegmente zwischen zwei Kreuzungen. Ein Graph kann als gerichtet oder ungerichtet definiert werden. Bei einem ungerichteten Graphen sind die Kanten bidirektional und können in beide Richtungen durchlaufen werden, während bei einem gerichteten Graphen die Kanten nur in eine Richtung durchlaufen werden können.

Der Graph in dieser Arbeit ist gerichtet und gewichtet. Die Kantenrichtung spiegelt dabei die Richtung des Verkehrs wieder, d. h. sie gibt an, ob eine Straße in einer bestimmten Richtung befahren werden darf oder nicht. Zusätzlich wird auf jede Kante eine Kostenfunktion angewandt, die dann ein Gewicht $w(e)$ festgelegt. In der Routenplanung ist normalerweise nicht die kürzeste Strecke von Interesse, sondern die Strecke mit der kürzesten Reisezeit. Diese lässt sich aus der Länge des Straßensegments S und der maximal erlaubten Geschwindigkeit V auf dieser Straße $t = \frac{V}{S}$ in Sekunden berechnen.

Es gibt generell zwei Möglichkeiten, einen Graphen als Datenstruktur darzustellen. Eine davon ist die Verwendung von Adjazenzlisten. Dabei wird für jeden Knoten $u \in V$ eine Liste der benachbarten Knoten bzw. ausgehenden Kanten $e(u, v) \in E$ gespeichert. Die Summe der Länge aller Adjazenzlisten in einem gerichteten Graph entspricht der Anzahl an Kanten $|E|$, bzw. $2|E|$ in einem ungerichteten Graph. Der Speicherverbrauch für die Adjazenzliste ist damit $\Theta(|V| + |E|)$ [9].
Die zweite Möglichkeit ist die Verwendung einer Adjazenzmatrix. Hier wird eine zweidimensionale Matrix der Größe $|V| \times |V|$ verwendet, bei der die Zeilen und Spalten den Knoten entsprechen. Der Eintrag an Position (i, j) in der Matrix gibt an, ob eine Kante zwischen den Knoten i und j existiert. Unabhängig von der Anzahl an Kanten ist der Speicherverbrauch $\Theta(|V|^2)$ [9]. Ein Beispiel für beide Darstellungen anhand eines einfachen Graph befindet sich in Abbildung 1.

Für diese Arbeit werden Adjazenzlisten verwendet, da zum einen Straßennetze dünne Graphen sind ($|E| << |V|^2$), und damit die Speichereffizienz gegenüber einer Adjazenzmatrix deutlich effizienter ist. Zum anderen ist für es wichtig, für den Aufbau der CHs und der Suche, schnell auf alle Nachbarn eines Knotens zuzugreifen zu können. Der Zugriff auf eingehende Kanten ist allerding nur sehr aufwendig möglich, wird aber für den Aufbau der Kontraktionshierarchien benötigt. Das Problem lässt sich jedoch lösen indem auch Adjazenzlisten für alle eingehenden Kanten eines Knotens angelegt werden. Damit erhöht sich der Speicherverbrauch auf $\Theta(|V|+2|E|)$, was aber immer noch deutlich effizienter ist als eine Darstellung als Adjazenzmatrix.



(a)

	A	B	C	D	E
A	0	3	5	0	3
B	3	0	3	0	0
C	0	0	0	2	0
D	0	0	2	0	3
E	3	0	0	3	0

(b)

A → [B | C | E]
B → [A | C]
C → [D]
D → [C | E]
E → [A | D]

(c)

Abbildung 1: Der oben gezeigte gerichtete Graph (a) dargestellt als Adjazenzmatrix(b) oder Adjazenzliste (c). Viele Einträge der Matrix sind 0, da der Graph dünn ist, d. h. $|E|$ um einiges kleiner ist als $|V^2|$.

