Schnelle Wegsucheverfahren auf digitalen Straßenkarten
–
Entwicklung, Implementierung und Anwendungsbeispiel bei einem Logistikdienstleister

Dissertation zur Erlangung des Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)
des Fachbereichs IV
– Mathematik, Naturwissenschaften, Wirtschaft und Informatik –
der Universität Hildesheim

Curt Nowak

18. März 2014

Versicherung gemäß §3 Abs. 2a der Promotionsordnung in der Beschlussfassung vom 11.07.1991
Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit mit dem Titel

Schnelle Wegsucheverfahren auf digitalen Straßenkarten
–
Entwicklung, Implementierung und Anwendungsbeispiel bei einem Logistikdienstleister

im Bereich Betriebswirtschaft mit Schwerpunkt in der Logistik selbstständig und ohne unerlaubte Hilfe verfasst und die benutzten Hilfsmittel vollständig angegeben habe.
Ich habe bisher an keiner in- oder ausländischen Universität einen Antrag auf Zulassung zur Promotion gestellt, noch die vorliegende oder eine andere Arbeit als Dissertation vorgelegt.

Hannover, 18. März 2014

Zusammenfassung

Die Lösung vieler straßengebundener Transportplanungsprobleme aus der Praxis ist kombinatorisch aufwendig, komplex und manche der notwendigen Informationen liegen nicht hinreichend strukturiert vor. Standard-Lösungsverfahren sind daher für sie oft ungeeignet. Dies gilt auch für die operative Disposition bei Express- und Direkt-Kurierdienstleistern, bei der Entscheidungen über die Zuweisung von Fahrzeugen zu Aufträgen getroffen werden. Üblicherweise kommen für diese Aufgabe dialogorientierte Entscheidungsunterstützungssysteme (EUS) zum Einsatz, welche Vorschläge generieren, aus denen menschliche Disponenten unter Einbezug von domänenspezifischem Wissen den günstigsten auswählen. Lösungen, bei denen die Frachten mehrerer Aufträge auf einem Fahrzeug konsolidiert werden, sind dabei häufig besonders wirtschaftlich.

Diese Arbeit beschäftigt sich mit der Optimierung der Disposition im Hinblick auf das Konsolidierungspotential der Frachten. Zum Aufdecken möglicher Ersparnisse durch Konsolidierung müssen viele alternative Fahrzeugrouten ermittelt und verglichen werden. Hierfür sind schnelle Verfahren zur Wegsuche von zentraler Bedeutung. Diese Verfahren bilden den ersten Schwerpunkt der Arbeit.

Aufbauend auf einem vergleichenden Überblick „klassischer“ sowie aktueller Wegsucheverfahren wird mit den *Contraction Hierarchies* (CH) ein hierarchisches Verfahren in den Mittelpunkt gestellt, das ein besonders vorteilhaftes Verhältnis von Suchgeschwindigkeit zu benötigtem Speicherbedarf aufweist. Untersucht werden Optimierungen der Hierarchie-Erzeugung sowie neue Erweiterungen der Wegsuche in CH. In ausführlichen Benchmarks auf digitalen Straßenkarten des OpenStreetMap-Projekts werden die deutlichen Verbesserungen durch diese Erweiterungen empirisch nachgewiesen.

Eine größere Realitätsnähe der berechneten Routen ergibt sich durch die Berücksichtigung von Abbiegebeschränkungen bei der Wegsuche. Nach der Vorstellung der in der Literatur hierfür üblichen Ansätze wird mit der *adaptiven Wegsuche* ein leistungsstarkes neues Verfahren für diesen Zweck präsentiert und für den Einsatz in CH angepasst. Die oben genannten Erweiterungen der Wegsuche sind mühelos auf dieses Verfahren übertragbar. Weitere Benchmarks unterstreichen die Vorteile der adaptiven gegenüber der in früherer Literatur eingesetzten pfeilbasierten Suche nach Wegen mit Abbiegebeschränkungen in CH.

Für den zweiten Schwerpunkt der Arbeit, das Konsolidierungsproblem, wird zunächst ein ausführliches mathematisches Modell entwickelt. Es folgt eine Extraktion realitätsnaher Modellannahmen aus operativen Vergangenheitsdaten der IN tIME Express Logistik GmbH. Mit dem *rekursiven Savingsverfahren* wird schließlich eine neue Heuristik präsentiert, die es ermöglicht, Konsolidierungsvorschläge im Rahmen eines EUS zu generieren. Sie offenbart in mehreren, auf den Vergangenheitsdaten basierenden Benchmarks ein deutliches Kosteneinsparungspotential gegenüber früheren Dispositionsentscheidungen.

Abstract

The solution to many real-world transportation planning problems is combinatorially comprehensive, complex, and some of the necessary information is not available in a sufficiently structured form. Hence, standard solving methods are often not applicable. This holds for courier and express providers' operational dispatching, where vehicles are assigned to customer orders. Typically, dialog-oriented decision support systems (DSS) are used to generate recommendations from which a human dispatcher selects the most profitable one by means of domain-specific knowledge. In this process, solutions that consolidate the freight of multiple customer orders onto a single vehicle are usually particularly favorable.

This work examines optimization of the dispatching process with particular attention to the potential of freight consolidation. In order to uncover possible savings by consolidating, many alternative vehicle routes need to be determined and compared with each other. For this purpose, fast routing algorithms are essential. These algorithms form the first focus of this work.

Based on a comparative literature overview of "classical" as well as current routing algorithms, the center of attention is put on *contraction hierarchies* (CH) as they provide a particularly favorable ratio of query speed-up to additional memory requirement. Both optimizations of hierarchy creation and new enhancements for queries in CH are studied. The considerable merits of these approaches are empirically shown through extensive benchmarks on digital road maps provided by the OpenStreetMap project.

More realistic routes are obtained when routing algorithms take turn restrictions into account. First, common approaches from literature are presented for this purpose. Next, the *adaptive search* is introduced as a more efficient new approach, and adjusted for usage in CH. The above-mentioned query enhancements can easily be applied to this method. The advantages of the adaptive search in CH over the edge-based search that is employed for turn restriction aware routing in earlier literature is emphasized by further benchmarks.

For the second focus of this work – the vehicle routing problem with freight consolidation – first, a thorough mathematical model is developed. Next, model assumptions are extracted from past operational real-world data provided by IN tIME Express Logistik GmbH. Eventually, the *recursive savings algorithm* is introduced as a heuristic that can generate freight consolidation recommendations in the context of a DSS. Employed in multiple benchmarks based upon the past operational data, it reveals considerable cost savings compared to former dispatching decisions.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Verzeichnis der Verfahren

Glossar

ALT	A* with landmarks and triangle inequality
API	Application Programming Interface
ArbZG	Arbeitszeitgesetz
CH	Contraction Hierarchy
DARP	Dial A Ride Problem
EUS	Entscheidungsunterstützungssystem
Flag	Bezeichnung für einen booleschen Wert, der häufig im Kontext der Programmierung verwendet wird
FPersV	Fahrpersonalverordnung
GB	Gigabyte
GPS	Global Positioning System
GPX	GPS Exchange Format; offenes, lizenzfreies und XML-basiertes Datenformat zur Speicherung von GPS-Daten
HH	Highway Hierarchies
HL	Hub-Based Labeling
HNR	Highway Node Routing
HTTP	Hypertext Transfer Protocol
JOSM	Java OpenStreetMap Editor
JVM	Java Virtual Machine
MB	Megabyte
Orthodrome	Bezeichnung der kürzesten Linie zwischen zwei Punkten auf einer Kugeloberfläche
OSM	OpenStreetMap
Osmosis	Java-Kommandozeilen-Programm zum Filtern von OpenStreetMap-Daten
pbf	Protocol Buffer Format
PDP	Pickup And Delivery Problem
PDVRP	Pickup and Delivery Vehicle Routing Problem

Point of Interest	häufig bei Navigationsgeräten: Ort, der einen Besucher interessieren könnte
Potlatch	Browserbasierter Online-Editor für OpenStreetMap-Daten, der Echtzeitbearbeitung erlaubt
SQL	Datenbanksprache sowohl für Abfragen als auch Definitionen innerhalb einer relationalen Datenbank; häufig als Abkürzung für *Structured Query Language* interpretiert
TNR	Transit Node Routing
VO (EG) 561/2006	Verordnung (EG) Nr. 561/2006 des Europäischen Parlaments und des Rates vom 15. März 2006 zur Harmonisierung bestimmter Sozialvorschriften im Straßenverkehr und zur Änderung der Verordnungen (EWG) Nr. 3821/85 und (EG) Nr. 2135/98 des Rates sowie zur Aufhebung der Verordnung (EWG) Nr. 3820/85 des Rates
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows
Wikipedia	Offene (im Sinne von *public domain*) Online-Enzyklopädie
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation und Gegenstand dieser Arbeit

Moderne Logistik umfasst heutzutage mehr als den reinen Gütertransport. Als Querschnittsfunktion ist sie – ähnlich wie das Personal- oder das Finanzwesen – in nahezu allen Unternehmen zu finden und erreicht dabei alle Bereiche heutiger Wirtschaftskreisläufe. Sie beinhaltet sowohl Planung, Steuerung als auch Kontrolle der Material-, Personen-, Energie- und Informationsflüsse (vgl. [Jünemann u. a., 1989, S. 11]). Und doch bleibt das Transportwesen möglicherweise ihre augenfälligste Facette, was nicht zuletzt an seiner Sichtbarkeit im Straßen-, Luft-, Schiffs- und Bahnverkehr liegt.

So ist auch zu erklären, dass logistische Problemstellungen im Bereich der Transportplanung gängige Forschungsgebiete des Operations Research sind. Die Probleme gehören auf der einen Seite häufig zur Klasse der kombinatorischen Optimierungsprobleme, was ein hohes Potential für Optimierungen verspricht und gleichzeitig den Einsatz computergestützter Lösungsverfahren begründet. Auf der anderen Seite sind ihre Lösungen besonders anschaulich.

Für viele dieser Problemstellungen sind zwar exakte Verfahren denkbar, doch ihr kombinatorischer Charakter macht diese Verfahren impraktikabel. Der zur Berechnung einer optimalen Lösung nötige Aufwand wäre oft deutlich zu groß. Vielmehr werden solche Probleme mit Heuristiken gelöst.

Heuristiken sind Lösungsverfahren, welche nur einen Teil des zu einem Optimierungsproblem gehörigen Lösungsraums betrachten. Als Vorteil gelingt es dadurch schneller eine Lösung zu generieren. Der Nachteil ist jedoch, dass diese Lösung folglich auch nur ein suboptimales Ergebnis garantiert. In Abhängigkeit von der Problemstellung und der eingesetzten Lösungsstrategie weichen die Ergebnisse teils jedoch nur wenig von einem tatsächlichen Optimum ab. (vgl. [Domschke, 1997, S. 21 ff.])

Einige logistische Probleme sind allerdings derart komplex, dass sie kaum noch in einem überschaubaren mathematischen Modell zu erfassen sind. Andere Logistik-Probleme sind unstrukturiert und entziehen sich damit einer vollständigen mathematischen Beschreibung. Die Berechnung optimaler Lösungen ist im ersten Fall mit sinnvollem Aufwand nicht mehr realisierbar und im zweiten Fall mangels der Formulierbarkeit einer Zielfunktion unmöglich.

Auf viele straßengebundene Transportplanungsprobleme aus der Praxis treffen alle drei oben genannten Eigenschaften zu. Sie sind kombinatorisch aufwendig, sehr komplex und manche der für eine Lösung notwendigen Informationen liegen nicht in strukturierter Form vor.

Die kombinatorische Vielfalt ergibt sich durch die Möglichkeiten bei der Zuordnung von Fahrzeugen zu Aufträgen, der so genannten *Disposition*. Die Komplexität der Probleme entsteht unter anderem durch spezifische Kundenwünsche und rechtliche Rahmenbedingungen wie Lenk- und Ruhezeiten. Die Unstrukturiertheit betrifft beispielsweise besondere Kundenanforderungen, die über eine bloße Angabe von Zeitfenstern zur Belieferungen hinausgehen und teils nur mündlich ausgetauscht werden. Auch dynamische Störungen wie Verkehrsstaus werden derzeit häufig noch wenig strukturiert erfasst. Details wie Sendungsstrukturen, d. h. die physische Beschaffenheit der Frachten von Aufträgen, können manchmal sogar erst bei der Abholung in Erfahrung gebracht werden.

EUS

Zur Lösung dieser Probleme werden im Rahmen praktischer Anwendungen häufig dialogorientierte *Entscheidungsunterstützungssysteme* eingesetzt. Der Begriff der Entscheidungsunterstützungssysteme[1] lässt sich auf die Arbeiten von Gorry u. Scott Morton [1971] und Scott Morton [1971] zurückverfolgen. Obwohl die Computersysteme der frühen 1970er Jahre in Bezug auf die Leistungsfähigkeit der Datenverarbeitung und Geschwindigkeit kaum mit heutigen Systemen vergleichbar sind, erkannten die Autoren bereits früh das Potential der Computerunterstützung bei Entscheidungsproblemen.

Der Fokus von EUS liegt auf „einer gesteigerten individuellen und organisatorischen Effektivität“ und ist von einer „gesteigerten Effizienz in der massenhaften Datenverarbeitung“ abgegrenzt ([Alter, 1980, S. 3]).[2] Damit heben sich EUS als Weiterentwicklung früherer reiner Datenverarbeitungssysteme hervor. Gaul u. Both [1990, S. 49] sprechen bei einem EUS sogar von einem „Intelligenzverstärker“.

Die Anwendungsfälle für ein EUS lassen sich danach unterteilen, ob sie eher daten- oder modellorientierte Anfragen an das System stellen (vgl. [Alter, 1980, S. 73 ff.]). Tabelle 1.1, S. 25, listet die Anfragekategorien in der Sortierung von datenorientiert nach modellorientiert. Die Liste ist um Beispiele ergänzt, die zeigen, wie sehr das Konzept von EUS Einzug in moderne Anwendungen gehalten hat.

[1] im Englischen *decision support systems*; kurz: DSS

[2] im englischen Original: „Thus the emphasis of DSSs is on increased individual and organizational effectiveness rather than on increased efficiency in processing masses of data.“

Anwendungsfall	Beispiel
Reine Datenabfragen	SQL-Abfragen in Datenbanken
Ad-hoc-Datenanalysen	Einfache Tabellenkalkulationen
Datenaggregation in Form von Berichten	Automatisierte Berichte in Datenbanken
Abwägen der Folgen von Entscheidungen	Durchführen von Simulationen
Generierung von Entscheidungsvorschlägen	Routenplanung in Navigationssystemen
Erzeugung von Entscheidungen	Vollautomatisierte Bestellungen in modernen Lagersystemen

Tabelle 1.1: Anwendungsfälle für EUS von datenorientiert (erste Zeile) nach modellorientiert (letzte Zeile)

Der strukturelle Aufbau eines EUS unterscheidet drei wesentliche Systemkomponenten. Bei Bonczek u. a. [1981, S. 69 ff.] bzw. Gaul u. Both [1990, S. 203] sind diese:

- das Sprachsystem bzw. die Benutzerschnittstelle,
- das Wissenssystem bzw. die Datenverwaltung und
- das Problemlösungssystem[3] bzw. die Methoden- und Modellverwaltung.[4]

Abbildung 1.1 zeigt den Aufbau schematisch. Pfeile symbolisieren Beziehungen und Informationsflüsse zwischen den Komponenten.

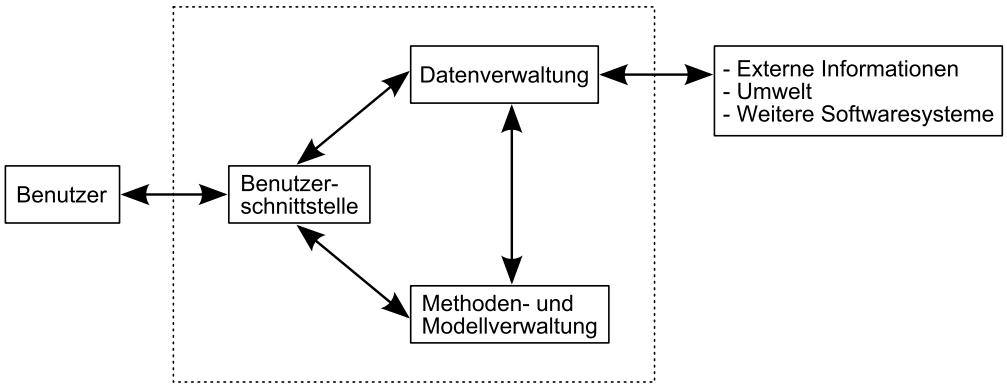


Abbildung 1.1: Schematischer Aufbau eines Entscheidungsunterstützungssystems Durch gepunkteten Rahmen hervorgehoben: EUS im engeren Sinne Quelle: in Anlehnung an Gaul u. Both [1990, S. 203]

Während Benutzerschnittstelle und Datenverwaltung zu den meisten interaktiven Softwaresystemen gehören, stellt die Methoden- und Modellverwaltung eine Besonderheit von EUS dar (vgl. [Gaul u. Both, 1990, S. 204]). Sie beinhaltet gleichsam die Erkenntnisse der Forschung aus dem Bereich des Operations Research für den zugrunde liegenden Anwendungszweck des EUS.

[3] im englischen Original: *language system*, *knowledge system* und *problem-processing system*
[4] Diese Arbeit verwendet die Begriffe nach Gaul u. Both [1990].

Für logistische Problemstellungen wie die Disposition enthält die Datenverwaltung eines EUS beispielsweise Stammdaten für Kunden, Fahrzeugflotte und offene sowie abgeschlossene Aufträge. Die Methoden- und Modellverwaltung umfasst in diesem Kontext dagegen Algorithmen (Methoden) zur Berechnung einer kostengünstigen Zuordnung von Fahrzeugen zu Aufträgen. Grundlage hierfür kann eine digitale Straßenkarte (Modell) sein, auf der Wegsucheverfahren (Methoden) schnellste oder auch kostengünstigste Fahrzeugrouten liefern.

Über ein derartiges EUS verfügt auch die IN tIME Express Logistik GmbH (fortan auch: IN tIME). Der europaweit tätige Kurierdienst ist auf den Bereich *Expressversand* spezialisiert, was bedeutet, dass Kundenaufträge in der Regel wenig Vorlaufzeit besitzen und sowohl aus einer *Abholung* als auch einer *Auslieferung* bestehen. Als Grundlage für eine empirische Untersuchung stellte IN tIME dem Autor einen Datensatz operativer Vergangenheitsdaten zur Verfügung. Anhand dieser Daten sind frühere Aufträge sowie die zugehörigen Transporte über einen Zeitraum mehrerer Jahre hinweg nachvollziehbar.

Gegenstand dieser Arbeit sind zwei logistische Problemstellungen aus der Praxis, die sowohl bei IN tIME als auch bei den meisten anderen Transportdienstleistern unter Zuhilfenahme von dialogorientierten EUS gelöst werden:

1. die Suche nach optimalen Wegen auf digitalen Straßenkarten und
2. die Suche nach kostenoptimalen Dispositionen.

Die erste Problemstellung taucht dabei häufig wiederholt als Teilproblem innerhalb der zweiten Problemstellung auf. Denn im Rahmen der Disposition wird über die Zuordnung von Fahrzeugen zu Aufträgen entschieden. Um eine möglichst günstige Auswahl zu treffen, werden dabei viele verschiedene Zuordnungen miteinander verglichen. Hierdurch sind ebenso vielfältige Routen denkbar. Einige dieser Routen lassen sich zwar im Vorfeld durch geeignete Abschätzungen als ineffizient ausschließen. Doch für eine Gegenüberstellung müssen die meisten mithilfe von Wegsucheverfahren auf digitalen Straßenkarten ermittelt werden.

Darüber hinaus befinden sich bei Transportdienstleistern wie IN tIME im laufenden Betrieb nicht alle Fahrzeuge notwendigerweise an einem zentralen Ort, wie einem Depot, sondern stehen oft direkt nach Auslieferung eines Auftrags wieder für eine neue Abholung zur Verfügung. Dadurch entsteht für jedes Fahrzeug und jeden Auftrag eine individuelle Anfahrtsroute, die ihrerseits berechnet werden muss. Die Geschwindigkeit des eingesetzten Wegsucheverfahrens ist daher ein kritischer Faktor für ein Dispositions-EUS, sofern die Aufträge nicht über längere Zeit im Voraus bekannt sind.

Wenn es die Ladekapazität eines bereits disponierten Fahrzeugs zulässt, ist es grundsätzlich möglich, diesem noch weitere Aufträge zuzuordnen. Hierdurch entsteht ein gleichzeitiger Transport der Sendungen unterschiedlicher Aufträge durch ein einzelnes Fahrzeug. Eine solche Disposition wird in dieser Arbeit auch als *Frachtkonsolidierung* bezeichnet.

Durch Frachtkonsolidierungen kann ein Transportunternehmen mehrere Vorteile parallel erzielen. Einerseits erlauben sie eine bessere Auslastung der Fahrzeugflotte. Dies verringert andererseits die entstehenden direkten Fahrtkosten und erhöht somit die Wirtschaftlichkeit des Unternehmens. Mit einer höheren Fahrzeugauslastung können außerdem mehr Aufträge bearbeitet werden, was sich erneut positiv auf die Wirtschaftlichkeit

auswirkt.[5] Nicht zuletzt geht mit einer erhöhten Fahrzeugauslastung auch eine geringere Umweltbelastung durch den Transportbetrieb je Fracht einher.

Allerdings erweitert die Möglichkeit von Konsolidierungen gleichzeitig die Anzahl denkbarer Transportzuordnungen und somit die Komplexität bei der Disposition. Gesucht sind daher geschickte, d. h. schnelle und effiziente Verfahren, die im Rahmen eines EUS menschlichen Disponenten sinnvolle Entscheidungsvorschläge unterbreiten und sie so bei der Planung unterstützen.

Die vorliegende Arbeit liefert hierzu einen Beitrag durch die Untersuchung und Erweiterung vorhandener Wegsucheverfahren sowie darauf aufbauend durch die Entwicklung eines eigenen Verfahrens zur direkten Unterstützung der operativen Disposition. Vor diesem Hintergrund ist auch die Gliederung der Arbeit zu verstehen.

1.2 Aufbau der Arbeit

Im nachfolgenden Kapitel 2 werden die wesentlichen Datengrundlagen vorgestellt, anhand derer in späteren Kapiteln empirische Untersuchungen durchgeführt werden. Sie werden auch hinsichtlich ihrer Qualität diskutiert. Das inhaltlich an der konkreten Implementierung orientierte Kapitel 3 beschreibt dann die vorgenommenen Interpretationen und Transformationen der digitalen Straßenkarten des OpenStreetMap-Projekts.

Als wesentliche Komponente eines Disposition-EUS in Kapitel 4 werden im Anschluss zuerst ausgewählte „klassische“ sowie moderne Wegsucheverfahren für digitale Straßenkarten aus der Literatur vorgestellt, diskutiert und miteinander verglichen. Eine besondere Rolle spielt dabei das Contraction-Hierarchies-Verfahren, welches gesondert in Kapitel 5 behandelt wird, denn es bietet ein ausgesprochen günstiges Verhältnis von Suchgeschwindigkeit zu Speicherbedarf. Es werden zu diesem Verfahren mehrere Erweiterungen präsentiert, um die Wegsuche zu beschleunigen. Ihr Einfluss wird anhand ausführlicher Benchmarks evaluiert.

Für praxistaugliche Routen sind auch Abbiegebeschränkungen bei der Wegsuche zu berücksichtigen. In Kapitel 6 wird daher gezeigt, wie sich Contraction Hierarchies effizient an diese Problemstellung adaptieren lassen. Weitere Benchmarks dienen dabei der Überprüfung der theoretischen Überlegungen.

Unterstützt durch die schnellen Wegsucheverfahren wird in Kapitel 7 eine Heuristik zur Entscheidungsunterstützung bei der Disposition bei einem Transportdienstleister wie IN tIME entwickelt. Das Potential dieses Ansatzes wird anhand einer empirischen Studie erörtert, deren Grundlage die vom Unternehmen zur Verfügung gestellten Vergangenheitsdaten bilden. Der Aufbau der Arbeit ist als Übersicht in Abbildung 1.2, S. 28, dargestellt.

[5] Eine Ausnahme bilden angenommene Aufträge, die nicht kostendeckend bearbeitet werden. Doch auch diese können je nach Unternehmensstrategie sinnvoll sein, beispielsweise um Kunden zu binden.

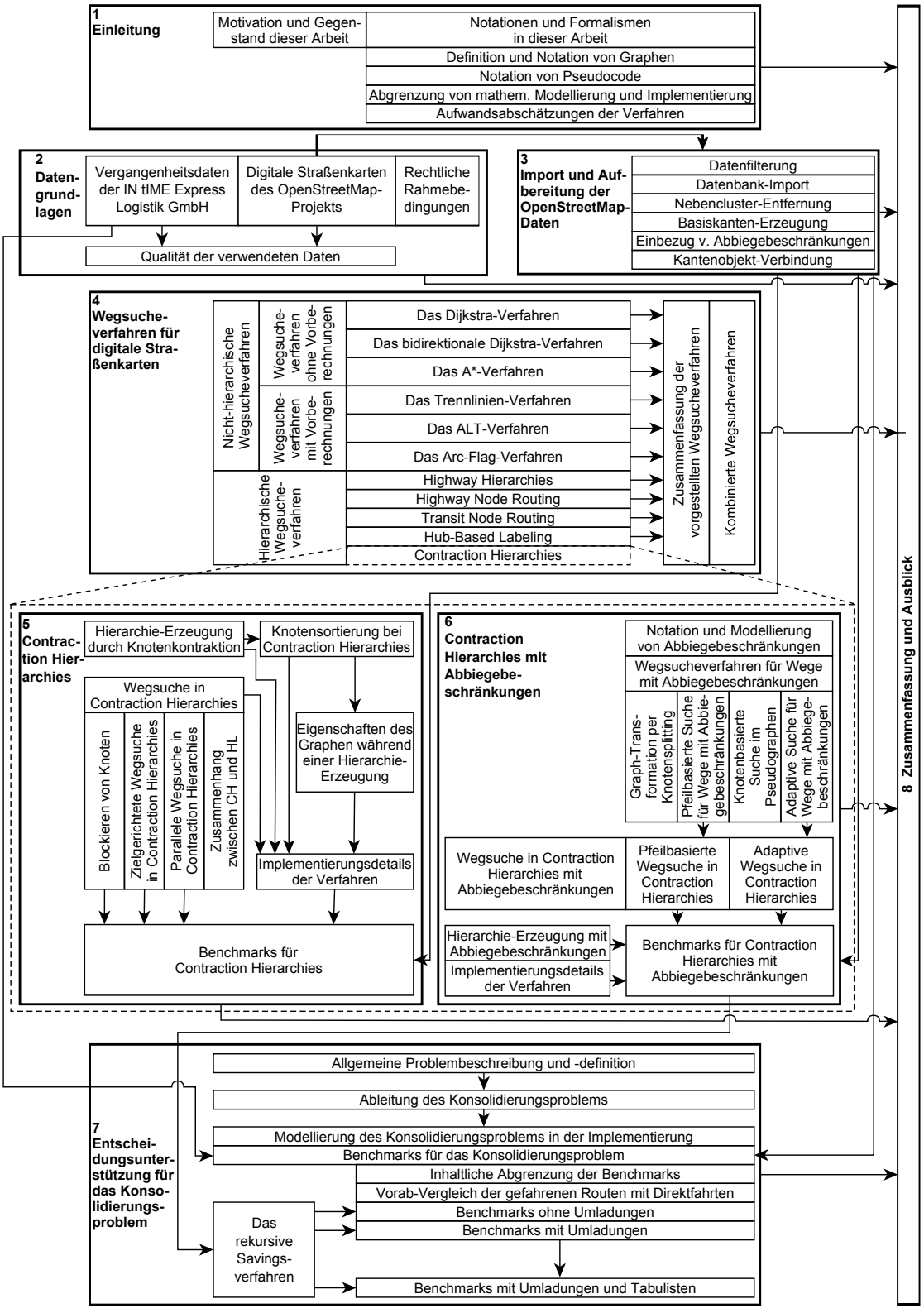


Abbildung 1.2: Aufbau der Arbeit
Pfeile symbolisieren gegenseitige Einflüsse

1.3 Notationen und Formalismen in dieser Arbeit

Nachfolgend werden die Notationen vorgestellt, die in dieser Arbeit Verwendung finden. Da die hier untersuchten Wegsucheverfahren mit *Graphen* arbeiten, wird zunächst deren Aufbau kurz erläutert. Daran schließt sich eine kurze Einführung des hier verwendeten *Pseudocodes* an und es wird der Unterschied zwischen mathematischer Darstellung und Implementierung herausgearbeitet. Das Kapitel endet mit der Darstellung von Aufwandsabschätzungen, wie sie in dieser Arbeit verwendet werden.

Am Ende dieser Arbeit erfolgt eine empirische Untersuchung zum Konsolidierungspotential bei Transporten eines Logistikdienstleisters. Die genaue Problemstellung und ihre Notation werden gesondert in Abschnitt 7.1, S. 262 ff., formuliert.

1.3.1 Definition und Notation von Graphen

Diese Arbeit orientiert sich in Bezug auf die Notation und Bezeichnung von Graphen und ihren Elementen an der Schreibweise, die auch Domschke u. Drexl [2007, S. 65 ff.] verwenden. Ein *Graph* G besteht aus einer nicht-leeren Menge von *Knoten* V (vom englischen *vertices*), die über *Kanten* $E \subseteq V \times V$ (vom englischen *edges*) miteinander verbunden sind. Ist die Reihenfolge der Knoten einer Kante relevant, ist dies eine *gerichtete Kante* und wird auch *Pfeil* genannt. Die Schreibweise für eine *ungerichtete Kante* zwischen u und v sei $[u, v]$; die für einen *Pfeil* vom Knoten u in Richtung Knoten v sei (u, v). Für den Pfeil (u, v) ist u der *Startknoten* und v sein *Endknoten*.

Ein Graph ist *ungerichtet*, falls er ausschließlich ungerichtete Kanten beinhaltet, und *gerichtet*, falls er ausschließlich Pfeile beinhaltet. Ein ungerichteter Graph wird mit $G = [V, E]$ notiert und ein gerichteter mit $G = (V, E)$. Ungerichtete Graphen lassen sich leicht in gerichtete Graphen überführen, indem jede ungerichtete Kante durch zwei gegensätzlich ausgerichtete Pfeile ersetzt wird. Aus diesem Grund werden in dieser Arbeit bis auf wenige Beispiele nur *gerichtete Graphen* behandelt. Der Unterschied ist jedoch stets anhand der Schreibweise eindeutig.

Existieren in einem Graphen zwei verschiedene Pfeile mit denselben Start- und Zielknoten, werden diese Pfeile *parallel* genannt; ein Pfeil (u, u) mit $u \in V$ ist eine *Schlinge* (vgl. [Domschke u. Drexl, 2007, S. 66]).[6] Besitzt ein Graph weder parallele Pfeile noch Schlingen, wird er als *schlicht* bezeichnet (vgl. ebd.). Ein *Digraph*[7] ist ein gerichteter, schlichter Graph mit endlicher Knotenmenge (vgl. ebd.). Digraph

Ein Knoten $v \in V$ heißt *Nachfolgerknoten* von Knoten u, wenn es einen Pfeil (u, v) gibt, der u mit v verbindet. $\mathcal{N}(u) = \{v \mid (u, v) \in E\}$ beinhaltet die Menge aller Nachfolgerknoten von Knoten u. Analog zur Definition von Nachfolgerknoten werden *Vorgängerknoten* festgelegt. Die Schreibweise ist $\mathcal{V}(u) = \{v \mid (v, u) \in E\}$. Die Vereinigungsmenge aus Nachfolger- und Vorgängerknoten eines Knotens u wird in dieser Arbeit Nachfolgerknoten; Vorgängerknoten; Nachbarknoten

[6] Wie auch bei Domschke u. Drexl [2007, S. 66] soll an dieser Stelle darauf hingewiesen werden, dass diese Notation streng formal nicht ausreicht, um mehrere parallele Pfeile zwischen denselben Knoten zu unterscheiden. Im Rahmen dieser Arbeit reicht diese vereinfachte Darstellung jedoch aus. Parallele Pfeile werden zunächst durch die Konstruktion der Graphen ausgeschlossen (Kapitel 3–5). Sie treten erst mit der Betrachtung von Abbiegebeschränkungen in Kapitel 6 auf. Dort wird jedoch eine Notation eingeführt, die Mehrdeutigkeiten ausschließt.

[7] vom englischen *directed graph*

Nachbarknoten genannt und mit $\mathcal{NB}(u) = \mathcal{N}(u) \cup \mathcal{V}(u)$ notiert. (vgl. [Domschke, 1997, S. 2])

Knotengrad

Der *Knotengrad* $g_u = |\mathcal{NB}(u)|$ eines Knotens $u \in V$ gibt an, mit wie vielen Pfeilen u verbunden ist. In gerichteten Graphen kann der Knotengrad aufgeteilt werden in die beiden Komponenten *positiver Knotengrad* $g_u^+ = |\mathcal{N}(u)|$ und *negativer Knotengrad* $g_u^- = |\mathcal{V}(u)|$. (vgl. ebd.)

Darstellung von Wegen

Wege bzw. *Pfade*[8] in Graphen werden meist als geordnete Folge von Pfeilen dargestellt. Hierbei müssen die beinhalteten Pfeile paarweise durch einen Knoten verbunden sein, so dass der Endknotens eines Pfeils der Startknoten des nächsten Pfeils des Weges ist.

Ein Weg kann darüber hinaus auch als Pfeilmenge interpretiert werden. Dies ermöglicht eine Schreibweise, wie sie weiter unten in Gleichung (1.1) verwendet wird. In diesem Sinne beschreibt $|p|$ die Anzahl der Pfeile, aus denen der Weg p besteht.

Eine allgemeine Notation für einen Weg p in $G = (V, E)$ lautet zusammenfassend: $p = \left\langle (u_0, u_1), (u_1, u_2), \ldots, (u_{|p|-1}, u_{|p|}) \right\rangle$ mit $(u_i, u_{i+1}) \in E$ für $0 \leq i < |p|$. (vgl. [Domschke u. Drexl, 2007, S. 67])

Alternativ ist eine Notation für Wege außerdem auch als Knotenfolge möglich, so dass sich das vorangegangene Beispiel wie folgt darstellen ließe: $p = \left\langle u_0, u_1, \ldots, u_{|p|} \right\rangle$ mit $(u_i, u_{i+1}) \in E$ für $0 \leq i < |p|$ (vgl. ebd.). Existieren im Graphen jedoch Schlingen oder parallele Pfeile, entstehen in beiden Schreibweisen Mehrdeutigkeiten, so dass bei Bedarf ergänzende Informationen mitgeliefert werden müssen, um zu entscheiden, welchen der parallelen Pfeile der Weg beinhaltet.

In den meisten Anwendungsfällen sind die Pfeile eines Graphen *gewichtet* bzw. *bewertet*[9], das heißt, jedem Pfeile wird ein Zahlenwert zugewiesen. Dies können anschauliche Werte wie die Pfeillänge in Metern oder auch abstraktere Größen wie die Kosten sein, die bei Verwendung der Pfeile entstehen. Die Bewertung des Pfeils zwischen Knoten u und v wird üblicherweise mit $c(u, v)$ (verallgemeinernd vom englischen *costs*) notiert und $c : E \to \mathbb{R}$ (vgl. [Domschke u. Drexl, 2007, S. 68]). Da die Bewertungsfunktion c zum Graphen gehört, geht sie in dessen Beschreibung mit ein, so dass $G = (V, E, c)$ einen gewichteten, gerichteten Graphen definiert (vgl. ebd.).

Die *Kosten* $c(p)$ eines Weges p ergeben sich als Summe über alle dessen zugrunde liegende Pfeile (vgl. ebd.):

$$c(p) = \sum_{(u,v) \in p} c(u, v) \tag{1.1}$$

Ein *kürzester Weg* $p_{s,t}^*$ zwischen den Knoten s und t ist ein Weg, der diese Kosten minimiert:

$$c(p_{s,t}^*) = \min \{c(p_{s,t}) \mid p_{s,t} \text{ ist Weg von } s \text{ nach } t\} = \text{dist}(s, t) \tag{1.2}$$

Die Länge eines solchen Weges wird auch mit $\text{dist}(s, t)$ beschrieben.

Aus Gleichung (1.2) wird auch deutlich, warum ein Wegsucheverfahren, dass *kürzeste* Wege liefert, ebenso für die Suche nach *schnellsten* Wegen einsetzbar ist. Anstelle

[8] Die beiden Begriffe werden in dieser Arbeit synonym verwendet.

[9] beide Begriffe werden in der Literatur in diesem Kontext synonym verwendet

der Weglänge als Pfeilbewertung muss lediglich die Fahrtdauer über die Pfeile verwendet werden. Auch Linearfaktorkombinationen beider oder weiterer Funktionen sind so möglich.

Metriken

Oft wird im Zusammenhang mit der Pfeilbewertung auch der Ausdruck *Metrik* verwendet. Bei den hier untersuchten Graphen auf Basis digitaler Straßenkarten stehen zwei wesentliche Bewertungsmöglichkeiten für die Pfeile, d. h. für die Straßenzüge, zur Verfügung. Bei der *Kürzeste-Wege-Metrik* werden die Pfeile eines Graphen gemäß ihrer Länge bewertet. Im Gegensatz dazu spielt bei der *Schnellste-Wege-Metrik* zusätzlich die Geschwindigkeit eine Rolle, mit der ein Fahrzeug die entsprechende Straße befahren kann.

Abbildung 1.3 zeigt ein Beispiel für einen bewerteten Digraphen mit vier Knoten. Für diesen Graph G gilt[10]:

- $G = (V, E, c)$,
- $V = \{v, w, x, y\}$,
- $E = \{(x, w), (w, v), (v, x), (v, y), (y, v)\}$,
- $c(x, w) = 12$,
- $c(w, v) = 4$,
- $c(v, x) = 15$,
- $c(v, y) = 17$,
- $c(y, v) = 13$ und beispielhaft
- $\mathcal{N}(v) = \{x, y\}$.

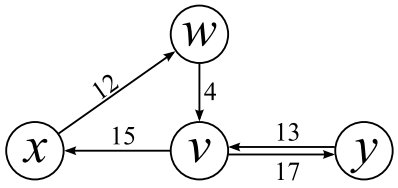


Abbildung 1.3: Beispiel für einen bewerteten Digraphen

Im Beispielgraph existiert nur ein einziger Weg von Knoten x nach Knoten y und das ist $p_{x,y} = \langle x, w, v, y \rangle$. Seine Kosten $c(p_{x,y})$ betragen $12 + 4 + 17 = 33$.

1.3.2 Notation von Pseudocode

In dieser Arbeit ist an mehreren Stellen *Pseudocode* angegeben. Es handelt sich hierbei um eine vereinfachte, abstrahierte Darstellung von Programmiercode. Ihr Zweck ist es, ein Verfahren möglichst präzise zu definieren, ohne dabei jedoch Formulierungen zu gebrauchen, die einer vollständigen Implementierung gleichkommen. Grundlegende Kontrollstrukturen wie Variablen-Zuweisungen und Fallunterscheidungen per *if* und *else* finden sich hier genauso wieder wie *for*- und *while*-Schleifen. Es sind auch *natürlichsprachliche* und *mathematische* Formulierungen vertreten, wenn sie der Lesbarkeit dienen. Kommentare sind ebenfalls für diesen Zweck zulässig.

Da moderne Programmiersprachen weitgehend objektorientiert sind, wie das in dieser Arbeit verwendete *Java*, befinden sich objektorientierte Notationen auch im Pseudocode. In Anlehnung an viele dieser Sprachen[11] wird hier der Punkt-Operator verwendet, um

[10] Hinweis zur Variablenbezeichnung: In Kapitel 7, S. 261 ff., tauchen x und y als Variablen erneut im Zusammenhang mit der Formulierung von Tourenplanungsproblemen auf. Dort sind sie jedoch mit weiteren Indizes versehen, so dass sie in den Beispielen zu den Wegsucheverfahren in der Schreibweise ohne Index stets für Knoten stehen.

[11] wie zum Beispiel C++, Java, Pascal oder Visual Basic

auf Attribute und Methoden eines Objekts zuzugreifen. Die grundlegenden Notationen können dem Beispiel aus Verfahren 1 entnommen werden.

Verfahren 1 Beispiel für Pseudocode

Gegeben: Hier werden die Eingabeparameter gelistet.

1: Dies ist eine Codezeile ▸ Dies ist ein Kommentar
2: $x := 1$ ▸ Variable x wird Wert 1 zugewiesen
3: **while** Bedingung der while-Schleife **do**
4: Schleifenrumpf
5: $einObjekt.counter := einObjekt.counter + 1$ ▸ erhöht Attribut *counter* vom Objekt *einObjekt* um 1
6: **end while**

Ergebnis: Hier werden Ausgabewerte und Endzustände gelistet.

1.3.3 Abgrenzung von mathematischer Modellierung und Implementierung

Neben der Vorstellung ausgewählter Wegsucheverfahren wird im Rahmen dieser Arbeit teilweise auch deren konkrete Implementierung diskutiert. Diese ist von der mathematischen Darstellung deutlich abzugrenzen. Während die mathematische Darstellung Sachverhalte und insbesondere *Ergebnisse* formuliert, ist der Zweck der Implementierung diese Ergebnisse möglichst *effizient zu erreichen*.

Die Anwendung der Verfahren auf digitale Straßenkarten macht es erforderlich, aus den Straßenkarten gewichtete, gerichtete Graphen zu interpretieren. Die Modellierung dieser Graphen erfolgt bei der Implementierung objektorientiert, d. h., Knoten, ungerichtete Kanten sowie Pfeile liegen als *Objekte* vor. Beide kommen innerhalb von digitalen Straßenkarten aufgrund von Einbahnstraßen und in beide Richtungen befahrbare Straßen vor.

Da gerichtete Graphen gemäß Definition[12] keine ungerichteten Kanten besitzen, werden in der *mathematischen Darstellung* sämtliche Straßenzüge, die in beide Richtungen befahrbar sind, als zwei gegensätzlich ausgerichtete Pfeile interpretiert. Im Gegensatz dazu arbeitet die *Implementierung* mit einem Modell, das sowohl ungerichtete Kanten als auch Pfeile erlaubt.[13] Dort existiert eine vereinheitlichte Sicht auf ungerichtete Kanten und Pfeile. Im UML-Klassendiagramm in Abbildung 1.4, S. 33, ist dieser Zusammenhang durch das Interface namens `IKante` ausgedrückt.

