



Hochschule für angewandte Wissenschaften München  
Fakultät für Geoinformation

## Bachelorarbeit

# Untersuchung der Implementierung von Contraction Hierarchies in Straßennetzwerken

**Verfasser:** Daniel Holzner  
**Matrikelnummer:** 26576714  
**Studiengang:** Geoinformatik und Navigation  
**Betreuer:** Prof. Dr. Thomas Abmayr  
**Abgabedatum:** 9. 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

<b>Abbildungsverzeichnis</b>	<b>5</b>
<b>Tabellenverzeichnis</b>	<b>6</b>
<b>Quellcodeverzeichnis</b>	<b>7</b>
<b>Abkürzungsverzeichnis</b>	<b>8</b>
<b>1 Einleitung</b>	<b>9</b>
1.1 Motivation und Kontext . . . . .	9
1.2 Stand der Forschung . . . . .	10
1.3 Beitrag der Arbeit . . . . .	11
1.4 Struktur der Arbeit . . . . .	11
<b>2 Theoretische und technische Grundlagen</b>	<b>12</b>
2.1 Datenstrukturen . . . . .	12
2.1.1 Graph . . . . .	12
2.1.2 Vorrangwarteschlangen . . . . .	13
2.2 Kürzeste-Wege-Algorithmen . . . . .	14
2.2.1 Grundlegende Technik . . . . .	14
2.2.2 Zielorientiert . . . . .	14
2.2.3 Hierarchisch . . . . .	14
2.3 OpenStreetMap . . . . .	14
<b>3 Schlussbetrachtung</b>	<b>15</b>
<b>Literatur</b>	<b>16</b>

## Abbildungsverzeichnis

1	Der oben gezeigte gerichtete Graph (a) dargestellt als Adjazenzmatrix(b) oder Adjazenzliste (c). . . . .	13
---	--	----

## **Tabellenverzeichnis**

## **Quellcodeverzeichnis**

### **Todo list**

## **Abkürzungsverzeichnis**

**CHs** Contraction Hierarchies

**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  $\text{dist}(s,t)$  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 Straßennetze 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) `todo[]` Add ref 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 Straßennetzen 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 Straßennetzen 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 Straßennetzen 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. todo[]Shorten

## 2 Theoretische und technische Grundlagen

### 2.1 Datenstrukturen

#### 2.1.1 Graph

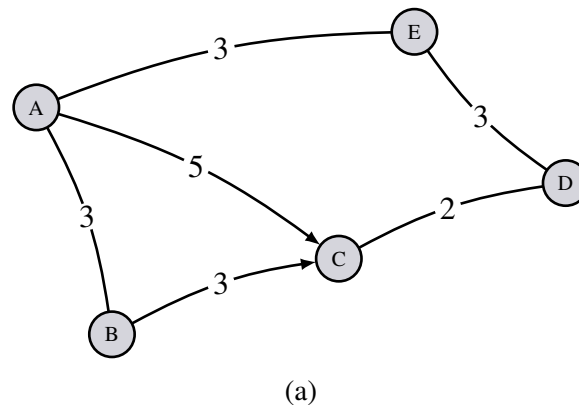
Um das kürzeste-Wege-Problem zu lösen, muss zunächst das reale Straßennetz in eine abstrakte Form gebracht werden. Hierzu wird das Straßennetz als Graph modelliert. Ein Graph  $G = (V, E)$  besteht aus einer Menge von Knoten  $V$  und einer Menge von Kanten  $E$ . Jede Kante  $e = (u, v) \in E$  verbindet 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 für jede Kante ein Gewicht  $w(e)$  festgelegt. In diesem Fall wird die durchschnittliche Zeit  $t = \frac{V}{S}$  in Sekunden verwendet, die sich aus der Länge des Straßensegments  $S$  und der maximal erlaubten Geschwindigkeit  $V$  auf dieser Straße berechnet. Der kürzeste Weg zwischen zwei Knoten ist damit der Weg mit der minimalen Zeit.

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 ist  $|E|$ , 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 ( $|E| \ll |V|^2$ ), 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 allerdings 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. [todo\[\]Ref genaue Datenstruktur?](#)



	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	→	<span style="border: 1px solid black; padding: 0 2px;">B</span>	<span style="border: 1px solid black; padding: 0 2px;">C</span>	<span style="border: 1px solid black; padding: 0 2px;">E</span>
B	→	<span style="border: 1px solid black; padding: 0 2px;">A</span>	<span style="border: 1px solid black; padding: 0 2px;">C</span>	
C	→		<span style="border: 1px solid black; padding: 0 2px;">D</span>	
D	→		<span style="border: 1px solid black; padding: 0 2px;">C</span>	<span style="border: 1px solid black; padding: 0 2px;">E</span>
E	→	<span style="border: 1px solid black; padding: 0 2px;">A</span>	<span style="border: 1px solid black; padding: 0 2px;">D</span>	

(c)

Abbildung 1: Der oben gezeigte gerichtete Graph (a) dargestellt als Adjazenzmatrix(b) oder Adjazenzliste (c).

### 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 Warteschlange 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 binärem Min-Heap verwendet mit einer Zeitkomplexität für das Einfügen von  $\theta(1)$  und für das Entfernen von  $\theta(\log n)$ .

## 2.2 Kürzeste-Wege-Algorithmen

Haha das ist doch wahnsinn

### 2.2.1 Grundlegende Technik

### 2.2.2 Zielorientiert

### 2.2.3 Hierarchisch

## 2.3 OpenStreetMap

### 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. ISBN: 978-0-262-03384-8. URL: <http://mitpress.mit.edu/books/introduction-algorithms>.



## **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 9. Juli 2023

A handwritten signature in black ink, consisting of a stylized 'Z' followed by a horizontal line with a small upward tick at the end.