2.1.2 Vorrangwarteschlangen

Eine Vorrangwarteschlange (eng. Priority Queue) (PQ) ist eine Datenstruktur, in der nur auf das Element mit der höchsten Priorität zugegriffen werden kann. Eine PQ unterstützt in der Regel die folgenden Operationen:

1. Einfügen (Push): Ein Element wird mit seiner zugehörigen Priorität in die Warteschlan-

ge eingefügt. Das Element wird entsprechend seiner Priorität platziert.

2. Entfernen (Pop): Das Element mit der aktuell höchsten Priorität wird aus der Warteschlange entfernt und zurückgegeben.

PQs werden allen nachfolgenden Suchalgorithmen verwendet, daher hat die Implementierung der Warteschlange einen großen Einfluss auf die Laufzeit der Algorithmen. In der Arbeit wird die Implementierung als binärern Min-Heap verwendet mit einer Zeitkompexität für das Einfügen von $\theta(1) \sim$ und für das Entfernen von $\theta(\log n)$.

2.2 Kürzeste-Wege-Algorithmen

Grundsätzlich arbeiten kürzeste-Wege-Algorithmen daran, den kürzesten Weg zwischen einem Startknoten s und einem Zielknoten t in einem gewichteten Graph zu finden. Der kürzeste Weg P bezieht sich dabei auf den Weg mit dem geringsten Gesamtgewicht $dist(s,t)$, der sich aus der Summe der Kosten zum Überqueren der einzelnen Kanten ergibt. Neben dem Punkt-zu-Punkt-kürzeste-Wege-Problem existieren noch weitere Varianten, wie das Eins-zu-Viele-Problem, bei dem der kürzeste Weg von einem Knoten s zu allen anderen Knoten im Graphen gesucht wird, oder das Viele-zu-Viele-Problem, bei dem jeder kürzeste Weg zwischen einer Knotenmenge S und einer Knotenmenge T gesucht wird [6]. In dieser Arbeit wird sich hauptsächlich auf das Punkt-zu-Punk-Problem fokusiert.

2.2.1 Grundlegende Technik

Der Algorithmus von Dijkstra löst das Eins-zu-Viele-Problem auf einem gewichteten gerichten Graphen. Dabei ist zu beachten dass alle Kanten $e \in E$ nicht-negativ gewichtet sind, also $e(u,v) \geq 0$.
Der Algorithmus besteht aus einer Initialisierungsphase, in der die Kosten *dist* für alle Knoten auf den Wert ∞ gesetzt werden. Bereits besuchte Knoten werden in der Menge S gespeichert, um zu verhindern, dass Knoten doppelte besucht werden. Um am Ende nicht nur die Kosten, sondern auch den Weg zu erhalten wird zusätzlich der Vorgänger P für jeden Knoten gespeichert. In folgender Implementierung wird eine Min-Vorrangwarteschlange Q verwendet, in der zu Beginn der Startknoten s mit Kosten 0 eingefügt wird. In der Hauptschleife wird nun solange ein Knoten $u \in V - S$ aus der Warteschlange entnommen, bis diese leer ist. Wenn ein Nachbarknoten v von u noch nicht besucht wurde, werden die Kosten $dist(v)$ aktualisiert, falls der Weg über u kürzer ist. Der Vorgänger $P(v)$ wird ebenfalls

aktualisiert und der Knoten v in die Warteschlange eingefügt. Der Algorithmus terminiert, wenn alle Knoten besucht wurden.

Algorithm 1 DIJKSTRA(G,s)

```
Kosten dist für alle Knoten außer s mit ∞ bewerten
dist(s) = 0
Besuchte Knoten S = ∅
Vorgänger P = ∅
Q.push(s)
while Q ≠ ∅ do
    u = Q.pop()
    M = M ∪ {u}
    for alle ausgehenden Kanten e(u,v) ∈ Adj[u] do
        if v ∉ S und dist(u) + w(u,v) < dist(v) then
            dist(v) = dist(u) + w(u,v)
            P(v) = u
            Q.push(v)
        end if
    end for
end while
```

Die Laufzeit des Algorithmus hängt hauptsächlich von der Implementierung der Prioritätswarteschlange ab. Mit der verwendeten Implementierung ergibt sich eine Laufzeit von $\Theta(|E|+|V|\log|V|)$, wenn Adjazenzlisten verwendet werden [9].