[12] siehe Seite 29

[13] In der Literatur finden sich zwar auch Notationen für Graphen, in denen gleichzeitig ungerichtete Kanten und Pfeile existieren. Stiege [2006] spricht in diesem Zusammenhang beispielsweise von „allgemeinen Graphen“ und Domschke [1997, S. 177] von „gemischten Graphen“. Doch auf diese eher unübliche Betrachtungsweise wird in dieser Arbeit o. B. d. A. zugunsten der weiter verbreiteten Darstellung verzichtet.

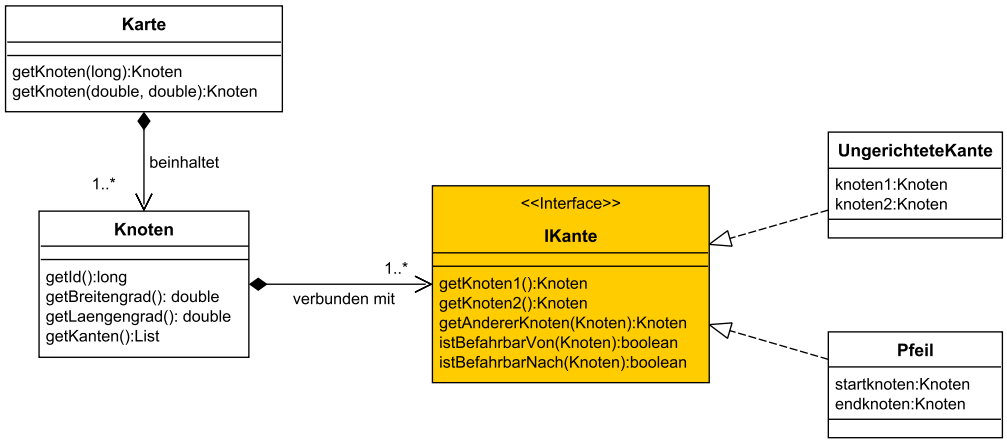


Abbildung 1.4: Einfaches UML-Klassendiagramm für eine digitale Straßenkarte

Aus Perspektive der Implementierung sind – im Gegensatz zur mathematischen Darstellung – Knotenobjekte grundsätzlich mit Objekten verbunden, welche das Interface `IKante` implementieren. D. h., der Zugriff auf ungerichtete Kanten als auch auf Pfeile erfolgt ausschließlich über das Interface. Dies führt zu einem geringeren Speicherbedarf, da für ungerichtete Kanten nur ein Objekt vom Typ `UngerichteteKante` gespeichert werden muss und nicht zwei Objekte vom Typ `Pfeil`. Gleichzeitig bleibt der Quellcode lesbar, da keine umständlichen Typunterscheidungen zwischen ungerichteten Kanten und Pfeilen notwendig sind. Beide lassen sich mit den beiden Methoden `istBefahrbarVon(Knoten)` und `istBefahrbarNach(Knoten)` wie Pfeile ansprechen.

Die tatsächlichen Implementierungen dieser beiden Methoden ist trivial für die Klasse `UngerichteteKante`, da in jedem Fall `wahr` zurückgeliefert wird. Bei Objekten vom Typ `Pfeil` ist das Ergebnis lediglich davon abhängig, ob die zugelassene Fahrtrichtung der abgefragten Richtung entspricht.

Ungerichtete Kanten, Pfeile und Kantenobjekte

Um zwischen der Implementierung und der mathematischen Beschreibung zu unterscheiden, sind die Begriffe daher sprachlich voneinander zu trennen. Der Begriff *Kantenobjekt* bezieht sich stets auf die konkrete Implementierung und die Begriffe *ungerichtete Kante* und *Pfeil* auf die mathematische Modellierung eines Graphen. Wenn von einem Kantenobjekt gesprochen wird, kann sich dahinter in der Implementierung sowohl eine ungerichtete Kante als auch ein Pfeil verbergen.

1.3.4 Aufwandsabschätzungen der Verfahren

Landau-Notation

In dieser Arbeit wird an vielen Stellen für die demonstrierten Wegsuche- und Optimierungsverfahren eine Abschätzung des Laufzeitverhaltens nach der in der Informatik weit verbreiteten *Landau-Notation* angegeben. Die Notation ist auf Bachmann [1894, S. 401] zurückzuführen, der mit ihr Größenordnungen von mathematischen Funktionen im Allgemeinen beschreibt. In der Informatik findet sie spätestens seit der Verwendung bei Knuth [1976, S. 1] und den von ihm dort vorgeschlagenen Erweiterungen große Verbreitung. $\mathcal{O}(f(n))$ steht hier für eine obere Schranke des Laufzeitverhaltens eines Algorithmus'; es ist daher eine *worst-case*-Betrachtung (vgl. ebd.).

Sei n die variable Eingangsgröße eines Verfahrens, beispielsweise die Anzahl der Ele-

mente einer zu sortierenden Liste, und μ ein konstanter Faktor in Zeiteinheiten. Besitzt das Verfahren dann eine Laufzeitabschätzung von $\mathcal{O}(f(n))$, bedeutet dies, dass dessen *tatsächliche* Laufzeit ab einer unbestimmten Größe von n aufwärts stets unterhalb des Werts von $\mu \times f(n)$ bleibt. Dadurch, dass der Schwellwert für n nicht näher spezifiziert wird, ermöglicht die Landau-Notation lediglich eine Aussage über *asymptotisches* Laufzeitverhalten. Für kleinere n wird explizit keine Aussage getroffen.

2 Datengrundlagen dieser Arbeit

Inhalt

In den folgenden Abschnitten werden die Daten vorgestellt, auf denen diese Arbeit beruht. Beginnend mit den Vergangenheitsdaten der IN tIME Express Logistik GmbH im nachfolgenden Abschnitt und den digitalen Straßenkarten des OpenStreetMap-Projekts in Abschnitt 2.2, werden rechtlich relevante Aspekte der Lenk- und Ruhezeiten in Abschnitt 2.3 beleuchtet. Das Kapitel schließt mit einer Diskussion der Qualität dieser Daten in Abschnitt 2.4.

2.1 Vergangenheitsdaten der IN tIME Express Logistik GmbH

Die IN tIME Express Logistik GmbH ist ein europaweit tätiger Express- und Direkt-Kurierdienst mit Hauptsitz in Hannover. Das 1987 gegründete Unternehmen verfügt über 15 Niederlassungen in Deutschland, fünf weitere in Schweden, Ungarn, Tschechien, Rumänien und Polen sowie sechs Vertretungen bei Partnerunternehmen in Frankreich, Portugal, Großbritannien Italien, Österreich und Spanien[1].

Die Transporte werden im Wesentlichen in der Manier eines so genannten Pickup-And-Delivery-Dienstleisters durchgeführt: Die Sendungen werden zunächst an einem Ort abgeholt und dann – grundsätzlich direkt – zu einem anderen Ort transportiert. Dabei greift das Unternehmen auf eine heterogene Fahrzeugflotte aus sowohl eigenen

[1] vgl. [IN tIME Express Logistik GmbH, 2013]

Fahrzeugen als auch Fahrzeugen von Subunternehmern zurück. Neben einigen eigenen hauptberuflichen Fahrern werden auch die zuletzt genannten Subunternehmer eingesetzt, die ihrerseits häufig Fahrer stellen.

Für diese Arbeit stellte IN tIME einen Auszug der Unternehmensdatenbank des operativen Geschäfts zur Verfügung, der sich auf die Jahre 2004–2008 inklusive erstreckt. In diesem Zeitraum zählten 17 Filialen in Deutschland zu IN tIME[2] sowie jeweils eine in Swadlincote (England), Györ (Ungarn), Timisoara (Rumänien) und Helsingborg (Schweden). Abbildung 2.1 zeigt ihre Positionen in einer Übersicht.

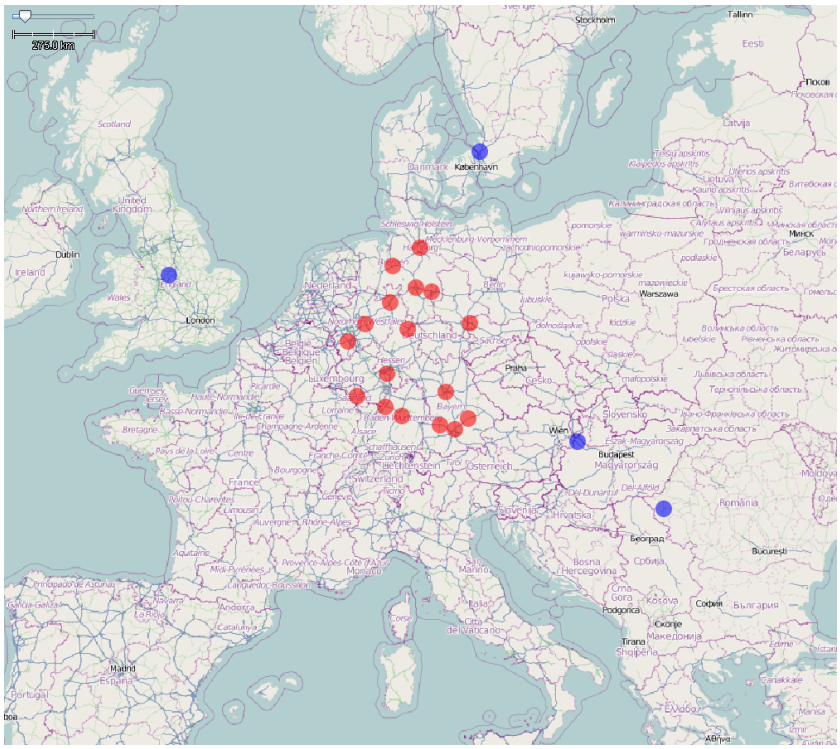


Abbildung 2.1: Verteilung der Niederlassungen von IN tIME
Rot dargestellt sind die 17 Niederlassungen innerhalb Deutschlands; blau diejenigen im Ausland.
Der Maßstab oben links im Bild zeigt eine Länge von 275 Kilometern.

[2] allerdings an teils anderen Orten als heute

Die zur Verfügung gestellten Daten umfassen:

- Auftragsdaten wie
  - die Eingangszeitstempel der Aufträge,
  - die Orte der Be- und Entladungen sowie
  - Sendungsinformationen zur Fracht wie Struktur, Gewicht oder Gefahrenklassifikation,
- Stammdaten zur Fahrzeugflotte und
- Fahrzeugortungsdaten.

Informationen zu Kundennamen, -adressen und Rechnungsbeträgen sind von den Vergangenheitsdaten aus Datenschutzgründen explizit ausgenommen.

Frachtbriefe Für diese Arbeit am bedeutendsten sind die Vergangenheitsdaten der so genannten *Frachtbriefe*. In der Praxis sind dies standardisierte Begleitdokumente zum Transport, über die eine Abrechnung erfolgen kann. Sie sind in § 408 HGB geregelt und beinhalten insbesondere Namen und Anschriften von Absender, Frachtführer und Empfänger sowie eine Beschreibung der Fracht. Die Daten umfassen insgesamt 893 449 anonymisierte Frachtbriefe, die sich, wie in Tabelle 2.1 dargestellt, auf die einzelnen Jahre verteilen.

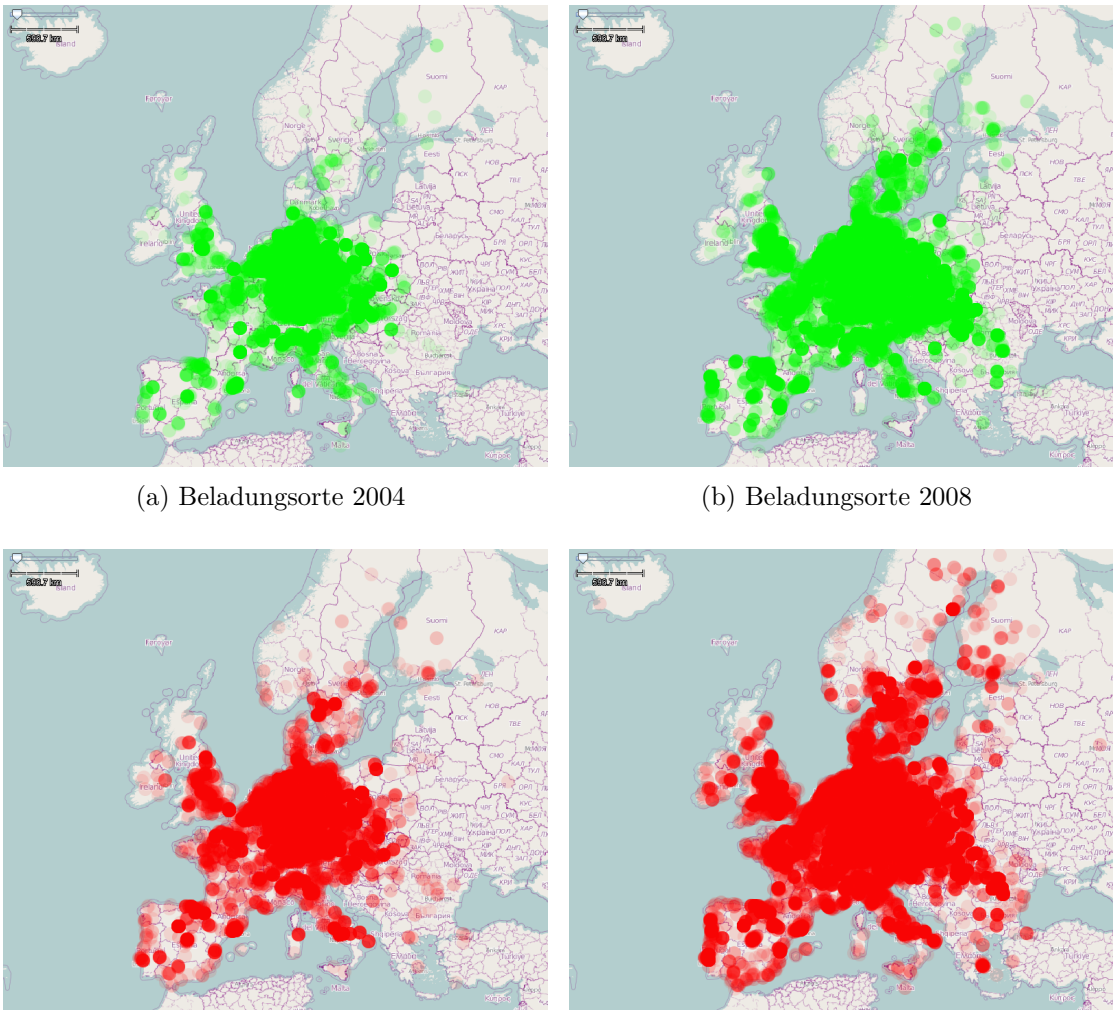
	2004	2005	2006	2007	2008
Frachtbriefe	116 239	136 008	192 281	207 239	241 682
Dokumentierte Frachten	158 505	176 787	225 422	255 365	284 732
Frachten : Frachtbriefe	1,36	1,30	1,17	1,23	1,18
Ortungsdaten	118 418	719 136	1 607 549	7 878 946	8 791 758

Tabelle 2.1: Jahresweise Statistiken über die Vergangenheitsdaten der IN tIME Express Logistik GmbH

Frachten Informationen über die den Frachtbriefen zugehörigen Frachten umfassen deren Verpackungsart, Stückzahl und Bruttogewicht. Insgesamt liegen Informationen zu 1 100 811 Frachten vor, also etwas mehr als eine pro Frachtbrief. Tatsächlich nimmt dieses Verhältnis im zeitlichen Verlauf innerhalb der Vergangenheitsdaten tendenziell ab, was entweder an einer weniger detaillierten Dokumentation der Frachten oder aber einer zunehmenden Homogenität derselben liegen kann (siehe Zeile *Frachten : Frachtbriefe* in Tabelle 2.1).

Ortungsdaten Die mit Abstand größte Datenmenge stellen die Ortungsdaten dar. Die 19 115 807 Ortungen wurden größtenteils automatisiert erzeugt mithilfe der in den Fahrzeugen installierten, auf dem Global Positioning System (GPS) basierenden Ortungsgeräte. Zusätzliche Einträge wurden ebenso automatisch generiert, wenn sich die Fahrer gemäß einer IN tIME-Richtlinie telefonisch meldeten, sobald sie einen Auftrag aufgeladen bzw. abgeliefert hatten. Deutlich erkennbar in Tabelle 2.1 ist die stark zunehmende Verwendung der automatisierten Ortung seit deren Einführung 2004.

Die vom Unternehmen zur Verfügung gestellten Vergangenheitsdaten stammen aus einer Phase, in der IN tIME europaweit erfolgreich expandieren konnte. Abbildung 2.2 stellt exemplarisch hierfür die Beladungs- bzw. Entladungs-Orte aus 2004 denjenigen aus 2008 gegenüber. Deutlich erkennbar ist etwa die Zunahme der Aufträge von und nach Großbritannien, Schweden, Frankreich und Spanien.



(a) Beladungsorte 2004

(b) Beladungsorte 2008

(c) Entladungsorte 2004

(d) Entladungsorte 2008

Abbildung 2.2: Kundenaufträge bei IN tIME 2004 vs. 2008
grünlich markiert: Beladungsorte der Frachtbriefe
rötlich markiert: Entladungsorte der Frachtbriefe
Der Maßstab im oberen linken Bereich aller vier Teil-Abbildungen zeigt einen Abstand von 596,7 Kilometern.

2.2 Digitale Straßenkarten des OpenStreetMap-Projekts

Die zugrunde liegenden Realdaten im Rahmen dieser Arbeit erfordern zwangsläufig auch die Verwendung von Realstrecken und -distanzen bei den Untersuchungen. Zwar ist zum Zeitpunkt dieser Arbeit eine ganze Reihe von Routingdiensten im Internet verfügbar[3]. Diese liefern Navigationsanweisungen für Wege, Satellitenbilder und so genannte *Points of Interest* (kurz: *POI*), nie jedoch das darunter liegende Kartenmaterial. In den Nutzungsbestimmungen dieser Dienste wird der automatisierte Zugriff sowohl auf das Kartenmaterial als auch auf die so genannten *Bildkacheln*[4] explizit untersagt (vgl. beispielsweise [Google Inc, 2009, Abschnitt 10.6]). Denn die Internetdienste sind in den meisten Fällen selbst nicht Eigentümer der Karten, sondern Lizenznehmer kommerzieller Kartenanbieter.[5] Darüber hinaus ist der automatisierte Zugriff auf diese Dienste ebenfalls untersagt, wodurch sich ihr Einsatz im Rahmen dieser Arbeit erheblich einschränkt.

OpenStreetMap: Die „Wikipedia“ für Straßenkarten

Eine frei verfügbare Alternative stellt das OpenStreetMap-Projekt dar, das sich zum Ziel gesetzt hat, den gesamten Planeten digital zu kartographieren. Gegründet im Juli 2004, basiert es grundsätzlich auf dem gleichen Gedanken von freiem Zugang zu Informationen, der auch die Online-Enzyklopädie Wikipedia[6] erfolgreich gemacht hat, und lebt von der aktiven Beteiligung seiner Nutzer. Jeder Benutzer kann – wie bei der Wikipedia – grundsätzlich alle Daten auslesen, verändern und neue Daten hinzufügen. Im Gegensatz zur Online-Enzyklopädie, die hauptsächlich aus Texten besteht, die auch Laien verfassen können, liegt die Einstiegshürde bei OpenStreetMap (OSM) spürbar höher. Denn um beitragen zu können, muss ein Benutzer sich wahlweise mit dem zugrunde liegenden Datenmodell (siehe auch Abschnitt 2.2.1, S. 41 ff.) oder spezieller Karteneditor-Software vertraut machen. Letztere Variante ist dabei die von den meisten Benutzern bevorzugte. Zur Auswahl stehen hier diverse Editoren; wobei der *Java OpenStreetMap Editor (JOSM)* und *Potlatch* die beliebtesten sind[7]. Während JOSM Kartenausschnitte herunterlädt, offline bearbeitet und die Veränderungen als so genannte „Change Sets“ wieder in die Online-Karten-Datenbank hoch lädt, ermöglicht Potlatch eine sofortige Kartenbearbeitung direkt aus einem Internetbrowser. Abbildung 2.3, S. 40, zeigt einen Screenshot von JOSM und Abbildung 2.4, S. 40, einen Screenshot von Potlatch.

[3] u.a. von der Navteq AG (`www.map24.de`), der Microsoft Corporation (`www.bing.com/maps/`) und allen voran der Google AG (`www.maps.google.de`)

[4] aus Performancegründen werden sämtliche angezeigten Karten und Satellitenbilder aus Einzelteilen, den Bildkacheln, zusammengesetzt, so dass ein Internetbrowser möglichst schnell ein Teilbild anzeigen kann und nicht warten muss, bis der anzuzeigende Kartenausschnitt komplett geladen ist

[5] Die größten beiden Unternehmen dieser Art sind die seit Ende 2007 zur Nokia AG gehörende Navteq AG sowie die TomTom Global Content AG (ehemals bekannt unter dem Namen „Tele Atlas“).

[6] `http://www.wikipedia.org`

[7] vgl. `http://lists.openstreetmap.org/pipermail/dev/2009-September/017195.html`

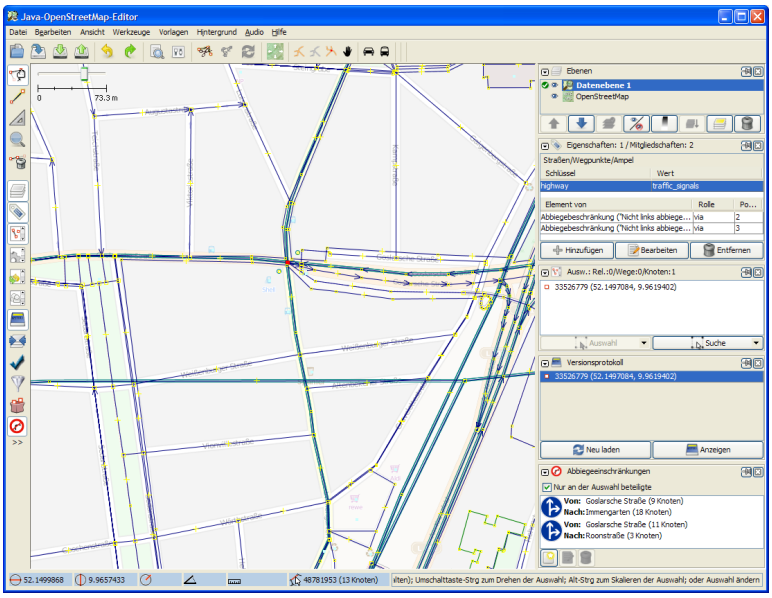


Abbildung 2.3: Screenshot des OSM-Karteneditors JOSM

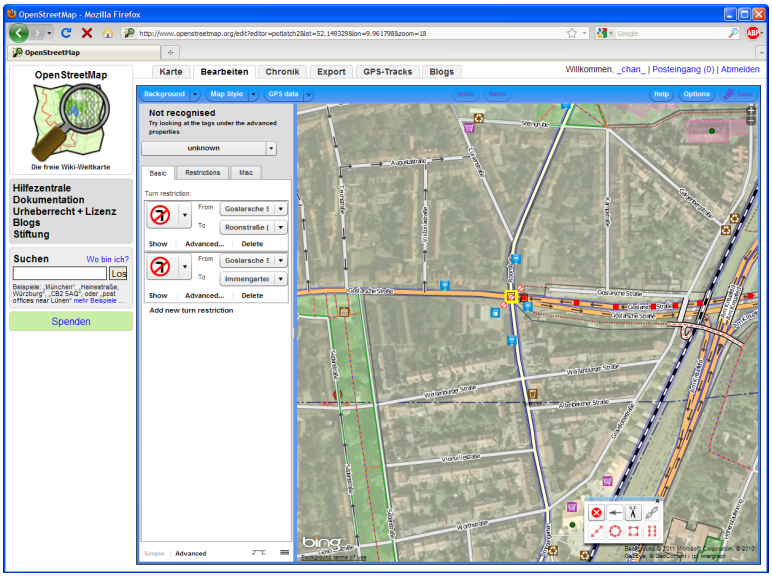


Abbildung 2.4: Screenshot des OSM-Karteneditors Potlatch2

Trotz des bei der Beteiligung anfallenden Mehraufwands hat sich das Projekt mittlerweile etabliert. Anfang 2011 zählte es bereits 362 773 registrierte Benutzer; das Kartenmaterial umfasste zu diesem Zeitpunkt annähernd eine Milliarde Knoten (vgl. [OpenStreetMap contributors, 2011]). Spätestens seit Ende des Jahres 2010, als die Kooperation mit Microsofts Kartendienst *Bing* verkündet wurde, beginnt OSM auch abseits von Privatnutzern Akzeptanz zu finden (vgl. [Kimchi, 2010]).

2.2.1 Einführung in das OpenStreetMap-Datenmodell

Das Datenmodell des OpenStreetMap-Projekts wurde bewusst sehr schlicht entworfen, um auch bei großen Datenmengen noch beherrschbar zu sein. Natürlich lohnt es sich, für eigene Anwendungen, diese Daten individuell zu filtern, zu interpretieren und zu portieren[8]. Das Standard-Modell basiert auf nur drei Bausteinen, die auch *Primitive* genannt werden:

- *Knoten* (im Original: *nodes*),
- *Wegen* (im Original: *ways*) und
- *Relationen* (im Original: *relations*).

Zu jedem der Primitive können beliebig viele so genannte *Tags* hinzugefügt werden, um sie näher zu beschreiben. Im Folgenden werden zunächst die Primitive und anschließend das Konzept von Tags vorgestellt.

Knoten Mithilfe von *Knoten* lassen sich bei OSM auf einer Karte relevante Punkte notieren. Alles, was durch eine Geo-Koordinate, also einem aus Breiten- und Längengrad bestehenden Koordinaten-Paar, lokalisierbar ist, kann als Knoten in eine OSM-Karte eingepflegt werden. Beispiele hierfür sind Straßen-, Fluss-, Grenz- oder sonstige Verläufe und so genannte *Points of Interest* wie Banken, Schulen, Krankenhäuser oder Polizeistationen. Dem Detailgrad sind hier bis auf die Auflösung der Geo-Koordinaten[9] keine Grenzen gesetzt, so dass in Karten durchaus auch Knoten für Windräder, Ampeln, Geldautomaten oder auch Briefkästen oder sogar einzelne Bäume verzeichnet sind. Knoten bilden damit das Fundament für eine OSM-Karte. Ohne sie kann eine Karte keine Informationen beinhalten.

Der Begriff *Knoten* wurde bereits im graphentheoretischen Kontext eingeführt[10]. Um Mehrdeutigkeiten zu vermeiden, wird in dieser Arbeit daher auch der Präfix *OSM* wie im nachfolgenden Satz verwendet.

Ein OSM-Knoten besteht im Minimalfall nur aus Längen- und Breitengrad, einer Versionsnummer, dem Zeitstempel seiner letzten Änderung und einer Identifikationsnummer. Für welches Objekt ein Knoten tatsächlich steht, wird erst durch seine Tags (siehe unten) und/oder seine Verwendung innerhalb von Wegen oder Relationen klar.

Wege Durch Knoten allein können Straßen noch nicht hinreichend beschrieben werden. Das Primitiv *Weg* erlaubt es, mehrere Knoten zu gruppieren und mit einer gemeinsamen Semantik zu versehen. Ohne diese Ordnung wäre es der Karte nicht zu entnehmen, zwischen welchen Knoten beispielsweise eine Straße verläuft. Knoten allein könnten nur angeben, *dass* auf ihnen eine Straße ist.

Wege sind im OSM-Jargon mitnichten ausschließlich mit Straßen, Feldwegen oder Ähnlichem gleichzusetzen; sie sind abstrakter zu verstehen: Wege sind geordnete Aneinanderreihungen von Knoten. Durch Wege lassen sich daher neben Autobahnen, Bundes-,

[8] siehe hierzu auch Kapitel 3, S. 61 ff.

[9] aktuell werden Längen- und Breitengrad mit einer Genauigkeit von sieben Nachkommastellen gespeichert – entsprechend einem größtmöglichen Fehler von etwa einem Zentimeter am Äquator

[10] siehe Abschnitt 1.3.1 auf Seite 29

Land- und sonstigen Straßen auch Bahngleise, Seilbahnen, Flüsse, ober- und unterirdische Stromleitungen, Pipelines, Küstenlinien etc. beschreiben.

Flächen aus Wegen

Eine Besonderheit stellen Wege dar, deren *Start-* und *Endknoten* identisch sind. Mit ihnen können einfache Flächen modelliert werden, sofern dies durch entsprechende Tags (siehe unten) signalisiert wird. Für komplexere Flächen sind jedoch Relationen Wegen vorzuziehen.

Auch der Begriff *Weg* wurde in dieser Arbeit bereits im graphentheoretischen Kontext eingeführt[11]. Wie auch bei *Knoten* werden Mehrdeutigkeiten daher mithilfe des *OSM-*Präfix ausgeräumt.

Relationen *Relationen* sind übergeordnete und abstrakte Primitive. Sie bestehen entweder aus OSM-Knoten, OSM-Wegen, anderen Relationen oder einer beliebigen Kombination dieser drei. Durch eine Relation lässt sich damit beispielsweise elegant eine Landesgrenze darstellen, ohne hierfür einen besonders langen OSM-Weg erzeugen zu müssen. Stattdessen können einfach bereits vorhandene OSM-Wege entlang der Grenze referenziert werden. Nur bei Bedarf – nämlich, falls für einen Teil der Grenze kein Weg existiert – müssen so zusätzliche OSM-Wege angelegt werden. Weitere (Haupt-)Einsatzgebiete für Relationen sind die Modellierung von Flächen und Abbiegeverboten.

Flächen aus Relationen

Flächeninformationen sind besonders für das Rendern von Karten von hoher Bedeutung. Durch sie lassen sich neben Wäldern, Seen und Feldern auch Gebäudeumrisse in die Karte schreiben. Im Vergleich zur Verwendung von Wegen können mithilfe von Relationen weitaus komplexere Flächen erstellt werden. Selbst verschachtelte Flächen, d. h. Flächen mit „Lücken“, wie zum Beispiel ein Wald mit einer Lichtung, lassen sich auf diese Weise modellieren. Im OSM-Jargon werden diese Relationen *Multipolygone* genannt und sind eher für fortgeschrittene Benutzer geeignet.

Rollen

Optional kann jedem Element einer Relation eine Freitext-*Rolle* zugewiesen werden. Relationen für Abbiegeverbote bestehen zumeist aus zwei OSM-Wegen und einem OSM-Knoten: Der Knoten kennzeichnet die Straßenkreuzung, an der das Abbiegeverbot von einem Weg kommend in Richtung des anderen Weges gilt. Die Wege besitzen die Rollen *from* und *to*; der Knoten die Rolle *via*. Der naheliegende Zweck ist der Einsatz in straßengebundenen Navigationsanwendungen.

Tags Alle oben vorgestellten Primitive können durch beliebig viele textliche Attribute – die so genannten *Tags* – genauer definiert werden. Ein einzelner Tag besteht aus einem eindeutigen Schlüssel-Wert-Paar, wobei zu einem Primitiv nicht mehrfach derselbe Schlüssel vergeben werden kann. Zu den häufigsten Schlüsseln gehört „highway“. Hiermit werden in den meisten Fällen Straßen näher spezifiziert. Als Beispiel beschreibt der Schlüssel „highway“ zusammen mit dem Wert „primary“ den am weitesten ausgebauten Straßentyp der Karte. In Deutschland werden die Bundesautobahnen durch diesen Tag gekennzeichnet.

Die übliche Art, eine Einbahnstraße zu kennzeichnen, besteht darin, einen OSM-Weg um den Tag mit dem Schlüssel „oneway“ zu ergänzen. Die häufigsten Werte sind dann „yes“ oder „true“.

Notation für Tags

In dieser Arbeit wird im Folgenden eine durch ein Gleichheitszeichen verkürzte Schreib-

[11] siehe Abschnitt 1.3.1 auf Seite 30

weise für Tags eingeführt. Anstatt Schlüssel „key“ und Wert „value“ wird fortan nur noch „key=value“ geschrieben.

Wie oben erwähnt, können Tags gleichermaßen auch zu Knoten und Relationen hinzugefügt werden. Ein häufiger Tag für Knoten ist beispielsweise „highway=traffic_signals“. Dieser gibt an, dass sich am zugehörigen Knoten eine Ampelanlage befindet.

Die oben beschriebenen Abbiegeverbote enthalten im Regelfall ebenfalls einen Tag, der die übergeordnete Relation als Abbiegeverbot kennzeichnet. Dies geschieht meist entweder durch „restriction=no_left_turn“ oder analog durch „restriction=no_right_turn“. Ein solcher Tag ist dringend notwendig, da ansonsten Mehrdeutigkeiten entstehen könnten. Denn auch Abbiege*gebote* sind in der Karte in Form von Relationen verzeichnet. Sie unterscheiden sich von Abbiege*verboten* lediglich hinsichtlich des beschreibenden Tags: Ein Abbiege*gebot* ist üblicherweise mit dem Tag „restriction=only_left_turn“, „restriction=only_right_turn“ oder aber „restriction=only_straight_on“ markiert.

Offensichtlich ergibt sich hierdurch eine gewisse Redundanz, denn es ist möglich, ein Abbiege*gebot* durch zwei oder je nach Kreuzungsgrad mehr Abbiege*verbote* zu formulieren. In Abbildung 2.5, S. 44, ist eine prototypische Kreuzung dargestellt. Sollte hier von Süden nach Norden fahrend ein Geradeausfahr-Gebot bestehen, müsste hierfür eine Relation erstellt werden, die Weg 1, Weg 3 und den weiß dargestellten *Kreuzungsknoten* 1 beinhaltet. Die notwendigen Tags könnten auf zwei Arten hinterlegt werden; entweder mit einem einzigen Gebots-Tag („restriction=only_straight“) oder mit zwei Verbots-Tags („restriction=no_left_turn“ und „restriction=no_right_turn“).

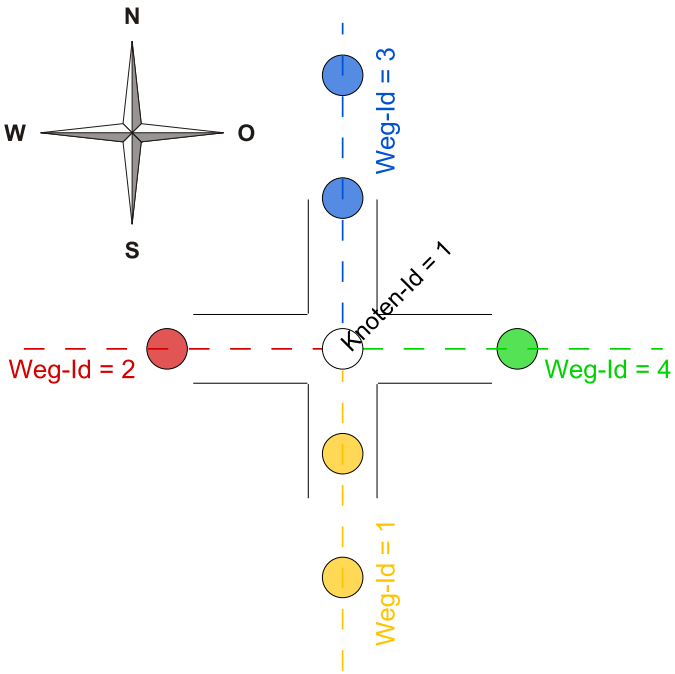


Abbildung 2.5: Beispiel-Szenario für Abbiegebeschränkungen

2.2.2 Arbeiten mit dem OpenStreetMap-Kartenmaterial

Sämtliches Kartenmaterial des Projekts inklusive der Daten-Historie liegt in Form einer zentralen Datenbank vor. Der Zugriff auf die Datenbank erfolgt von außen über eine Programmierschnittstelle oder auch Application Programming Interface (API), die seit April 2009 in der Version 0.6 vorliegt[12]. Obwohl das API sowohl für lesende als auch schreibende Zugriffe verwendet werden kann, ist es hauptsächlich für letztere gedacht. Für rein lesenden Zugriff existiert das so genannte „Extended API“ (kurz *XAPI*)[13]. Das XAPI unterstützt dieselben Abfragen wie das API, bietet darüber hinaus aber noch weitere Nutzungsmöglichkeiten.

Die Kommunikation mit dem (X)API erfolgt per so genanntem *Representational State Transfer*[14], bei dem hier über das Hypertext Transfer Protocol (HTTP) Dokumente per Extensible Markup Language (XML) ausgetauscht werden, deren Aufbau sich im Wesentlichen an den in Abschnitt 2.2.1, S. 41, vorgestellten OSM-Primitiven orientiert.

XML-Codierung der Daten

Für jedes Primitiv existiert ein XML-Element mit den notwendigen zusätzlichen Informationen wie der Identifikations- und der Versionsnummer als XML-Attribut; OSM-Tags (vgl. S. 42) werden als Kinder-XML-Elemente mit exakt zwei XML-Attributen angegeben: Das erste besitzt den XML-Attribut-Namen „k“ und als Wert den Tag-Schlüssel; das zweite besitzt den XML-Attribut-Namen „v“ und als Wert den Tag-Wert. Listing 2.1, S. 45, zeigt ein Beispiel für ein vollständiges solches XML-Dokument, das einen einzigen OSM-Knoten beinhaltet.

[12] Für Details: siehe `http://wiki.openstreetmap.org/w/index.php?title=API_v0.6&oldid=648881`
[13] siehe `http://wiki.openstreetmap.org/w/index.php?title=Xapi&oldid=675755`
[14] Vereinfacht ausgedrückt beschreibt dies eine grundsätzliche Vereinbarung über einen wohldefinierten Informationsaustausch im Internet für Client-Server-Architekturen.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' generator='Osmosis 0.39'>
  <node id='391051182' version='2' lat='52.117198' lon='9.899422'>
    <tag k='highway' v='traffic_signals'/>
    <tag k='beliebiger Schlüssel' v='beliebiger Wert'/>
    ...
  </node>
</osm>
```

Listing 2.1: Beispiel für die XML-Darstellung eines einzelnen OSM-Knotens

Bei den verschachtelten Primitiven Weg und Relation werden die beinhalteten Primitive über ihre Identifikationsnummern referenziert. Listing 2.2 zeigt Ausschnitte für das XML-Dokument, das die Robert-Bosch-Straße in Hildesheim mit der Identifikationsnummer 4915290 beschreibt. Es greift auf den im vorangegangenen Beispiel definierten OSM-Knoten mit der Identifikationsnummer 391051182 zurück. Listing 2.3, S. 46, beschreibt schematisch ein Abbiegeverbot für die in Abbildung 2.5, S. 44, dargestellten OSM-Wege.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' generator='Osmosis 0.39'>
  ...
  <node id='391051182' version='2' lat='52.117198' lon='9.899422'>
    ...
  </node>
  ...
  <way id='4915290' version='27'>
    <nd ref='391051182'/>
    ...
    <nd ref='32176001'/>
    <tag k='ref' v='K 103'/>
    <tag k='highway' v='secondary'/>
    <tag k='name' v='Robert-Bosch-Straße'/>
    <tag k='oneway' v='yes'/>
    <tag k='maxspeed' v='50'/>
  </way>
</osm>
```

Listing 2.2: Beispiel für die XML-Darstellung eines OSM-Weges

Wichtig ist, dass das XML-Dokument *sortiert* vorliegt, das heißt, zuerst werden sämtliche Knoten gelistet, anschließend alle Wege und zuletzt – sofern vorhanden – die Relationen. So ist für ein OSM-XML-Dokument sichergestellt, dass für jedes verschachtelte Primitiv vorab dessen beinhaltete Primitive eingelesen werden können, so dass ein vollständiges Kartengerüst mit einmaligem Lesen des Dokuments, d. h. ohne Rücksprünge beim Lesen, erzeugt werden kann.

Neue Knoten, Wege und Relationen können in Echtzeit über Schnittstellen wie den im vorherigen Abschnitt in Abbildung 2.4, S. 40, vorgestellten Online-Editor Potlatch eingepflegt werden. Der Editor kommuniziert hierbei mit der zentralen Datenbank mithilfe

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' generator='Osmosis 0.39'>
  ...
  (Definition der Knoten und Wege)
  ...
  <relation id='1'>
    <member type='way' ref='3' role='from' />
    <member type='node' ref='1' role='via' />
    <member type='way' ref='4' role='to' />
    <tag k='restriction' v='no_left_turn' />
    <tag k='type' v='restriction' />
  </relation>
</osm>
```

Listing 2.3: Beispiel für die XML-Darstellung eines Abbiegeverbots

der oben genannten API.

Für den Fall, dass ein oder mehrere Primitive gleichzeitig von mehreren Benutzern geändert werden, stellen Editoren wie Potlatch Konflikt-Bereinigungsmechanismen bereit. Das Erkennen derartiger Konflikte erfolgt über transaktionsbasierte Versionsnummern eines jeden OSM-Primitivs: Falls ein Benutzer gleichzeitig mit einem zweiten Benutzer die Position eines Knotens verschiebt und der erste Benutzer die Änderung speichert, erhält dieser Knoten eine neue Versionsnummer. Versucht nun der zweite Benutzer ebenfalls zu speichern, erkennt die Datenbank, dass die Änderung auf Grundlage einer veralteten Versionsnummer des Knotens erfolgen würde und meldet den Konflikt. Der Potlatch-Editor zeigt in derartigen Fällen dem zweiten Benutzer eine Fehlermeldung und bietet an, die in der Zwischenzeit veränderten Primitive neu zu laden.

Das osm-Dateiformat Für den Offline-Gebrauch existiert ein Datenaustausch-Format auf Dateibasis, das durch die Dateiendung „.osm“ gekennzeichnet wird. Es handelt sich hierbei im Wesentlichen um das oben beschriebene XML-Format, in dem ein Kartenausschnitt kodiert ist. Offline-Editoren wie das in Abbildung 2.3, S. 40, gezeigte JOSM erlauben über die API das Herunterladen von Kartenausschnitten in diesem Format, können die Informationen anschließend lokal als .osm-Dateien speichern und darauf aufbauend das Kartenmaterial zum Bearbeiten rendern. Auch Osmosis[15], eines der populärsten Kommandozeilen-Programme zur (Massen-)Bearbeitung von OSM-Daten, beherrscht das .osm-Format. In vielen Fällen ist es sinnvoll, die Größe der .osm-Dateien vor dem Transport mit geeigneten Kompressionsverfahren zu verringern. Da die Dateien XML-Daten in Textform und nicht binär speichern, lassen sie sich besonders gut komprimieren.

Das pbf-Dateiformat Seit April 2010 findet ein weiteres OSM-Dateiformat zunehmend Verbreitung, welches auf den so genannten Protocol Buffer Format (pbf) des Internet-

[15] siehe `http://wiki.openstreetmap.org/w/index.php?title=Osmosis&oldid=652103` für weitere Details

konzerns Google basiert[16]. Die mit der Endung „.pbf“ gekennzeichneten Binärdateien ermöglichen eine erheblich beschleunigte Verarbeitung im Vergleich zu ihren .osm-Pendants. Gleichzeitig benötigen sie nur einen Bruchteil deren Speicherbedarfs. Die auch für Menschen vergleichsweise einfache Lesbarkeit des XML-Formats wurde zugunsten dieser Vorteile freilich aufgegeben. Das oben genannte Osmosis beherrscht mittlerweile ebenfalls diesen „Datei-Dialekt“.

Einen beispielhaften Vergleich des osm-Formats mit dem pbf-Format zeigt Tabelle 2.2. Auf Basis der Deutschland-Karte[17] wurden hier die Dateigrößen sowie die benötigte Dauer für das einmalige Parsen der Dateien gemessen.[18]

	Dateigröße in KB	Dauer für Parsen
.osm-Format	10 854 191	100,0 %
.pbf-Format	636 959	12,4 %

Tabelle 2.2: Vergleich der beiden bei OSM am weitesten verbreiteten Dateiformate für eine Deutschlandkarte

Der *Planet File* OSM bietet Kartenmaterial für den gesamten Globus. Es existieren verschiedene Möglichkeiten für den Zugriff auf diese Daten. Benutzer, die über ausreichende Speicher- und Rechnerkapazität verfügen, können sich den so genannten *Planet-File* aus dem Internet herunterladen.[19] Der Download umfasst sämtliches aktuelles Kartenmaterial. Am 1. September 2011 bestand er aus einer mit bzip2[20] komprimierten osm-Datei und besaß eine Größe von über 18 Gigabyte (GB).[21]

Filtern per Osmosis

Kartenextrakte können von dieser Datei anhand des bereits erwähnten Programms *Osmosis* erstellt werden. Hierfür ist die Angabe einer Bounding Box möglich, welche die Randpunkte des gewünschten Bereichs in Form von Längen- und Breitengraden definiert. Das Programm bietet darüber hinaus auch die Verwendung eines in analoger Art definierten Polygons an.

Weiterhin können die Kartendaten mithilfe von Osmosis – und vielen weiteren frei verfügbaren Programmen – auf mehrerlei Arten gefiltert werden. In den meisten Fällen ist dies sinnvoll, um die Dateigröße weiter zu reduzieren und für den Einsatzzweck unnötige Informationen – wie beispielsweise die in Abschnitt 2.2.1, S. 42, erwähnten Stromleitun-

[16] zur Ankündigung des Formats in der OSM-Entwickler-Mailingliste: siehe `http://lists.openstreetmap.org/pipermail/dev/2010-April/019303.html`; Protocol Buffers: siehe `http://code.google.com/p/protobuf/`

[17] Kartendownload vom 30. September 2010

[18] Verwendet wurde die Version 0.39 von Osmosis; Befehl für die Zeitmessung des Parsens:
`osmosis -read-xml germany_20100930.osm -write-null` bzw.
`osmosis -read-pbf germany_20100930.osm.pbf -write-null`.

[19] Zum Beispiel bei `http://planet.openstreetmap.org/` oder bei `ftp://ftp.spline.de/pub/openstreetmap/`; im Projekt-Wiki unter `http://wiki.openstreetmap.org/wiki/Planet.osm` werden darüber hinaus noch weitere (inklusive experimenteller Bittorrent-Distributionen) genannt.

[20] `www.bzip.org`

[21] Trotz der Vorzüge des pbf-Formats finden sich immer noch viele Anbieter von Kartendownloads, die nach wie vor das mit bzip2 komprimierte osm-Format anbieten. Der Grund liegt vermutlich im verhältnismäßig jungen Alter des pbf-Formats, so dass viele Tools möglicherweise noch nicht mit ihm kompatibel sind und Entwickler die Etablierung des Formats noch abwarten wollen.

gen – herauszufiltern. Die Details zur Benutzung des Programms sind dessen Eintrag im OSM-Wiki zu entnehmen[22].

Datenextrakte aus der OSM-Karte Glücklicherweise ist kein Benutzer gezwungen, auf den Planet File zurückzugreifen, um mit OSM-Karten zu arbeiten. Verschiedene Internetseiten bieten als Download auch abgegrenzte Regionen wie Kontinente, Länder und, besonders für Deutschland und die USA, auch Bundesländer bzw. -staaten an.[23] Für besonders kleine Regionen eignet sich ebenfalls ein Download per API, beispielsweise mithilfe von JOSM, das es dem Benutzer erlaubt, per Maus ein rechteckiges Gebiet der Karte auszuwählen und herunterzuladen. Das Programm nimmt dafür via API Kontakt mit dem zentralen Server auf. Bei zu detailreichen Kartenausschnitten verweigert dieser den Download allerdings. Daher richtet sich JOSM an Benutzer, die am zielgerichteten Editieren eines kleinen Kartenausschnitts interessiert sind. Abbildung 2.6 zeigt einen Screenshot der Auswahl- und Download-Funktion.

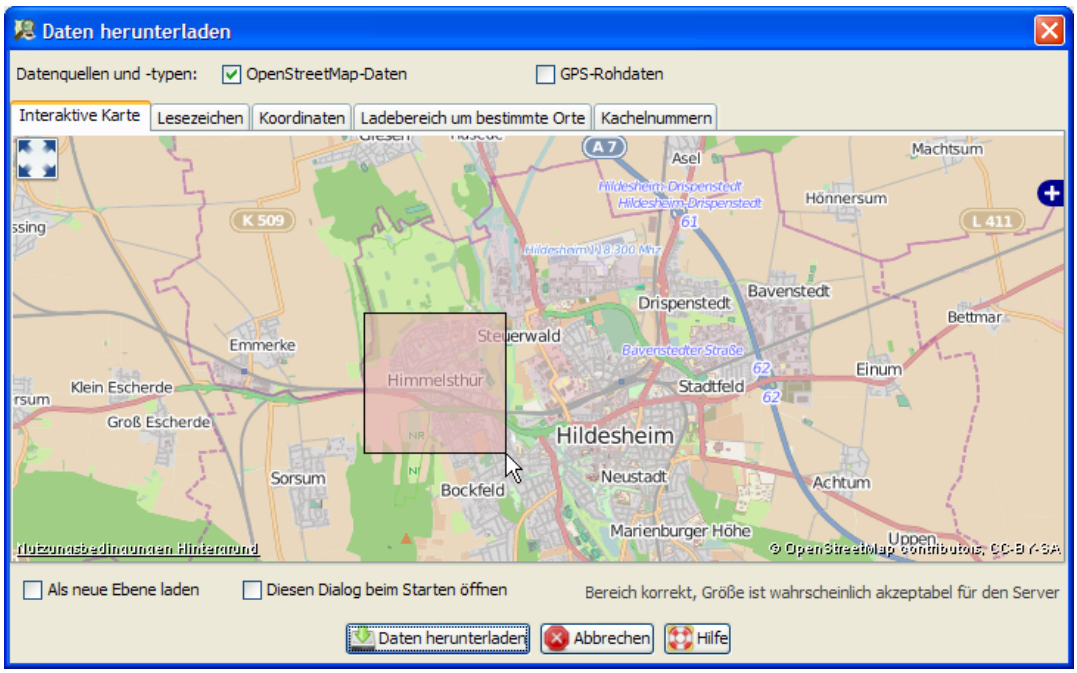


Abbildung 2.6: Screenshot der Direkt-Download-Funktion bei JOSM

[22] `http://wiki.openstreetmap.org/w/index.php?title=Osmosis&oldid=682437`

[23] Als prominente Beispiele seien hier `http://download.geofabrik.de/osm/` und `http://downloads.cloudmade.com/` genannt.

2.3 Rechtliche Rahmenbedingungen

Im Kontext von straßengebundenen Transporten sind eine ganze Reihe gesetzlicher Vorgaben zu berücksichtigen, deren vornehmlicher Zweck im Arbeitnehmerschutz und in der Verkehrssicherheit besteht. Unterschieden wird hier zwischen den allgemein für Arbeitnehmer geltenden *Arbeitszeiten* und den speziell auf die Gruppe der angestellten Fahrer ausgerichteten *Lenk- und Ruhezeiten*. Der Fokus liegt im Folgenden auf den für deutsche Unternehmen relevanten Regelungen.

Seit dem 11. April 2007 gilt hier insbesondere die Verordnung (EG) Nr. 561/2006 des Europäischen Parlaments und des Rates vom 15. März 2006 zur Harmonisierung bestimmter Sozialvorschriften im Straßenverkehr und zur Änderung der Verordnungen (EWG) Nr. 3821/85 und (EG) Nr. 2135/98 des Rates sowie zur Aufhebung der Verordnung (EWG) Nr. 3820/85 des Rates (VO (EG) 561/2006) (vgl. [Europäisches Parlament, 2006]). Sie regelt die Transporte mit Fahrzeugen, deren zulässiges Gesamtgewicht inklusive eines Anhängers 3,5 Tonnen übersteigt. Hervorzuheben ist, dass die tatsächliche Beladung dabei unerheblich ist. Durch die Fahrpersonalverordnung (FPersV) gelten in Deutschland nahezu identische Regelungen für Fahrzeuge mit einem zulässigen Gesamtgewicht ab 2,8 Tonnen (vgl. [FPersV, 2005]). Damit ist die überwiegende Mehrheit der für gewerbliche Transporte verwendbaren Fahrzeuge erfasst.[24] Ergänzend ist das Arbeitszeitgesetz (ArbZG) – insbesondere § 21a (Beschäftigung im Straßentransport) – zu berücksichtigen (vgl. [ArbZG, 2012]).

VO (EG) 561/2006

Die Vorgaben für Lenk- und Ruhezeiten sind komplex und zum großen Teil miteinander verschachtelt. Es folgt ein Überblick über die wichtigsten Regelungen der Lenk- und Ruhezeiten.[25]

2.3.1 Zeitarten bei Lenk- und Ruhezeiten

Die VO (EG) 561/2006 unterscheidet zwischen einer Reihe klar definierter Zeitarten für Fahrer. Sowohl Arbeits- als auch Ruhezeiten werden dabei noch genauer unterteilt. Die Ruhezeiten bestehen aus *täglichen* und *wöchentlichen Ruhezeiten*; die *Arbeitszeit* ist unterteilt in *Lenkzeit* und sonstige *Arbeitszeit*. Als drittes existiert die Kategorie der *Bereitschaftszeit* nach § 21a ArbZG, die explizit weder als *Arbeitszeit* noch als *Ruhepause* zu bewerten ist.

Die Taxonomie der verschiedenen Zeitarten wird in Abbildung 2.7, S. 50, verdeutlicht. Die Darstellung lässt sich von oben nach unten betrachtet als immer feinere Spezialisierung interpretieren: *Bereitschaftszeit* ist eine spezielle *Zeitart*; *Lenkzeit* ist eine spezielle Art der *Arbeitszeit* und *sonstige Arbeitszeit* ist die andere. Jede *Ruhezeit* ist gleichzeitig auch eine *Ruhepause*; jede *tägliche Ruhezeit* gilt sowohl als eine *Ruhepause* als auch als *Ruhezeit*.

[24] Abschließende Auflistungen von Ausnahmen sind in Artikel 3 VO (EG) 561/2006 respektive § 18 FPersV aufgeführt.

[25] Eine gute Einführung in die Thematik findet sich bei Leuker u. Daniel [2007].

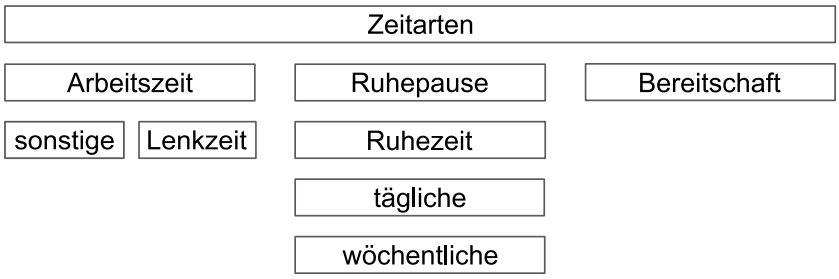


Abbildung 2.7: Taxonomie der Zeitarten

Etwas kontraintuitiv ist die Tatsache, dass eine *wöchentliche Ruhezeit* im Sinne der Verordnung eine spezielle Form der *täglichen Ruhezeit* darstellt. Denn anstelle einer *täglichen* kann ein Fahrer gemäß der Verordnung auch eine *wöchentliche Ruhezeit* nehmen. Da Arbeitnehmer geschützt werden sollen, sind längere Pausen stets zulässig. Es folgt eine Beschreibung der wichtigsten Zeitarten in Bezug auf angestellte Fahrer.

Arbeitszeiten Die Arbeitszeit darf gemäß ArbZG grundsätzlich weder 6 Stunden ohne Pause noch 48 Stunden pro Woche durchschnittlich überschreiten. Doch das ArbZG nennt für Beschäftigte im Straßentransport gewisse Ausnahmen und verweist dabei auf die VO (EG) 561/2006[26]. Für die *Lenkzeit* sind in der EU-Verordnung vier Größenordnungen vorgegeben: Die maximale ununterbrochene *Lenkdauer* beträgt 4,5 Stunden. Spätestens dann ist eine vollständige Ruhepause einzuhalten (s. u.). Die *tägliche Lenkzeit* darf grundsätzlich 9 Stunden nicht überschreiten. Ihr Bezug ist nicht der Kalendertag, sondern der Zeitraum zwischen zwei Ruhezeiten (s. u.). Die *wöchentliche Lenkzeit* ist bezogen auf eine Kalenderwoche und beträgt maximal 56 Stunden. Innerhalb zweier aufeinander folgenden Kalenderwochen liegt das Maximum bei 90 Stunden.

Ruhezeiten Als Ruhezeiten gelten Zeiträume, in denen ein Fahrer frei über seine Zeit verfügen kann. Dieser Aspekt grenzt sie von der *Bereitschaftszeit* ab. Es wird dabei zwischen *wöchentlicher* und *täglicher* Ruhezeit unterschieden. Tägliche Ruhezeiten dauern grundsätzlich mindestens 11 Stunden und bilden das Gerüst für die oben genannten täglichen Lenkzeiten: Einerseits beginnt nach jeder vollständigen Ruhezeit eine neue tägliche Lenkzeit. Andererseits muss innerhalb von 24 Stunden nach Ende einer Ruhezeit wiederum die nächste tägliche Ruhezeit bereits vollendet sein.

Spätestens nach sechs dieser 24-Stunden-Zeiträume muss eine *wöchentliche Ruhezeit* erfolgen. Diese umfasst im Regelfall einen Zeitraum von mindestens 45 Stunden. Während in der Praxis zwar zumeist darauf geachtet wird, wöchentliche Ruhezeiten auf das Ende einer Kalenderwoche zu legen, ist dies jedoch nicht zwingend notwendig. Wie bereits erwähnt, können tägliche Ruhezeiten stets durch wöchentliche ersetzt werden (siehe Abbildung 2.7).

Ruhepausen Das ArbZG schreibt Ruhepausen von mindestens 30 Minuten nach einer Arbeitszeit von 6–9 Stunden vor. Übersteigt die Arbeitszeit 9 Stunden, ist die Ruhepause

[26] siehe § 21a ArbZG

auf mindestens 45 Minuten zu verlängern. Ruhepausen können in Zeitabschnitte von jeweils mindestens 15 Minuten aufgeteilt werden.

Für Fahrer gelten ergänzende Bestimmungen aus VO (EG) 561/2006: Eine Ruhepause trennt im Normalfall zwei ununterbrochene Lenkzeiten von je 4,5 Stunden und dauert mindestens 45 Minuten. Wie in Abbildung 2.7, S. 50, dargestellt, kann eine Ruhepause stets durch eine Ruhezeit ersetzt werden.

2.3.2 Sonderfälle der Zeitarten

Die *allgemeine Arbeitszeit* kann auf bis zu 60 Stunden verlängert werden, sofern innerhalb von vier Kalendermonaten *oder* 16 Wochen im Durchschnitt 48 Stunden wöchentlich nicht überschritten werden.

Die *tägliche Lenkzeit* kann auf maximal 10 Stunden erhöht werden. Von dieser Möglichkeit darf jedoch zwischen je zwei *wöchentlichen Ruhezeiten* nur bis zu dreimal Gebrauch gemacht werden. Bei genauerer Betrachtung zeigt sich die Überschneidung dieser Forderung mit der Obergrenze für die Lenkzeit einer Kalenderwoche: Wird die Lenkzeit zwischen zwei wöchentlichen Ruhezeiten vollends ausgeschöpft, ergibt dies 2×10 Stunden $+ 4 \times 9$ Stunden $= 56$ Stunden.

Komplementär zur Verlängerung der *Arbeitszeiten* besteht der Sonderfall bei den *Ruhezeiten* aus einer Verkürzung. So ist es möglich, die *tägliche Ruhezeit* auf bis zu 9 Stunden zu reduzieren. Es sind allerdings höchstens drei verkürzte tägliche Ruhezeiten zwischen je zwei wöchentlichen Ruhezeiten zulässig. Da für eine derartige Verkürzung kein Ausgleich erforderlich ist, beschreibt diese Ausnahme nach Erfahrung des Autors in der Praxis den Normalfall. Zu beachten ist, dass bei dieser Regelung allein derjenige Teil einer täglichen Ruhezeit gilt, der sich im 24-Stunden-Zeitfenster nach der direkt vorangegangenen vollständigen Ruhezeit befindet. Abbildung 2.8 zeigt ein Beispiel für eine tägliche Ruhezeit, die sich zwar über 12 Stunden erstreckt, jedoch als *verkürzt* zu bewerten ist.

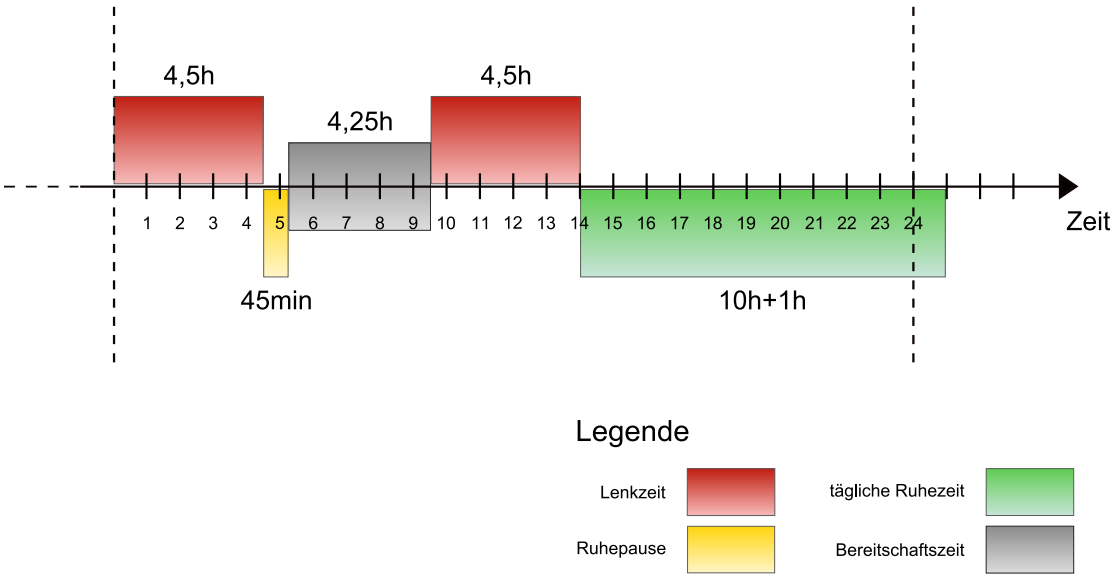


Abbildung 2.8: Beispiel für verkürzte tägliche Ruhezeit

Weiterhin ist es möglich, eine tägliche Ruhezeit in zwei Abschnitte aufzuteilen, ohne dass sie als verkürzt gilt. Hierbei muss der erste Abschnitt eine Länge von mindestens 3 und der zweite von mindestens 9 Stunden besitzen. Diese Option wird in der Praxis eher selten wahrgenommen. Sie ist bestenfalls geeignet, um kurzfristig auf Verkehrsstörungen zu reagieren. So könnte sich ein Fahrer bei Kenntnis eines Staus auf seiner Strecke dazu entschließen, auf dem nächstgelegenen Rasthof 3 Stunden zu ruhen, in der Hoffnung, dass sich die Verkehrslage danach entspannt haben wird.

Wöchentliche Ruhezeiten können bei Bedarf ebenfalls verkürzt werden. Sie dürfen jedoch 24 Stunden nicht unterschreiten und jede Verkürzung muss ausgeglichen werden. Der Ausgleich erfolgt durch eine Ruhepause, die ohne Unterbrechung noch vor Ende der dritten Kalenderwoche nach der *betreffenden* Kalenderwoche genommen werden muss. Diese Ruhepause muss mindestens der Dauer entsprechen, um die verkürzt wurde, und zusätzlich ist sie an eine mindestens 9-stündige Ruhezeit anzuhängen. Ferner müssen innerhalb zweier direkt aufeinander folgenden Kalenderwochen zumindest entweder zwei vollständige oder aber eine vollständige und eine verkürzte wöchentliche Ruhezeit eingehalten werden. Für letztere muss wie beschrieben ein Ausgleich erfolgen. Wöchentliche Ruhezeiten, die sich über das Ende einer Kalenderwoche erstrecken, zählen dabei wahlweise für jene oder die darauf folgende Kalenderwoche, nicht jedoch für beide.

In Abbildung 2.9 ist ein Beispiel für einen Ausgleich dargestellt. Die erste wöchentliche Ruhezeit ist hier um 15 Stunden verkürzt; die vierte tägliche Ruhezeit gleicht sie aus. Hervorzuheben ist, dass in diesem Beispiel das Maximum von drei verkürzten täglichen Ruhezeiten zwischen zwei wöchentlichen Ruhezeiten voll ausgeschöpft wurde. Denn die vierte tägliche Ruhezeit erstreckt sich zwar über 24 Stunden, doch davon müssen 15 Stunden für den Ausgleich gewertet werden. Somit ist diese tägliche Ruhezeit eine reduzierte.

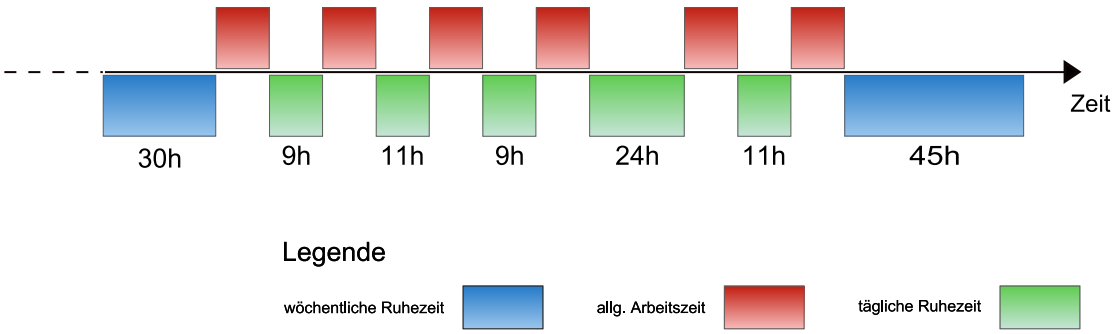


Abbildung 2.9: Beispiel für Ausgleich einer verkürzten wöchentlichen Ruhezeit (Zeitachse nicht maßstabsgetreu)

Ruhepausen können ähnlich wie die täglichen Ruhezeiten in exakt zwei Abschnitte geteilt werden. Hierbei muss der erste Teil über wenigstens 15 Minuten und der zweite Teil muss wenigstens 30 Minuten umfassen. Dies ermöglicht Konstellationen, die auf den ersten Blick nicht zulässig erscheinen, einer genaueren Untersuchung aber standhalten. Abbildung 2.10, S. 53, zeigt ein konstruiertes Beispiel für eine Einteilung von Lenk- und Pausenzeiten. Es beinhaltet ein hervorgehobenes Zeitfenster, das aus mehr als 4,5 Stunden Lenkzeit ohne vollständige Ruhepause besteht und ist dennoch zuläs-

sig.[27] Bei Prüfungen muss daher stets ein ausreichend großer Zeitraum berücksichtigt werden.

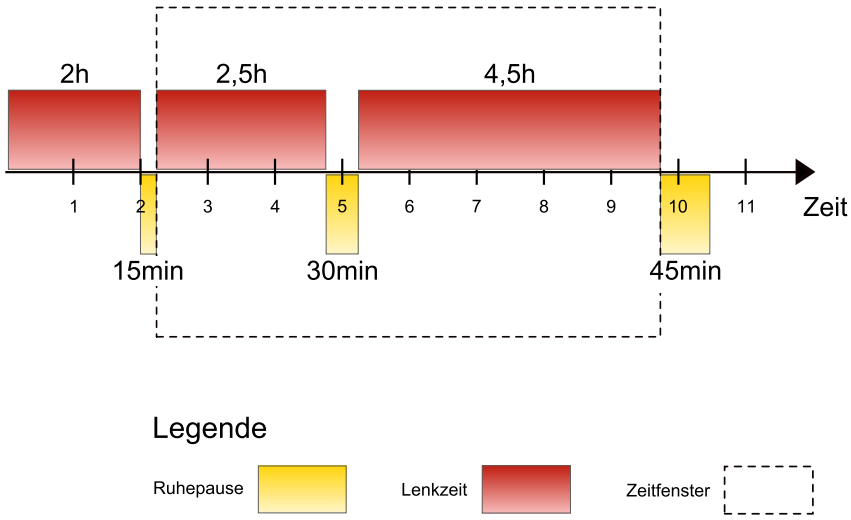


Abbildung 2.10: Beispiel für zweigeteilte Ruhepause

Mehrfahrerbetrieb Werden mindestens zwei Fahrer gleichzeitig auf demselben Fahrzeug eingesetzt, kann dies als *Mehrfahrerbetrieb* gewertet werden. Hierbei verlängert sich für beide Fahrer der maximale Zeitraum, innerhalb dessen eine tägliche Ruhezeit einzuhalten ist, von sonst 24 auf 30 Stunden. Darüber hinaus können Ruhepausen als *Beifahrer* auch während der Fahrt genommen werden; *Ruhezeiten* sind während der Fahrt jedoch nicht möglich. Mehrfahrerbetrieb ist daher geeignet, längere Fahrtstrecken schneller zu bewältigen. Denn während der Ruhepausen eines Fahrers kann der andere[28] weiterfahren. Für diesen Modus gilt allerdings, dass der zweite Fahrer spätestens eine Stunde nach dem ersten Fahrer im Fahrzeug anwesend sein muss. So wird verhindert, dass ein zweiter Fahrer nachträglich eingesetzt werden kann, um die „Schichtzeit“ des ersten spontan zu verlängern.

2.3.3 Kritik an den rechtlichen Rahmenbedingungen

Die oben beschriebenen Regelungen wurden vor dem Hintergrund eines natürlichen Zielkonflikts erstellt. Die Priorität liegt zwar auf dem (Arbeits-)Schutz angestellter Fahrer. Denn diese müssen in der Praxis häufig Fehler kompensieren, die bereits in der Phase der Tourenplanung entstanden. Als Resultat fahren und arbeiten sie oftmals erheblich länger als ursprünglich geplant, um Verspätungen beim Transport zu verhindern.

Auf der anderen Seite sollten die Vorschriften den Arbeitsablauf im Transportgewerbe jedoch nicht unverhältnismäßig einschränken. Die oben beschriebene Komplexität der Vorgaben zu den Lenk- und Ruhezeiten steht daher häufig unter Kritik. Es stellt sich die Frage, ob es für Fahrer und Disponenten sowohl inhaltlich als auch zeitlich zumutbar ist, sämtliche Vorschriften zu kennen und auf deren Einhaltung zu achten.

[27] Ein analoges Beispiel ließe sich ebenfalls für eine *geteilte tägliche Ruhezeit* erstellen.
[28] Im Regelfall werden nicht mehr als zwei Fahrer eingesetzt.

Trotz der hohen Komplexität der Vorschriften bleiben indes noch Unklarheiten. Die wohl auffälligsten drei werden im Folgenden kurz vorgestellt.

Mangelnde Definition des Werktags Zwar wird im ArbZG die maximale „werktägliche“ Arbeitszeit vorgegeben; eine Definition des „Werktags“ findet sich dort allerdings nicht. Dabei ergibt sich eine Mehrdeutigkeit aus der Tatsache, dass bei den Sozialvorschriften nach VO (EG) 561/2006 der „24-Stunden-Zeitraum“ hervorgehoben wird, der sich explizit von einem Kalendertag unterscheiden kann.

Bezug der Lenkzeit Das zweite Beispiel betrifft ebenfalls den zu bewertenden Kontext: Die tägliche Lenkzeit darf höchstens zweimal in der Woche auf höchstens 10 Stunden verlängert werden. Es wird jedoch keine Aussage darüber getroffen, zu welcher Woche eine *verlängerte Lenkzeit* gehört, die sich über das Ende einer Kalenderwoche erstreckt.

Betroffen hiervon sind besonders Fahrer von Fahrzeugen, die unter das in Deutschland geltende Fahrverbot an Sonn- und Feiertagen fallen. Da das Fahrverbot regelmäßig bis sonntags 22 Uhr gilt, beginnen direkt im Anschluss häufig die *täglichen Lenkzeiten* dieser Fahrer. Erstrecken sich diese Lenkzeiten über mehr als 9 Stunden, ergibt sich die Unklarheit.

Auswirkungen von Urlaub oder Krankheit In den Sozialvorschriften wurde eine klare Aussage zu Urlaubs- und Krankheitstagen versäumt. Es bleibt unklar, ob ein Verstoß vorliegt, wenn ein Fahrer – im Extremfall – im Anschluss an eine reduzierte wöchentliche Ruhezeit seinen kompletten Jahresurlaub nimmt und der Ausgleich für diese Ruhezeit deshalb nicht innerhalb der geforderten drei Kalenderwochen erfolgt.

Die Klärung dieser und ähnlicher offener Fragen wird sich voraussichtlich erst durch Gerichtsurteile ergeben, auch und gerade weil die Interpretation der Vorschriften unter Umständen durch unterschiedliche Prüf-Softwareprodukte auf unterschiedliche Weise geschehen kann.

2.3.4 Erfassung der Lenk- und Ruhezeiten durch den digitalen Tachographen

Seit 1. Mai 2006 müssen gemäß VO (EG) 561/2006 europaweit neu zugelassene Nutzfahrzeuge über 3,5 Tonnen Gesamtgewicht und alle Omnibusse mit mehr als neun Sitzplätzen – einschließlich Fahrersitz – mit einem elektronischen Fahrtenschreiber, dem digitalen Tachographen, ausgestattet werden. Dessen Funktionen und Bauweise sind weitgehend durch Verordnung (EG) Nr. 2135/98 des europäischen Rates vom 24. September 1998 zur Änderung der Verordnung (EWG) Nr. 3821/85 über das Kontrollgerät im Straßenverkehr und der Richtlinie 88/599/EWG über die Anwendung der Verordnungen (EWG) Nr. 3820/85 und (EWG) Nr. 3821/85 (fortan: *VO (EG) 2135/1998*) vorgegeben. Hauptzweck ist dabei die automatische, lückenlose und fälschungssichere Erfassung von Lenk- und Ruhezeiten. Die digitalen Tachographen ersetzen damit die analogen Kontrollgeräte (siehe Abbildung 2.11, S. 55). Die analogen Tachoscheiben (siehe Abbildung 2.12, S. 56, und Abbildung 2.13, S. 56) werden damit ebenfalls in absehbarer Zeit obsolet.

(a) analog (b) digital

Abbildung 2.11: Beispiele für Tachographen
Quellen: [Continental Automotive GmbH, 2008] und [Continental Automotive GmbH, 2007]

Um ein Fahrzeug mit digitalem Kontrollgerät einsetzen zu dürfen, muss der Fahrer im Besitz einer so genannten *Fahrerkarte* sein. Diese muss er vor Antritt der Fahrt in den dafür vorgesehenen Leseschacht des digitalen Tachographen einschieben (siehe Abbildung 2.11b unterhalb des Displays). Am Gerät selbst muss in der Folge immer der korrekte Betriebsmodus manuell eingestellt werden: Lenkzeit, Arbeitszeit (zum Beispiel während der Be- oder Entladung), Ruhepause oder Ruhezeit. Sobald sich das Fahrzeug jedoch bewegt, wird automatisch Lenkzeit ausgewählt. Die Daten werden sowohl auf der eingelegten Fahrerkarte personengebunden als auch auf dem digitalen Kontrollgerät fahrzeuggebunden gespeichert. Sie müssen gemäß VO (EG) 2135/1998 spätestens nach 28 Tagen ausgelesen und für mindestens ein Jahr archiviert werden. Die Archivierung kann auch über externe Dienstleister erfolgen. Am Markt gibt es dazu bereits eine Reihe von Unternehmen, die ihr Angebot diesbezüglich erweitert haben[29]. Fahrerkarte

Die Fahrerkarte wird je nach Bundesland von verschiedenen Instanzen ausgestellt. In den meisten Fällen sind dies die Fahrerlaubnisbehörden.[30] Neben der Fahrerkarte existieren noch drei weitere Arten von Karten für die digitalen Tachographen[31]:

- die *Werkstattkarte* zur Prüfung, Reparatur und Kalibrierung des digitalen Kontrollgerätes sowie zum Herunterladen der Daten und zur Datensicherung,
- die *Unternehmenskarte* zur Authentifizierung des zugehörigen Unternehmens, damit die Daten des Kontrollgeräts zwecks Archivierung heruntergeladen beziehungsweise im Fahrzeug ausgedruckt werden können und
- die *Kontrollkarte*, die einen unbeschränkten Zugriff auf die im digitalen Tachograph gespeicherten Daten erlaubt und von den Kontrollinstanzen verwendet wird.

[29] beispielsweise das Softwarebüro Zauner & Partner (`http://www.zamik.de/`), die DAKO GmbH (`http://www.tachoweb.eu/`) oder auch die Continental Automotive GmbH (`http://dtco.vdo.de/data-management/data-evaluation/archiving/tis-web-archiving.htm`)

[30] Eine vollständige Liste findet sich bei Kraftfahrt-Bundesamt [2005].

[31] vgl. Verordnung (EG) Nr. 1360/2002 der Kommission vom 13. Juni 2002 zur siebten Anpassung der Verordnung (EWG) Nr. 3821/85 des Rates über das Kontrollgerät im Straßenverkehr an den technischen Fortschritt

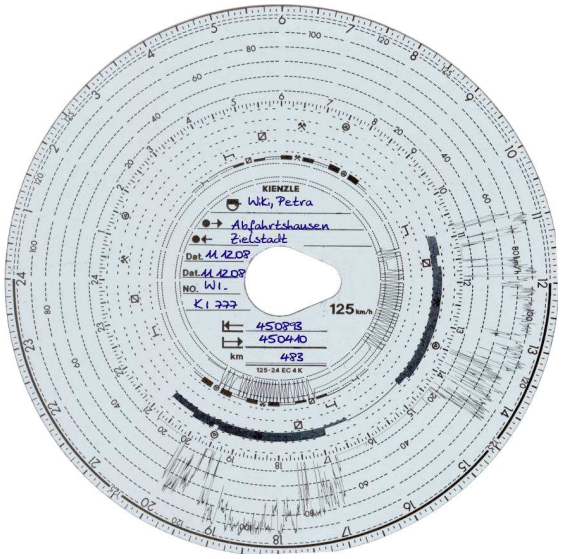


Abbildung 2.12: Beispiel für eine analoge Tachoscheibe
Quelle: [Seidler, 2009a]

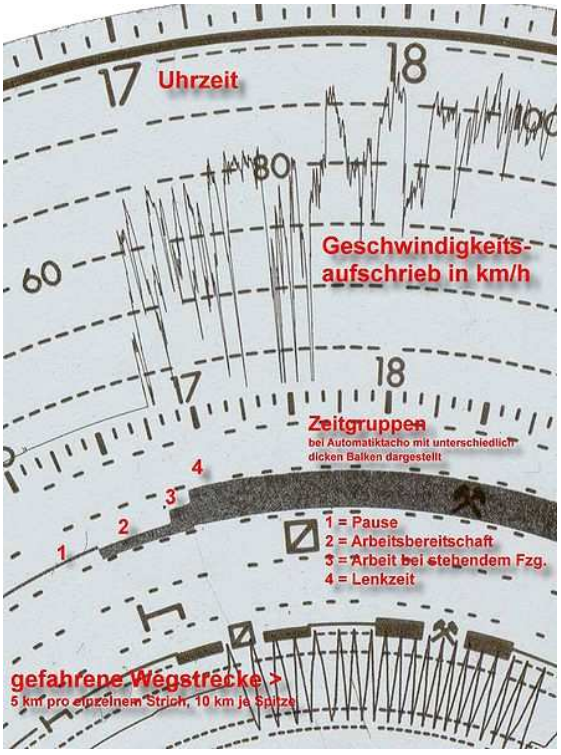


Abbildung 2.13: Ausschnitt einer Tachoscheibe mit Erläuterungen
Quelle: [Seidler, 2009b]

Über die Nutzbarkeit der Daten per Fernübertragung Aus praktischer Sicht ergibt sich noch ein Mangel: Derzeit können Lenk- und Ruhezeiten zwar mithilfe der digitalen Tachographen erfasst, aber nur begrenzt in Echtzeit ausgewertet werden. Technisch ist eine direkte Übertragung der Daten aus den Geräten zum Beispiel an die Dispositionszentrale einer Spedition grundsätzlich realisierbar. Für diesen Zweck bietet der digitale Tachograph neben der frontseitigen *Download-Schnittstelle* eine so genannte *Info-Schnittstelle*. Über diese stehen die Daten aus dem Tachographen weiteren telematischen Anwendungen zur Verfügung.

Allerdings werden sie bei dieser Übertragungsart nicht digital signiert und können daher nicht zur gesetzlich vorgeschriebenen Archivierung verwendet werden (vgl. [Winkler, 2007]). Diese Lücke soll das ergänzende *Remote Interface* schließen, das eine Datenfernübertragung via Mobilfunk in Echtzeit sowohl mit einer digitalen Signatur als auch einer Verschlüsselung ermöglicht (vgl. [VDO, 2008]). Digitale Tachographen der „zweiten Generation“, die eine Unterstützung dieser Schnittstelle anbieten, sind jedoch erst seit Mitte 2009 flächendeckend verfügbar (vgl. [VDO, 2011]).

2.4 Qualität der verwendeten Daten

Nachdem in den ersten Abschnitten dieses Kapitels die Herkunft der dieser Arbeit zugrunde liegenden Daten erläutert wurde, steht in diesem Abschnitt eine Erörterung der Qualität dieser Daten im Vordergrund. Insbesondere alle Untersuchungen auf Basis der Vergangenheitsdaten der IN tIME Express Logistik GmbH in Kapitel 7, S. 261 ff., müssen in diesem Zusammenhang gesehen werden.

2.4.1 Qualität der Vergangenheitsdaten der IN tIME Express Logistik GmbH

Die Vergangenheitsdaten liegen in Form einer relationalen Datenbank vor. Sie wurden sowohl automatisch erhoben als auch manuell von den Mitarbeitern des Unternehmens in die hauseigenen Dispositionssoftware eingepflegt. Während systemseitig generierte Daten wie Ortungsdaten weitgehend fehlerfrei sind, schwanken die manuellen Einträge mitunter in ihrer Datenqualität. Dies hat mehrere Ursachen, die im Folgenden näher dargelegt werden.

Unstrukturierte Daten Manche Felder in den Datenbanktabellen beinhalten textuelle Beschreibungen anstelle strukturierter Datentypen. Dies erschwert insbesondere die Interpretation der Fracht: Frachtverpackungen und -abmessungen können in der Dispositionssoftware als Freitexte hinterlegt werden. Dies führt zu unterschiedlichen Schreibweisen. Beispielsweise finden sich als Multiplikationszeichen bei den Abmessungen ein „x“ als auch ein „*“. Außerdem beeinträchtigen Rechtschreibfehler in den Freitexten die Datenqualität.

Unklare Inhalte und Dimensionen Die Semantik der Frachtbeschreibungen ist nicht immer eindeutig. So finden sich bei Beschreibungen der Frachtabmessungen zwar überwiegend Einträge wie „60x40x30“ oder „280 x 250 x 140“, aber es existieren darüber

hinaus auch Einträge wie „2,25*1,18*0,98“, was auf Meter statt Zentimeter als Referenz schließen lässt. Auch Einträge wie „stapelbar“, „ 120*100*70 nichtstabelbar“[32] oder auch „4,6 Lademeter“ erschweren eine automatisierbare Interpretation.

Eingabe-Fehler Da das Datenmodell der Vergangenheitsdaten auf einer relationalen Datenbank basiert, sind damit logische Überprüfungen grundsätzlich auf das Sicherstellen der referentiellen Integrität beschränkt. Zu jedem Verweis eines Feldes in einer Tabelle auf eine andere Tabelle muss zwangsläufig ein entsprechender Datensatz in letzterer existieren. *Semantische* Fehler bei der Dateneingabe können dagegen allenfalls durch Datenbank-Trigger, d. h. kleine Programme, die automatisch bei Datenänderungen gestartet werden, oder ähnliche Mechanismen entdeckt werden.

An einigen Stellen sind in den Vergangenheitsdaten daher offensichtliche Falscheingaben zu beobachten. Beispielhaft seien hierfür die nachfolgenden, ausgewählten Extremfälle aufgeführt:

- Falsche Eingaben bezüglich der Fahrzeugkapazität
  - Bei den Frachten zu sechs Aufträgen ist das Bruttogewicht mit einem Wert angegeben, der größer ist als „60 000“. Die vom Kunden gewünschte Fahrzeugkategorie ist dagegen nur mit „BUS“ bzw. „PKW“ notiert.
  - Gemäß den Stammdaten besitzen drei Fahrzeugen der Kategorie „PKW“ jeweils mehr als 200 Palettenstellplätze. Zwei Fahrzeuge der Kategorie „BUS“ sind mit je einmal 330 und einmal sogar 1 000 Palettenstellplätzen angegeben.
- Falsche Eingaben bezüglich der zeitlichen Reihenfolge
  - Bei 479 Aufträgen liegt der Zeitpunkt der Abholung zeitlich *nach* dem Auslieferungszeitpunkt.
  - Bei 6 276 Aufträgen, die an einer der Niederlassungen des Unternehmens umgeladen wurden, ist der Zeitpunkt dieser Umladung früher angegeben als der gespeicherte Zeitpunkt der Erstbeladung.

Fehlende Daten Bisweilen enthalten die Daten auch Lücken. Diese stellen eine fehlende Datenintegrität oder Informationslücken dar. Hierzu gehören unter anderem

- 3 656 Kundenstandorte, 247 808 Fahrzeugortungen sowie 1 541 Standorte der Subunternehmer, denen 0-Werte als Geokoordinaten zugewiesen sind,
- ein Fahrzeug, dem keine Fahrzeugkategorie zugewiesen wurde,
- 996 Fahrzeuge, für die 0 Palettenstellplätze hinterlegt sind,
- 54 Transporte, die sich nicht vollständig nachvollziehen lassen, da sie teilweise außerhalb des Zeitfensters der Vergangenheitsdaten liegen,
- 44 645 Aufträge, zu denen keine Frachtinformationen hinterlegt ist und deren Fracht daher lediglich anhand der vom Kunden gewünschten Fahrzeugkategorie abgeschätzt werden kann.

[32] Das Leerzeichen vor der „120“ ist kein Tippfehler des Autors, sondern im betroffenen Datensatz ebenfalls enthalten.

Darüber hinaus ist zu vielen Aufträgen kein lückenloser Transportweg nachvollziehbar. Einerseits betrifft dies natürlich Aufträge, deren Bearbeitung an den zeitlichen „Rändern“ der Vergangenheitsdaten liegt. Andererseits finden sich in den manuellen Ortungsdaten gelegentlich widersprüchliche Angaben, wenn ein Auftrag etwa nach seiner Abholung bei einer Niederlassung entladen und im Anschluss – gemäß der Vergangenheitsdaten – an einem *anderen* Ort zur Weiterfahrt beladen wurde.

2.4.2 Qualität der digitalen Straßendaten des OpenStreetMap-Projekts

Das OpenStreetMap-Projekt basiert im Wesentlichen auf der freiwilligen Mitarbeit der Nutzer. Es bleibt nicht aus, dass deren Beiträge Fehler beinhalten. Ebenso können sie keine Vollständigkeit der Kartendaten garantieren. Zwar gab es des Öfteren größere Imports von offiziellen Stellen[33], doch auch diese sind zu großen Teilen bereits manuell überarbeitet worden.

Eine detaillierte Analyse der Datenqualität im deutschen Bundesgebiet durch den Vergleich mit einer kommerziellen digitalen Straßenkarte der Firma TomTom aus dem Jahr 2011 unternahmen Neis u. a. [2012]. Die Autoren legen hierfür den Stand der OSM-Daten von Juni 2011 zugrunde. Ihre wesentlichen Ergebnisse sind:

1. Die OpenStreetMap-Daten umfassten 27 % mehr Wegkilometer als die zum Vergleich herangezogene kommerzielle Karte (vgl. [Neis u. a., 2012, S. 9]). Werden allerdings nur Straßen betrachtet, die mit dem Auto befahrbar sind, d. h. insbesondere keine Rad- oder Fußwege, umfasst die kommerzielle Karte 9 % mehr Kilometer (vgl. ebd.). Sofern die Rate der Beiträge bei OSM in etwa konstant bleibt, lässt sich jedoch abschätzen, dass die OSM-Daten Mitte 2012 annähernd eine vollständige Abdeckung erreichen (vgl. [Neis u. a., 2012, S. 11]).
2. Grundsätzlich gilt: Je dichter ein Gebiet besiedelt ist, desto besser ist es bei OSM abgedeckt (vgl. ebd.).
3. In der kommerziellen Karte sind etwa sechs Mal so viele Abbiegebeschränkungen verzeichnet wie in der OSM-Karte. Zwar ist der Unterschied in dicht besiedelten Gebieten geringer als in dünn besiedelten, doch bei diesem Aspekt besteht offenbar noch deutlich Verbesserungspotential. (vgl. [Neis u. a., 2012, S. 16 f.])

Auf einige Mängel im Datenmaterial der hier verwendeten OSM-Karten wird im nachfolgenden Kapitel gesondert eingegangen. Es wird erläutert, inwieweit sie besonders relevant für diese Arbeit sind und zusätzlich, wie sie behandelt werden.

Abschätzung von Fahrtdauern auf der digitalen Straßenkarte Die Ermittlung der Fahrtdauer stellt eine nicht zu unterschätzende Herausforderung dar. Die wohl präzisesten Werte könnten über eine Realdaten-Erhebung gewonnen werden. Der Ansatz ist leicht zu beschreiben, jedoch schwierig in der Durchführung: Benötigt wird hierfür ein Datensatz real in der Vergangenheit gefahrener Wege, beispielsweise in Form von GPX-Daten. Im Anschluss erfolgt ein Matching der Wege auf eine digitale Straßenkarte.

[33] Hier ist besonders die Integration der Kartendaten des United States Census Bureau, namentlich das „Topologically Integrated Geographic Encoding and Referencing system“ (TIGER), zwischen 2007 und 2008 zu nennen, welches das gesamte U.S.-Gebiet abdeckte (vgl. `http://wiki.openstreetmap.org/w/index.php?title=TIGER&oldid=811997`).

Für jeden Pfeil des Graphen der Straßenkarte, die in einer oder mehreren dieser Wege verwendet wurde, könnten auf diese Weise Fahrtzeiten gemittelt werden. Bei dieser Herangehensweise ließe sich schnell feststellen, dass die Fahrtzeiten nach Fahrzeugtyp, Fahrertyp und Tageszeit – und möglicherweise auch *Jahreszeit* – variieren.

Offenbar kann dieser Ansatz im Rahmen dieser Arbeit nicht verfolgt werden. Wie die Fahrtdauern stattdessen ermittelt werden, beschreibt Abschnitt 5.5.4 auf Seite 172 ff.

3 Import und Aufbereitung der OpenStreetMap-Daten

Inhalt

Sofern nicht anders erwähnt, werden im Rahmen dieser Arbeit ausschließlich Straßenkarten des OpenStreetMap-Projekts verwendet, die von `www.geofabrik.de/` heruntergeladen wurden. Dieses Kapitel beschreibt die Reihenfolge und Details der durchgeführten Arbeitsschritte, um aus den OSM-Rohdaten einen Graphen zu interpretieren, auf den Wegsucheverfahren aus der Literatur anwendbar sind. Der Ablauf ist in Abbildung 3.1 skizziert und stellt gleichzeitig den Aufbau dieses Kapitels dar.

Zusätzlich sind Messungen und Statistiken zu den Karten angegeben. Mit ihrer Hilfe werden unter anderem die Überlegungen, die zu den Präprozess-Schritten (Arbeitsschritte 1–6 in Abbildung 3.1) geführt haben, untermauert. Sie dienen dem besseren Verständnis des zugrunde liegenden Kartenmaterials.

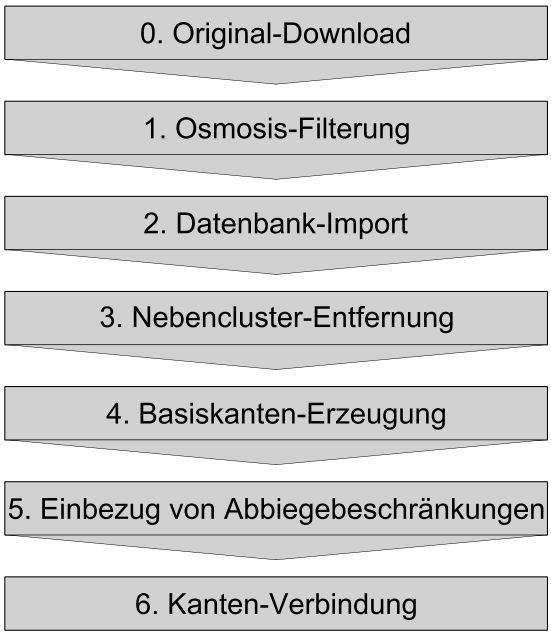


Abbildung 3.1: Präprozess aus sechs Schritten

Dieses Kapitel bezieht sich an vielen Stellen auf die konkrete Implementierung der hier vorgestellten Schritte. Daher sei an den sprachlichen Unterschied zwischen *Pfeilen des Graphen* und *Kantenobjekte* erinnert[1].

3.1 Datenfilterung

Wie in Abschnitt 2.2.2 beschrieben, sind in den Straßenkarten des OpenStreetMap-Projekts neben den für Navigation unverzichtbaren Straßeninformationen noch viele weitere Objekte der realen Welt verzeichnet. Die für diese Arbeit relevanten Daten sind daher zu extrahieren. Die Filterung erfolgte mithilfe der frei verfügbaren Software namens *Osmosis*[2]. Der zugehörige Aufruf ist in Listing 3.1 wiedergegeben und besteht aus den rot hervorgehobenen Programmparametern und den ihnen jeweils angefügten Argumenten. Der Befehl wird von Osmosis sequenziell abgearbeitet. Die Weiterleitung der Zwischenergebnisse von einem Parameter zu einem anderen erfolgt anhand so genannter *Pipes*, deren Namen frei wählbar sind. Die Programmparameter sind nachfolgend im Einzelnen erläutert[3]:

```
1  osmosis
2  --read-pbf original.pbf outPipe.0="filePipe"
3  --way-key-value inPipe.0="filePipe" outPipe.0="wayFilterPipe"
4  keyValueList=
5  highway.motorway,
6  highway.motorway_link,
7  highway.trunk,
8  highway.trunk_link,
9  highway.unclassified,
10 highway.secondary,
11 highway.secondary_link,
12 highway.primary,
13 highway.primary_link,
14 highway.tertiary,
15 highway.tertiary_link,
16 highway.residential,
17 route.ferry,
18 highway.service
19 --tag-filter reject-ways motorcar=no inPipe.0="wayFilterPipe" outPipe.0="tagFilterPipe1"
20 --tag-filter reject-ways access=agricultural,forestry,no,private,emergency inPipe.0="tagFilterPipe1"
       outPipe.0="tagFilterPipe2"
21 --tag-filter reject-ways service=parking_aisle,alley,siding,spur,drive-through,emergency_access,
       firefighters,driveway inPipe.0="tagFilterPipe2" outPipe.0="tagFilterPipe3"
22 --used-node idTrackerType=BitSet inPipe.0="tagFilterPipe3" outPipe.0="usedPipe"
23 --write-pbf omitmetadata=true file=original_gefiltered.pbf inPipe.0="usedPipe"
```

Listing 3.1: Osmosis-Befehl zum Filtern der OSM-Highways

[1] siehe Seite 33, Stichwort „Ungerichtete Kanten, Pfeile und Kantenobjekte“

[2] siehe auch Abschnitt 2.2.1 auf Seite 46

[3] Eine detailreichere Beschreibung kann unter `https://wiki.openstreetmap.org/w/index.php?title=Osmosis/Detailed_Usage_0.43&oldid=930761` nachgeschlagen werden (letzter Zugriff: 18.9.2013)

--read-pbf Der Programmparameter gibt das Einlesen der Quelldatei im pbf-Format[4] an. Sein Argument ist die Quelldatei; in Listing 3.1 ist es *original.pbf*. Das Einlese-Ergebnis wird über die hier *filePipe* genannte Pipe dem nächsten Programmparameter zur Verfügung gestellt.

--way-key-value Über diesen Parameter filtert Osmosis OSM-Wege der Quelldatei. Es bleiben nur Wege mit den als Argument aufgeführten Tags übrig. Im Gegensatz zu der auf Seite 43 beschriebenen Notation verlangt Osmosis eine Tag-Schreibweise, bei der Schlüssel und Werte durch einen Punkt getrennt sind. In Listing 3.1 wird das Filterergebnis in die Pipe namens *wayFilterPipe* weitergeleitet.

Für diese Arbeit wurden neben den Straßen auch Fährverbindungen berücksichtigt (siehe Zeile 17 in Listing 3.1). Da Fähren oft nur über spezielle Zufahrten erreichbar sind, wurden als *Service*-Strecken markierte Wege ebenfalls zugelassen (siehe Zeile 18).

--tag-filter Weiteres Filtern der Wege erfolgt über diesen Parameter. Wege, die nicht für „motorcar“ zugelassen sind, werden mittels des Arguments *reject-ways* entfernt (siehe Zeile 19 in Listing 3.1). Anschließend entfallen Wege, die nicht dem allgemeinen Straßenverkehr zugängig sind. Hierbei werden zuerst Tags mit dem „access“-Schlüsselwort und danach Tags mit dem „service“-Schlüsselwort verwendet (siehe Zeilen 20 und 21).

--used-node Die Entfernung sämtlicher Knoten, die zu keinem OSM-Weg gehören, wird mit diesem Parameter veranlasst. Zweck des Parameters ist hier die Datenreduktion. Denn ein isolierter Knoten ist für die weiteren Betrachtungen in dieser Arbeit irrelevant.

--write-pbf Dieser Parameter sorgt dafür, dass Osmosis das gefilterte Ergebnis in eine Datei im pbf-Format schreibt. Das Argument *original_gefiltert.pbf* in Listing 3.1 ist die Ausgabedatei.

Die auf diese Weise gewonnenen gefilterten Kartendateien besitzen nur noch etwa 15 % der Größe der zugehörigen Originale. Die pbf-Dateigröße der Deutschland-Karte vom 10. Mai 2012[5] verringert sich so von 1,3 GB auf 182 Megabyte (MB). Die Statistiken für die Anzahl der Knoten, Wege und Relationen der in dieser Arbeit verwendeten Karten sind in Tabelle 3.6, S. 85, zusammengefasst. Aus Speicherplatzgründen wurde die Europa-Karte zusätzlich dem in Listing 3.2, S. 64, wiedergegebenen Bounding-Box-Filter unterzogen, der sie im Wesentlichen auf das Gebiet reduziert, in dem die in dieser Arbeit betrachteten Aufträge der IN tIME Express Logistik GmbH stattfanden. Abbildung 3.2, S. 64, visualisiert den gewählten Kartenausschnitt als JOSM-Screenshot.

[4] siehe hierzu S. 46 f., Stichwort „Das pbf-Dateiformat“
[5] Stand von 03:30 Uhr

```
1 osmosis
2 --read-pbf original.pbf outPipe.0="filePipe"
3 --bounding-box inPipe.0="filePipe" outPipe.0="polygonFilterPipe" idTrackerType=BitSet
      completeWays=no completeRelations=no cascadingRelations=no clipIncompleteEntities=true
      top=71.2 left=-24.7 bottom=34.5 right=33.1
4 --way-key-value inPipe.0="polygonFilterPipe" outPipe.0="wayFilterPipe"
5 keyValueList=
6 highway.motorway,
7 ...
```

Listing 3.2: Bounding-Box-Filter für die Europa-Karte
Ab Zeile 7 ist der Rest identisch zu Listing 3.1, Zeile 6 ff.

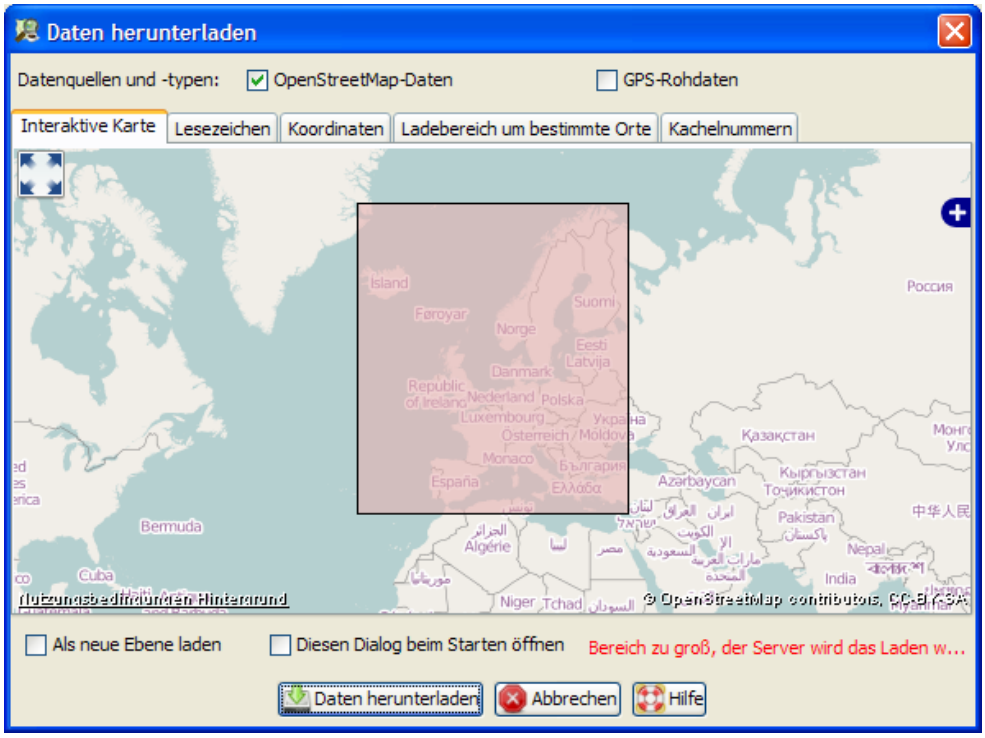


Abbildung 3.2: Bounding Box um Europa
Der Screenshot aus JOSM zeigt den verwendeten Bounding-Box-Filter, der bei der Europa-Karte zusätzlich eingesetzt wurde. JOSM warnt den Benutzer jedoch vor Downloads dieser Größenordnung zu, da dies den zentralen Datenbankserver von OSM überlasten würde. Die entsprechende Meldung ist unten rechts auf dem Bild ebenfalls erkennbar. Die hier verwendeten Daten stammen daher, wie zu Beginn dieses Kapitels erwähnt, von `www.geofabrik.de/`.

3.2 Datenbank-Import

Für die weitere Arbeit mit den Kartendaten empfiehlt es sich, diese in eine relationale Datenbank zu importieren. Dies bietet besonders während der Karten-Analyse und der Programm-Entwicklung mehrere Vorteile:

- Die Speicherung der Daten und der direkte Zugriff auf sie kann mithilfe standardisierter Software-Tools erfolgen[6].
- Auf einzelne Knoten/Wege/Relationen kann per SQL schnell zugegriffen werden.
- Insbesondere das Parsen der Daten wird vereinfacht, da die meisten Programmiersprachen – wie das in dieser Arbeit verwendete Java – eine gute SQL-Integration anbieten.
- Das Datenmaterial lässt sich – beispielsweise durch zusätzliche Datenbanktabellen – einfach erweitern, ohne Rücksicht auf eventuelle Anforderungen des Dateiformats nehmen zu müssen.

Für diese Arbeit wurde das Open-Source-Software-Projekt *TravelingSalesman*[7] verwendet und überarbeitet. Es kommt sowohl beim Datenimport als auch bei der Visualisierung zum Einsatz. Das Datenbankschema ist in Abbildung 3.3 dargestellt. Es wird in den nachfolgenden Schritten noch um weitere Tabellen ergänzt.

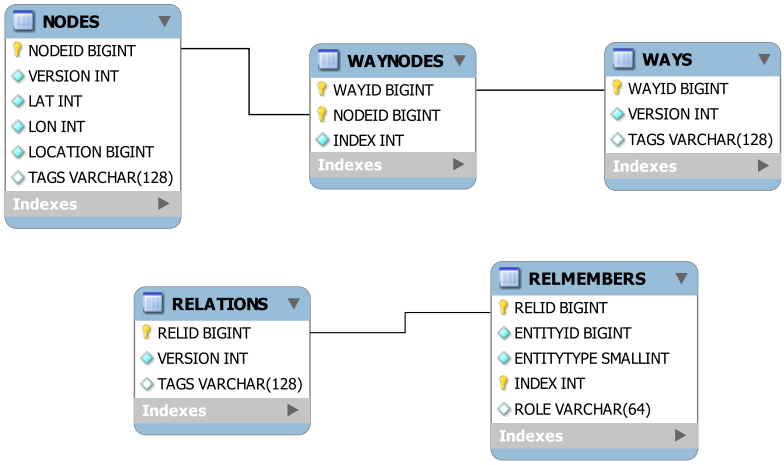


Abbildung 3.3: Datenbankschema aus TravelingSalesman
Gelbe Schlüssel symbolisieren den Primärschlüssel der jeweiligen Datenbanktabelle; Verbindungslinien zwischen den Tabellen stellen Fremdschlüsselbezüge dar.

Die verwendete Datenbank namens *H2*[8] ist ebenfalls eine Open-Source-Lösung. Für die in Abschnitt 2.2.1 vorgestellten OSM-Primitive *Knoten*, *Wege* und *Relationen* existiert im verwendeten Datenbankschema jeweils eine eigene Tabelle. Wege und Relationen

[6] siehe z. B. die Entfernung der Nebencluster in Abschnitt 3.3 oder Kantenobjekt-Verbindung in Abschnitt 3.6

[7] siehe auch `http://wiki.openstreetmap.org/wiki/TravelingSalesman` sowie `http://sourceforge.net/projects/travelingsales/`

[8] `http://www.h2database.com`

benötigen jeweils noch eine weitere Tabelle, um die in ihnen beinhalteten Objekte zu referenzieren. Die `INDEX`-Spalte der Tabellen `WAYNODES` und `RELMEMBERS` geben die Reihenfolge der Tabelleneinträge an; beispielsweise die der Knoten entlang eines Weges. Einträge der `INDEX`-Spalte beginnen bei Null.

Etwas weniger intuitiv gestaltet sich die Tabelle `RELMEMBERS`, da eine Relation sowohl aus Knoten, Wegen sowie weiteren Relationen bestehen kann. Die drei Sorten der Bestandteile werden in der Spalte `ENTITYID` referenziert. Um sie auseinanderhalten zu können, wird zusätzlich noch in der Spalte `ENTITYTYPE` vermerkt, ob es sich um einen Knoten (Wert: 1), einen Weg (Wert: 2) oder eine Relation (Wert: 3) handelt.

Ebenfalls ungewöhnlich ist die Modellierung von OSM-Tags. Denn diese sind nicht in der so genannten *dritten Normalform*[9] hinterlegt. Stattdessen werden sie als eine einzelne fortlaufende Zeichenkette in einer Spalte gespeichert. Tag-Schlüssel und -Werte werden durch ein doppeltes Gleichheitszeichen aneinandergehängt und jeder Tag durch eine doppelte Raute (##) terminiert.

3.3 Nebencluster-Entfernung

Die Karten von OpenStreetMap sind an vielen Stellen detailreicher und aktueller als kommerzielle Straßenkarten. Da es sich aber um ein Freiwilligen-Projekt handelt, ist die Qualitätssicherung weniger professionell. Dies führt unter anderem dazu, dass das Straßennetz der Karte nicht vollständig *zusammenhängend* ist. Es existieren vielmehr einzelne, kleinere Straßennetz-Cluster, die – um bei dem Begriff zu bleiben – um den und zwischen dem mit Abstand größten *Hauptcluster* verteilt sind. Abbildung 3.4, S. 67, zeigt am Beispiel der Niedersachsenkarte Zahl und Lage derjenigen OSM-Wege, die zu Nebenclustern gehören.

Nicht immer liegt bei einem Nebencluster ein Fehler in der Karte vor. Ein valides Beispiel findet sich beim Steinhuder Meer in Abbildung 3.5, S. 67. Hier existiert eine Fährverbindung, die korrekt einen „route=ferry“-Tag besitzt. Da die Fähre aber nicht per Auto erreichbar ist, gehört sie zu einem Nebencluster. Es ließe sich allenfalls darüber streiten, ob ein zusätzliches Tagging dieser Wege sinnvoll wäre, um zu kennzeichnen, dass es sich um reine Personen-Fähren handelt.

[9] siehe hierfür beispielhaft [Elmasri u. Navathe, 2009]

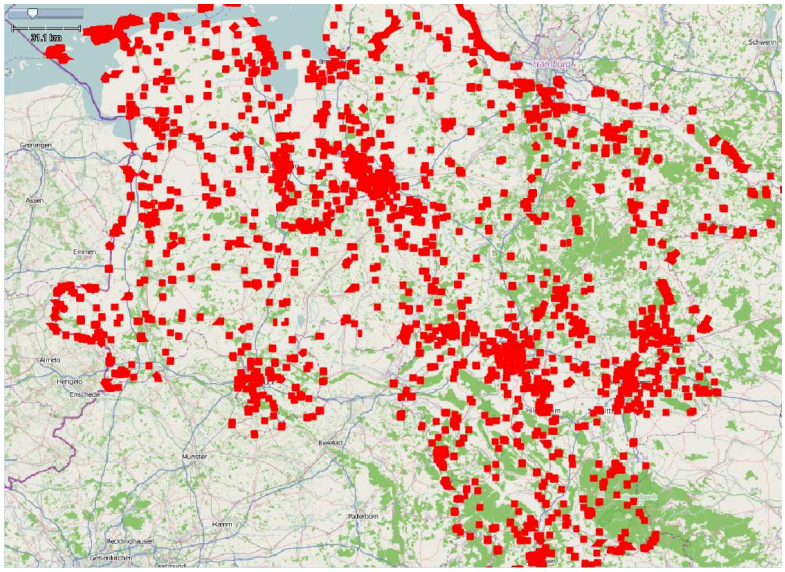


Abbildung 3.4: Nebencluster auf der Niedersachsenkarte
Rot markiert sind die OSM-Wege, die nicht zum Hauptcluster gehören. Im Nord-Osten ist der größte Nebencluster der Karte mit den durch eine Fährlinie verbundenen Inseln Juist und Norderney zu erkennen.

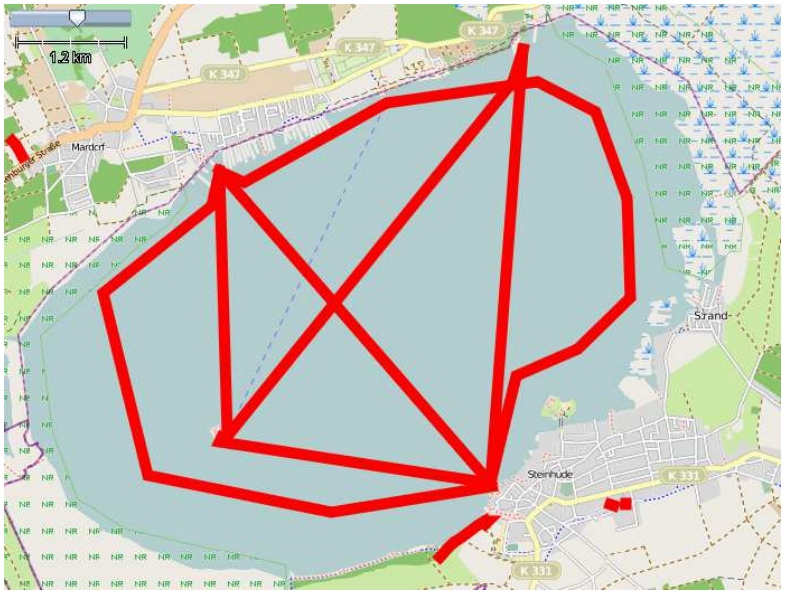


Abbildung 3.5: Beispiel für einen Nebencluster am Steinhuder Meer
Die Fährverbindung ist in der Karte als route=ferry markiert, hat aber keinen Anschluss an den Hauptcluster Niedersachsens. Oben links und unten rechts im Bild sind noch insgesamt drei weitere Nebencluster zu sehen.

Problembedeutung für die Wegsuche Offensichtlich können Wegsuchen, die bei einem Startknoten innerhalb eines Clusters beginnen, unmöglich einen Weg zu einem Zielknoten innerhalb eines anderen Clusters finden. Darüber hinaus kann eine solche Wegsuche ganz andere praktische Probleme nach sich ziehen.

Betrachtet werde die Wegsuche von einem Knoten des Hauptclusters C_H zu einem Zielknoten in einem Nebencluster C_N. Die in Abschnitt 4.1.1.1 vorgestellte Dijkstra-Wegsuche erreicht gemäß Verfahren 6, S. 93, *sämtliche Knoten* aus C_H, bevor sie abbricht. Dies ist nicht nur langsam, sondern kann bei größeren Graphen wie der Straßenkarte von Deutschland und Europa zu Speicherproblemen auf dem System führen, auf dem die Wegsuche vollzogen wird. Denn da die Zwischenergebnisse im Allgemeinen im Arbeitsspeicher gehalten werden, wird bei einer derartigen Suche beinahe noch einmal der Speicher für die Karte notwendig, da der Hauptcluster gewissermaßen dupliziert wird. Auch der erheblich reduzierte Suchraum bei neueren Wegsucheverfahren, wie den in Kapitel 5, S. 131 ff., vorgestellten Contraction Hierarchies, ändert dieses Problem nicht grundlegend.

Deshalb wurde in dieser Arbeit von den Straßenkarten jeweils nur der Hauptcluster verwendet und die Nebencluster verworfen. Die Identifikation der einzelnen Cluster erfolgte iterativ über alle OSM-Wege in der Datenbanktabelle `WAYS`[10]. Jedem Knoten eines Weges wird derselbe Cluster zugeordnet. Wird hierbei ein Knoten erreicht, dem bereits ein anderer Cluster zugeordnet ist, werden beide Cluster vereint. Am Ende werden alle Knoten verworfen, die nicht zum größten Cluster gehören. Verfahren 2, S. 69, stellt das Vorgehen im Pseudocode dar.[11] Die Kenngrößen der Straßenkarten von Niedersachsen, Deutschland und Europa bei der Nebencluster-Entfernung sind Tabelle 3.1 zu entnehmen. Die Zusammenfassung in Tabelle 3.6, S. 85, am Ende dieses Kapitels beinhaltet in der Zeile *„Nebencluster-Entfernung“* Informationen über die größten Nebencluster.

	Niedersachsen	Deutschland	Europa
Anzahl Cluster	2 149	22 326	75 869
Anzahl Knoten im größten Nebencluster	1 464[†]	4 374[‡]	101 397[※]

[†] Inseln Juist und Norderney – verbunden durch eine Fährlinie.
[‡] Insel Sylt
[※] Insel Zypern

Tabelle 3.1: Cluster der verwendeten Straßenkarten

[10] siehe Datenbankschema in Abbildung 3.3, S. 65
[11] Es sei darauf hingewiesen, dass die verwendete Implementierung effizienter arbeitet als der hier dargestellte Pseudocode. Auf die genaue Wiedergabe der Implementierung wurde aber zugunsten der Lesbarkeit des Pseudocodes verzichtet.

Verfahren 2 Nebencluster-Entfernung

Gegeben: Eine OpenStreetMap-Karte mit Knoten N und Wegen W

1: $aktuelleClusterId \coloneqq 0$
2: $v.clusterId \coloneqq 0, \quad \forall v \in N$ ▸ Initialisierung
3: **for all** $weg \in W$ **do**
4: $\quad aktuelleClusterId \coloneqq aktuelleClusterId + 1$
5: $\quad$ **for all** $u \in weg$ **do**
6: $\quad\quad$ **if** $u.clusterId = 0$ **then** ▸ Knoten u ist noch keinem Cluster zugewiesen
7: $\quad\quad\quad u.clusterId \coloneqq aktuelleClusterId$
8: $\quad\quad$ **else** ▸ Knoten u ist bereits Teil eines „alten“ Clusters
9: $\quad\quad\quad$ **for all** $v \in \{v' \in N \mid v'.clusterId = u.clusterId\}$ **do** ▸ Knoten des „alten“ Clusters
10: $\quad\quad\quad\quad v.clusterId \coloneqq aktuelleClusterId$ ▸ Übernimmt diese Knoten in den Cluster *aktuelleClusterId*
11: $\quad\quad\quad$ **end for**
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: Zähle für alle vergebenen Cluster-Ids die Anzahl der zugehörigen Knoten
16: Entferne alle Knoten aus der Karte, die nicht zum größten Cluster gehören
17: Entferne alle Wege, die dadurch keine Knoten mehr beinhalten

Ergebnis: Die Karte besteht nur noch aus dem größten zusammenhängenden Cluster.

Betrachtung der größten Nebencluster Eine genauere Untersuchung der größten Nebencluster zeigt, dass sie für alle drei betrachteten Karten Inseln repräsentieren. Bei der Niedersachsenkarte handelt es sich um einen gemeinsamen Cluster aus den ostfriesischen Inseln Juist und Norderney. Diese sind im Kartenmaterial zwar durch eine Fährlinie verbunden; es ist sogar eine weitere Fähre Richtung Festland bei der Stadt Norden verzeichnet. Letzte ist aber nicht an das restliche Festland-Straßennetz angebunden (siehe Abbildung 3.6, S. 70). Da Juist ohnehin eine autofreie Insel ist, stellt der Nebencluster nur für die Navigation von und nach Norderney einen Fehler dar.

Ähnlich gestaltet es sich für die Deutschland- und Europakarte. Bei ersterer liegt der größte Nebencluster auf der Insel Sylt. Auch bei dieser Insel sind die Fährlinien im Kartenmaterial nicht mit dem Straßennetz des Festlandes verbunden. Bei der Europakarte bildet ein Teil Zyperns den größten Nebencluster. Hier kommt hinzu, dass die ausgewählte Bounding Box bei der Europakarte Zypern zerteilt[12].

Keine manuelle Kartenkorrektur

Zugunsten eines reproduzierbaren, automatisierten Import-Prozesses wurde im Rahmen dieser Arbeit auf eine manuelle Anbindung der besagten Nebencluster an den Hauptcluster explizit verzichtet. Ansonsten ließe sich nur schwer argumentieren, warum andere Kartenfehler nicht ebenfalls bereinigt wurden.

Beschränkung des Verfahrens Die Nebencluster-Entfernung kann trotz ihrer Zielsetzung nicht vollständig sicherstellen, dass jeder Weg zwischen je zwei Knoten der Karte

[12] zur Definition der verwendeten Bounding Box: siehe Abschnitt 3.1 und insbesondere Listing 3.2, S. 64

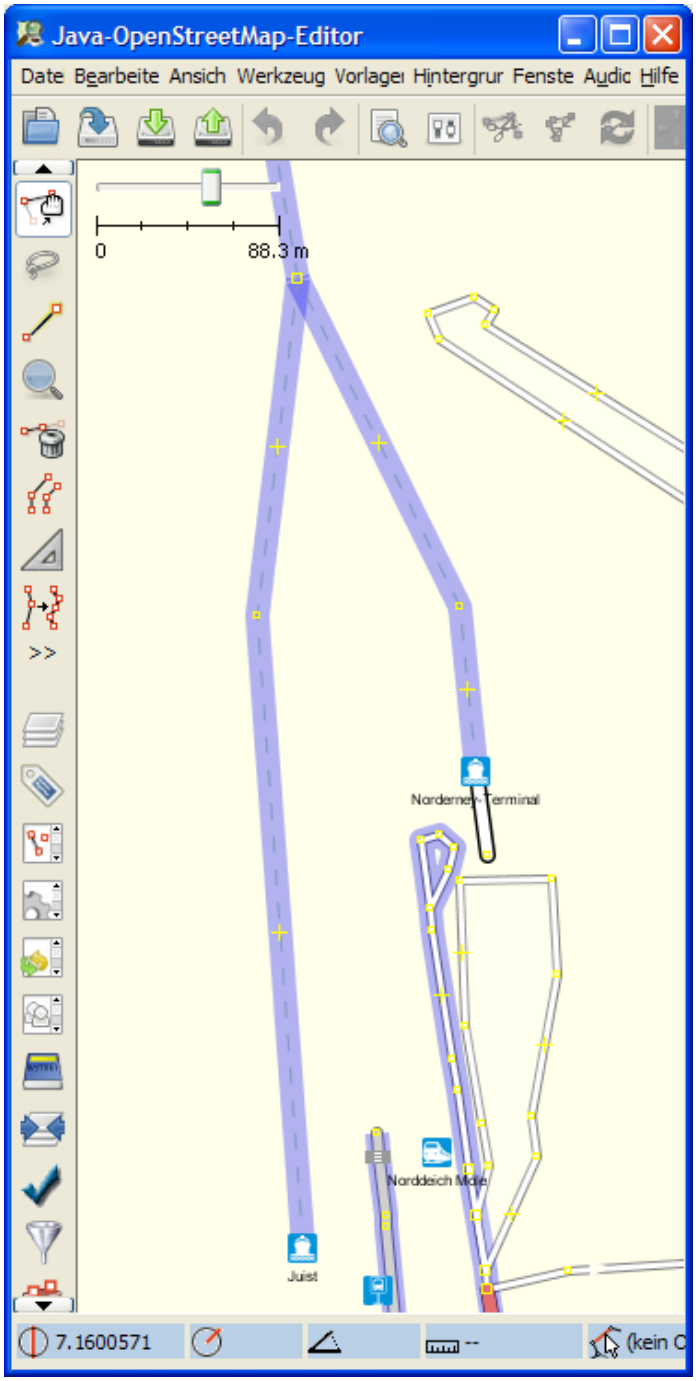


Abbildung 3.6: Fehlende Verbindung zwischen der Fähre aus Juist und dem Festland bei der Stadt Norden (Screenshot aus JOSM)
Die kreisförmigen Markierungen kennzeichnen die Stellen der Karte, bei denen die Verbindung vom Straßenzubringer zur von Norden eingehenden Fähre fehlt.

existiert und daher gefunden werden kann. Denn es ist zumindest denkbar, dass Knoten am äußeren Rand des Clusters existieren, die mit dem Rest des Clusters nur durch eine Einbahnstraße verbunden sind. Zeigt diese in die „falsche“ Richtung, separiert dies den Knoten vom Rest des Clusters. Dieses Szenario ist allerdings unwahrscheinlich, so dass es an dieser Stelle vernachlässigt wird.

3.4 Basiskanten-Erzeugung

In den vorangegangenen Abschnitten wurde beschrieben, wie das OSM-Kartenmaterial gefiltert und in eine Datenbankstruktur überführt wurde. Die in dieser Arbeit vorgestellten Wegsucheverfahren benötigen jedoch als Grundlage einen gewichteten, gerichteten Graphen mit Knoten und Pfeilen[13]. Dieser muss noch aus dem OSM-Datenmodell extrahiert werden.

- Die Knoten des Graphen können direkt aus den OSM-Knoten übernommen werden.
- Die Pfeile des Graphen ergeben sich aus den OSM-Wegen.

Mithilfe der in diesem Arbeitsschritt extrahierten Kantenobjekte[14] ließe sich ein strukturell geeigneter Graph interpretieren. Allerdings wird im übernächsten Arbeitsschritt in Abschnitt 3.6, S. 78 ff., gezeigt, dass es sinnvoll ist, mehrere dieser Kantenobjekte zu verbinden. Aus diesem Grund erhalten die hier extrahierten Objekte die Bezeichnung *Basiskanten*. Dies unterstreicht ihre Eigenschaft als Grundlage derjenigen Kantenobjekte, aus denen letztlich der gesuchte Graph erstellt wird. Basiskanten beinhalten sowohl gerichtete als auch ungerichtete Straßenabschnitte. Sie werden in einer eigenen Datenbanktabelle gespeichert, welche aus den nachfolgend beschriebenen acht Spalten besteht.

Begriff: Basiskanten

`NodeId1` beinhaltet die Knoten-Id des ersten Knotens der Basiskante.[15] Die Spalte stellt einen Fremdschlüsselbezug zur Tabelle `NODES` her.

`NodeId2` beinhaltet die Knoten-Id des zweiten Knotens der Basiskante; sie stammt ebenfalls aus Tabelle `NODES`.

`Length` speichert die Länge der Basiskante in Kilometern.

`OkToGoTo1` ist ein boolescher Wert, d. h. ein so genanntes Flag, das angibt, ob die Basiskante von ihrem zweiten Knoten in Richtung ihres ersten Knotens befahrbar ist. Hierüber werden Einbahnstraßen gekennzeichnet.

[13] zur Definition: siehe Abschnitt 1.3.1, S. 29 ff.

[14] zur sprachlichen Unterscheidung zwischen ungerichteten Kanten, Pfeilen und Kantenobjekten siehe Seite 33

[15] Für das Datenmodell ist es unerheblich, ob ein Knoten der „erste“ oder „zweite“ des Kantenobjekts ist. Ob eine Basiskante in graphentheoretischer Hinsicht ein *Pfeil* ist, wird durch die Spalten `OkToGoTo1` und `OkToGoTo2` modelliert.

`OkToGoTo2` ist wie Spalte `okToGoTo1`; allerdings für die Gegenrichtung[16].

`SpeedLimitedCountryHighwayId` speichert eine Referenz für die Kategorie der Basiskante.

`WayId` ist die Id desjenigen OSM-Weges, aus dem die Basiskante ermittelt wurde. Der Wert wird bei der Erweiterung des Datenmodells um Abbiegebeschränkungen in Abschnitt 3.5, S. 75 ff., verwendet.

`EdgeId` beinhaltet eine eindeutige Id der Basiskante und dient damit hauptsächlich Debugging-Zwecken.

Über die Berechnung der Kantenlängen Sowohl für die Ermittlung kürzester als auch schnellster Wege in Graphen muss selbstverständlich die Länge der Kanten und Pfeile bekannt sein. Dieser Wert ist allerdings nicht explizit in den OSM-Daten vorhanden und muss daher separat berechnet werden. Für diese Arbeit wurden die Längen der Basiskanten als Längen der Orthodromen zwischen ihren Endknoten ermittelt, d. h. als kürzeste Linie zwischen zwei Punkten auf einer Kugel. Als Erdradius wurde dabei ein Wert von 6 367 km angesetzt. Damit ergibt sich die Länge der Kante mit den Endknoten $(lat1, lon1)$[17] und $(lat2, lon2)$ in Kilometern anhand der nachfolgenden Rechenschritte[18]:

$$\begin{aligned} \Delta lat &:= lat2 - lat1 \\ \Delta lon &:= lon2 - lon1 \\ länge &:= 6\,367\text{km} \times 2 \times \arcsin\left(\sqrt{\sin^2(\Delta lat/2) + \cos(lat1) \times \cos(lat2) \times \sin^2(\Delta lon/2)}\right). \end{aligned} \tag{3.1}$$

Über die Interpretation von Einbahnstraßen OSM-Wege können nur auf Grundlage ihrer Tags als Einbahnstraßen erkannt werden. Einen Überblick hierzu liefert das OpenStreetMap-Wiki[19]. Demnach ist das Befahren eines Weges nur in Richtung in der Reihenfolge seiner zugehörigen Knoten zulässig, wenn der Weg einen Tag mit dem Schlüssel „oneway“ und einen der folgenden Werte besitzt: „yes“, „true“ oder „1“. Eine Einbahnstraße in Gegenrichtung wird üblicherweise mit den Werten „reverse“ oder „-1“ angegeben.

[16] Zwar erlaubt dieses Datenmodell prinzipiell, dass sowohl in Spalte `OkToGoTo1` als auch in Spalte `OkToGoTo2` kein Flag gesetzt und bei der Basiskante damit eigentlich die Fahrt in *beide* Richtungen verboten ist. Doch die übliche OSM-Notation für Einbahnstraßen besteht darin, einen OSM-Weg um das Tag „oneway=yes“ zu erweitern. Da ein Schlüssel eindeutig ist, kann ein OSM-Weg auch nur einen Tag mit Schlüssel „oneway“ besitzen. In dieser Hinsicht kann es also nicht zu der oben beschriebenen Situation einer unbefahrbaren Basiskante kommen.

[17] *lat* steht hier für *Latitude*, also geographische Breite und *lon* für *Longitude*, also geographische Länge

[18] gemäß der *Haversine-Formel* bei `http://www.movable-type.co.uk/scripts/gis-faq-5.1.html`
Die Quelle erweitert das Argument der Arkussinus-Funktion in der vierten Zeile als das Minimum aus 1 und $\sqrt{a}$. Hierdurch sollen Rundungsfehler vermieden werden, die entstehen können, wenn die beiden betrachteten Punkte sich auf beinahe entgegengesetzten Seiten der Erde befinden. Dieser Fall tritt im Rahmen der hier beschriebenen Berechnung der Kantenlängen jedoch nicht auf.
Zu beachten ist weiterhin, dass die Formel die eingehenden Breiten- und Längengrade im Bogenmaß benötigt.

[19] `http://wiki.openstreetmap.org/w/index.php?title=Key:oneway&oldid=761672`

Zusätzlich gelten manche Highway-Sorten implizit als Einbahnstraße, so dass bei ihnen häufig der entsprechende Tag fehlt. Das markanteste Beispiel hierfür bilden OSM-Wege, die als „highway=motorway“ notiert sind. In der zusammenfassenden Tabelle 3.6, S. 85, am Ende dieses Kapitels befindet sich ein Überblick über die Anzahl der so ermittelten Basiskanten, die nur in eine Richtung befahrbar sind (Zeilen *„davon Pfeile“* und *„Anteil Pfeile“*).

Über die Kantenkategorien In Hinsicht auf die Suche nach schnellsten Wegen muss die Geschwindigkeit, mit der ein Fahrzeug über eine Straße fahren darf, zur Verfügung stehen. Aber nur ein geringer Teil der OSM-Wege der für diese Arbeit zugrunde liegenden Karten ist mit entsprechenden Informationen versehen.

Schätzung der Höchstgeschwindigkeiten

Wie in Tabelle 3.2 ersichtlich, besitzt in Deutschland ein knappes Viertel aller Wege Tags zur erlaubten Höchstgeschwindigkeit; in ganz Europa sind es nur etwa 11 %. Die Daten wurden mithilfe der nachfolgenden SQL-Anfrage ermittelt und beinhalten reguläre Ausdrücke, damit sie möglichst viele verschiedene sinnvolle Varianten des „`maxspeed`“-Tags einbeziehen[20]:

```
1 SELECT count(*)
2 FROM ways
3 WHERE tags REGEXP 'maxspeed==(\d+)((.*km/h.*)|(.*mph.*)|(.*kmh.*)|(.*kph.*)|(\s*))'
```

Listing 3.3: SQL-Anfrage zur Ermittlung der Anzahl der OSM-Wege mit Informationen zur Höchstgeschwindigkeit

	Niedersachsen	Deutschland	Europa
Anzahl OSM-Wege mit HG	46 809	630 007	1 290 177
Anteil OSM-Wege mit HG	16,4 %	24,1 %	11,1 %

Tabelle 3.2: Angaben zur Höchstgeschwindigkeit (HG) bei OSM-Wegen

Um die Karten also für die Suche nach schnellsten Wegen aufzubereiten, müssen die meisten Höchstgeschwindigkeiten geschätzt werden. Für diese Arbeit erfolgt dies nach Straßenkategorie gemäß OSM-Tag und dem Land, in dem der betreffende Straßenabschnitt liegt.

Länderzuordnung der Basiskanten

Die Identifikation der Straßenkategorie ist dabei einfach, da die Karten nach der Datenfilterung[21] ohnehin nur aus den in Listing 3.1, S. 62, angegebenen Highway-Sorten bestehen. Da aber die Geschwindigkeitsbegrenzungen pro Straßenkategorie je nach Land unterschiedlich sind, muss jede Basiskante noch einem Land zugeordnet werden, zu dem sie gehört. Für diesen Zweck wurden die geographischen Positionen der Landesgrenzen der Firma Cloudmade[22] verwendet. Das Unternehmen bietet diese als Datei-Download

[20] Die hier verwendete Syntax für reguläre Ausdrücke ist für die Verwendung in einer H2-Datenbank spezifisch.
[21] siehe Abschnitt 3.1, S. 62
[22] `http://downloads.cloudmade.com/`

an, der seinerseits für die weitere Verwendung mit Osmosis geeignet ist[23].

Aus Speicher- und Performance-Gründen ist es zweckmäßig, die Kantenattribute für die Höchstgeschwindigkeit, Landeszugehörigkeit und Straßenkategorie zusammenzufassen. Tabelle `SpeedLimitedCountryHighways` dient hier als Container für alle verschiedenen Datentripel der zugrunde liegenden Karte. So wird in der Basiskanten-Tabelle in Spalte `SpeedLimitedCountryHighwayId` nur noch ein einziger Eintrag referenziert. Aus Tabelle 3.6, S. 85, (Zeile *„Basiskanten-Erzeugung“*) lässt sich erahnen, wie viel Speicher dadurch eingespart werden kann im Vergleich zu drei einzelnen Spalten für Länderzugehörigkeit, Straßenkategorie und Höchstgeschwindigkeit jeder einzelnen Basiskante.

Durchschnittsgeschwindigkeiten

In der Realität kommt es vermutlich nur selten vor, dass bei einer Autofahrt jeder Streckenabschnitt mit der zulässigen Höchstgeschwindigkeit zurückgelegt wird. Die Durchschnittsgeschwindigkeit liegt im Allgemeinen deutlich darunter. Sie ist jedoch ausschlaggebend für die tatsächliche Fahrtdauer. Kantenkategorien tragen diesem Umstand Rechnung, indem sie eine Durchschnittsgeschwindigkeit für jede Kombination aus Highway-Sorte und Land definieren.

Darüber hinaus ermöglichen sie, Durchschnitts- und Höchstgeschwindigkeiten für mehr als nur eine Fahrzeugart effizient zu hinterlegen. Dies ist besonders in Hinsicht auf die Anwendung der Wegsucheverfahren als Bestandteil von Entscheidungsunterstützungssystemen in der Praxis sinnvoll. Denn in vielen Fällen müssen hier heterogene Fahrzeugflotten betrachtet werden. Auch die empirische Studie zur Aufdeckung von Konsolidierungspotential bei der IN tIME Express Logistik GmbH in Kapitel 7 benötigt diese Flexibilität.

[23] `http://support.cloudmade.com/answers/downloads`

3.5 Einbezug von Abbiegebeschränkungen

In Abschnitt 2.2.1 (S. 42 f.) wurde beschrieben, wie bei OpenStreetMap im Allgemeinen Abbiegebeschränkungen modelliert sind. Dieser Abschnitt erklärt, wie diese Informationen in das Datenbankmodell überführt werden. Die nachfolgenden Relationen-Tags finden dabei Berücksichtigung:

- restriction=only_left
- restriction=no_left_turn
- restriction=only_right_turn
- restriction=no_right_turn
- restriction=only_straight_on
- restriction=no_straight_on.

Bei genauerer Betrachtung zeigt sich, dass einzig eine Unterscheidung nach „restriction=only_X“ und „restriction=no_X“ nötig ist, wobei „X“ für die Richtung steht, auf die sich die Abbiegebeschränkung bezieht. Wichtig ist also nur, zwischen Verboten und Geboten zu trennen. Denn zu den Elementen einer Relation, die eine Abbiegebeschränkung darstellt, gehören neben dem hierfür notwendigen „restriction“-Tag grundsätzlich zwei OSM-Wege mit den Rollen *from* und *to* sowie ein OSM-Knoten mit der Rolle *via*. Die Richtung („left“, „right“ oder „straight“) ist dabei nebensächlich und bisweilen sogar mehrdeutig. So ist beispielsweise bei einer Kreuzung mit mehr als vier Straßen oder mehreren annähernd parallel adjazenten Straßen unklar, auf welche sich ein „no_right_turn“-Verbot bezieht. Die Rollen schaffen hier Abhilfe.

Um mit einem Graphen mit Abbiegebeschränkungen arbeiten zu können, müssen diese aus dem OSM-Modell extrahiert und übersetzt werden. Ziel ist es, sie als sortiertes Knoten-Tripel aus Startknoten, Kreuzungsknoten und Zielknoten angeben zu können. Dabei müssen der Start- und Zielknoten jeweils über genau eine Basiskante mit dem Kreuzungsknoten verbunden sein. Zusätzlich muss vermerkt werden, ob es sich um ein Abbiegeverbot oder -gebot handelt. Dafür werden in der Datenbank zwei neue Tabellen angelegt[24]: `TURNRESTRICTION_NO` und `TURNRESTRICTION_ONLY`. Sie besitzen jeweils drei Spalten, nämlich `FromNodeId`, `ViaNodeId` und `ToNodeId`, für das beschriebene Knoten-Tripel.

Verfahren 3, S. 76, beschreibt schematisch, wie die Überführung der Abbiegebeschränkungen vom OSM- ins Datenbank-Modell erfolgt. Dazu wird über alle relevanten OSM-Relationen iteriert. Bei den OSM-Wegen mit der Rolle *from* muss dann der letzte Knoten identifiziert werden, der vor dem Kreuzungsknoten (*via*) liegt. Für Wege mit der *to*-Rolle ist spiegelbildlich der erste Knoten *nach* dem Kreuzungsknoten gesucht. Das hier beschriebene Verfahren vereinfacht diese Suche, indem es auf die in Abschnitt 3.4 beschriebenen Basiskanten zurückgreift. Im Pseudocode sind die Datenfelder einer Basiskante als Attribute notiert.

[24] für das vollständige Datenbankschema siehe Abbildung 3.10, S. 84

Verfahren 3 Extraktion der Abbiegebeschränkungen als Knotentripel

Gegeben: Eine OpenStreetMap-Datenbank mit Knoten N, Wegen W, Relationen *Rel* sowie erzeugten Basiskanten *BK*

```
1: for all rel ∈ Rel do
2:   fromKnoten := NULL          ▸ Platzhalter für den Startknoten einer Abbiegebeschränkung
3:   viaKnoten := NULL           ▸ Platzhalter für den Knoten, an dem eine Abbiegebeschränkung existiert
4:   toKnoten := NULL            ▸ Platzhalter für den End knoten einer Abbiegebeschränkung
5:   if rel besitzt „restriction=only_X“-Tag tag then
6:     if tag verweist mit Rolle via nicht auf einen OSM-Knoten then
7:       goto: 1
8:     end if
9:     if tag verweist mit Rolle from auf einen OSM-Knoten then
10:      fromKnoten := from
11:    else if tag verweist mit Rolle from auf einen OSM-Weg then
12:      Suche in BK nach einer Basiskante mit Endknoten via und wayId = from
13:      if Es existiert genau eine solche Basiskante then
14:        fromKnoten := anderer Endknoten der gefundenen Basiskante
15:      else
16:        goto: 1              ▸ Mehrdeutigkeit oder Datenfehler; d. h., Abbiegebeschränkung wird ignoriert
17:      end if
18:    end if
19:    if tag verweist mit Rolle to auf einen OSM-Knoten then
20:      toKnoten := to
21:    else if tag verweist mit Rolle to auf einen OSM-Weg then
22:      Suche in BK nach Basiskante mit Endknoten via und wayId = to
23:      if Es existiert genau eine solche Basiskante then
24:        toKnoten := anderer Endknoten der gefundenen Basiskante
25:      else
26:        goto: 1              ▸ Mehrdeutigkeit oder Datenfehler; d. h., Abbiegebeschränkung wird ignoriert
27:      end if
28:    end if
29:    Speichere Abbiegegebot als Tripel aus (fromKnoten, viaKnoten, toKnoten)
30:  else if rel besitzt „restriction=no_X“-Tag then
31:    Verfahre analog zum Fall „restriction=only_X“
32:    Speichere Abbiegeverbot als Tripel aus (fromKnoten, viaKnoten, toKnoten)
33:  end if
34: end for
```

Ergebnis: Alle Abbiegegebote und -verbote wurden als Knoten-Tripel gespeichert.

Datenqualitätsmängel und ihre Behandlung Das OSM-Kartenmaterial weist für dieses Vorgehen an manchen Stellen Mängel auf. Grundsätzlich sollten zwei OSM-Wege sich ausschließlich an einem ihrer Endknoten kreuzen. Dies wird aber nicht vom OSM-Datenmodell forciert, so dass es zu Mehrdeutigkeiten bei der Interpretation von Abbiegebeschränkungen kommen kann. Abbildung 3.7 illustriert dies anhand eines einfachen Beispiels. Auf dem Kartenmaterial ist es für den menschlichen Betrachter eindeutig, worauf sich das Abbiegeverbot bezieht. Ein Computerprogramm müsste aber noch zusätzlich zwischen den möglichen Abbiege-Richtungen unterscheiden, um zu einer plausiblen Interpretation zu gelangen, und selbst dann wären noch Szenarien mit Mehrdeutigkeiten an Kreuzungen mit vielen angrenzenden Straßen möglich. Verfahren 3, S. 76, ignoriert daher derartige Abbiegebeschränkungen, da sie darüber hinaus nur selten sind (Zeilen 16 und 26).

Mehrdeutigkeiten

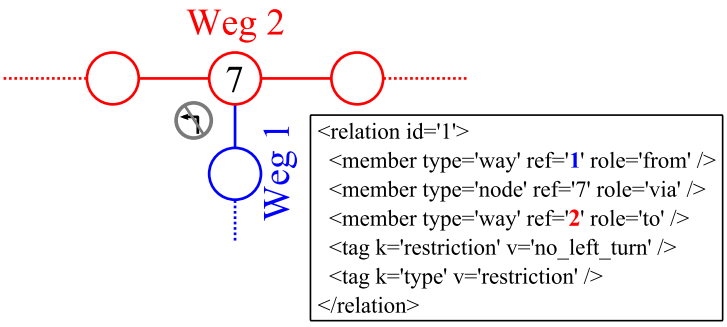


Abbildung 3.7: Mehrdeutigkeit bei einem Abbiegeverbot
Da der Kreuzungsknoten 7 kein Endknoten von OSM-Weg 2 ist, kann die Interpretation des Abbiegeverbots nur mithilfe geometrischer Analyse erfolgen. Ferner entspricht das dargestellte Szenario ohnehin nicht der bevorzugten Modellierung bei OSM. Demnach sollte Weg 2 besser bei Knoten 7 geteilt werden.

Sinnlose Abbiegebeschränkungen

In den Karten finden sich mitunter auch sinnlose Abbiegebeschränkungen wie beispielsweise ein Abbiegegebot bei einem Knoten, der ohnehin nur mit zwei Kantenobjekten verbunden ist. Aus praktischen Gründen werden diese temporär mitgespeichert, da in diesem Präprozess-Schritt nicht das gesamte Kartenmaterial im Speicher gehalten wird und somit der Knotengrad eines Knotens nicht ermittelt werden kann. Stattdessen werden diese Abbiegebeschränkungen beim nachfolgenden Präprozess-Schritt *Kantenobjekt-Verbindung* erkannt – und infolgedessen ignoriert[25].

[25] siehe Abschnitt 3.6 auf Seite 83, Stichwort „Auswirkungen auf Abbiegebeschränkungen“

Kehrtwenden

Keinen Mangel im Sinne dieser Arbeit stellen Abbiegeverbote dar, bei denen die Rollen *from* und *to* auf denselben Weg verweisen. Denn derartige Kehrtwenden werden von den hier implementierten Wegsucheverfahren ohnehin vermieden[26]. Bei der Extraktion der Abbiegeverbote werden sie daher ebenfalls ignoriert.

Eine abschließende Übersicht über die gefundenen Tags für Abbiegebeschränkungen in den verschiedenen Straßenkarten liefert Tabelle 3.3. Fett hervorgehoben sind diejenigen Beschränkungen, die beim Kartenimport der OpenStreetMap-Karten letztendlich Verwendung finden.

	Niedersachsen	Deutschland	Europa
Anzahl Tags „restriction=only_X“	2 736	25 982	40 744
Anzahl Tags „restriction=no_X“	571	7 961	20 781
Summe	3 307	33 943	61 525
mehrdeutige Beschränkungen	7	83	375
Beschr. mit Bezug auf obsolete Elemente[†]	182	1 997	4 311
extrahierte Beschränkungen	**3 048**	**31 679**	**55 012**
sonstige (nicht verwendete)[‡]	70	184	1 827

[†] Gemeint sind OSM-Wege und -Knoten, die bei der Nebencluster-Entfernung (siehe Abschnitt 3.3) gelöscht wurden.

[‡] Hierunter fallen auch unvollständige Tags und solche, die auf nicht vorhandene Wege oder Knoten verweisen.

Tabelle 3.3: Abbiegebeschränkungen in den OSM-Karten

3.6 Kantenobjekt-Verbindung

OSM-Straßenkarten besitzen im Allgemeinen eine inhomogene Informationsdichte. Während an manchen Stellen besonders detailliert kartographiert wurde, sind an anderen Stellen nur rudimentär die Straßen aufgezeichnet. Ein Extrembeispiel zeigt Abbildung 3.8, S. 79. Die vier hervorgehobenen OSM-Knoten westlich des Autobahndreiecks „Ahlhorner Heide“[27] liegen nur wenige Zentimeter auseinander. Ein derart hoher Detailgrad ist für die meisten Navigationsanwendungen nicht notwendig.

[26] siehe auch Abschnitt 6.2 auf Seite 203, Stichwort „Wenden vermeiden“

[27], bei dem die Autobahnen 1 und 29 in der Nähe von Cloppenburg aufeinander treffen,

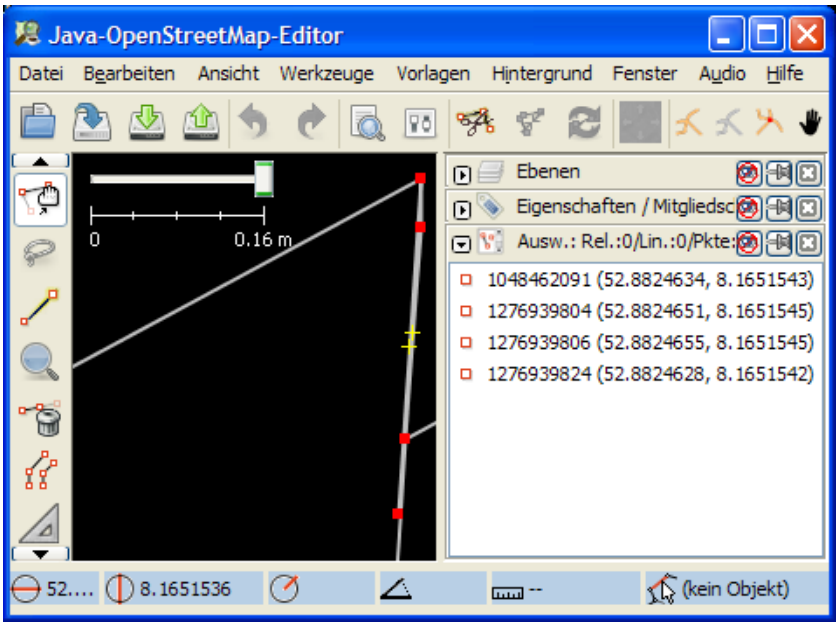


Abbildung 3.8: Beispiel für besonders kurze Basiskanten (Screenshot aus JOSM)
Wie aus dem Maßstab oben links im Bild erkennbar ist, liegen die vier rot markierten OSM-Knoten nur wenige Zentimeter auseinander.

Nicht zuletzt aus diesem Grund bleiben große Kartenausschnitte wie die Deutschland- oder Europa-Karte trotz Datenfilterung ohne weitere Bearbeitung für aktuelle Desktop-Computer noch schwer bis unmöglich zu handhaben. Dies gilt insbesondere, wenn die komplette Karte – wie bei den Software-Programmen für diese Arbeit – im Arbeitsspeicher eines Computers vorliegen soll. Die nachfolgende Überlegung veranschaulicht dies.

Abschätzung des Speicherbedarfs Eine untere Schranke für den Speicherbedarf einer – wie in dieser Arbeit verwendeten – objektorientierten Modellierung lässt sich anhand von Tabelle 3.4, S. 80, grob abschätzen: Pro OSM-Knoten müssen dessen Id, Längen- und Breitengrad sowie ein Verweis auf seine adjazenten Kantenobjekte hinterlegt sein. Die Knoten-Ids benötigen seit dem 9. Februar 2013[28] einen Long-Wertebereich; Längen- und Breitengrad sind Fließkommazahlen. Werden pro Knoten lediglich zwei adjazente Kantenobjekte angenommen, ergibt sich für die 86 834 164 Knoten der Europakarte auf diese Weise bereits ein Speicherbedarf von 3 473 366 560 Bytes[29]; also beinahe 3,5 GB.

[28] vgl. `http://wiki.openstreetmap.org/w/index.php?title=64-bit_Identifiers&oldid=890112`
[29] $86\,834\,164 \times (\text{Id} + \text{Breitengrad} + \text{Längengrad} + 2 \times \text{Kanten-Referenz} + \text{Objekt-Overhead}) = 86\,834\,164 \times (8 + 8 + 8 + 2 \times 4 + 8) = 3\,473\,366\,560$

Datentyp	Speicherbedarf (in Bytes)
long	8
double	8
Referenz	4
Overhead pro Objekt	$8^{\dagger}$

† Bezugsgröße ist ein 32-Bit-Betriebssystem; für 64-Bit-Systeme verdoppelt sich dieser Wert (vgl. [Wilson u. Kesselman, 2000]).

Tabelle 3.4: Speicherbedarf ausgewählter Java-Datentypen
Quelle: [Wilson u. Kesselman, 2000]

Für jede Basiskante müssen darüber hinaus ihre Id, ihre beiden Randknoten, ihre Länge sowie ihre Kantenkategorie[30] vorliegen. Die 90 726 380 Basiskanten der Europakarte erfordern daher mindestens noch weitere 2 903 244 160 Bytes[31], also weitere knappe drei GB Arbeitsspeicher. Diese in Summe mehr als sechs GB Speicherbedarf decken jedoch nur das interpretierte OSM-Kartenmaterial ab. Moderne Wegsucheverfahren[32] reichern den Graphen, mit dem sie arbeiten, in einem Vorberechnungsschritt mit weiteren Zusatzinformationen an. Diese können ihrerseits teilweise sogar mehr Speicher benötigen als der zugrunde liegende Graph selbst. Im praktischen Einsatz wird darüber hinaus noch ein Index für die Knoten benötigt, um zu einem gegebenen Längen- und Breitengrad effizient den nächsten OSM-Knoten zu finden, und auch die Abbiegebeschränkungen müssen kodiert werden.

Der in diesem Abschnitt vorgestellte Arbeitsschritt verringert das Volumen der Kartendaten daher noch einmal, indem die in Abschnitt 3.4 beschriebenen Basiskanten zusammengefasst werden. Um möglichst viel Speicher zu sparen, werden Knoten mit genau zwei adjazenten Kantenobjekten aus der Karte entfernt und ihre Kantenobjekte zu einem einzigen zusammengefasst. Der Vorgang ähnelt inhaltlich der in Abschnitt 5.1, S. 132 ff., beschriebenen Knotenkontraktion, die im Rahmen der Vorberechnungen einer Contraction Hierarchy durchgeführt wird.

Eine Ausnahme von der Vorgehensweise bilden dabei die Nachbarknoten von Kreuzungsknoten, an denen eine Abbiegebeschränkung existiert. Dies erleichtert die Fehlersuche im Zusammenhang mit Abbiegebeschränkungen.

Natürlich dürfen nur Kantenobjekte mit kompatiblen Einbahnstraßen-Regelungen verbunden werden. Darüber hinaus werden Kantenobjekte mit unterschiedlicher Kantenkategorie ebenfalls nicht verbunden, da ansonsten die Abschätzung der Fahrtdauer entlang des entstehenden Kantenobjekts an Genauigkeit verlöre.

Verfahren 4, S. 81, skizziert das Vorgehen zur Kantenobjekt-Verbindung explizit für gerichtete Graphen in der mathematischen Formulierung. Daher arbeitet das Verfahren ausschließlich mit *Pfeilen*. In der Implementierung[33] werden diese in Form von Kanten-

[30] siehe Seite 73, Stichwort „Über die Kantenkategorien“

[31] $90\,726\,380 \times (\text{Id} + 2 \times \text{Knoten-Referenz} + \text{Länge} + \text{Kantenkategorie-Referenz} + \text{Objekt-Overhead}) = 90\,726\,380 \times (4 + 2 \times 4 + 8 + 4 + 8) = 2\,903\,244\,160$

[32] siehe Kapitel 4, 5 und 6

[33] siehe hierzu auch Abbildung 1.4, S. 33

objekten gespeichert. Die übrig gebliebenen Kantenobjekte – also auch solche, die nicht verbunden werden konnten – werden in einer eigenen Tabelle der Datenbank gespeichert. So werden die Zwischenstände der Datenaufbereitung nicht überschrieben und stehen für spätere Analysen zur Verfügung. Das *Löschen* in Zeile 20, Verfahren 4 ist hier also nur temporär gemeint.

Verfahren 4 Pfeil-Verbindung

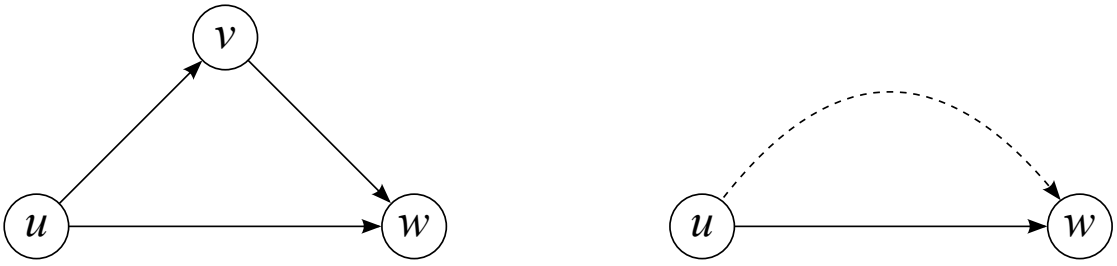
Gegeben: Ein gewichteter, gerichteter Graph $G = (V, E, c)$

1: $knotenListe \coloneqq \{v \in V \mid g_v^+ = 1 \wedge g_v^- = 1\}$ ▸ Initialisierung
2: **while** $knotenListe \neq \emptyset$ **do**
3: Entferne beliebigen Knoten v aus $knotenListe$
4: **if** $g_v^+ \neq 1$ **oder** $g_v^- \neq 1$ **then** ▸ Fall möglich durch Löschen paralleler Pfeile in Zeile17
5: **goto:** 2
6: **end if**
7: Lösche alle Abbiegebeschränkungen bei v ▸ Diese sind ohnehin sinnlos.
8: **if** Es existiert eine Abbiegebeschränkung an einem Nachbarknoten von v **then**
9: **goto:** 2
10: **end if**
11: Seien (u, v) und (v, w) die beiden zu v adjazenten Pfeile
12: **if** (u, v) und (v, w) besitzen verschiedene Kategorien **then**
13: **goto:** 2
14: **end if**
15: Erzeuge neuen Pfeil (u, w) mit $c(u, w) = c(u, v) + c(v, w)$
16: **if** Es existiert bereits ein Pfeil von u nach w **then** ▸ Behandlung paralleler Pfeile
17: Lösche den längeren der beiden Pfeile (bei gleicher Länge lösche den neuen)
18: Entferne alle Abbiegebeschränkungen, die sich auf den soeben gelöschten Pfeil beziehen
19: **end if**
20: Lösche v aus V und (u, v) sowie (v, w) aus E
21: Ersetze in allen Abbiegebeschränkungen (u, v) und (v, w) durch (u, w)
22: **if** $g_u^+ = 1 \wedge g_u^- = 1$ **und** $u \notin knotenListe$ **then**
23: Füge u zu $knotenListe$ hinzu
24: **end if**
25: **if** $g_w^+ = 1 \wedge g_w^- = 1$ **und** $w \notin knotenListe$ **then**
26: Füge w zu $knotenListe$ hinzu
27: **end if**
28: **end while**

Ergebnis: Die Pfeil-Verbindung des Graphen G ist abgeschlossen.

Im Verlauf des Verfahrens kommt es immer wieder vor, dass Kantenobjekte, die aufgrund einer vorangegangenen Verbindung zweier anderer Kantenobjekte entstanden, ihrerseits in einer Verbindung aufgehen. Es werden also nicht alle neu erzeugten Kantenobjekte tatsächlich gespeichert.

Behandlung paralleler Pfeile Durch das Verbinden der Pfeile entstehen an einigen Stellen parallele Pfeile. Abbildung 3.9 zeigt hierfür ein einfaches Beispiel zur Verdeutlichung. Parallele Pfeile führen jedoch bei den Suchverfahren zu Mehrdeutigkeiten. Aus diesem Grund – und um den benötigten Speicherbedarf weiter zu reduzieren – wird bei einer Pfeile-Verbindung nur der *kürzere* von zwei parallelen Pfeile behalten und der andere verworfen.



(a) Die Pfeile von Knoten v sollen verbunden werden.

(b) Der gestrichelt dargestellte verbundene Pfeil ist parallel zum Original-Pfeile (u, w).

Abbildung 3.9: Entstehung paralleler Pfeile durch Pfeil-Verbindung

Dies ist daher der erste Präprozess-Schritt, bei dem die verwendete Metrik[34] eine Rolle spielt: Zielt die Vorbereitung auf die Suche nach *schnellsten* Wegen ab, wird hier derjenige Pfeil verworfen, der langsamer traversiert wird. Die im Rahmen dieser Arbeit verwendeten Geschwindigkeitsprofile zeigt Tabelle 3.5, S. 83. Sämtliche nachfolgenden Statistiken zur den Karten beziehen sich – sofern nicht anders erwähnt – auf die Verwendung der Metrik „schnellste Wege für Transporter“. Die Entscheidung für diese Metrik basiert auf der Beobachtung, dass in über 73 % aller straßengebundenen[35] Transporte der in dieser Arbeit verwendeten Vergangenheitsdaten ein Fahrzeug aus der „Transporter“-Kategorie zum Einsatz kam.

[34] zur Begriffsdefinition siehe Seite 31

[35] In den Vergangenheitsdaten befinden sich auch ein paar Flugtransporte.

Straßenkategorie	Auto	Transporter	LKW<7to	LKW>7to	Gespann
`hw.motorway`	90	90	70	60	60
`hw.motorway_link`	70	70	65	60	60
`hw.trunk`	80	75	65	60	60
`hw.trunk_link`	60	60	65	60	60
`hw.primary`	80	80	70	60	60
`hw.primary_link`	70	70	65	60	60
`hw.secondary`	50	50	50	50	50
`hw.secondary_link`	50	50	50	50	50
`hw.tertiary`	40	40	40	40	40
`hw.tertiary_link`	30	40	40	40	40
`hw.residential`	30	30	25	25	25
`hw.unclassified`	30	30	30	30	30
`route.ferry`	5	5	5	5	5
`hw.service`	10	10	10	10	10

Tabelle 3.5: Geschwindigkeitsprofile in dieser Arbeit
Alle Angaben in km/h; „hw“ steht abkürzend für „highway“

Auswirkungen auf Abbiegebeschränkungen Die Verbindung zweier Pfeile (u, v) und (v, w) kann eine Abbiegebeschränkung bei u oder w obsolet machen. Dies passiert, wenn u oder w nach der Verbindung aufgrund der Entfernung eines parallelen Pfeils einen Knotengrad von zwei besitzen (siehe Zeile 18 in Verfahren 4, S. 81). Zusätzlich muss der Bezug von allen Abbiegebeschränkungen aktualisiert werden, die sich auf einen der verbundenen Pfeile beziehen (siehe Zeile 21). Zusammen mit sinnlosen Abbiegebeschränkungen (siehe Zeile 7)[36] wurden auf diese Weise bei Verwendung der Schnellste-Wege-Metrik bei der Niedersachsenkarte 54, bei der Deutschlandkarte 449 und bei der Europakarte 609 Abbiegebeschränkungen entfernt.

Abbildung 3.10 zeigt das vollständige Datenbankschema, das auch die Kantenobjekt-Verbindung beinhaltet. In Tabelle `MERGEDEDGES` sind die verbundenen Kantenobjekte gespeichert. Die Abbiegebeschränkungen, die sich auf diese neuen Kantenobjekte beziehen, befinden sich in der Tabelle `MERGEDEDGESTURNRESTRICTIONS_ONLY` und ihrem Pendant `MERGEDEDGESTURNRESTRICTIONS_NO`.

[36] beschrieben auf Seite 77

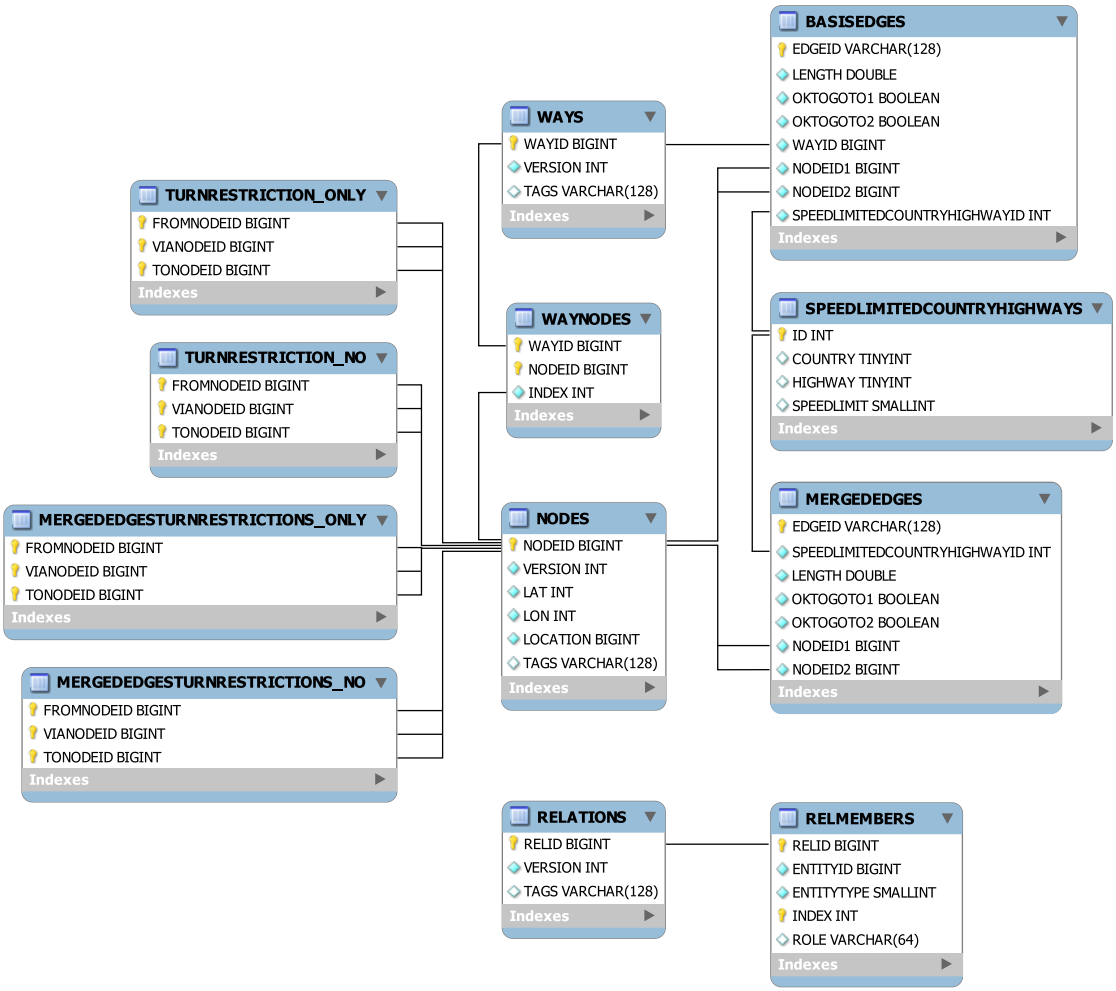


Abbildung 3.10: Vollständiges Datenbankschema
Gelbe Schlüssel symbolisieren den Primärschlüssel der jeweiligen Datenbanktabelle; Verbindungslinien zwischen den Tabellen stellen Fremdschlüsselbezüge dar.

Eine zusammenfassende Übersicht über die verbleibenden Kantenobjekte und Knotenobjekte nach jedem Arbeitsschritt, der in diesem Kapitel vorgestellt wurde, zeigt Tabelle 3.6, S. 85. Die Werte für die drei in dieser Arbeit wiederholt verwendeten Karten sind hier einzeln aufgeführt.

Nach Schritt		Niedersachsen[I]	Deutschland[II]	Europa[III]
Original-Download	Knoten	9 347 773	99 123 541	617 409 999
	Wege	1 441 942	14 782 071	73 678 784
	Relationen	23 313	231 451	881 873
	Kartengröße (MB)	114	1 331,2	7 065,6
Daten-Filterung	Knoten	1 440 399	15 082 153	88 196 411[IV]
	Ersparnis[V]	84,6 %	84,8 %	85,7 %
	Wege	288 756	2 644 305	11 761 176[IV]
	Ersparnis[V]	80,0 %	82,1 %	84,0 %
	Relationen	23 313	231 451	829 562[IV]
	Kartengröße (MB)	18,2	182	886[IV]
Nebencluster-Entfernung	Knoten	1 419 901	14 898 304	86 834 164
	Ersparnis[V]	1,4 %	1,2 %	1,5 %
	Wege	284 781	2 609 554	11 604 948
	Ersparnis[V]	1,4 %	1,3 %	1,3 %
Basiskanten-Erzeugung	Basiskanten	1 503 319	15 611 744	90 726 380
	davon Pfeile[VI]	114 007	1 279 024	8 149 077
	Anteil Pfeile[VI]	7,6 %	8,2 %	9,0 %
	Kantenkategorien[VII]	167	501	3 888
Kantenobjekt-Verbindung[VIII]	Knoten	362 040	3 282 166	15 462 474
	Ersparnis[V]	74,5 %	78,0 %	82,2 %
	Kantenobjekte	450 809	4 088 311	19 804 559
	Ersparnis[V]	70,0 %	73,8 %	78,2 %
	davon Pfeile[VI]	29 900	342 954	2 359 290
	Anteil Pfeile[VI]	6,6 %	8,4 %	11,9 %

[I] Stand: 11. Mai 2012 03:34 Uhr
[II] Stand: 10. Mai 2012 03:30 Uhr
[III] Stand: 19. Dezember 2012 01:53 Uhr
[IV] inklusive zusätzlicher Bounding-Box-Filterung:
nord-westlichster Punkt bei Lat = 71,2° N, Lon = 24,7° W;
süd-östlichster Punkt bei Lat = 34,5° N, Lon = 33,1° E (siehe Abbildung 3.2, S. 64)
[V] im Vergleich zum vorangegangenen Bearbeitungsschritt
[VI] d. h. Kantenobjekte, die nur als Pfeile und nicht als ungerichtete Kanten zu interpretieren sind
[VII] Eine Kantenkategorie umfasst ein eindeutiges Tripel aus Länderzugehörigkeit, Straßenkategorie und Höchstgeschwindigkeit für eine Kante (siehe hierzu auch Abschnitt 3.4 auf Seite 73).
[VIII] bei Verwendung der Schnellste-Wege-Metrik

Tabelle 3.6: Kennzahlen der verwendeten Straßenkarten

4 Wegsucheverfahren für digitale Straßenkarten

Inhalt

In diesem Kapitel werden Verfahren zur Suche nach optimalen Wegen in Graphen im Allgemeinen und in digitalen Straßenkarten im Speziellen vorgestellt. Bevor in Abschnitt 4.2 hierarchische Wegsucheverfahren vorgestellt werden, stehen zunächst die nicht-hierarchischen Wegsucheverfahren im nachfolgenden Abschnitt im Vordergrund. Auf ihnen basieren die hierarchischen Weiterentwicklungen. Unterschieden wird zwischen Verfahren, die als Eingabe lediglich einen Graphen benötigen (Abschnitt 4.1.1) und solchen, die neben dem Eingangsgraphen noch weitere Informationen benötigen, die sie über einen meist automatisierbaren Präprozess erhalten (Abschnitt 4.1.2). Das in dieser Arbeit in weiten Teilen zugrunde liegende Verfahren der *Contraction Hierarchies* wird gesondert im nachfolgenden Kapitel behandelt.

Pfeil- vs. knotenbasierte Verfahren

Verfahren zur Suche nach optimalen Wegen in Graphen können sowohl knotenbasiert, d. h., Start und Ziel sind Knoten, als auch pfeilbasiert, d. h., Start und Ziel sind bzw. liegen auf Pfeilen, erfolgen. In diesem Kapitel werden zunächst nur knotenbasierte Verfahren behandelt. Einige ausgewählte pfeilbasierte Verfahren werden später in Kapitel 6 diskutiert.

Häufig können jedoch pfeilbasierte Verfahren für Wegsuchen in knotenbasierte überführt werden: Falls der Startpunkt entlang eines Pfeils liegt, könnte der betroffene Pfeil für die Wegsuche an diesem Punkt in zwei temporäre Pfeile geteilt werden, die an einem temporären Knoten verbunden sind. Analoges gilt für den Endpunkt einer Wegsuche. Manche pfeilbasierten Problemstellungen wie beispielsweise die Einsatzplanung für eine innerstädtische Müllabfuhr lassen sich jedoch nicht auf diese Weise überführen.

Wenn von *optimalen Wegen* gesprochen wird, meint dies zwingend *optimal im Sinne einer Zielfunktion*. Häufig liegt hier die Suche nach kürzesten oder schnellsten Wegen zugrunde. Es sind aber auch beliebige andere Zielfunktionen denkbar. Im Zuge der Elektromobilität und der damit einhergehenden Einführung erster Elektrofahrzeuge könnte etwa die Suche nach einem Weg von Bedeutung sein, der eine möglichst große Fahrzeugreichweite verspricht. Solche *energiesparendsten* oder *verbrauchärmsten* Wege werden zunehmend relevant.

Im Rahmen dieser Arbeit werden die Begriffe *optimale* und *kürzeste* Wege weitgehend austauschbar verwendet. Als Zielfunktion wird – sofern nicht anders erwähnt – die Weglänge optimiert. In Abschnitt 1.3.1, S. 30, wurde bereits erläutert, warum dies in den meisten Fällen keine Einschränkung darstellt.

Einen Überblick über die in diesem Kapitel vorgestellten Verfahren liefert Tabelle 4.1. Da hierarchische Verfahren stets mit Vorberechnungen arbeiten, existieren keine hierarchischen Ad-hoc-Verfahren.

	Ad-hoc-Verfahren	**Verfahren mit Vorberechnungen**
Nicht-Hierarchische	unidirektionale Dijkstra-Suche bidirektionale Dijkstra-Suche A*-Verfahren	Trennlinien-Verfahren ALT-Verfahren Arc-Flags-Verfahren
Hierarchische	—	Highway Hierarchies Highway Node Routing Transit Node Routing Hub-Based Labeling

Tabelle 4.1: Vorschau auf die vorgestellten Wegsucheverfahren

4.1 Nicht-hierarchische Wegsucheverfahren

Dieser Abschnitt beschreibt ausgesuchte Verfahren, bei denen alle Knoten des zugrunde liegenden Graphen bzw. der digitalen Straßenkarte gleichrangig sind. Mit anderen Worten: Weder durch eine Vorbearbeitung noch bei der Wegsuche selbst ist in der Karte hinterlegt, dass manche Knoten oder Pfeile eine besondere Bedeutung haben, die sie von anderen abhebt. Hierarchische Verfahren werden in Abschnitt 4.2 auf Seite 110 ff. behandelt.

4.1.1 Wegsucheverfahren ohne Vorberechnungen

Die Wegsucheverfahren in diesem Abschnitt können direkt auf einen gewichteten, gerichteten Graphen $G = (V, E, c)$ angewendet werden. Abgesehen von den Geokoordinaten beim A*-Verfahren werden neben den Knoten-Pfeil-Relationen sowie natürlich den Pfeilbewertungen keine zusätzlichen Informationen benötigt. Vorgestellt werden hier das uni- und das bidirektionale Dijkstra-Verfahren sowie das A*-Verfahren.

Zur Notation im Pseudocode Eine Gemeinsamkeit der nachfolgend beschriebenen Verfahren ist, dass sie zu jedem Knoten die Länge des bisher kürzesten bekannten Weges zu ihm speichern. Im Pseudocode wird dies objektorientiert als Attribut `distanz` modelliert, so dass $v.distanz$ mit $v \in V$ diesen Wert referenziert. Der Vorgängerknoten entlang des bisher kürzesten bekannten Weges wird ebenfalls an den Knoten gespeichert. Hierzu dient das Attribut `vorgänger`.

4.1.1.1 Das Dijkstra-Verfahren

Ein Standard-Verfahren bei der Wegsuche ist der von Dijkstra [1959] veröffentlichte und nach ihm benannte Algorithmus.[1] Dieser bildet die Grundlage vieler weiterer Ansätze, die auch in dieser Arbeit Erwähnung finden. Ausgehend von einem Startknoten werden in einem zusammenhängenden, gewichteten, gerichteten Graphen die kürzesten Wege zu allen anderen Knoten des Graphen ermittelt.[2] Eine wichtige Anforderung an den Graphen ist, dass er keine negativen Pfeilbewertungen beinhalten darf. Für die Ermittlung kürzester oder schnellster Wege auf einer Straßenkarte stellt dies in der Praxis offensichtlich keine Einschränkung dar.

Im Verlauf des Verfahrens werden zwei Knotenmengen verwaltet: Die erste beinhaltet Knoten, zu denen *mit Sicherheit ein kürzester* Weg gefunden wurde, die andere Knoten, zu denen *überhaupt ein* Weg gefunden wurde. Die erste Menge wird im Folgenden als *Lösungsmenge* bezeichnet; die zweite als *Horizontmenge*, da mit ihr die Kenntnis des Verfahrens über die Struktur des Graphen wächst, sich also bildlich gesprochen der „Erkenntnis-Horizont“ des Verfahrens erweitert.[3] Die Knoten der Horizontmenge wer-

Lösungs- und Horizontmenge

[1] Dantzig [1960] beschrieb erst später unabhängig davon ein analoges Verfahren.

[2] Damit kann das Verfahren selbstverständlich auch bei zusammenhängenden *ungerichteten* Graphen verwendet werden, indem sämtliche ungerichtete Kanten als zwei gegensätzlich ausgerichtete Pfeile interpretiert werden.

[3] In der englischsprachigen Literatur werden die Knoten der Lösungsmenge als *settled nodes* bezeichnet und die Knoten der Horizontmenge als *reached nodes* (vgl. beispielsweise [Bauer u. a., 2008], [Bast u. a., 2009] oder [Geisberger u. a., 2008]).

den stets sortiert vorgehalten. Die Sortierung erfolgt aufsteigend nach der Länge des kürzesten bisher bekannten Weges vom Startknoten zu ihnen.

Das Verfahren hat zu Beginn nur „Kenntnis“ über den angegebenen Startknoten. Dieser wird in der Initialisierungsphase der Horizontmenge hinzugefügt. Anschließend werden immer wieder zwei grundlegende Schritte wiederholt, bis die Horizontmenge schließlich leer ist und das Verfahren terminiert:

Schritt 1: Entfernen des ersten Knotens v aus der Horizontmenge

Schritt 2: Expandieren von v

Da die Horizontmenge sortiert vorliegt, wird bei Schritt 1 stets derjenige Knoten $v \in \mathit{Horizontmenge}$ gewählt, der die Suche am wenigsten weit vom Startknoten entfernt und für den daher gilt:

$$v.\mathit{distanz} = \min\{v'.\mathit{distanz} \mid v' \in \mathit{Horizontmenge}\}. \tag{4.1}$$

Bei Schritt 2 werden alle zu v adjazenten Pfeile *expandiert*, wodurch alle Nachfolgerknoten in die Horizontmenge gelangen, sofern sie nicht bereits Teil der Lösungsmenge sind. Der Weg und die Distanz vom Startknoten zu den so erreichten Knoten werden gespeichert. Wird ein Knoten erreicht, der vorher bereits Teil der Horizontmenge war, existiert ein alternativer Weg vom Startknoten zu diesem Knoten. In diesem Fall wird nur der kürzere der beiden Wege gespeichert. Sollten mehrere gleichwertige kürzeste Wege zwischen zwei Knoten existieren, speichert das Verfahren nur einen von ihnen. Die Suche terminiert, sobald kürzeste Wege vom Startknoten zu allen anderen Knoten bekannt sind.

Verfahren 5, S. 91, zeigt den Dijkstra-Algorithmus in einer Pseudocode-Darstellung. Nach Terminierung des Verfahrens lassen sich die kompletten kürzesten Wege rekursiv, beim Endknoten beginnend, aus den `vorgänger`-Attributen rekonstruieren. Über dieses Attribut können darüber hinaus zu jedem Zeitpunkt während der Suche die Knoten der Horizont- und Lösungsmenge rekursiv bis hin zum Startknoten s in einer Baumstruktur sortiert werden, dem *Suchbaum*. Im Suchbaum bildet s die Wurzel und die Knoten der Horizontmenge die Blätter. Die inneren Knoten des Suchbaums sind aus der Lösungsmenge.[4] Im Suchbaum besitzt jeder Knoten v als *Kinderknoten*, d. h. Nachfolgerknoten im Teilbaum mit Wurzel v, eine Teilmenge aus $\mathcal{N}(v)$. Ein Weg von s zu einem Blattknoten t im Suchbaum beinhaltet sortiert alle Knoten des bislang kürzesten bekannten Weges von s nach t.

Bildung des Suchbaums

[4] In Ausnahmefällen kann auch ein Knoten v aus der Lösungsmenge ein Blatt des Suchbaums sein, nämlich wenn v keine Nachfolgerknoten besitzt oder die kürzesten Wege vom Startknoten zu allen Nachfolgerknoten von v *nicht* über v führen.

Verfahren 5 Knotenbasierter Dijkstra-Algorithmus (1:n-Suche)

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G=(V,E,c)$, Startknoten $s \in V$