2.2.2 Bidirektionale Suche

Eine Variante des klassischen Dijkstra-Algorithmus ist die Erweiterung um Bidirektionalität. Im Gegensatz zum herkömmlichen Algorithmus, der nur in einer Richtung vom Startknoten zum Zielknoten arbeitet, führt der bidirektionale Dijkstra-Algorithmus eine Suche von beiden Knoten gleichzeitig durch. Die Rückwärtssuche traversiert dabei den transponierten Graphen, d. h. die Richtung der Kanten sind invertiert.
Der Algorithmus verwendet zwei Prioritätswarteschlangen, eine für die Vorwärtsrichtung und eine für die Rückwärtsrichtung. Der Ablauf des Algorithmus ist in Abbildung 2 illustriert.

1. Initialisierung: Jede Prioritätswarteschlange wird mit dem Startknoten bzw. dem Zielknoten initialisiert. Die vorläufigen Werte werden auf 0 für den Startknoten bzw. auf unendlich für den Zielknoten gesetzt.

2. Expansionsschritt: In jedem Schritt wird der Knoten mit dem kleinsten vorläufigen Wert

von beiden Warteschlangen ausgewählt. In der Schlange mit dem kleineren Knoten wird eine Dijkstra-Iteration ausgeführt.

3. Überprüfung des Treffpunkts: Bei jedem Expansionsschritt wird überprüft, ob der aktuelle Knoten in beiden Richtungen erreicht wurde. Wenn ein Knoten in beiden Richtungen erreicht wurde, gibt es einen potenziellen Pfad von Start zu Ziel.

4. Terminierung: Der Algorithmus terminiert, wenn beide Prioritätswarteschlangen leer sind oder ein Treffpunkt gefunden wurde.

5. Kürzester Weg: Für jeden Knoten u der von beiden Seiten erreicht wurde (u besitzt eine endliche Distanz von s und t), wird die Summe $dist(s,t) = dist(s,u) + dist(t,u)$ berechnet. Der kürzeste Weg verläuft über den Knoten, bei dem die Summe minimal ist.

6. Rückverfolgung des kürzesten Pfades: Wenn ein Treffpunkt u gefunden wurde, kann der kürzeste Pfad durch Rückverfolgung der Vorgängerknoten von Startknoten bis zum Treffpunkt in der Vorwärtsrichtung und von Zielknoten bis zum Treffpunkt in der Rückwärtsrichtung rekonstruiert werden.