```
1:  s.distanz := 0                                                    ▸ Initialisierung
2:  v.distanz := ∞,          v.vorgänger := NULL      ∀v ∈ V \ {s}
3:  Lösungsmenge := ∅,       Horizontmenge := {s}
4:  while Horizontmenge ≠ ∅ do
5:    Wähle v mit v.distanz = min{v'.distanz | v' ∈ Horizontmenge}    ▸ Schritt 1
6:    Entferne v aus Horizontmenge
7:    Nimm v in Lösungsmenge auf
8:    for all v' ∈ N(v) do                                            ▸ Schritt 2
9:      if v' ∈ Lösungsmenge then
10:       goto: 8                          ▸ kürzester Weg zu v' ist bereits bekannt
11:     end if
12:     if v.distanz + c(v,v') < v'.distanz then
13:       v'.vorgänger := v                ▸ (kürzerer) Weg zu v' gefunden
14:       v'.distanz := v.distanz + c(v,v')
15:       if v' ∉ Horizontmenge then
16:         Nimm v' in Horizontmenge auf
17:       end if
18:     end if
19:   end for
20: end while
```

Ergebnis: Kürzeste Wege von s zu allen Knoten $v \in G$ sind rekursiv konstruierbar aus den `vorgänger`-Attributen der Knoten. Für sie gilt: $v.distanz = \text{dist}(s,v)$. Zu Knoten, für die das `vorgänger`-Attribut nicht gesetzt ist, existiert vom angegebenen Startknoten ausgehend kein Weg in G.

In Zeile 8 aus Verfahren 5 werden durch die Definition der Nachfolgerknoten[5] nur solche Knoten in die Horizontmenge aufgenommen, die über einen Pfeil in zulässiger Fahrtrichtung erreicht werden können. Auf diese Weise werden Einbahnstraßen berücksichtigt.

Aufwandsabschätzung

Das Verfahren in dieser Form expandiert jeden Knoten eines zusammenhängenden Graphen. Bei jedem Expansionsschritt werden alle Nachfolgerknoten des zu expandierenden Knotens betrachtet. Da dies theoretisch alle anderen Knoten des Graphen sein können, ergibt sich im ungünstigsten Fall eine Laufzeit von $\mathcal{O}(|V|^2)$.

In der Literatur wird jedoch häufig eine auf Fredman u. Tarjan [1987, S. 610] zurück verfolgbare Aufwandsabschätzung von $\mathcal{O}(|V|\log|V|+|E|)$ bei Verwendung so genannter *Fibonacci Heaps* als Datenstruktur für die sortierten Horizontknoten angegeben. Der Unterschied ergibt sich erst bei näherer Betrachtung, da Fredman u. Tarjan [1987, S. 598] explizit *dünn besetzte* Graphen zugrunde legen, bei denen gilt: $|V| \ll |E| \ll |V|^2$. Ist dies nicht gewährleistet, behält die oben genannte pessimistischere Abschätzung ihre

[5] siehe Seite 29

Gültigkeit.

Ausbreitung des Suchbaums

Die Horizontmenge wächst bei der Dijkstra-Suche in etwa kreisförmig im Gebiet um den Startknoten, falls der zugrunde liegende Graph *homogen* ist (vgl. [Hahne, 2000, S. 46 f.]). Die Homogenität eines Graphen ist dabei umso größer, je mehr folgende drei Eigenschaften erfüllt sind (vgl. ebd.):

1. Die Bewertung eines Pfeils (u, v) im Graphen unterscheidet sich nur wenig von der euklidischen Distanz zwischen u und v.

2. Im Suchgebiet befinden sich wenige Hindernisse wie Flüsse, Berge oder Zufahrtsbeschränkungen.

3. Die Pfeilbewertungen des Graphen weichen wenig voneinander ab.

Die genannten Kriterien werden bei Verwendung der *Kantenlänge* für die Pfeilbewertung besser erfüllt als beispielsweise bei Verwendung der *Fahrzeit*, die für den Straßenabschnitt benötigt wird. Das Anwachsen der Horizontmenge erfolgt demnach bei der Suche nach kürzesten Wegen auf einer digitalen Straßenkarte grundsätzlich kreisförmiger als bei der Suche nach schnellsten.

Suchbaum vs. Suchraum

Im Gegensatz zum Suchbaum beschreibt der *Suchraum* die Lage der Knoten aus der Horizont- und Lösungsmenge im Graphen. Die Begriffe liegen sprachlich nahe beieinander und auch die Größe des Suchbaums korreliert stark mit der des Suchraums. Doch inhaltlich beschreibt der Suchbaum eine sortierte Datenstruktur, während der Suchraum eine von der Suche abgedeckte „Fläche“ über die Knoten des zugrunde liegenden Graphen meint.

Wegsuchen der Form 1:1

Ist nur der kürzeste Weg von einem Startknoten zu *einem* anderen Zielknoten von Interesse, kann das Verfahren abgebrochen werden, sobald der Zielknoten in Schritt 1 ausgewählt wurde, d. h. als nächstes in die Lösungsmenge aufgenommen werden soll. Das abgeleitete Verfahren 6, S. 93, zeigt diese Änderung im Vergleich zu Verfahren 5, S. 91. Die Aufwandsabschätzung von $\mathcal{O}(|V|^2)$ bleibt auch für diese Form der Suche für den ungünstigsten Fall – wenn ein optimaler Weg alle Knoten des Graphen beinhaltet – bestehen. Alle nachfolgend beschriebenen Verfahren liefern ausschließlich kürzeste Wege von einem Start- zu einem Zielknoten.

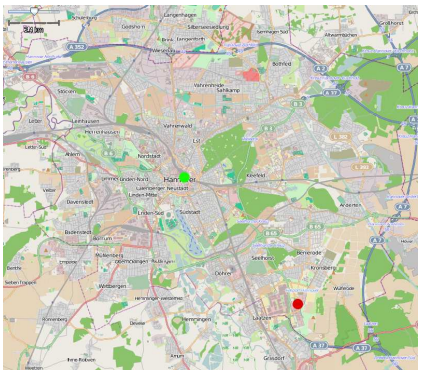
Verfahren 6 Knotenbasierter Dijkstra-Algorithmus (1:1-Suche)

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$, Startknoten s, Zielknoten t mit $s, t \in V$

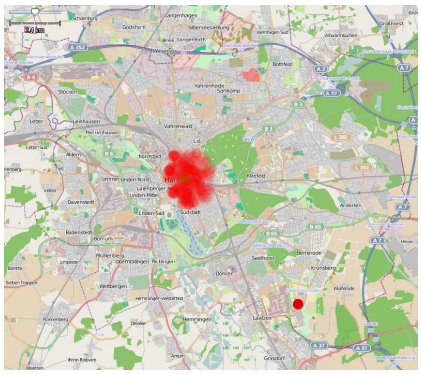
1: $s.distanz \coloneqq 0$ ▸ Initialisierung
2: $v.distanz \coloneqq \infty$, $v.vorgänger \coloneqq NULL$ $\forall v \in V \setminus \{s\}$
3: $Lösungsmenge \coloneqq \emptyset$, $Horizontmenge \coloneqq \{s\}$
4: **while** $Horizontmenge \neq \emptyset$ **do**
5: Wähle v mit $v.distanz = \min\{v'.distanz \mid v' \in Horizontmenge\}$ ▸ Schritt 1
6: **if** $v = t$ **then**
7: Abbruch des Verfahrens
8: **end if**
9: ... ▸ Rest identisch zu Verfahren 5, S. 91,
10: **end while**

Ergebnis: Ein kürzester Weg von s zu t ist rekursiv konstruierbar aus $t.vorgänger$. Es gilt: $t.distanz = \text{dist}(s, t)$. Ist $t.vorgänger$ nicht gesetzt, existiert kein Weg zwischen s und t in G.

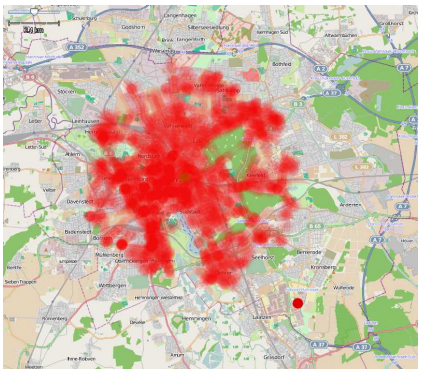
Durch die oben beschriebene Ausbreitung des Suchbaums beim Dijkstra-Verfahren werden bei 1:1-Wegsuchen in der Regel viele Knoten in die Horizont- und Lösungsmenge aufgenommen, die nicht zum optimalen Weg vom Start- zum Zielknoten gehören. Abbildung 4.1 veranschaulicht dies anhand eines Beispiels. Dargestellt ist die Wegsuche vom Hauptbahnhof Hannover in der Bildmitte zum Messegelände (unten rechts im Bild) in verschiedenen Stadien. Die zugrunde liegende Karte wurde direkt aus dem OpenStreetMap-Projekt entnommen, ohne die in Abschnitt 3.6, S. 78, beschriebene Kantenobjekt-Verbindung vorzunehmen, um die Ausbreitung der Suche deutlicher zu dokumentieren. Gut sichtbar zeigt sich hier das kreisförmige Anwachsen des Suchbaums. In Abbildung 4.1d ist erkennbar, dass fast ebenso weit in die dem Ziel entgegengesetzte Richtung gesucht wurde wie in Richtung des Ziels.



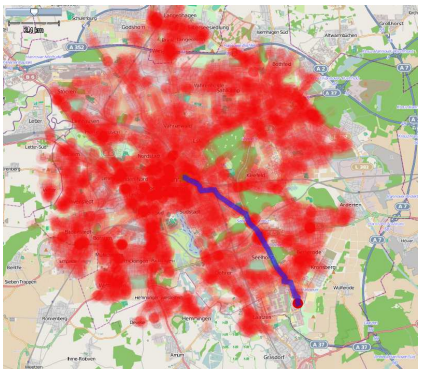
(a) Start- und Zielknoten



(b) Lösungsmenge nach 1 000 Iterationen



(c) Lösungsmenge nach 10 000 Iterationen



(d) Lösungsmenge nach 20 601 Iterationen
Ein kürzester Weg (blau) wurde gefunden.

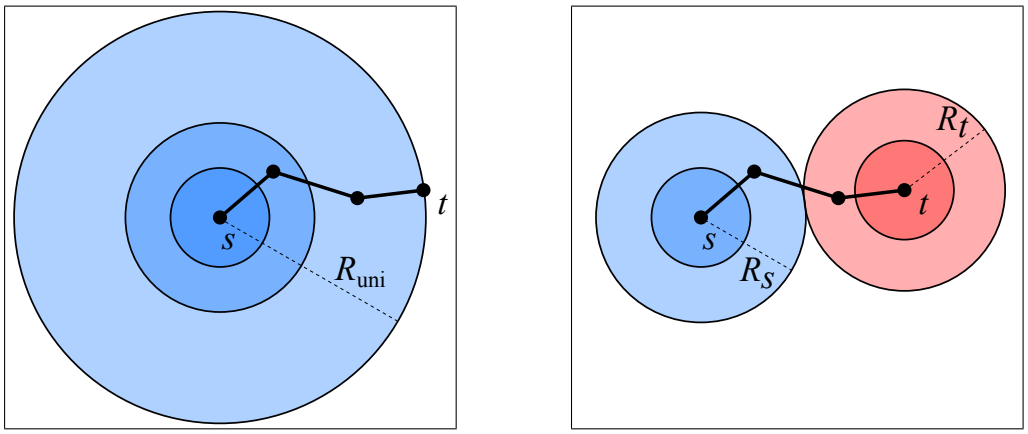
Abbildung 4.1: Ausdehnung der Lösungsmenge beim Dijkstra-Verfahren in verschiedenen Stadien der Wegsuche
Darstellung in Anlehnung an Hahne [2000, S. 47]

Sämtliche auf dem Dijkstra-Verfahren aufbauenden Verfahren versuchen daher, die Anzahl der in die Horizont- und/oder Lösungsmenge aufgenommenen Knoten zu verringern. Besonders erfolgreich sind hierbei zwei nachfolgend vorgestellte Verfahren. Das bidirektionale Dijkstra-Verfahren arbeitet mit zwei *parallelen* Suchen, während eine *zielgerichtete* Suche durch das A*-Verfahren ermöglicht wird.

Definition: Dijkstra-Rang In der Literatur zu Wegsucheverfahren wird als Maß für die „Nähe“ zweier Knoten zueinander häufig der Dijkstra-Rang angegeben (vgl. [Sanders u. Schultes, 2005], [Bast u. a., 2007a] oder [Geisberger u. a., 2008]). Formal beschreibt der Dijkstra-Rang $r_s(v)$ die Anzahl Iterationen, die das Dijkstra-Verfahren mit Startknoten s benötigt, bis v in die Lösungsmenge aufgenommen wird.

4.1.1.2 Das bidirektionale Dijkstra-Verfahren

Ein naheliegender Gedanke, den Suchraum beim Dijkstra-Verfahren zu beschränken, besteht darin, die Suche bidirektional anzugehen, also eine Dijkstra-Suche sowohl vom Start- zum Zielknoten als auch in umgekehrter Richtung *parallel* durchzuführen. Wie in Abbildung 4.2 skizziert, ist dabei die Hoffnung, dass anstatt eines großen Suchradius' R_{uni} zwei kleinere Radien R_s und R_t ausreichen, um den kürzesten Weg zu finden.



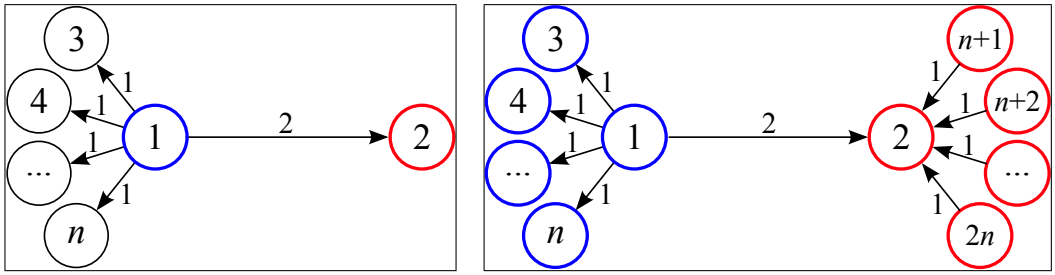
(a) Unidirektionale Dijkstra-Wegsuche (b) Bidirektionale Dijkstra-Wegsuche

Abbildung 4.2: Schematische Ausbreitung des Suchraums bei uni- und bidirektionaler Dijkstra-Wegsuche zwischen Startknoten s und Zielknoten t
blau: abgedeckter Suchraum der Vorwärtssuche
rot: abgedeckter Suchraum der Rückwärtssuche
Darstellung in Anlehnung an Salzmann [2011, S. 14]

Idealisiert: halbierter Suchraum

Die vom Suchraum abgedeckte Fläche A ist im schematischen Fall in Abbildung 4.2a $A_{\text{uni}} = \pi R_{\text{uni}}^2$. Bei der in Abbildung 4.2b schematisch dargestellten *bidirektionalen* Suche dagegen ist die von beiden Suchräumen abgedeckte Fläche A_{bi} erheblich kleiner. Da die Fläche quadratisch mit dem Radius wächst, tritt der günstigste Fall genau dann ein, wenn die beiden Teil-Suchräume der bidirektionalen Suche gleich groß sind, also gilt: $R_s = R_t = \frac{1}{2}R_{\text{uni}}$. Dann ist $A_{\text{bi}} = \pi R_s^2 + \pi R_t^2 = 2\pi(\frac{1}{2}R_{\text{uni}})^2 = \frac{1}{2}\pi R_{\text{uni}}^2 = \frac{1}{2}A_{\text{uni}}$; im Vergleich zur unidirektionalen Suche könnte der Aufwand also – im idealisierten homogenen Graphen – bestenfalls halbiert werden.

Dass diese Aussage nicht auf jede reale Wegsuche übertragbar ist, verdeutlicht Abbildung 4.3: Es sind inhomogene Graphen denkbar, in denen ein kürzester Weg im Vergleich mit dem unidirektionalen Dijkstra-Verfahren beliebig schneller oder aber auch bis zu halb so schnell gefunden wird.



(a) Bidirektionale Suche vorteilhaft (b) Bidirektionale Suche unvorteilhaft

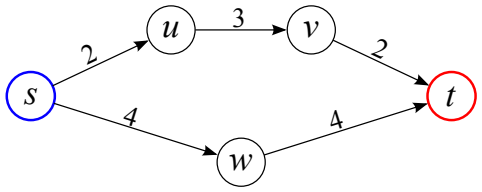
Abbildung 4.3: Extrembeispiele für die bidirektionale Dijkstra-Suche
In beiden Graphen werde die Wegsuche von Knoten 1 zu Knoten 2 betrachtet. Das linke Beispiel (4.3a) zeigt, wie ein Graph aufgebaut sein kann, so dass die bidirektionale Suche um ein beliebiges Vielfaches schneller ist als die unidirektionale. Im rechten Beispiel (4.3b) ist die Struktur von Graphen dargestellt, in denen die bidirektionale Suche nur halb so schnell ist wie die unidirektionale.
blau markiert: Vorwärtssuche
rot markiert: Rückwärtssuche

Grundsätzlich ist das Vorgehen einfach: Für die Rückwärtssuche müssen lediglich die Pfeilrichtungen umgekehrt berücksichtigt werden, damit beim Aufeinandertreffen der Teil-Suchen ein zulässiger Pfad zusammengesetzt werden kann. Für jeden Knoten existieren darüber hinaus die Attribute `distanz` und `vorgänger` notwendigerweise doppelt, nämlich einmal je Suchrichtung. Die Notation bei Knoten $v \in V$ ist hierfür $v.distanz_{\text{vor}}$ und $v.distanz_{\text{rück}}$ sowie analog $v.vorgänger_{\text{vor}}$ und $v.vorgänger_{\text{rück}}$.[6]

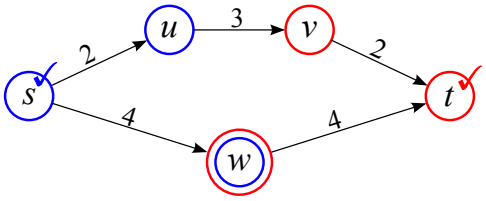
Das Abbruchkriterium ist hingegen weniger intuitiv. Dreyfus [1969, S. 398] weist explizit darauf hin, dass es mitnichten genügt, das Verfahren abzubrechen, sobald ein Knoten des Graphen sowohl in der einen als auch in der anderen Suchrichtung in die Lösungsmenge aufgenommen wurde.

Den Grund zeigt das Beispiel in Abbildung 4.4, S. 97,: Bei der dargestellten bidirektionalen Wegsuche wird auf diese Weise *nicht* der kürzeste Weg gefunden. In blauer Farbe sind die erreichten Knoten der Vorwärtssuche ausgehend von Knoten s dargestellt; in roter Farbe die der Rückwärtssuche ausgehend von t. Ein Häkchen in der entsprechenden Farbe gibt an, dass ein Knoten in die Lösungsmenge aufgenommen wurde. Offensichtlich ist Knoten w der erste, der sowohl in die Lösungsmenge der Vorwärts- als auch die der Rückwärtssuche aufgenommen wird. Der dadurch implizierte Weg $p = \langle s, w, t \rangle$ mit $c(p) = 8$ ist länger als der kürzeste Weg $p^* = \langle s, u, v, t \rangle$ mit $c(p^*) = 7$.

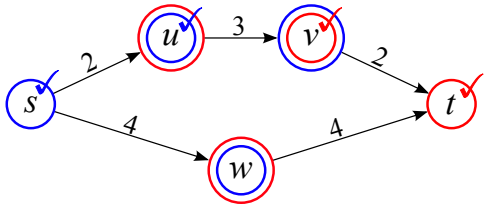
[6] Semantisch wird im Attribut $\texttt{vorgänger}_{\text{rück}}$ eines Knotens v zwar der *Nachfolger* von v entlang des kürzesten bislang bekannten Weges *zum* Zielknoten gespeichert. Es erscheint an dieser Stelle jedoch konsequenter, die Index-Schreibweise (*vor* und *rück*) weiter zu verwenden, anstatt ein neues `nachfolger`-Attribut für die Rückwärtssuche einzuführen.



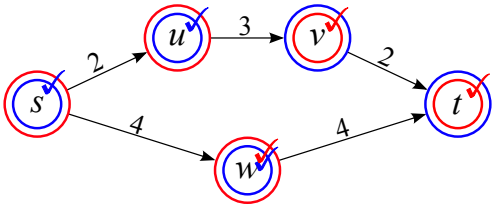
(a) Start- und Zielknoten (blau bzw. rot)



(b) Nach der 2. Iteration



(c) Nach der 4. Iteration



(d) Nach der 6. Iteration

Abbildung 4.4: Naive bidirektionale Dijkstra-Suche
blau markiert: Vorwärtssuche
rot markiert: Rückwärtssuche
Häkchen kennzeichnen Knoten der jeweiligen Lösungsmenge

Wie optimale Wege zuverlässig mit einer bidirektionalen Dijkstra-Suche gefunden werden können, beschreibt Nicholson [1966]. Es bedarf einer Trennung von Lösungs- und Horizontmenge für die Vorwärts- und Rückwärtssuche, was im Folgenden durch einen entsprechenden Index angezeigt wird.

Die Sortierung der Horizontmengen für die vorwärts bzw. rückwärts gerichtete Suche erfolgt analog zum Dijkstra-Verfahren aufsteigend nach der Länge des bisher kürzesten bekannten Weges vom Startknoten bzw. zum Zielknoten. Es wird stets derjenige Knoten als nächstes expandiert, dessen Distanz-Attribut den kleinsten Wert besitzt und erst daraus ergibt sich die Suchrichtung der aktuellen Dijkstra-Iteration. Formal wird in Vorwärtsrichtung expandiert, falls es einen Knoten $v \in \mathit{Horizontmenge}_{\text{vor}}$ gibt, für den gilt:

$$v.\mathit{distanz}_{\text{vor}} =$$
$$\min\Big\{ \min\left\{l.\mathit{distanz}_{\text{vor}} \mid l \in \mathit{Horizontm.}_{\text{vor}}\right\}, \min\left\{l.\mathit{distanz}_{\text{rück}} \mid l \in \mathit{Horizontm.}_{\text{rück}}\right\} \Big\}.$$

Ansonsten muss demnach gelten:

$$v.\mathit{distanz}_{\text{rück}} =$$
$$\min\Big\{ \min\left\{l.\mathit{distanz}_{\text{vor}} \mid l \in \mathit{Horizontm.}_{\text{vor}}\right\}, \min\left\{l.\mathit{distanz}_{\text{rück}} \mid l \in \mathit{Horizontm.}_{\text{rück}}\right\} \Big\}$$

und die Knotenexpansion erfolgt in Rückwärtsrichtung, d. h., es werden alle zu v adjazenten Pfeile expandiert, die *zu* v führen.

Im Gegensatz zum obigen naiven Verfahren liegt der Fokus beim von Nicholson [1966] beschriebenen Verfahren auf der Horizontmenge statt der Lösungsmenge: Nach jeder

Expansion wird untersucht, ob diejenigen Knoten, die neu in die Horizontmenge aufgenommen werden oder zu denen ein kürzerer als der bisher bekannte Weg gefunden wurde, im Suchbaum der Gegenrichtung ebenfalls bereits Teil der Horizontmenge sind. Derartige Knoten werden im Folgenden als *Brückenknoten* bezeichnet.

Brückenknoten

Die Identifikation eines Brückenknotens ist gleichbedeutend mit dem Finden *eines* Weges vom Start- zum Zielknoten. Daher bleibt ein Knoten Brückenknoten, selbst wenn er später im Verfahren noch aus der Horizontmenge entfernt und der Lösungsmenge hinzugefügt werden sollte. Die Weglänge des von Brückenknoten m induzierten Weges $p_m = \langle s, \ldots, m, \ldots, t \rangle$ ergibt sich als Summe der Distanz-Attribute zu m für die beiden Suchrichtungen:

$$c(p_m) = m.distanz_{\text{vor}} + m.distanz_{\text{rück}} \tag{4.2}$$

Abbruchkriterium

Im Verlauf des Verfahrens wurden die Knoten der Horizontmengen noch nicht genauer untersucht. Daher kann das Verfahren nicht ausschließen, dass zwischen einem Knoten aus $\text{Horizontmenge}_{\text{vor}}$ und einem Knoten aus $\text{Horizontmenge}_{\text{rück}}$ noch ein sehr kurzer Pfeil existiert. Die Suche darf daher erst abbrechen, sobald der kürzeste auf diese Weise *theoretisch* mögliche Weg bereits länger ist als der bisher kürzeste *bekannte* Weg, wenn also erstmals ein Brückenknoten m existiert, für den gilt (vgl. [Nicholson, 1966, S. 277]):

$$m.distanz_{\text{vor}} + m.distanz_{\text{rück}} \leq u.distanz_{\text{vor}} + v.distanz_{\text{rück}} \tag{4.3}$$

und $u \in Horizontmenge_{\text{vor}}$

und $v \in Horizontmenge_{\text{rück}}$

und $u.distanz_{\text{vor}} = \min\{w.distanz_{\text{vor}} \mid w \in Horizontmenge_{\text{vor}}\}$

und $v.distanz_{\text{rück}} = \min\{w.distanz_{\text{rück}} \mid w \in Horizontmenge_{\text{rück}}\}$

und $m.distanz_{\text{vor}} + m.distanz_{\text{rück}} = \min\{m'.distanz_{\text{vor}} + m'.distanz_{\text{rück}} \mid m' \text{ ist Brückenknoten}\}$

Die Suche könnte daher auch präziser als „bidirektionale Dijkstra-Suche mit einem *look-ahead* von 1“ bezeichnet werden. Denn im Gegensatz zur unidirektionalen Variante ist derjenige Brückenknoten, über den letztlich der Lösungsweg führt, nicht notwendigerweise Teil der Lösungsmengen von Vorwärts- und Rückwärtssuche. Verfahren 7, S. 99, formuliert die bidirektionale Dijkstra-Suche im Pseudocode.

Verfahren 7 Bidirektionaler knotenbasierter Dijkstra-Algorithmus

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$, Startknoten s, Zielknoten t mit $s, t \in V$

1: $s.distanz_{\text{vor}} \coloneqq 0$, $t.distanz_{\text{rück}} \coloneqq 0$ ▸ Initialisierung
2: $v.distanz_{\text{vor}} \coloneqq \infty$, $v.vorgänger_{\text{vor}} \coloneqq NULL$ $\forall v \in V \setminus \{s\}$
3: $v.distanz_{\text{rück}} \coloneqq \infty$, $v.vorgänger_{\text{rück}} \coloneqq NULL$ $\forall v \in V \setminus \{t\}$
4: $Lösungsmenge_{\text{vor}} \coloneqq \emptyset$, $Lösungsmenge_{\text{rück}} \coloneqq \emptyset$
5: $Horizontmenge_{\text{vor}} \coloneqq \{s\}$, $Horizontmenge_{\text{rück}} \coloneqq \{t\}$
6: **while** Es existiert kein Brückenknoten m, der Gleichung (4.3), S. 98, erfüllt **do**
7: **if** $Horizontmenge_{\text{vor}} \cup Horizontmenge_{\text{rück}} = \emptyset$ **then**
8: Abbruch des Verfahrens: Es existiert kein Weg in G von s nach t
9: **end if**
10: $minDist_{\text{vor}} \coloneqq \infty$
11: $minDist_{\text{rück}} \coloneqq \infty$
12: **if** $Horizontmenge_{\text{vor}} \neq \emptyset$ **then**
13: $minDist_{\text{vor}} \coloneqq \min\{v.distanz \mid v \in Horizontmenge_{\text{vor}}\}$
14: **end if**
15: **if** $Horizontmenge_{\text{rück}} \neq \emptyset$ **then**
16: $minDist_{\text{rück}} \coloneqq \min\{v.distanz \mid v \in Horizontmenge_{\text{rück}}\}$
17: **end if**
18: **if** $minDist_{\text{vor}} < minDist_{\text{rück}}$ **then** ▸ Dijkstra-Iteration in Vorwärtsrichtung
19: Wähle $v \in Horizontmenge_{\text{vor}}$ mit $v.distanz_{\text{vor}} = minDist_{\text{vor}}$
20: Entferne v aus $Horizontmenge_{\text{vor}}$
21: Nimm v in $Lösungsmenge_{\text{vor}}$ auf
22: Expandiere v in Vorwärtsrichtung
23: **else** ▸ Dijkstra-Iteration in Rückwärtsrichtung
24: Wähle $v \in Horizontmenge_{\text{rück}}$ mit $v.distanz_{\text{rück}} = minDist_{\text{rück}}$
25: Entferne v aus $Horizontmenge_{\text{rück}}$
26: Nimm v in $Lösungsmenge_{\text{rück}}$ auf
27: Expandiere v in Rückwärtsrichtung
28: **end if**
29: **end while**

Ergebnis: Ein kürzester Weg von s zu t über m ist für die erste Teilstrecke rekursiv konstruierbar aus $m.vorgänger_{\text{vor}}$ und für die zweite Teilstrecke aus $m.vorgänger_{\text{rück}}$. Es gilt: $m.distanz_{\text{vor}} + m.distanz_{\text{rück}} = \text{dist}(s, t)$. Wurde kein Brückenknoten gefunden, existiert kein Weg zwischen s und t.

Aufwands-abschätzung

Grundsätzlich gelten für die Abschätzung des Aufwands des Verfahrens dieselben Überlegungen wie für die 1:1-Dijkstra-Suche auf Seite 92: Enthält der optimale Weg alle Knoten eines vollverknüpften Graphen $G = (V, E, c)$, ergibt sich auch hier ein Aufwand von $\mathcal{O}(|V|^2)$.

Abbildung 4.5 zeigt die Wegsuche mit dem bidirektionalem Dijkstra-Verfahren für dieselbe Problemstellung wie Abbildung 4.1. Erneut wurde die hierfür zugrunde liegende OSM-Karte nicht der in Abschnitt 3.6, S. 78, beschriebenen Kantenobjekt-Verbindung unterzogen, damit die Suchausbreitung markanter darstellbar ist.

(a) Lösungsmenge nach 500 Iterationen

(b) Lösungsmenge nach 5000 Iterationen

(c) Lösungsmenge nach 8541 Iterationen
Ein kürzester Weg (blau) wurde gefunden.

Abbildung 4.5: Ausdehnung der Lösungsmenge beim bidirektionalen Dijkstra-Verfahren zu verschiedenen Stadien der Wegsuche
Darstellung in Anlehnung an Hahne [2000, S. 47]

Deutlich zu erkennen ist in Abbildung 4.5a und Abbildung 4.5b das unterschiedlich schnelle Wachstum von $\text{Lösungsmenge}_{\text{vor}}$ und $\text{Lösungsmenge}_{\text{rück}}$. Da im Umkreis um den Hauptbahnhof das zugrunde liegende Straßennetz Hannovers dichter und homogener ist als um das Messegelände, werden zunächst auffällig mehr Knoten in Vorwärtsrichtung zur entsprechenden Lösungsmenge hinzugefügt. Mit insgesamt 8 541 benötigten Iterationen läuft das Verfahren im Vergleich zum Standard-Dijkstra-Verfahren in diesem Beispiel mehr als doppelt so schnell.

4.1.1.3 Das A*-Verfahren

Das zuerst von Hart u. a. [1968] beschriebene A*-Verfahren erweitert das Dijkstra-Verfahren um eine Zielorientierung bei der Wegsuche, um den Suchraum zu verkleinern. Hierfür wird das Sortierkriterium für die Horizontmenge um eine Abschätzung der Restdistanz $\mathrm{rd}(v)$ von einem Knoten v zum Zielknoten ergänzt. Wie auch beim Dijkstra-Verfahren darf der zugrunde liegende gewichtete, gerichtete Graph keine negativen Pfeilbewertungen besitzen. Es wird nunmehr derjenige Knoten $v \in \mathit{Horizontmenge}$ als nächstes expandiert, für den gilt:

Restdistanz-Schätzer

$$v.\mathit{distanz} + \mathrm{rd}(v) = \min\{v'.\mathit{distanz} + \mathrm{rd}(v') \mid v' \in \mathit{Horizontmenge}\}. \tag{4.4}$$

Die linke Seite aus Gleichung (4.4) lässt sich als untere Schranke für die Länge des kürzest möglichen Weges vom Start- zum Zielknoten über Knoten v interpretieren. Hart u. a. [1968, S. 102 f.] zeigen, dass auf diese Weise optimale Wege gefunden werden, sofern $\mathrm{rd}(v)$ die tatsächliche Restdistanz niemals überschätzt.

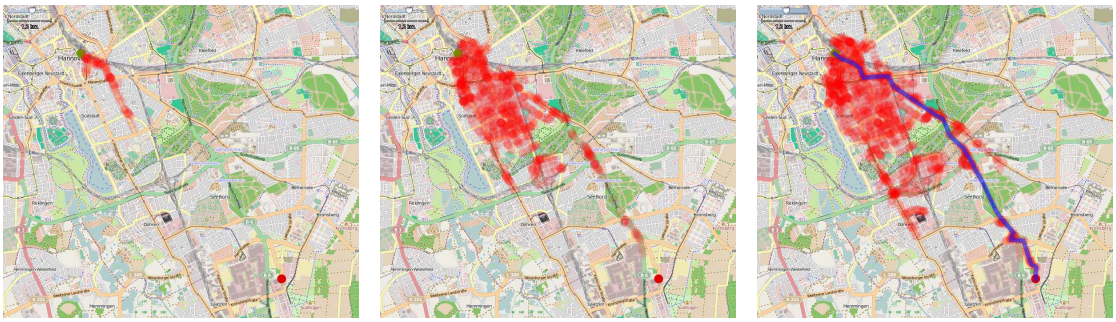
Schätzer für Rest-Fahrtdauer

Für eine Minimierung der Fahrtdauer anstelle der Weglänge mithilfe einer digitalen Straßenkarte bedarf es natürlich einer Anpassung der Restschätzung. Liegen die Straßen im Kartenmaterial nach Kategorien wie *Autobahn*, *Schnellstraße*, *Fußweg*, etc. vor, ergibt sich die Restschätzung als Produkt aus der Luftdistanz multipliziert mit der Geschwindigkeit, die auf der schnellsten Straßenkategorie hinterlegt ist. Andernfalls muss die größtmögliche Geschwindigkeit geschätzt und zugrunde gelegt werden. Denn nur so ist sichergestellt, dass der ermittelte Wert die tatsächliche Rest-Fahrtdauer nicht überschätzt. Die Qualität der Schätzung ist deshalb für die Schnellste-Wege-Metrik im Allgemeinen deutlich geringer als für die Kürzeste-Wege-Metrik.

Im Gegensatz zum Dijkstra-Verfahren ist das A*-Verfahren von vornherein nur auf eine Start-Ziel-Wegsuche ausgerichtet. Entsprechend muss somit lediglich in Verfahren 6, S. 93, das Sortierkriterium in Zeile 5 angepasst werden. Das A*-Verfahren wird hier daher nicht noch einmal explizit als Pseudocode angegeben.

Im Vergleich zum Dijkstra-Verfahren werden beim A*-Verfahren erheblich weniger Knoten expandiert. Die Suche in einem homogenen Graphen wächst hier typischerweise zunächst spitz auf den Zielknoten zulaufend und im Verlauf der weiteren Suche vom Startknoten ausgehend immer breiter werdend.

Das Beispiel in Abbildung 4.6, S. 102, zeigt in einem kleineren Kartenausschnitt dieselbe Problemstellung wie Abbildung 4.1, S. 94,; diesmal jedoch mit der Lösung durch das A*-Verfahren. Es liegt dieselbe Karte zugrunde wie in Abbildungen 4.1, S. 94 und 4.5, S. 100. Die Suche verläuft zunächst annähernd der Luftlinie zum Ziel folgend. Wie in Abbildung 4.6c ersichtlich, beinhaltet dieses Gebiet allerdings mitnichten den kürzesten Weg. Erst nach 1 000 Iterationen erreicht die Wegsuche den so genannten Messeschnellweg, der tatsächlich Teil des kürzesten Weges ist. Die im Verlaufe des Verfahrens immer stärkere Ausweitung des Suchraums ist beim Vergleich von Abbildung 4.6b mit Abbildung 4.6c gut zu erkennen.



(a) Lösungsmenge nach 100 Iterationen

(b) Lösungsmenge nach 1 000 Iterationen

(c) Lösungsmenge nach 1 566 Iterationen
Ein kürzester Weg (blau) wurde gefunden.

Abbildung 4.6: Ausdehnung der Lösungsmenge beim A*-Verfahren in verschiedenen Stadien der Wegsuche
Darstellung in Anlehnung an Hahne [2000, S. 47]

Hervorzuheben ist in diesem Beispiel ebenfalls der im Vergleich mit dem Dijkstra-Verfahren beträchtliche Geschwindigkeitsvorteil des A*-Verfahrens: Benötigte ersteres noch 20 601 Iterationen, sind es bei der A*-Suche nur noch 1 566, entsprechend einem Verhältnis von etwa 13:1.

Hahne [2000, S. 71 f.] gibt beim Verhältnis von Dijkstra-Iterationen zu A*-Iterationen für kürzeste Wege zwar ein Verhältnis von etwa 4:1 an. Der gefundene Weg aus dem hier gewählten Beispiel verläuft jedoch in weiten Teilen nahe der Luftlinie zwischen Start- und Zielknoten, was sich beschleunigend auf den A*-Algorithmus auswirkt.

Grundsätzlich ist die Wegsuche mit dem A*-Verfahren also schneller als beim Dijkstra-Verfahren; genauer gesagt ist sie gemessen an den notwendigen Iterationen stets *mindestens so schnell*. Die Beweisführung verwendet das Sortierkriterium beider Such-Arten und erfolgt per logischem Widerspruch.

Beweis für Geschwindigkeitsvorteil des A- gegenüber dem Dijkstra-Verfahren.* Es werde angenommen, es existiere ein Graph, in dem die Wegsuche von einem Startknoten s zu einem Zielknoten t per Dijkstra-Verfahren schneller verliefe als per A*-Verfahren. Für diese spezielle Wegsuche werden beide Verfahren unabhängig voneinander parallel ausgeführt. Für den Beweis sind zwei Erkenntnisse über das Dijkstra-Verfahren wichtig, die sich aus Gleichung (4.1), S. 90, ableiten:

Erkenntnis 1 Die Dijkstra-Suche findet mit jeder Iteration einen kürzesten Weg vom Startknoten aus zum aktuell ausgewählten Knoten.

Erkenntnis 2 Die Weglängen wachsen monoton: Der neu gefundene kürzeste Weg der aktuellen Iteration ist mindestens so lang wie der aus der vorangegangenen.

Weiterhin sind folgende Aussagen bei einer Wegsuche von s nach t gleichbedeutend:

Aussage 1 Die Dijkstra-Suche verläuft nach Iterationen schneller als die A*-Suche.

Aussage 2 Bei der A*-Suche werden mehr Knoten expandiert als bei der Dijkstra-Suche.

Aussage 3 Wenn – bei schrittweise paralleler Ausführung der beiden Verfahren – bei der Dijkstra-Suche t aus der Horizontmenge entfernt wird (siehe Zeile 5 in Verfahren 6, S. 93), existiert bei der A*-Suche noch mindestens ein Knoten v in der Horizontmenge, der gemäß A*-Sortierkriterium (Gleichung (4.4), S. 101) einen geringeren Wert liefert als t.[7]

Nach Aussage 3 gilt innerhalb der A*-Suche für v demnach:

$$t.distanz + \mathrm{rd}(t) > v.distanz + \mathrm{rd}(v).$$

Ferner ist $\mathrm{rd}(v) \geq 0$ und mit $\mathrm{rd}(t) = 0$ folgt

$$t.distanz > v.distanz. \tag{4.5}$$

Alle Knoten v, für die Gleichung (4.5) gilt, werden aber nach Erkenntnis 1 und 2 von der Dijkstra-Suche in einer früheren Iteration bereits erreicht und expandiert. Dies steht im Widerspruch zu Aussage 2, die somit *nicht gleichzeitig* mit Aussage 1 und 3 gelten kann. □

Aufwandsabschätzung

Diese Erkenntnis lässt sich jedoch nicht in der Landau-Notation ausdrücken. Denn nach wie vor wird der Aufwand des A*-Algorithmus maßgeblich vom Entfernen der Knoten aus der Horizontmenge und dem anschließenden Expandieren derselben bestimmt. Auch für das A*-Verfahren gilt daher die Aufwandsabschätzung von $\mathcal{O}(|V|^2)$ für Graphen mit Knotenmenge V.

Sonderfall: perfekter Schätzer

Offensichtlich gilt grundsätzlich: Je besser die Abschätzung der Restdistanz beim A*-Verfahren, desto schneller kann der optimale Weg gefunden werden. Ein perfekter Schätzer liefert stets die tatsächliche Restdistanz zum Ziel. Bei einer Wegsuche von Knoten s nach t führt dies dazu, dass sämtliche Knoten entlang des kürzesten Weges $p_{s,t}$ denselben Wert in Gleichung (4.4), S. 101, für die Knotensortierung erhalten, nämlich $\mathrm{dist}(s,t)$. Dieser Wert ist gleichzeitig stets der minimale unter allen erreichten Knoten, so dass ausschließlich Knoten entlang $p_{s,t}$ in die Lösungsmenge aufgenommen werden. Allerdings ist ein perfekter Schätzer nur von theoretischem Wert, da seine Kenntnis die Lösung ebenjenes Wegeproblems voraussetzt (vgl. [Hahne, 2000, S. 52 f.]).

Das parametrisierte A*-Verfahren (PA*) Eine Verallgemeinerung des Dijkstra-Verfahrens und des A*-Verfahrens ergibt sich, wenn Gleichung (4.4), S. 101, um einen Parameter $\alpha \in \mathbb{R}$ als Gewichtungsfaktor für die Restdistanz-Schätzung erweitert wird (vgl. [Hahne, 2000, S. 54 ff.]). Es wird dann immer derjenige Knoten $v \in \mathit{Horizontmenge}$ als nächstes expandiert, für den gilt:

$$v.distanz + \alpha \times \mathrm{rd}(v) = \min\{v'.distanz + \alpha \times \mathrm{rd}(v') \mid v' \in \mathit{Horizontmenge}\}. \tag{4.6}$$

[7] Denn falls t bei der A*-Suche der nächste zu expandierende Knoten wäre, liefen beide Suchen nach Iterationen genau gleich schnell ab.

Somit entsprechen die Werte $\alpha = 0$ dem Dijkstra-Verfahren und $\alpha = 1$ dem A*-Verfahren. Je größer der α-Wert, d. h., je stärker die Restdistanz gewichtet wird, desto stärker ist die Wegsuche zielgerichtet. α-Werte > 1 bewirken aber auch, dass die Restdistanz unter Umständen *überschätzt* wird. Dies geschieht, wenn der gesuchte optimale Weg sehr gerade Kantenzüge in Richtung des Zielknotens besitzt. Das Verfahren beschleunigt in diesen Fällen zwar stärker, wird aber gleichzeitig zur Heuristik[8].

Untersuchungen von Hahne [2000, S. 74 ff.] auf einer digitalen Straßenkarte der Stadt Hildesheim legen nahe, dass α-Werte bis 1,2 für kürzeste Wege einen guten Kompromiss zwischen der Beschleunigung des Verfahrens und der Fehleranfälligkeit bieten. Für die Suche nach *schnellsten* Wegen werden sogar α-Werte bis einschließlich 2 genannt.

4.1.2 Wegsucheverfahren mit Vorberechnungen

In diesem Abschnitt werden ausgesuchte Verfahren vorgestellt, die eine beschleunigte Wegsuche ermöglichen, indem sie auf Erkenntnisse zurückgreifen, die sie durch eine einmalige Vorab-Bearbeitung des zugrunde liegenden Graphen erhalten haben. Es ist leicht nachvollziehbar, dass sich durch diese Vorberechnungen nahezu beliebige Beschleunigungen erzielen lassen: Im Extremfall werden sämtliche optimalen Wege im Voraus berechnet, so dass sich die Wegsuche auf das Finden eines Eintrags in einer start- und zielknotenindexierten Wege-Menge beschränkt. Bereits für relativ kleine digitale Straßenkarten ist der Ansatz einer solchen vollständigen Enumeration jedoch impraktikabel, da er mit sehr hohem Speicherbedarf für die vorab ermittelten Wege verbunden ist. Schließlich steigt der benötigte Speicherplatz quadratisch mit der Anzahl der Knoten in der Karte. Die nachfolgenden Verfahren bieten daher einen Kompromiss aus Beschleunigung der Wegsuche im Vergleich zum Dijkstra-Verfahren und benötigtem Speicherplatz für die Vorberechnungen.

4.1.2.1 Das Trennlinien-Verfahren

Eine Möglichkeit, den beim A*-Verfahren eingesetzten Restdistanz-Schätzer zu verbessern, ohne dabei – wie beim PA*-Verfahren – eine heuristische Wegsuche zu erhalten, stellen Dubois u. Semet [1995] vor. Hierbei liegt die Erkenntnis zugrunde, dass auf Straßenkarten der euklidische Schätzer in einigen Fällen erheblich von der tatsächlichen Restdistanz abweicht. Dies geschieht, falls beispielsweise natürliche Hindernissen wie Seen, Flüsse oder Berge zwischen Start und Ziel liegen.

[8] siehe Aussage zum Schätzer auf Seite 101

Um die Qualität des Schätzers zu erhöhen, schlagen Dubois u. Semet [1995] daher vor, so genannte *Trennlinien* in das Kartenmaterial einzufügen. Dies sind gerade Linien, die auf die Grundfläche eines Hindernisses positioniert werden, so dass sie möglichst gut das Hindernis beschreiben (siehe Beispiele in Abbildung 4.7). Trennlinien

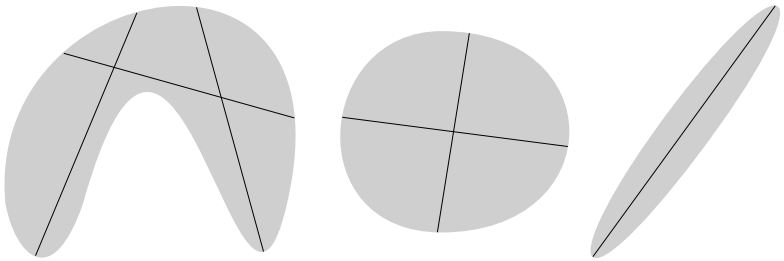


Abbildung 4.7: Beispiel für durch Trennlinien gekennzeichnete Hindernisse
Quelle: Dubois u. Semet [1995, S. 106]

Betrachtet werde der Fall einer s-t-Wegsuche nach dem A*-Verfahren. Eine verbesserte Abschätzung der Restdistanz von einem Knoten v zum Zielknoten t als per euklidischer Distanz erfolgt dann über einen ungerichteten Hilfsgraphen G', der anhand der Trennlinien T konstruiert wird, ohne diese jedoch zu beinhalten. G' hat dabei folgende Eigenschaften:

- Seine Knoten sind alle Endknoten aus T sowie v und t.
- Seine Kanten sind solche Luftlinien zwischen allen seinen Knoten, die keine Trennlinien schneiden.
- Die Kantenbewertung in G' erfolgt euklidisch.

Die Abschätzung der Restdistanz ergibt sich dann aus der Länge des kürzesten Weges von v nach t in G'. Abbildung 4.8 zeigt ein Beispiel für einen Hilfsgraphen.

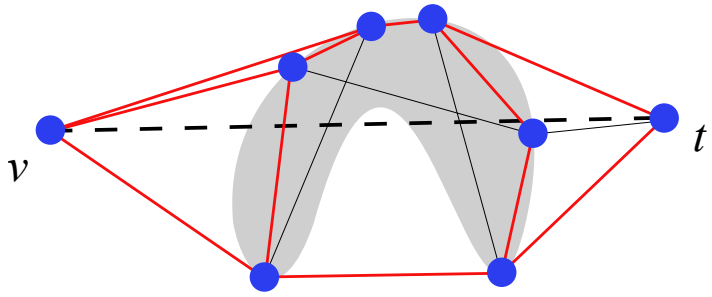


Abbildung 4.8: Beispiel für Hilfsgraph beim Trennlinien-Verfahren
Schwarze gestrichelte Linie: Luftlinie zwischen Knoten v und t
Schwarze durchgezogene Linien: Trennlinien
Rote Linien: Kanten des ungerichteten Hilfsgraphen
Blaue Punkte: Knoten des Hilfsgraphen
Graue Fläche: Hindernis
Quelle: in Anlehnung an Hahne u. a. [2008, S. 457]

Aufwandsabschätzung und Beschleunigung

Da das Verfahren auf dem A*-Verfahren basiert und ausschließlich auf eine Verbesserung des Schätzers der Restdistanz abzielt, gelten für den Aufwand dieselben Aussagen. Es ist daher insbesondere – gemessen an der Anzahl der Iterationen – niemals langsamer als das Dijkstra-Verfahren[9].

Dubois u. Semet [1995, S. 113] geben in ihren Messungen des auf diese Weise veränderten A*-Verfahrens im Vergleich zum Standard-A*-Verfahren eine durchschnittliche Verringerung der Iterationen bei der Wegsuche von knapp 25 % an. In der Praxis ist dies jedoch nicht gleichzusetzen mit einer *zeitlichen* Beschleunigung der Wegsuche. Denn die Ermittlung der kürzesten Wege in den Hilfsgraphen erzeugt ihrerseits einen Mehraufwand. Auf der anderen Seite ist der Aufwand für das vorbereitende Einfügen der Trennlinien in die Karte überschaubar und vergleichsweise gering. Der größte Vorteil des Trennlinien-Verfahrens entsteht daher in Situationen, in denen das Laden bzw. Nachladen von Kartenmaterial aufwendig ist, wie etwa auf mobilen Navigationsgeräten mit begrenztem Speicher.

Darüber hinaus sind die Trennlinien selbst weitgehend robust gegen Änderungen des Kartenmaterials. Sie behalten ihre Gültigkeit, wenn Straßen entfernt werden oder sich ihre Wegführung ändert. Sobald jedoch neue Straßen Trennlinien schneiden, müssen letztere entfernt bzw. verlegt werden.

Überlegungen zur weiteren Beschleunigung des Trennlinien-Verfahrens sowie zur automatisierten Lokalisierung von Trennlinien finden sich bei Hahne u. a. [2008]. So ist es effizienter, den Hilfsgraphen zwischen Knoten v und t nur mittels solcher Trennlinien zu erzeugen, die von der Luftlinie zwischen v und t geschnitten werden. Den Pseudocode für diesen effizienteren Schätzer zeigt Verfahren 8.

Verfahren 8 Abschätzung der Restdistanz beim Trennlinien-Verfahren

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$, Trennlinien T, Knoten v, Zielknoten t mit $v, t \in V$

1: Bilde die Luftlinie von v nach t.
2: $T' \coloneqq \{\tau \mid \tau \in T$ und die Luftlinie von v nach t schneidet $\tau\}$ ▸ geschnittene Trennlinien
3: $V' \coloneqq \{v' \mid v'$ ist einer der Endknoten von τ und $\tau \in T'\} \cup v \cup t$
4: $E' \coloneqq \{[u', v'] \mid$ die Luftlinie von u' nach v' schneidet kein $\tau \in T'$ und $u', v' \in V'\}$
5: Die Kostenfunktion c' für die Kanten aus E' gibt die euklidische Distanz wieder
6: Sei $G' = [V', E', c']$ der ungerichtete Hilfsgraph
7: Berechne Schätzer $\mathrm{rd}(v) \coloneqq$ Länge eines kürzesten Weges von v nach t in G'

Ergebnis: Eine präzisere Schätzung $\mathrm{rd}(v)$ für die Restdistanz von Knoten v zu t als die euklidische Distanz zwischen v und t.

[9] siehe Beweis auf Seite 102 f.

4.1.2.2 Das ALT-Verfahren

Wesentlich mehr Vorverarbeitung als das Trennlinien-Verfahren benötigt das von Goldberg u. Harrelson [2005] vorgestellte ALT-Verfahren. Es zielt ebenfalls auf eine verbesserte Restdistanz-Schätzung beim A*-Verfahren ab. Dazu wird auf der Karte vorab einer Menge M von Knoten eine besondere Rolle zugewiesen, die im Folgenden als *Landmarken* bezeichnet werden. Von den Landmarken werden zu allen Knoten des Graphen kürzeste Wege berechnet und gespeichert.

Landmarken

Die Distanz von Knoten v zu Knoten t wird dann, wie in Abbildung 4.9 skizziert, über die mehrfache Anwendung der Dreiecksungleichung unter Zuhilfenahme der vorab berechneten kürzesten Wege abgeschätzt.[10] Daher gilt für jede Landmarke $L_i \in M$:

$$\begin{aligned} \text{dist}(L_i, v) + \text{dist}(v, t) &\geq \text{dist}(L_i, t) \\ \Leftrightarrow \text{dist}(v, t) &\geq \text{dist}(L_i, t) - \text{dist}(L_i, v). \end{aligned} \tag{4.7}$$

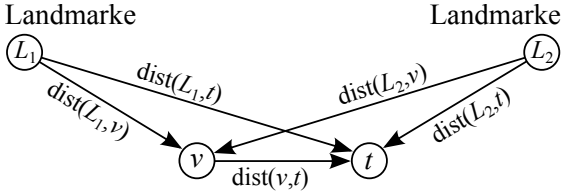


Abbildung 4.9: Restdistanz-Schätzung beim ALT-Verfahren anhand von Landmarken
Quelle: in Anlehnung an Hahne u. a. [2008, S. 3]

Die Differenzen der Form $\text{dist}(L_i, t) - \text{dist}(L_i, v)$ bilden somit jeweils eine untere Schranke für die wahre, unbekannte Restdistanz von v nach t. Die größte untere Schranke ist dann die beste Schätzung für die Restdistanz von v nach t:

$$\text{rd(v)} = \max\{\text{dist}(L_i, t) - \text{dist}(L_i, v) \mid L_i \in M\}. \tag{4.8}$$

Entscheidenden Einfluss auf das Verfahren hat die Auswahl geeigneter Landmarken. Sowohl ihre Anzahl als auch Position sind von großer Bedeutung. Insbesondere gilt, dass die Qualität der Restdistanz-Schätzung mit wachsender Anzahl an Landmarken zwar grundsätzlich steigt, aber gleichzeitig erhöht sich auch der Aufwand zur Ermittlung dieses Schätzers linear mit ihrer Anzahl.

Das ALT-Verfahren kann als Verallgemeinerung der zu Beginn von Abschnitt 4.1.2, S. 104, beschriebenen vollständigen Enumeration interpretiert werden. Statt *aller* kürzesten Wege wird hier eine *Teilmenge* berechnet und dieses Wissen geschickt mit folgenden Wegsuchen verknüpft.

[10] Hieraus erschließt sich dann auch das englische Akronym: A* with landmarks and triangle inequality.

Aufwandsabschätzung und Beschleunigung

Da auch dieses Verfahren auf dem A*-Verfahren basiert und dessen Restdistanz-Schätzer verbessert, gelten dieselben Aussagen bezüglich des Aufwands und beweisbarer Suchbeschleunigung. Besonders hervorzuheben ist aber, dass keine euklidischen Distanzen für den Restdistanz-Schätzer berechnet werden müssen. Das ALT-Verfahren benötigt daher im Gegensatz zum A*-Verfahren keine Knoten-Koordinaten im Graphen.

Goldberg u. Harrelson [2005, S. 162] beobachten beim Vergleich des ALT-Verfahrens mit dem Dijkstra-Verfahren eine Beschleunigung von bis zu Faktor 22. Dieser Wert ist jedoch nur im engen Zusammenhang mit dem konkreten Beispiel zu sehen. So wurden 16 Landmarken verwendet – eine andere Zahl von Landmarken hätte mit hoher Wahrscheinlichkeit zu anderen Ergebnissen geführt. Weiterhin wurde die maximale Beschleunigung auf einer Karte mit etwa 2,2 Mio. Knoten und 5,2 Mio. Pfeilen erreicht. Sowohl bei kleineren als auch bei größeren Karten finden sich in den Ergebnissen aber geringere Werte, d. h., die Suchbeschleunigung ist offenbar nicht streng proportional zur Kartengröße. Für die Auswertung wird daher auf Goldberg u. Harrelson [2005] verwiesen.

4.1.2.3 Das Arc-Flag-Verfahren

Eine weitere Variante, den Suchraum beim Dijkstra-Verfahren zu verkleinern, stellen Köhler u. a. [2009] vor. Sie unterteilen die Knoten eines nicht-negativ bewerteten, gerichteten Graphen $G = (V, E, c)$ zunächst in $\hat{r}$ disjunkte Regionen $R_1, R_2, \ldots, R_{\hat{r}}$, so dass $R_1 \cup R_2 \cup \ldots \cup R_{\hat{r}} = V$ und $R_i \cap R_j = \emptyset$ für $i = 1, \ldots, \hat{r}$, $j = 1, \ldots, \hat{r}$ und $i \neq j$. Ferner sei $\mathrm{reg} : V \to \{1, \ldots, \hat{r}\}$ eine Funktion, die für jeden Knoten den Index „seiner“ Region liefere.

Die Idee des Verfahrens besteht darin, pro Pfeil und Region des Graphen in einem einmaligen Präprozess zu vermerken, ob *ein optimaler Weg* existiert, der über diesen Pfeil *in* diese Regionen führt. Bei einer Dijkstra-Wegsuche müssen dann nur solche Pfeile berücksichtigt werden, die Teil eines optimalen Weges in die Region sind, in welcher der *Zielknoten* liegt.

Im Präprozess müssen dazu *alle* kürzesten Wege in G berechnet werden, ohne sie jedoch explizit zu speichern. Stattdessen wird pro Pfeil $(u, v) \in E$ in einem Array f der Länge $\hat{r}$ an Position $f_{(u,v)}(i)$ der Wert `wahr` gespeichert, falls ein mit (u, v) beginnender kürzester Weg existiert, dessen Endknoten in R_i liegt. Ansonsten erhält $f_{(u,v)}(i)$ den Wert `falsch`. Diese Arrays sind die namensgebenden *Arc-Flags*. Pro Pfeil des Graphen sind somit $\hat{r}$ zusätzliche Bits für die Vorberechnung nötig. (vgl. [Köhler u. a., 2009, S. 45])

Bei jeder nachfolgenden Wegsuche von einem Startknoten s zu einem Zielknoten t wird dann zunächst die Region $R_{\mathrm{reg}(t)}$ ermittelt, in der t liegt. Die Suche erfolgt nach dem Dijkstra-Verfahren. Allerdings können all jene Pfeile ignoriert werden, bei denen das Flag für $R_{\mathrm{reg}(t)}$ nicht gesetzt ist, d. h., ein Pfeil $(u, v) \in E$ wird genau dann berücksichtigt, wenn gilt: $f_{(u,v)}(\mathrm{reg}(t)) =$ `wahr`. Hierdurch ergeben sich sehr starke Einschränkungen des Suchraums, die nach Köhler u. a. [2009, S. 55] zu einer bis zu über 1 400-fachen Beschleunigung gegenüber einer Dijkstra-Suche führen.[11]

Beschleunigung

[11] Erneut sei für die detaillierte Auswertung auf die Veröffentlichung hingewiesen, da dort Geschwindigkeitsvorteile anhand der Größe des Suchbaums gemessen werden. Diese Messgröße korreliert jedoch sehr stark mit der in dieser Arbeit verwendeten Größe der Anzahl der Iterationen bei der Suche.

Da die Suche mit dem Dijkstra-Algorithmus vollzogen wird und manche Pfeile *nicht* berücksichtigt werden, ist das Verfahren – bezogen auf die Anzahl der Iterationen – stets mindestens so schnell wie eine Dijkstra-Suche. Es besitzt im ungünstigsten Fall jedoch dieselbe Aufwandsabschätzung von $\mathcal{O}(|V|^2)$.

Aufwandsabschätzung

Abbildung 4.10 zeigt schematisch die notwendigen zusätzlichen Informationen beispielhaft für den grau dargestellten Pfeil (u, v) eines Graphen. In diesem Beispiel verläuft offenbar kein kürzester Weg mit Zielknoten in der weiß, grün oder schwarz gekennzeichneten Region über diesen Pfeil. Wegsuchen mit Zielen in der gelben oder roten Region müssen diesen Pfeil allerdings berücksichtigen.

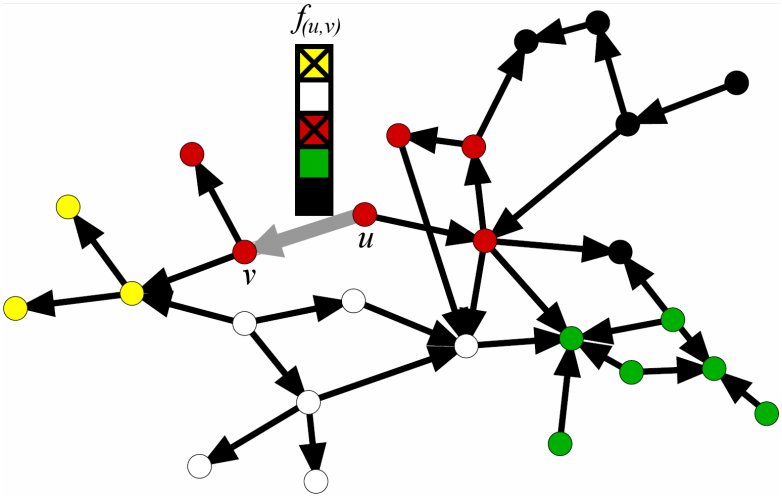


Abbildung 4.10: Pfeil-Zusatzinformationen beim Arc-Flag-Verfahren
Dargestellt sind die für Pfeil (u, v) gespeicherten Flags zu den fünf farblich gekennzeichneten Regionen des Graphen. Ein Kreuz bei der entsprechenden Farbe bedeutet, dass im Graphen ein mit (u, v) beginnender kürzester Weg existiert, der in der zur Farbe gehörenden Region endet. Pfeil (u, v) kommt also ausschließlich in kürzesten Wegen mit Ziel in der gelben oder roten Region vor.
Quelle: Köhler u. a. [2009, S. 46]

Entscheidend für das Arc-Flag-Verfahren sind die Auswahl und Anzahl der Regionen. Für den Sonderfall, dass pro Knoten des Graphen eine Region erzeugt wird, ergibt sich dasselbe Suchverhalten wie beim A*-Verfahren mit perfektem Schätzer[12]. Heuristische Vorgehensweisen zur Auswahl der Regionen werden von Köhler u. a. [2009, S. 51 ff.] angegeben bzw. referenziert.

Das Verfahren kann in seiner Grundform weiter beschleunigt werden, indem die Suche bidirektional angegangen wird. Die größten von Köhler u. a. [2009] ermittelten Suchbeschleunigungen wurden auf diese Weise erreicht.

Des Weiteren lässt sich die *Vorberechnung* der Arc-Flags drastisch beschleunigen, indem vorab nicht mehr alle kürzesten Wege berechnet werden, sondern nur solche zwischen allen Randknoten der Regionen, d. h. zwischen Knoten, die Teil eines Pfeils *zwi-*

[12] siehe Seite 103

schen zwei Regionen sind. Dahinter steckt die Erkenntnis, dass alle Wege, die in eine Region hineinführen, selbstverständlich einen dieser Randknoten passieren müssen. (vgl. [Köhler u. a., 2009, S. 47])

4.2 Hierarchische Wegsucheverfahren

In diesem Abschnitt werden Verfahren vorgestellt, welche zur Wegsuche nicht den Originalgraphen, sondern eine Graphen-Hierarchie verwenden, an deren unterster Ebene der Originalgraph steht. Der Graph einer Ebene l stellt dabei im Wesentlichen einen *Ausschnitt* des Graphen der Ebene $l-1$ dar. Allerdings können je nach Verfahren auch zusätzliche Anreicherungen erforderlich sein. In der Literatur findet sich für die Graphen einer Hierarchie oft der Begriff der *Overlay-Graphen*[13], der weiter unten definiert wird.

Die Suche selbst erfolgt prinzipiell in der Graph-Hierarchie aufsteigend, so dass sich der Suchraum kontinuierlich verkleinert. In der Folge werden die Lösungswege deutlich schneller gefunden als im Originalgraphen.

Eine weitere Beschleunigung ergibt sich durch die Verwendung von *Shortcut-Pfeilen*. Durch geschicktes Einfügen zusätzlicher Pfeile, die ihrerseits zwei oder mehrere Pfeile überbrücken, in die Overlay-Graphen lassen sich bei der Wegsuche kleinschrittige Iterationen vermeiden.

Sowohl das Erstellen von Graphen-Hierarchien als auch das Einfügen von Shortcut-Pfeilen erfolgt bei den hier vorgestellten Verfahren innerhalb eines Präprozesses. Daher werden in diesem Abschnitt keine Ad-hoc-Verfahren diskutiert.

Da sie in mehreren Verfahren verwendet werden, folgen zunächst formale Definitionen für Shortcut-Kanten, Shortcut-Pfeile und Overlay-Graphen.

Definition: Shortcut-Kanten Seien $G = [V, E, c]$ ein ungerichteter, gewichteter Graph und $p = \langle u_0, u_1, \ldots, u_i \rangle$ ein Weg in G. Eine *ungerichtete Shortcut-Kante* ist dann eine künstliche, ungerichtete Kante zwischen u_0 und u_i mit der Bewertung $c(p)$. Die Schreibweise dieser Shortcut-Kante ist $[u_0 \ldots u_i]$.[14]

Analog dazu kennzeichnet $(u \ldots v)$ einen *Shortcut-Pfeil* zwischen den Knoten u und v in einem gerichteten Graphen. Im Gegensatz dazu wird $(u \to v)$ geschrieben, wenn explizit der Original-Pfeil zwischen den beiden Knoten gemeint ist. Ist diese Unterscheidung unerheblich, wird weiterhin (u, v) für (Shortcut-)Pfeile und $[u, v]$ für ungerichtete (Shortcut-)Kanten notiert.

Wie auch bei Wegen können bei Shortcut-Pfeilen die überbrückten Original-Pfeile als Bestandteile referenziert werden. Mittels dieser Schreibweise lassen sich die Kosten eines Shortcut-Pfeils alternativ wie folgt formulieren[15]:

$$c(u \ldots v) = \sum_{(u' \to v') \in (u \ldots v)} c(u' \to v'). \tag{4.9}$$

[13] z. B. bei Abraham u. a. [2010a]; Bauer u. a. [2008, 2010]; Geisberger u. a. [2008]; Schultes [2008]
[14] Diese Punkt-Notation erfolgt in Anlehnung an Geisberger u. Vetter [2011, S. 103].
[15] Dabei wird $(u \ldots v)$ auch als Menge der zugrunde liegenden Original-Pfeile interpretiert.

Definition: Overlay-Graph[16] Sei $G = (V, E, c)$ ein gerichteter, gewichteter Graph. Ein *Overlay-Graph* $G' = (V', E', c')$ ist dann ein Graph, der die nachfolgenden beiden Forderungen erfüllt:

1. $V' \subseteq V$
2. Für alle $u, v \in V'$ gilt: $\text{dist}(u, v) = \text{dist}'(u, v)$, wobei $\text{dist}(u, v)$ bzw. $\text{dist}'(u, v)$ die Länge eines kürzesten Weges zwischen u und v in G bzw. in G' ist.

Anschaulich formuliert ist G' damit ein kürzeste Entfernungen bewahrender „Knotenausschnitt“ aus G. Die Pfeilmenge E' in G' muss geschickt gewählt werden, um diesen Anforderungen zu genügen. Häufig sind die Pfeile aus E' Shortcut-Pfeile über die Pfeile aus E.

Seien $p^*_{u,v}$ bzw. $p'^*_{u,v}$ kürzeste Wege in G bzw. G' zwischen den Knoten $u, v \in V'$. Wenn sich beide Wege auf dieselben Pfeile aus E beziehen[17], lässt sich aus obiger Definition ableiten: $|p'^*_{u,v}| \leq |p^*_{u,v}|$, wobei $|p|$ für die Anzahl der in p enthaltenen Pfeile steht[18]. Insbesondere gilt deshalb auch für die Dijkstra-Ränge in den beiden Graphen für alle $u, v \in V'$: $r'_u(v) \leq r_u(v)$. Mit anderen Worten: Dijkstra-Wegsuchen zwischen Knoten aus V' benötigen in G' stets *höchstens genau so viele Iterationen* wie in G. Diese Eigenschaft von Overlay-Graphen wird von vielen hierarchischen Wegsucheverfahren intensiv ausgenutzt. Abbildung 4.11 zeigt ein Beispiel für einen gerichteten, gewichteten Graphen mit einem zugehörigen Overlay-Graphen.

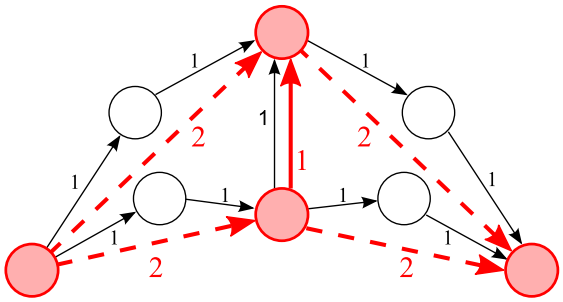


Abbildung 4.11: Beispiel für einen Overlay-Graphen
Dargestellt ist ein gerichteter, gewichteter Originalgraph (schwarz) und ein zugehöriger Overlay-Graph (rot). Gestrichelte symbolisieren Shortcut-Pfeile des Overlay-Graphen.
Quelle: in Anlehnung an Schultes [2008, S. 94]

[16] in Anlehnung an Schultes [2008, S. 99 f.]
[17] d. h., alle Pfeile aus $p^*_{u,v}$ sind entweder explizit in $p'^*_{u,v}$ enthalten oder werden durch Shortcut-Pfeile aus $p'^*_{u,v}$ überbrückt
[18] siehe auch Seite 30

4.2.1 Highway Hierarchies

Der hierarchische Ansatz dieses von Sanders u. Schultes [2005] vorgestellten Verfahrens bezieht sich auf die *Pfeile* des Graphen. Der Grundgedanke bei Highway Hierarchies (HH) ist, dass bei nicht-lokalen Wegsuchen häufig nur noch eine Teilmenge der Pfeile des Graphen für kürzeste Wege eine Rolle spielt. Es sind diese Pfeile, die das so genannte *Highway-Netzwerk* des Graphen bilden. Der Begriff bezieht sich also nicht nur auf die Straßenkategorien, sondern ist abstrakter zu verstehen.

Formal wird hierbei die Definition einer *Radius-Nachbarschaft* für die Knoten des gerichteten, gewichteten Graphen $G = (V, E, c)$ eingeführt. Die Menge der Radius-Nachbarknoten $\mathcal{N}_r(u)$ zu einem Knoten u beinhaltet alle Knoten, zu denen die Distanz von u aus einen gewissen Wert r nicht übersteigt. Es gilt also $\mathcal{N}_r(u) = \{v \in V \mid \text{dist}(u, v) \leq r\}$ und der Radius r ist ein Stellparameter für das Verfahren (vgl. [Sanders u. Schultes, 2006, S. 807]).[19]

Definition des Highway-Netzwerks

Ein Highway-Netzwerk $G_1 = (V_1, E_1, c_1)$ des Graphen $G = (V, E)$ mit Radius r ist dann wie nachfolgend erläutert definiert. Die Menge E_1 besteht aus all jenen Pfeilen $(u, v) \in E$ mit der folgenden Eigenschaft: Es existieren zwei Knoten $s, t \in V$, so dass ein kürzester Weg zwischen ihnen über (u, v) führt und $v \notin \mathcal{N}_r(s)$ und $u \notin \mathcal{N}_r(t)$ (vgl. [Sanders u. Schultes, 2005, S. 571 f.]). Zudem ist $V_1 = \{u \mid \exists (u, v) \in E_1\} \cup \{u \mid \exists (v, u) \in E_1\}$ (vgl. ebd.). Der Graph G_1 lässt sich interpretieren als Netzwerk derjenigen Pfeile, die *nicht nur* für lokale Wegsuchen Bedeutung besitzen.[20]

Kontrahiertes Highway-Netzwerk

Ausgehend vom Originalgraphen $G = G_0$ ist damit eine neue Hierarchie-Ebene entstanden. Die nächste Ebene, $G_2 = (V_2, E_2, c_2)$, wird nicht aus G_1, sondern aus dem *kontrahierten Highway-Netzwerk* $G_1' = (V_1', E_1', c_1')$ berechnet. Hierbei werden Shortcut-Pfeile über alle Knoten aus V_1 gelegt, die mit genau zwei Pfeilen verbunden sind. V_1' enthält dann alle Knoten aus V_1 abzüglich der durch Shortcuts überbrückten. E_1' besteht aus allen Pfeilen, die zwischen den Knoten aus V_1' verlaufen (vgl. [Sanders u. Schultes, 2005, S. 572]). Inhaltlich entspricht G_1' damit übrigens – abgesehen von der Berücksichtigung der Abbiegebeschränkungen – dem Ergebnis aus Verfahren 4, S. 81, mit Input G_1.

Zusätzliche Hierarchie-Ebenen $G_{l+1} = (V_{l+1}, E_{l+1}, c_{l+1})$ können nach Belieben ausgehend von Ebene l mit $G_l = (V_l, E_l, c_l)$ über $G_l' = (V_l', E_l', c_l')$ erzeugt werden (vgl. ebd.). Dabei ist stets $V_{l+1} \subseteq V_l$. Die Anzahl der Hierarchien ist eine weitere Stellgröße des Verfahrens; Delling u. a. [2009b] arbeiten mit einer einstelligen Zahl; Schultes [2008, S. 165] variiert zwischen sechs und elf Ebenen.

Wegsuche

Alle nachfolgenden Wegsuchen von einem Startknoten s zu einem Zielknoten t erfolgen bidirektional mit einem modifizierten Dijkstra-Verfahren und beginnen auf der untersten Knoten-Hierarchie-Ebene $l = 0$, d. h. auf dem Originalgraphen. Sobald die Vorwärtssuche die Menge $\mathcal{N}_r(s)$ der aktuellen Hierarchie-Ebene l verlässt, wird sie in G_{l+1} fortgeführt. Analoges gilt für die Rückwärtssuche beim Verlassen von $\mathcal{N}_r(t)$. So erreicht die bidirektionale Suche Hierarchie-Ebenen in monoton steigender Reihenfolge,

[19] Delling u. a. [2009a, S. 120] beschreiben eine Variation, bei der die Nachbarschaft $\mathcal{N}_n'(u)$ aus den n zu u nächsten Knoten besteht. Anhand des Dijkstra-Ranges (siehe Abschnitt 4.1.1.1 auf Seite 94) ließe sich formulieren: $\mathcal{N}_n'(u) = \{v \in V \mid r_u(v) \leq n\}$. Hier ist n die Stellgröße.

[20] Sanders u. Schultes [2005] beschreiben das Verfahren o. B. d. A. für ungerichtete Graphen. Gemäß der in dieser Arbeit gegebenen Darstellungsweise müssen daher ungerichtete Kanten eines Graphen vorab in zwei gegensätzlich ausgerichtete Pfeile überführt werden.

woraus aufgrund von $V_{l+1} \subseteq V_l$ ein stark verkleinerter Suchraum entsteht. (vgl. [Sanders u. Schultes, 2005, S. 573 f.])

Ähnlich wie bei der klassischen bidirektionalen Dijkstra-Suche (vgl. Abschnitt 4.1.1.2, S. 95 ff.) ist das Abbruchkriterium nicht erreicht, wenn ein Knoten sowohl von der Vorwärts- als auch von der Rückwärtssuche in die Lösungsmenge aufgenommen wurde. Stattdessen dient die Länge des von einem solchen Brückenknoten induzierten Weges $\hat{p}_{s,t}$ zwischen s und t fortan als obere Schranke für die Länge des gesuchten kürzesten Weges. Die Vorwärtssuche wird so lange weitergeführt, bis kein Knoten der Horizontmenge mehr in einer kürzeren Distanz erreicht werden kann, d. h., es gilt also: $v.distanz_{\text{vor}} \geq c(\hat{p}_{s,t}),\ \forall v \in Horizontmenge_{\text{vor}}$. Analog wird die Rückwärtssuche abgebrochen, sobald erstmals gilt: $v.distanz_{\text{rück}} \geq c(\hat{p}_{s,t}),\ \forall v \in Horizontmenge_{\text{rück}}$. Abbildung 4.12 zeigt den schematischen Ablauf einer Vorwärts-Wegsuche, bei der drei Hierarchie-Ebenen erreicht werden.

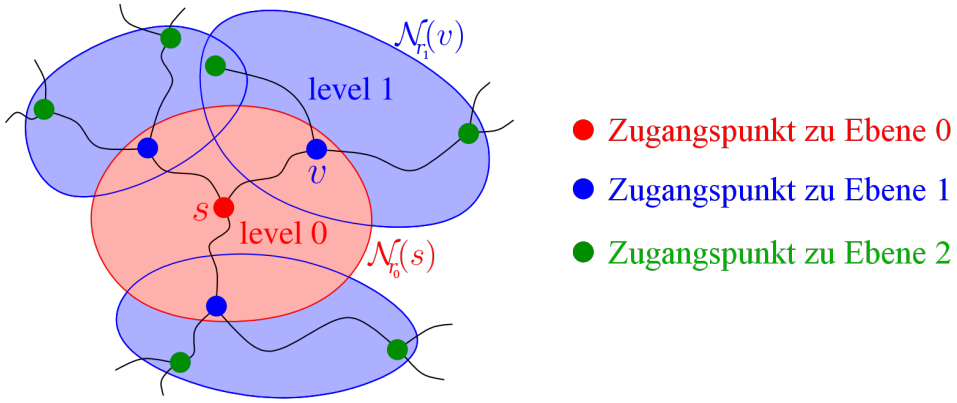


Abbildung 4.12: Schematische Wegsuche beim Highway-Hierarchies-Verfahren
Dargestellt ist die Vorwärtssuche ausgehend von Knoten s. Sie beginnt in der rötlich markierten Radius-Nachbarschaft $\mathcal{N}_{r_0}(s)$ von s und arbeitet in dieser Phase auf dem Originalgraphen, d. h. Level 0. Die äußeren Knoten aus $\mathcal{N}_{r_0}(s)$ führen zur blau markierten, nächsthöheren Hierarchie-Ebene, Level 1. An ihrem Rand erreicht die Suche mit Level 2 die höchste, hier grün dargestellte, Ebene.
Quelle: in Anlehnung an Delling u. a. [2009b, S. 144]

Aufwandsabschätzung

Der stark verkleinerte Suchraum ändert jedoch nichts an der Landau-Notation für den Aufwand des Verfahrens. Wegen der zugrunde liegenden Dijkstra-Suche gilt auch für HH in einem Originalgraphen $G = (V, E, c)$ eine Aufwandsabschätzung von $\mathcal{O}(|V|^2)$.

Für eine zusätzliche Verdopplung der Suchgeschwindigkeit schlagen Sanders u. Schultes [2006] vor, für das höchstrangige – und damit kleinste – Highway-Netzwerk eine vollständige Distanztabelle zu berechnen. So muss keine Dijkstra-Suche mehr erfolgen, sobald die Wegsuche diese Hierarchie-Ebene erreicht.

Beschleunigung

Das Verfahren erreicht dadurch insgesamt eine mehr als 10 000-fache Beschleunigung im Vergleich zum unidirektionalen Dijkstra-Verfahren (vgl. [Sanders u. Schultes, 2006,

S. 812]).[21] Damit war das Verfahren zum Zeitpunkt seiner Veröffentlichung – nach eigenen Angaben – das einzige, mit dem es möglich war, Wegsuche-Anfragen auf großen digitalen Straßenkarten im Millisekundenbereich zu beantworten (vgl. [Delling u. a., 2009a, S. 120]).[22]

Trotz der enormen Suchbeschleunigung ist das HH-Verfahren nicht universell einsetzbar. Die Vorteile werden gewissermaßen durch den für die Vorberechnungen benötigten Speicherplatz „erkauft“. In ihren Messungen geben Sanders u. Schultes [2006, S. 812] einen zusätzlichen Speicherbedarf von knapp 70 Bytes pro Knoten des Graphen an. Bei der von ihnen verwendeten Straßenkarte von Europa mit über 18 Mio. Knoten umfassen die Hierarchie-Informationen demnach mehr als ein Gigabyte Datenspeicher. Steht auf einem (Navigations-)Gerät nicht ausreichend schneller Arbeitsspeicher zur Verfügung, führt dies zwangsläufig dazu, dass nicht das gesamte Geschwindigkeitspotential des Verfahrens ausgenutzt werden kann.

Overhead

4.2.2 Highway Node Routing