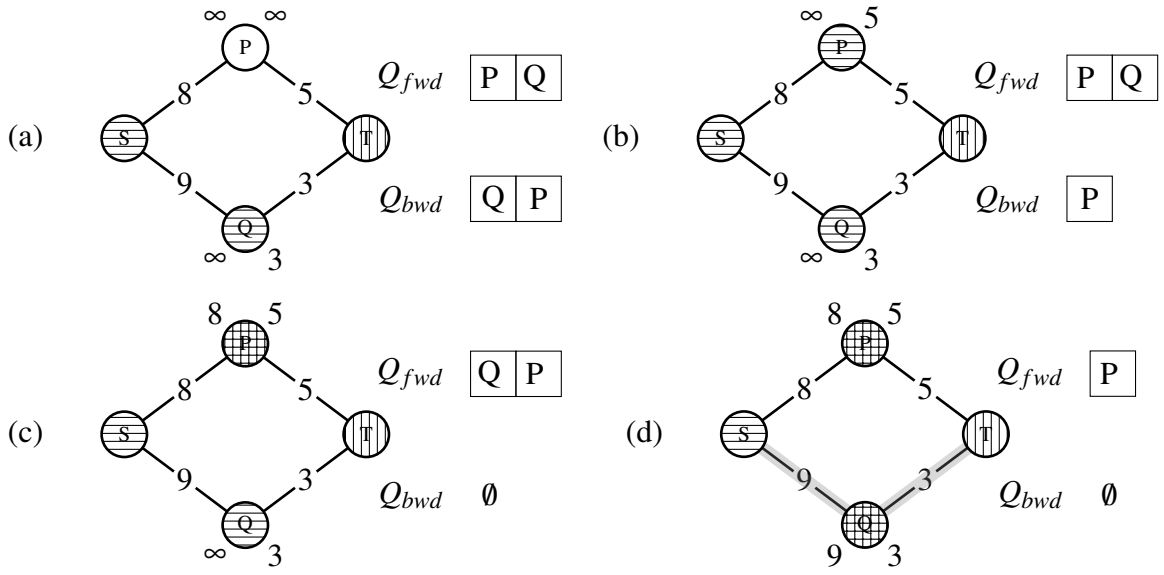


Abbildung 2: Bidirektionale Suche von S nach T: In Schritt (a) und (b) wird zuerst Knoten Q und P von der Rückwärtssuche relaxiert. In den Schritten (c) und (d) wird P und Q von der Vorwärtssuche relaxiert. Obwohl P der erste Knoten ist, der von beiden Suchen relaxiert wurde, ist der Weg über Q am kürzesten.

Der bidirektionale Dijkstra Algorithmus kann die Anzahl der untersuchten Knoten im Vergleich zum herkömmlichen Dijkstra reduzieren, insbesondere in großen Graphen. Durch

die gleichzeitige Suche in beiden Richtungen kann er die Laufzeit verbessern, indem er die Anzahl der Expansionsschritte und die Anzahl der Knoten, die in Betracht gezogen werden, reduziert. In der Praxis wird der Suchraum etwa um die Hälfte reduziert (siehe Abbildung 3):

$$A = \pi r^2$$
$$A_{bidir} = 2\pi(\frac{r}{2})^2 = \frac{A}{2}$$

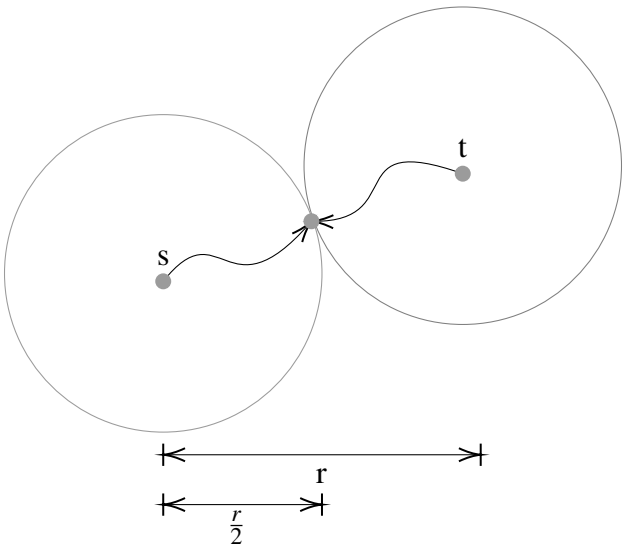


Abbildung 3: Suchraum einer bidirektioanlen Suche. Die Anzahl der Knoten die untersucht werden müssen, halbiert sich.

Korrektheitsbeweis

Lemma 2.1. *Für alle Knoten u die von beiden Seiten erledigt wurden, gilt:*
$$dist(s,t) = min\{dist(s,u) + dist(t,u)\}.$$

Beweis. Gegeben sei der Graph in Abbildung 4. Die Länge des kürzesten Pfads von s nach t wird als D bezeichnet. Der Knoten p ist der erste Knoten der von beiden Suchen erledigt wurde. Wenn gilt $dist(s,p) = dist(t,p)$, dann ist p garantiert auf dem kürzesten Weg.
Angenommen $dist(s,p)$ und $dist(t,p)$ sind ungleich D/2, dann muss entweder $dist(s,p) < D/2$ oder $dist(t,p) < D/2$ gelten. Dies wiederum bedeutet , dass alle Knoten mit einem kürzesten-Pfad-Wert, der kleiner oder gleich $D/2$ ist, bereits gesetzt wurden.
Knoten r und q liegen auf dem kürzesten Weg von s nach t und $dist(s,r) \leq D/2$ und $dist(t,q) \leq D/2$. Knoten r wurde von s erledigt und Knoten q von t. Dann wurde die Kante

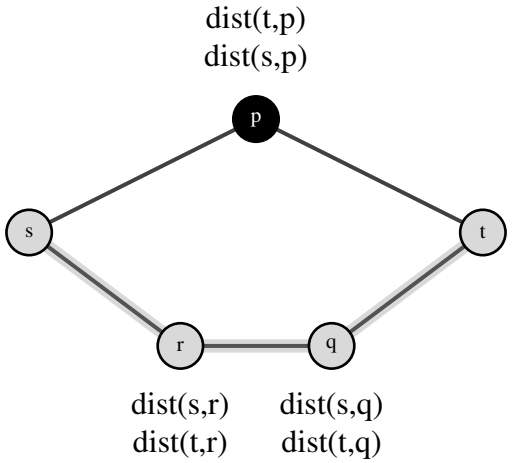


Abbildung 4: Graph für den Korrektheitsbeweis der bidirektionalen Suche

$e(r,q)$ bereits von beiden Seiten aus relaxiert uns somit haben r und q Distanzwerte von beiden Richtungen erhalten.
Die länge des kürzesten Weges von s nach t ist der kleinere Wert aus $dist(s,r)+dist(t,r)$ und $dist(s,q)+dist(t,q)$, welche in diesem Fall den gleichen Wert haben.
Somit ist der Knoten, der zuerst von beiden Seiten erledigt wurde nicht unbedingt auf dem kürzesten Weg, aber es wurde zumindest ein Knoten mit einer kürzeren Distanz gefunden, der auf dem kürzesten Weg liegt und einen Distanzwert von beiden Seiten erhalten hat. □

2.3 Contraction Hierarchies

2.3.1 Überblick

Straßennetzwerke sind extrem hierarchisch aufgebaut. Es gibt „wichtigere“ Straßen wie z. B. Autobahnen und „unwichtigere“ Straßen wie z. B. Straßen in Wohnsiedlungen (siehe Abbildung 5). Um von einem weit entfernten Ort zum anderen zu gelangen, macht es daher nur Sinn sich über immer wichtiger werdende Straßen zu bewegen. Bei einer Suche könnte die Information des Straßentyps und der damit verbundenen Wichtigkeit als Heuristik verwendet werden, um bestimmte Kanten, die auf weniger wichtige Straßen führen, zu ignorieren und damit die Suche zu beschleunigen. Das Problem hierbei ist allerding, dass mit dieser Methode keine Garantie besteht, dass der gefundene Weg auch wirklich der *exakt* kürzeste Weg ist. Die Methode der Contraction Hierarchies nach Geisberger et al. [3] [10] [11] löst dieses Problem, indem in einer Vorverarbeitungsphase Abkürzungskanten in den Graph eingefügt werden, die in der Suche ausgenutzt werden. Die Abkürzungen erhalten dabei die kürzesten Wege [6]. Während der Suche wird ein modifizieren bidirektioanlen Dijkstra-Algorithmus angewendet,

Abbildung 5: Hierarchie in Straßennetzen. Autobahnen (schwarz) sind ganz oben in der Hierarchie und sind sehr „wichtig“. Dagegen sind Straßen in Wohnsiedlungen weniger wichtig. © OpenStreetMap contributors

der Kanten die zu Knoten mit niedrigerem Level führen ignoriert. Dadurch wird der Suchraum extrem verkleinert, was zu schnellen Antwortzeiten führt. Der CH-Algorithmus lässt sich in zwei Komponenten unterteilen:

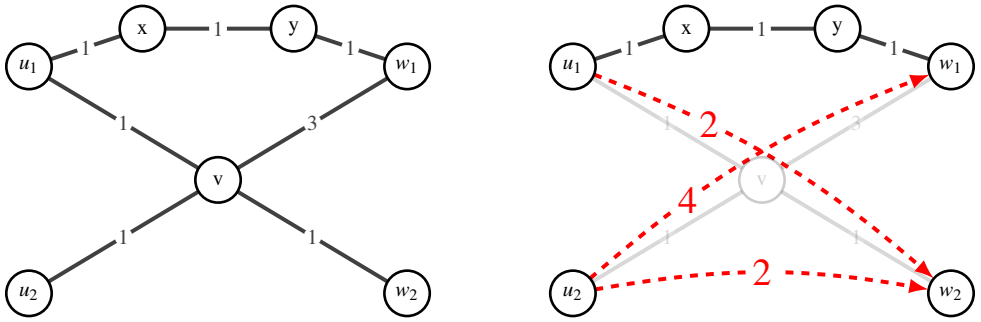
1. Vorverarbeitung: In dieser Phase werden die Knoten geordnet und die Hierarchie augebaut.

2. Suche: Ausführung der bidirektionale Suche auf dem erweiterten Graph.

2.3.2 Vorverarbeitung

In dieser Phase wird der Graph $G = (V, E)$ um zusätzliche Abkürzungskanten erweitert. Die Abkürzungskanten werden im Prozess der Knotenkontraktion (eng. Node Contraction) eingefügt. Wenn ein Knoten $v \in V$ *kontraktiert* wird, dann wird er und alle Kanten die mit v inzident sind aus dem Graphen entfernt. Der Zeitpunkt des Entfernens ist gleichzeitig das *Level* des Knotens. Je später der Knoten entfernt wird, desto höher ist sein Level und damit seine Relevanz. Wenn v auf dem kürzesten Weg zwischen zwei benachbarten Knoten u und w liegt, dann wird eine Abkürzungskante $e(u,w)$ mit $w(u,w) = w(u,v) + w(v,w)$ eingefügt, um den kürzesten Weg zu erhalten.
Abbildung 6 demonstriert den Prozess: Knoten v soll kontraktiert werden und damit er und seine inzidenten Kanten entfernt werden. Sei U die Menge aller eingehenden Kanten und W die Menge aller ausgehenden Kanten, dann muss für jedes Paar überprüft werden, ob v auf dem

kürzesten Weg <u,v,w> zwischen zwei benachbarten Knoten $u \in U$ mit $Level(u) > Level(v)$ und $w \in W$ mit $Level(w) > Level(v)$ liegt. Zum Beispiel ist dies zwischen u_1 und w_2 der Fall, denn es gibt keine andere Möglichkeit w_2 zu erreichen, als über v. Um den kürzesten Weg zu erhalten, wird eine Abkürzungskante $e(u_1,w_2)$ mit dem Gewicht $w(u_1,w_2) = w(u_1,v) + w(v,w_2) = 1+1 = 2$ eingefügt. Das gleiche gilt für <u_2,v,w_1> und <u_2,v,w_2>. Der Weg von u_1 nach w_1 kann allerding über <u_1,x,y,w_1> schneller erreicht werden (Kosten 3 sind kleiner als 4) und es muss daher keine Abkürzungskante eingefügt werden.



(a) Graph $G = (V,E)$ (b) Graph $G' = (V',E')$ mit $V' = V - \{v\}$

Abbildung 6: Beispiel einer Knotenkontraktion

Nachdem jeder Knoten kontraktiert wurde, erhält man einen neuen Graph $G^* = (V,E')$, der als *Overlaygraphh* bezeichnet wird. E' enthält alle ursprünglichen Kanten sowie die neu hinzugefügten Abkürzungskanten (siehe Abbildung 7).
Algorithmus zur Erstellung einer Contraction Hierarchie (CH)

Algorithm 2 RUN_CONTRACTION(G)

```
for each v ∈ V geordnet nach Level do
    for each (u,v) ∈ E mit Level(u) > Level(v) do
        for each (v,w) ∈ E mit Level(w) > Level(v) do
            if <u,v,w> ist kürzester Weg von u nach w then
                E = E ∪ {e(u,w)} mit Gewicht w(u,w) = w(u,v) + w(v,w)
            end if
        end for
    end for
end for
```

Beweis: Kontraktion erhält kürzeste Wege

Lemma 2.2. *Sei $G = (V,E)$ ein beliebiger Graph und $G' = (V',E')$ der Graph nach Kontratkion von einem beliebigen Knoten $v \in V$ mit $V' = V - \{v\}$.*
Dann gilt für alle $s,t \in V'$: $dist_{G'}(s,t) = dist_G(s,t)$.

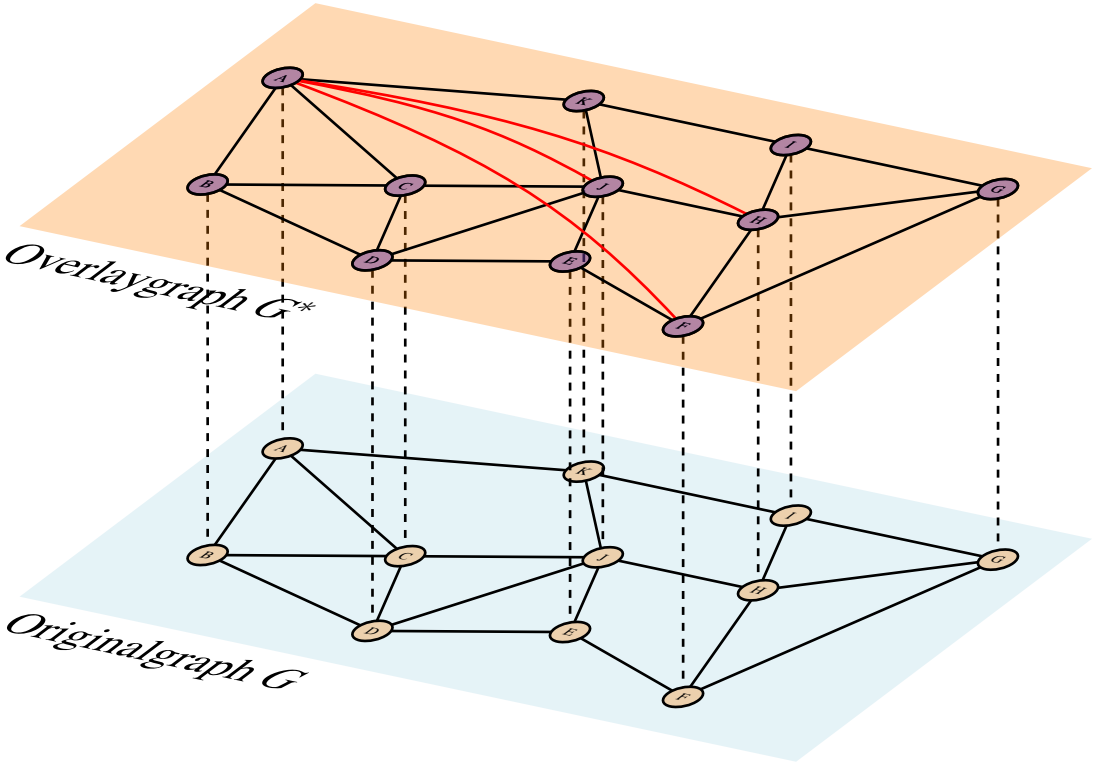


Abbildung 7: Overlaygraph nach Kontraktionsprozess. Der Graph G^* enthält alle ursprünglichen Kanten sowie die neu hinzugefügten Abkürzungskanten.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed euismod, nisl quis tincidunt pellentesque, nunc nisl ultrices ipsum, quis aliquam nunc nisl ut nunc. Nulla facilisi. Nulla

2.3.3 Suche

2.4 OpenStreetMap

OpenStreetMap (OSM) ist ein Projekt, das sich der Erstellung und Bereitstellung von freien geografischen Daten verschrieben hat. Es handelt sich dabei um Daten die von einer weltweiten Gemeinschaft von Freiwilligen erstellt und gepflegt wird [12].
Das grundlegende Konzept von OpenStreetMap basiert auf OpenData, was bedeutet, dass die Daten frei verfügbar und für jeden zugänglich sind. Im Gegensatz zu kommerziellen Kartenanbietern, die ihre Daten schützen und für den Zugriff hohe Gebühren verlangen, ermutigt OpenStreetMap Menschen dazu, ihre eigenen Daten beizutragen und von den vorhandenen Daten zu profitieren.
Das OpenStreetMap-Projekt verwendet eine Kombination aus verschiedenen Datenquellen, um eine detaillierte und umfassende Karte zu erstellen. Dazu gehören zum Beispiel Satelliten-

bilder, GPS-Tracks, Luftaufnahmen und auch öffentlich verfügbare geografische Daten. Die Daten werden von Freiwilligen erfasst, indem sie Straßen, Gebäude, Gewässer, Landnutzung und andere geografische Merkmale auf der Karte markieren oder Informationen darüber hinzufügen.
Die OpenStreetMap-Daten sind unter einer offenen Lizenz, der Open Data Commons Open Database Lizenz (ODbL), verfügbar. Das bedeutet, dass die Daten frei verwendet, kopiert, modifiziert und weiterverbreitet werden können, solange die Lizenzbedingungen eingehalten werden [13]. Dadurch wird Entwicklern ermöglicht, die OpenStreetMap-Daten in ihre eigenen Anwendungen und Dienste zu integrieren.

3 Schlussbetrachtung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Literatur

[1] E. W. Dijkstra. „A note on two problems in connexion with graphs“. In: *Numerische Mathematik* 1 (1959), S. 269–271.

[2] Peter Hart, Nils Nilsson und Bertram Raphael. „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[3] Robert Geisberger u. a. „Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks“. In: *Experimental Algorithms*. Hrsg. von Catherine C. McGeoch. Springer, 2008, S. 319–333.

[4] „9th DIMACS Implementation Challenge: Shortest Paths“. In: *International Workshop on Experimental and Efficient Algorithms*. DIMACS. 2006. URL: http://www.dis.uniroma1.it/~challenge9/.

[5] Daniel Delling u. a. „Engineering Route Planning Algorithms“. In: *Algorithmic of Large and Complex Networks*. Bd. 5515. Springer, 2009, S. 117–139.

[6] Hannah Bast u. a. *Route Planning in Transportation Networks*. URL: http://arxiv.org/pdf/1504.05140v1.

[7] Thomas Liebig u. a. „Dynamic route planning with real-time traffic predictions“. In: *Information Systems* 64 (2017), S. 258–265. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2016.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0306437916000181.

[8] Steve Kalbink und Carol Nichols. *The Rust Programming Language*. Bd. 2. 2022.

[9] Thomas H. Cormen u. a. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009, S. 589–592, 661–662. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[10] Robert Geisberger. „Diploma Thesis - Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks“. In: 2008.

[11] Robert Geisberger u. a. „Exact Routing in Large Road Networks using Contraction Hierarchies“. In: *Transportation Science*. Bd. 46. 2012, S. 388–404.

[12] OpenStreetMap Foundation. *About OpenStreetMap*. 2023. URL: https://www.openstreetmap.org/about (besucht am 16. 07. 2023).

[13] OpenStreetMap Foundation. *OpenStreetMap License*. 2023. URL: https://www.openstreetmap.org/copyright (besucht am 16. 07. 2023).

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

München, den 18. Juli 2023