Einen mit dem Highway-Hierarchies-Verfahren verwandten Ansatz verfolgt Highway Node Routing (HNR), das von der gleichen wissenschaftlichen Arbeitsgruppe entwickelt wurde. Wie der Name andeutet, bilden hier nicht die Pfeile, sondern die Knoten des zugrunde liegenden gerichteten, gewichteten Graphen $G = (V, E, c)$ die Grundlage für die Hierarchie. Die Hierarchie selbst besteht aus den L Overlay-Graphen[23] $G_0, G_1, \ldots, G_{L-1}$. Dabei ist $G_0 = G$ und G_l ist Overlay-Graph zu G_{l-1} für alle $1 \leq l \leq L$. Die Anzahl der Hierarchie-Ebenen L ist ein Stellparameter für das Verfahren. (vgl. [Schultes u. Sanders, 2007, S. 72])

Abdeckungsknoten

Bevor der Aufbau der Overlay-Graphen näher erläutert wird, ist es notwendig das Konzept der so genannten *Abdeckungsknoten*[24] nach Schultes u. Sanders [2007, S. 69] vorzustellen. Während einer Dijkstra-Wegsuche in $G = (V, E, c)$ mit Startknoten $s \in V$ und gegebener Knotenmenge $V' \subseteq V$ gelte jeder Knoten $v \in \mathit{Horizontmenge} \cup \mathit{Lösungsmenge}$ als *von V' abgedeckt*, wenn der aktuell kürzeste bekannte Weg zu v einen Konten $v' \in V'$ beinhaltet. Die Menge der Abdeckungsknoten $\mathcal{C}_G(V', s)$ bestehe dann aus denjenigen Knoten aus $V' \cap \mathit{Lösungsmenge}$, für die gilt, dass der zu ihnen gefundene kürzeste Weg keinen von V' abgedeckten Knoten beinhaltet (vgl. [Schultes u. Sanders, 2007, S. 70]). Anschaulicher formuliert ist $\mathcal{C}_G(V', s)$ die kleinste Teilmenge derjenigen Knoten aus V', mit der Eigenschaft, dass jeder mit dem Dijkstra-Verfahren ermittelbare kürzeste Weg mit Startknoten s, der über einen Knoten aus V' führt, ein Element aus $\mathcal{C}_G(V', s)$ enthält. Abbildung 4.13, S. 115, zeigt ein einfaches Beispiel für das Konzept.

[21] Die Autoren verwenden als Messgröße die Anzahl der Knoten in der Lösungsmenge nach Terminierung des Verfahrens. Dieser Wert entspricht grundsätzlich der in dieser Arbeit verwendeten Messgröße der Anzahl der Dijkstra-Iterationen, da in jeder Iteration genau ein Knoten in die Lösungsmenge aufgenommen wird. Ferner beziehen sich die Angaben auf die Suche nach *schnellsten* Wegen bei einer Straßenkarte. Die Messergebnisse für *kürzeste* Wege sind nicht explizit angegeben.

[22] Hieraus erklärt sich auch, warum Schultes [2008, S. 165] die schnellsten Wegsuchen für nur sechs Hierarchie-Ebenen verzeichnet und die langsamsten bei elf. Denn auf der obersten Ebene kommen die oben beschriebenen Distanztabellen zum Einsatz, deren Geschwindigkeitsvorteil überragt.

[23] zur Definition: siehe Seite 111

[24] im englischen Original: *covering nodes*

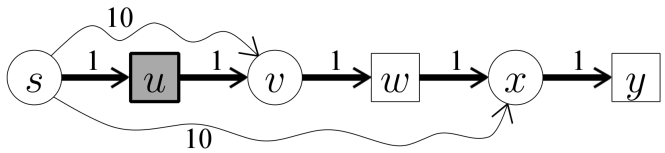


Abbildung 4.13: Beispiel für Abdeckungsknoten bei Highway Node Routing
In dem Beispiel sei $V' = \{u, w, y\}$ (Knoten, um die ein Quadrat gezeichnet ist). Dick markiert sind Pfeile entlang kürzester Wege nach Durchlauf einer Dijkstra-Suche von s nach Verfahren 5, S. 91. Dann ist $\mathcal{C}_G(V', s) = \{u\}$ (grau hinterlegt).
Quelle: in Anlehnung an Schultes u. Sanders [2007, S. 71]

Aufbau der Overlay-Graphen

Ausgehend von der Hierarchie-Ebene l mit $G_l = (V_l, E_l, c_l)$ – wobei G_0 der Original-Graph ist – ergibt sich Ebene $l + 1$ dann zu $G_{l+1} = (V_{l+1}, E_{l+1}, c_{l+1})$ mit

- V_{l+1} = Menge der Knoten aus dem *kontrahierten Highway-Netzwerk* zu G_l[25] und
- $E_{l+1} = \{(u, v) \mid u \in V_{l+1} \text{ und } v \in \mathcal{C}_{G_l}(V_{l+1} \setminus \{u\}, u)\}$

Diejenigen Pfeile $(u, v) \in E_{l+1}$, die nicht in E_l existieren, sind Shortcut-Pfeile über einen kürzesten Weg $p^*_{u,v}$ zwischen u und v in G_l. Ihre Pfeilbewertung entspricht der Länge dieses Weges, d. h. $c_{l+1}(u, v) = c_l(p^*_{u,v})\ \forall (u, v) \in E_{l+1}$.

Da Abdeckungsknoten beim Aufbau der Hierarchie sehr häufig berechnet werden müssen – exakt $|V_l|$ mal pro Hierarchie-Ebene l –, sollte dieser Vorgang möglichst effizient implementiert werden. Schultes u. Sanders [2007, S. 70 f.] führen zu diesem Zweck vier Varianten an und diskutieren deren Vor- und Nachteile.

Grundsätzlich kann V_{l+1} anstatt mit den Knoten aus dem kontrahierten Highway-Netzwerk zu G_l mit *jeder* Knotenauswahl $\subset V_l$ gebildet werden. Allerdings hat die Auswahl einen entscheidenden Einfluss auf sowohl auf die Dauer für den Aufbau einer Hierarchie als auch für nachfolgende Wegsuchen in diesen (vgl. [Schultes u. Sanders, 2007, S. 72]).

Wegsuche
Aufwandsabschätzung

Die Suche erfolgt – analog zum Highway-Hierarchies-Verfahren[26] – mit einem modifizierten, bidirektionalen Dijkstra-Verfahren. Der Aufwand für die Suche ist daher ebenfalls $\mathcal{O}(|V|^2)$. Auch hier wird ein Knoten einer Ebene ausschließlich in Richtung derjenigen Pfeile expandiert, die zu Knoten in gleich- oder höherrangigen Ebenen führen.

Blockieren von Knoten

Auf diese Weise kann es vorkommen, dass Knoten über einen suboptimalen Weg erreicht werden. Schultes u. Sanders [2007, S. 71] tragen diesem Umstand mit einer Technik Rechnung, die sie *stall on demand* (sinngemäß: *Blockieren bei Bedarf*) nennen. Sie entdecken dadurch Knoten, die im aktuellen Suchbaum auf einem nicht-optimalen Weg erreicht wurden und verhindern, dass die Suche über diese Knoten weitergeführt wird; *blockieren* also diese Knoten. Diese Technik wird ausführlicher in Abschnitt 5.2.1, S. 146 ff., vorgestellt.

[25] siehe Seite 112 – vermutlich stammt der Name des Verfahrens aus diesem Zusammenhang
[26] zur Wegsuche beim HH-Verfahren siehe Seite 112, Stichwort „Wegsuche“

Beschleunigung und Overhead

Im Vergleich zur Dijkstra-Wegsuche erreicht das HNR-Verfahren eine über 9 000-fache Beschleunigung (vgl. [Schultes, 2008, S. 165 und 179]).[27] In dieser Variante wird für die Vorberechnung der zugrunde liegenden zwölfstufigen Hierarchie Speicher in der Größenordnung von 9,5 Byte pro Knoten des Graphen benötigt (vgl. [Schultes, 2008, S. 179]). Schultes [2008, S. 179] gibt zwar auch eine Platz sparendere Variante mit elf Hierarchie-Ebenen und 4 Byte pro Knoten an, doch diese ist um den Faktor 1,7 langsamer.

Zusammenfassend arbeitet das HNR-Verfahren im Vergleich zum HH-Verfahren also erheblich Platz sparender. Gleichzeitig erreicht es ähnliche Suchgeschwindigkeiten.

4.2.3 Transit Node Routing

Bei hinreichend langen Autofahrten führt der Weg fast immer nur über eine gewisse Menge von Verkehrsknotenpunkten in der Nähe vom Startpunkt, um von dort weiter zu verzweigen. Aus dieser Beobachtung heraus entstand das Transit Node Routing (TNR). In den meisten Fällen beinhalten die dem Verfahren namensgebenden *Transitknoten* in Deutschland beispielsweise die dem Startpunkt nächstgelegenen Autobahnauffahrten. Gleichsam lassen sich Transitknoten auch für das Ziel der Reise identifizieren. (vgl. [Bast u. a., 2007a, S. 179])

Zugangsknoten

So kann die Gesamtstrecke als Weg vom Startpunkt über einen „seiner“ Transitknoten – genannt *Zugangsknoten* – zu einem Transitknoten des Ziels und schließlich zum Ziel aufgefasst werden. Da die Zugangsknoten in der Nähe ihrer Bezugsknoten liegen, verläuft erwartungsgemäß der längste Teil des Weges zwischen den Zugangsknoten von Start und Ziel. Das TNR-Verfahren nutzt diese Erkenntnis und berechnet einmalig im Voraus für jeden Knoten des Graphen dessen Zugangsknoten, die kürzesten Wege zu diesen, sowie eine vollständige Distanztabelle für die kürzesten Wege zwischen allen Transitknoten-Paaren. Auf diese Weise reduzieren sich nachfolgende Wegsuchen auf den Vergleich weniger Einträge aus vorab berechneten Distanztabellen. Denn ein kürzester Weg ergibt sich dann grundsätzlich als Minimum über alle durch die Zugangsknoten von Start und Ziel erzeugbaren Wege. Ein Sonderfall sind nur noch diejenigen Wege, die über *keine* Transitknoten führen. Diese müssen erkannt und über ein anderes Verfahren ermittelt werden.

Als erstes beschrieben Bast u. a. [2009] das Verfahren. Unabhängig davon entwickelte Schultes [2008] eine Variante, die sich im Wesentlichen nur bei der Auswahl der Transitknoten unterscheidet[28]. Schließlich taten sich beide Forschungsgruppen zusammen und veröffentlichten gemeinsam ihre Ergebnisse (vgl. [Bast u. a., 2007a]).[29]

[27] Der Wert findet sich nur implizit bei Schultes [2008]: Er gibt sowohl für HH als auch für HNR die im Rahmen seiner Benchmarks ermittelte durchschnittliche Anzahl der Knoten in der Lösungsmenge an. Zusätzlich nennt er für HH den Vergleich zum Dijkstra-Verfahren, woraus sich der oben genannte Wert berechnen lässt.

[28] für Details siehe Seite 120 f., Stichwort „Auswahl der Transitknoten“

[29] Zur Klärung der scheinbar nicht chronologischen Jahreszahlen bei den Quellen: Bast u. a. [2009] bezieht sich auf einen wissenschaftlichen Wettbewerb, der bereits 2006 stattfand, die „9. DIMACS Implementation Challenge“ (`http://www.dis.uniroma1.it/~challenge9/`). Der Beitrag ist somit der älteste zum Thema *Transit Node Routing*.

Formal werden für das Verfahren in einem gerichteten, gewichteten Graphen $G = (V, E, c)$ drei zusätzliche Hilfsmittel benötigt:

- die Menge der Transitknoten $\mathcal{T} \subseteq V$,
- die Mengen der Zugangsknoten $\mathcal{Z}_v \subseteq \mathcal{T},\ \forall v \in V$ und
- eine Entscheidungsfunktion[30] $\mathcal{L} : V \times V \to \{wahr, falsch\}$, die für jedes Knotenpaar angibt, ob der kürzeste Weg zwischen ihnen über einen Transitknoten führt (*falsch*) oder nicht (*wahr*).[31]

Der Suchalgorithmus lässt sich dank dieser Hilfsmittel wie in Verfahren 9 (in Anlehnung an Schultes [2008, S. 130 f.]) äußerst kompakt beschreiben. Der oben erwähnte Vergleich zwischen Einträgen aus Distanztabellen findet in Zeile 4 des Verfahrens statt. Wegsuche

Verfahren 9 Weglängensuche beim Transit Node Routing

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$, Startknoten s, Zielknoten t mit $s, t \in V$

1: **if** $\mathcal{L}(s, t)$ **then**
2: Verwende ein anderes Suchverfahren ▸ Lokale Suche ist erforderlich
3: **else**
4: $\text{dist}(s, t) = \min\{\text{dist}(s, u) + \text{dist}(u, v) + \text{dist}(v, t) \mid u \in \mathcal{Z}_s, v \in \mathcal{Z}_t\}$
5: **end if**

Ergebnis: Die Länge des kürzesten Weges zwischen s und t ist bekannt. Der Weg selbst lässt sich im nicht-lokalen Fall durch Auflösen ableiten (siehe Seite 122). <bei der lokalen Suche steht er je nach eingesetztem Suchverfahren schon fest.

Es sei darauf hingewiesen, dass die Entscheidungsfunktion $\mathcal{L}$ keineswegs für alle Knotenpaare korrekt entscheiden muss, damit das Verfahren korrekte Ergebnisse liefert. Es muss allerdings sichergestellt sein, dass $\mathcal{L}$ stets auf *wahr* entscheidet, wenn der kürzeste Weg zwischen Start- und Zielknoten *nicht* über einen Transitknoten führt, da das Verfahren ansonsten einen Umweg liefern würde – nämlich über einen Transitknoten (vgl. [Schultes, 2008, S. 131]).

Auf eine markante Eigenschaft der Zugangsknoten weisen Bast u. a. [2007a, S. 48] zusätzlich hin: Im Rahmen einer lokalen Suche, d. h., $\mathcal{L}$ liefert den Wert *wahr*, können sie als *Landmarken*[32] dienen. Bei Wegsuchen zwischen Knoten s und t gilt diese Aussage allerdings ohne zusätzliche Berechnungen nur für die Knoten aus $\mathcal{Z}_s \cap \mathcal{Z}_t$.

[30] Schultes [2008] und Bast u. a. [2007a] sprechen hier übereinstimmend von einem „locality filter“ (*Lokalitätsfilter*)

[31] Ein Beispiel für die Entscheidungsfunktion findet sich auf Seite 122, Stichwort „Beispiel für Entscheidungsfunktion“. Dieses ergibt sich direkt aus der Auswahl der Transitknoten bei Bast u. a. [2009, S. 180 ff.] und Bast u. a. [2007a, S. 50 f.].

[32] siehe Abschnitt 4.1.2.2, S. 107 ff.

Aufwandsabschätzung

Der asymptotische Aufwand des Verfahrens richtet sich nach der Entscheidungsfunktion $\mathcal{L}$. Liefert sie *wahr*, entspricht er dem Aufwand der Alternativsuche (Zeile 2 in Verfahren 9). Ansonsten müssen die Wege für alle vom Startknoten erreichbaren Transitknoten zu allen vom Endknoten erreichbaren Transitknoten miteinander verglichen werden. Aufgrund der notwendigen Vergleiche in Verfahren 9, S. 117, Zeile 4 wächst der Aufwand in $\mathcal{O}(|\mathcal{T}|^2)$, wobei allerdings $|\mathcal{T}| \ll |V|$.

Abbildung 4.14 zeigt schematisch den Verlauf einer Wegsuche zwischen Startknoten s und Zielknoten t. Angegeben sind die Zugangsknoten von s (blau markiert) und t (rot markiert). In Abbildung 4.15, S. 119, ist ergänzend ein konkretes Beispiel für die Wegsuche in einer Straßenkarte Deutschlands gegeben. Dargestellt sind dort ebenfalls diejenigen Bereiche des Graphen, für die lokale Suchen notwendig sind.

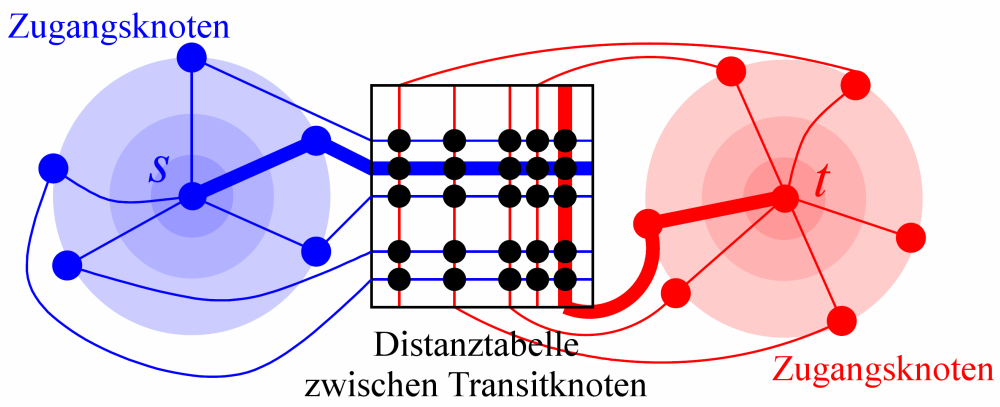


Abbildung 4.14: Schematische Wegsuche beim Transit-Node-Routing-Verfahren
Dargestellt sind Vorwärts- und Rückwärtssuche (blau bzw. rot). Start- und Zielknoten besitzen jeweils fünf Zugangsknoten (fett markiert). Für den kürzesten Weg müssen in diesem Fall nur 25 potentielle Wege (genauer: *Weglängen*) verglichen werden.
Quelle: Schultes [2008, S. 132]

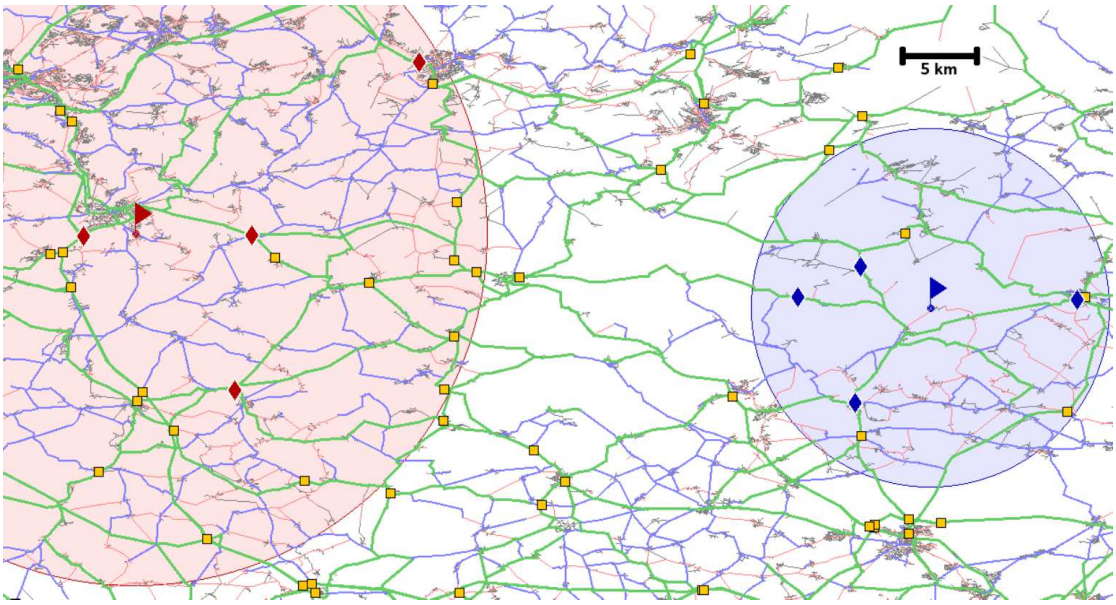


Abbildung 4.15: Beispielhafte Ausprägung und Verteilung der Transitknoten beim Transit-Node-Routing-Verfahren
Dargestellt ist die Wegsuche zwischen den mit Fähnchen markierten Start- und Zielknoten. Rauten einer Farbe symbolisieren die jeweils vier Zugangsknoten des zugehörigen Start- bzw. Zielknotens. Die Runden Flächen stehen für die Bereiche der Karte, für die eine *lokale Suche* anstelle des TNR-Verfahrens durchzuführen wäre. Gelbe Quadrate stehen für Transitknoten, die im Rahmen der aktuellen Wegsuche nicht relevant sind.
Quelle: Bast u. a. [2007b, S. 47]

Aufbau einer Hierarchie (I)

Um die lokalen Suchen zu beschleunigen, kann mittels zusätzlicher Transitknoten gewissermaßen eine weitere Hierarchie-Ebene erzeugt werden (vgl. [Bast u. a., 2007a, S. 49], [Schultes, 2008, S. 133]). In dieser sind die Menge der Transitknoten $\mathcal{T}_2 \supset \mathcal{T}$, die Menge der Zugangsknoten $\mathcal{Z}_{2_v} \supseteq \mathcal{Z}_v$, $\forall v \in V$ sowie die Entscheidungsfunktion $\mathcal{L}_2 : V \times V \rightarrow \{wahr, falsch\}$ zu betrachten. Bei einer Wegsuche zwischen Knoten s und t wird dann zuerst geprüft, ob der Weg bereits anhand der ersten Ebene gefunden werden kann. Dies ist nach wie vor der Fall, falls $\mathcal{L}(s,t)$ den Wert *falsch* liefert. Ansonsten wird die Suchanfrage an die zweite Ebene und $\mathcal{L}_2$ weitergereicht. Der zusätzliche Aufwand der Vorberechnungen für die zweite Ebene beschränkt sich dabei auf kürzeste Wege zwischen allen $u, v \in \mathcal{T}_2$, für die gilt: $\mathcal{L}(u,v) = wahr$ (vgl. ebd.). Weitere Ebenen lassen sich analog hierzu berechnen. Dabei gilt jedoch – entgegen der sonstigen in dieser Arbeit beschriebenen hierarchischen Verfahren –, dass Ebene $l+1$ *mehr* Knoten beinhaltet als Ebene l.

Aufbau einer Hierarchie (II)

Der Speicherplatz für die zusätzlichen Informationen, insbesondere für die Distanztabelle zwischen den Transitknoten, nähme für das Verfahren in der bisher beschriebenen Form immer noch große Ausmaße an. Daher schlägt Schultes [2008] einen Kompromiss vor: Aus den identifizierten Transitknoten und den Pfeilen, die zu kürzesten Wegen zwi-

schen ihnen gehören, erzeugt er mittels des in Abschnitt 4.2.2 vorgestellten Highway Node Routing einen Overlay-Graphen. Offensichtlich lassen sich die Pfeile dabei in größerem Maße zu Shortcut-Pfeilen aggregieren. Anstelle von Distanztabellen erfolgt die Wegsuche zwischen zwei Transitknoten dann mit dem HNR-Verfahren, das seinerseits natürlich über mehrere Hierarchie-Ebenen verfügen kann. (vgl. [Schultes, 2008, S. 143 f.])

Dieser Speicherplatz sparende Ansatz stellt somit eine *alternative* hierarchische Ordnung dar und ist explizit vom oben beschriebenen Ansatz[33] zu unterscheiden, mit dessen Hilfe die *lokalen* Suchen beschleunigt werden sollen. Natürlich sind beide Ansätze auch gemeinsam im selben Graphen anwendbar.

Schultes [2008, S. 148] verwendet eine insgesamt vierstufige Hierarchie, deren unterste Ebene der Originalgraph bildet. Während er damit bei einer Straßenkarte von Europa mit über 18 Mio. Knoten auf der vierten und damit höchsten Ebene nur 14 001 Transitknoten identifiziert, sind es auf der dritten noch 179 972 (vgl. [Schultes, 2008, S. 191]).[34] Dies lässt erahnen, wie viel Speicherplatz durch den Einsatz von Overlay-Graphen anstelle von Distanztabellen eingespart werden kann: Eine vollständige Distanztabelle der vierten Ebene enthält über 196 Mio. Einträge und bei der dritten Ebene wären es über 32 Mrd. Einträge.

Overhead

Der Bedarf an zusätzlichem Speicher für das TNR-Verfahren variiert sehr stark mit der Auswahl der Transitknoten und der Entscheidung darüber, ob Distanztabellen eingesetzt werden, oder die Wege zwischen Transitknoten einer Hierarchie-Ebene durch andere Verfahren wie Highway Node Routing berechnet werden sollen. Bei Schultes [2008, S. 199 und S. 191] finden sich etwa Angaben von 21 bis 301 Bytes pro Knoten des Graphen[35]. Diese Aussagen bezieht Schultes [2008, S. 159] im Vergleich zu einer „Platz sparenden Implementierung des bidirektionalen Dijkstra-Verfahrens“, ohne jedoch näher darauf einzugehen.[36]

Auch Bast u. a. [2009, S. 185] sowie Bast u. a. [2007a, S. 53] nennen den Wert von 21 Bytes pro Knoten. Das große Spektrum des Platzbedarfs unterstreicht nicht zuletzt die Bedeutung einer geschickten Auswahl der Transitknoten – einer zentralen Stellgröße des Verfahrens, auf die nachfolgend näher eingegangen wird.

Auswahl der Transitknoten

Schultes [2008] schlägt vor, die Auswahl mithilfe des HH-Verfahrens zu treffen. Die Knoten eines kontrahierten Highway-Netzwerks (siehe S. 112) bilden bei ihm die Menge der Transitknoten. Die Berechnung der zugehörigen Entscheidungsfunktion ist dadurch jedoch komplex (vgl. [Schultes, 2008, S. 145 ff.]).

Einen anderen Weg stellen Bast u. a. [2009, S. 180 ff.] und Bast u. a. [2007a, S. 50 f.] vor, der hier zunächst für ungerichtete Graphen $G = [E, V, c]$ erläutert wird. Ein Gitternetz wird über den Graphen gelegt und damit jedem Knoten eine Zelle zugewiesen. Die Idee besteht darin, zu definieren, dass alle Knoten einer Zelle Z dieselben Zugangsknoten erhalten und dass diese Zugangsknoten auf einer geschlossenen Linie um Z liegen müssen.

[33] Seite 119, Stichwort: „Aufbau einer Hierarchie (I)“

[34] Die Zahlen der Transitknoten der ersten und zweiten Ebene sind nicht angegeben.

[35] jeweils ohne zusätzliche Transitknoten-Erzeugung für die lokalen Suchen

[36] Im Original: „When we specify the memory consumption of one of our approaches, we usually give the overhead, which accounts for the additional memory that is needed by our approach compared to a space-efficient bidirectional implementation of Dijkstra's algorithm.“

Dann werden systematisch alle kürzesten Wege betrachtet, die aus Z herausführen und diese Linie überqueren. Alle Knoten, an denen die Linie „geschnitten“ wird, sind dann Zugangsknoten zu den Knoten aus Z.

Betrachtet werde Zelle Z. Die Menge der Randknoten[37] $V_Z \subset V$ der Zelle Z sind dann solche Knoten $u \in Z$, die zu Kanten $[u, v] \in E$ gehören, so dass $v \notin Z$. Nun wird die aus 5×5 Zellen bestehende, quadratische Verbundzelle um Z herangezogen. Deren Randknoten seien $V_{inner} \subset V$. Analog beschreibe $V_{outer} \subset V$ die Menge der Randknoten der quadratischen Verbundzelle aus den 9×9 Zellen um Z.[38]

Abbildung 4.16 visualisiert diese Einteilung des Graphen. In der Bildmitte liegt Z mit schwarz hervorgehobenen Randknoten V_Z. Um diese herum liegen die rot dargestellten Knoten aus V_{inner} und ganz außen erneut schwarz markiert die Knoten aus V_{outer}. Die Bestimmung der Zugangsknoten, die in Abbildung 4.16 durch ihre Größe hervorgehobenen sind, wird nachfolgend beschrieben.

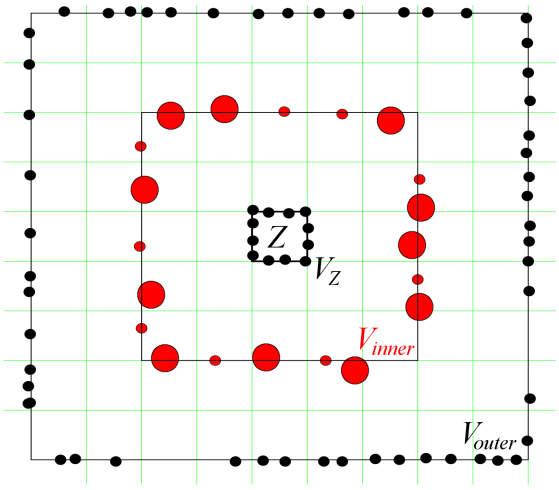


Abbildung 4.16: Bestimmung von Zugangsknoten beim Transit Node Routing
Rot markiert ist die Knotenmenge V_{inner}, aus denen die Zugangsknoten für sämtliche Knoten aus Zelle Z durch ihre Größe hervorgehoben sind. Jeder kürzeste Weg zwischen einem Knoten aus V_Z (Bildmitte, schwarz markiert) zu einem Knoten aus V_{outer} (Bildrand, schwarz markiert) führt über einen dieser Zugangsknoten.
Quelle: in Anlehnung an Bast u. a. [2007a, S. 50]

$W_{Z,outer}$ umfasse die Menge aller kürzesten Wege mit Startknoten aus V_Z und Zielknoten aus V_{outer}. Die Zugangsknoten für alle $u \in Z$ sind dann $\{v \mid v \in V_{inner}$ und $\exists\, p^* \in W_{Z,outer}$, so dass v von p^* überquert wird$\}$. Die Menge der Transitknoten ergibt sich anschließend direkt aus der Vereinigung aller Zugangsknoten. (vgl. [Bast u. a., 2007a, S. 50 f.])

[37] im englischen Original: *crossing nodes*

[38] Selbstverständlich könnten auch andere Größen und Formen der um Z liegenden Verbundzellen gewählt werden. So sind beispielsweise bei Bast u. a. [2009, S. 81] quadratische Verbundzellen aus $2^i \times 2^i$ Zellen mit $i \in \mathcal{N}$ genannt.

Wenn das Verfahren mit einem gerichteten Graphen verwendet wird, müssen die Zugangsknoten pro Zelle auch für die Rückwärtsrichtung ermittelt werden. Hierfür sind also die kürzesten Wege mit Startknoten aus V_{outer} und Zielknoten aus V_Z zu bestimmen.

Beispiel für Entscheidungsfunktion

Ein Vorteil dieser Art der Transitknoten-Auswahl besteht darin, dass die Entscheidungsfunktion $\mathcal{L}$ leicht anhand der vorab bestimmten Zellen zu definieren ist. Für eine Wegsuche muss $\mathcal{L}$ lediglich prüfen, ob zwischen der Zelle, in welcher sich der Startknoten befindet, und der Zelle, in welcher sich der Zielknoten befindet, mindestens vier weitere Zellen in horizontaler oder vertikaler Richtung entlang des Gitters liegen. Im Beispiel von Abbildung 4.16, S. 121, liefert $\mathcal{L}$ daher den Wert *falsch* für alle Wegsuchen von einem Startknoten aus Zelle Z zu einem Zielknoten außerhalb des dargestellten Bereichs des Graphen. Liegt der Zielknoten dagegen innerhalb des dargestellten Bereichs, liefert $\mathcal{L}$ den Wert *wahr*.

Auflösung des Pfads

Im Gegensatz zu den bisher vorgestellten Verfahren liefert das TNR-Verfahren nur die *Länge* eines optimalen Weges, nicht jedoch den Weg selbst. Dieser muss bei Bedarf noch rekonstruiert werden, was daher einen zusätzlichen Aufwand darstellt. Vorgestellt wird hier das Verfahren nach Bast u. a. [2009, S. 182]. Gegeben sei eine Wegsuche von Knoten s nach t und die Lösung wurde anhand der Zugangsknoten $u \in \mathcal{Z}_s$ und $v \in \mathcal{Z}_t$ beantwortet, d. h., der Weg hat die Form $p_{s,t} = \langle s, \ldots, u, \ldots, v, \ldots, t \rangle$ und $c(p_{s,t}) = \text{dist}(s,t)$.[39]

Der Weg kann sowohl in Vorwärts- als auch in Rückwärtsrichtung rekonstruiert werden. O. B. d. A. wird hier das Vorgehen für die Vorwärtsrichtung erläutert. In $p_{s,t}$ folgt direkt auf s genau derjenige Knoten $s' \in \mathcal{N}(s)$, für den gilt: $c(s,s') + \text{dist}(s',t) = \text{dist}(s,t)$.[40] Offensichtlich ist hierbei nur $\text{dist}(s',t)$ zu berechnen, was jedoch nach dem TNR-Verfahren sehr schnell gelingt. Alle weiteren Knoten des gesuchten Pfades werden rekursiv auf die gleiche Weise identifiziert.

Sobald allerdings $\mathcal{L}(s',t) = \textit{wahr}$ gilt, muss $\text{dist}(s',t)$ mithilfe des Wegsucheverfahrens für lokale Suchen bestimmt werden. Es ist daher sinnvoll, den Pfad sowohl in Vorwärts- als auch Rückwärtsrichtung parallel zu rekonstruieren, um lokale Suchen so lange wie möglich zu vermeiden.

Anders als bei allen bisher vorgestellten Verfahren erzeugen beim TNR-Verfahren nicht mehr die absolut längsten Wege die längsten Suchdauern. Stattdessen benötigen die längsten derjenigen Wege, die *nicht über Transitknoten* gebildet werden, d. h. die längsten Wege der lokalen Suchen, die größte Suchdauer (vgl. [Bast u. a., 2007a, S. 51]).

Beschleunigung

Das Verfahren entzieht sich aufgrund des massiven Einsatzes von Distanztabellen dem bisher in dieser Arbeit verwendeten Geschwindigkeitsvergleich mit dem unidirektionalen Dijkstra-Verfahren. Einzig die Größenordnung der notwendigen Vergleiche in Zeile 4 aus Verfahren 9, S. 117, lassen sich abschätzen.

Bast u. a. [2009, S. 184] nennen dazu die folgenden Werte für eine digitale Straßenkarte der USA mit 24 Mio. Knoten und 58 Mio. Pfeilen und die Suche nach *schnellsten*

[39] Wenn der kürzeste Weg zwischen s und t dagegen *ohne Transitknoten* ermittelt wurde, geschah dies – im Rahmen der lokalen Suche – mit einem anderen Verfahren. Der Pfad muss in diesen Fällen natürlich in Abhängigkeit des für die lokale Suchen eingesetzten Verfahrens rekonstruiert werden.

[40] Existiert mehr als ein kürzester Weg von s nach t, lässt sich *einer* von diesen rekonstruieren, indem stets der *erste* Nachfolgeknoten von s gewählt wird, der die obige Gleichung erfüllt.

Wegen: Ausgehend von einer Transitknoten-Bestimmung mithilfe eines Gitternetzes aus 1024×1024 Zellen, ergeben sich pro Knoten des Graphen durchschnittlich 9,1 Zugangsknoten. Bei einem Gitternetz aus 64×64 Zellen sind es dagegen im Mittel 11,4 Zugangsknoten. Für den durch die Transitknoten definierten mittleren Teil eines gesuchten Weges (siehe Abbildung 4.14, S. 118) erfordert dies daher durchschnittlich 82,81 bzw. 129,96 Vergleiche – sofern keine hierarchischen Overlay-Graphen verwendet werden[41], die Distanztabelle für die Transitknoten also vollständig vorliegt. Wie oben erwähnt, ist dieser Fall jedoch aus Gründen des Speicherplatzes impraktikabel. Die Abschätzung muss also als untere Schranke für den Aufwand verstanden werden.

Konkrete Messwerte für die Wegsuche sind bei Schultes [2008, S. 194] nur für die tatsächliche Suchdauer angegeben, d. h. ohne Pfadauflösung. Für zufällige Suchanfragen auf der von ihm verwendeten Straßenkarte von Europa[42] bewegen sich die Antwortzeiten im ein- bis dreistelligen Mikrosekundenbereich; durchschnittlich jedoch niedrig zweistellig.

Bast u. a. [2009, S. 185] trennen ihre Ergebnisse für die US-Straßenkarte zwischen denjenigen Wegen, die über Transitknoten führen, und den „lokalen“ Suchen. Für erstere geben sie eine durchschnittliche Suchdauer von 12 Mikrosekunden an. Die lokalen Suchen lösen sie mit dem Dijkstra-Verfahren und benötigen dafür im Schnitt 5 112 Mikrosekunden.

4.2.4 Hub-Based Labeling

Die bislang in dieser Arbeit vorgestellten hierarchischen Wegsucheverfahren entstanden weitgehend aus experimentellen Erkenntnissen. Abraham u. a. [2010b] entwickelten daher für mehrere Wegsucheverfahren, darunter auch das TNR-Verfahren sowie das im nachfolgenden Kapitel vorgestellten *Contraction Hierarchies*, eine vereinheitlichte, systematische Sichtweise, mit der sie den empirischen Erfolg der Verfahren begründen. Sie beschreiben einen skalaren Wert für Graphen, den sie *Highway-Dimension* des Graphen nennen. Je größer dieser Wert ist, desto schwieriger ist es, allgemeine Wegsuchen im Graphen durchzuführen, d. h. desto geringer ist die durchschnittliche Wegbeschleunigung, die durch ein Verfahren erreicht werden kann. Die Highway-Dimension entspricht vereinfachend formuliert der größten Anzahl Zugangsknoten, die beim TNR-Verfahren für einen Knoten nötig sind (siehe Abschnitt 4.2.3).

Formal ist die Highway-Dimension in Anlehnung an Abraham u. a. [2010b, S. 784] für einen ungerichteten Graphen $G = [V, E, c]$, in dem ausschließlich eindeutige kürzeste Wege existieren, wie folgt definiert: Sei $\mathcal{W}_{u,v}$ die Menge aller Knoten des kürzesten Weges zwischen u und v und $\mathcal{N}_r(u)$ die Menge der Radius-Nachbarknoten zu Knoten u.[43] Die *Highway-Dimension* von G ist dann die kleinste Zahl h, für die gilt

$$\begin{aligned}
&\forall r \in \mathbb{R}^+, \forall u \in V, \exists S \subseteq \mathcal{N}_{4r}(u) \text{ so dass} \\
&|S| \leq h \text{ und} \\
&\forall v, w \in \mathcal{N}_{4r}(u) \text{ gilt: } \Big(\mathrm{dist}(v, w) > r \text{ und } \mathcal{W}_{v,w} \subseteq \mathcal{N}_{4r}(u)\Big) \Rightarrow \mathcal{W}_{v,w} \cap S \neq \emptyset.
\end{aligned}$$

[41] siehe Seite 119, Stichwort „Aufbau einer Hierarchie (II)“

[42] mit 18 Mio. Knoten, siehe Seite 120

[43] Die Radius-Nachbarschaft wurde in dieser Arbeit im Kontext der Highway Hierarchies eingeführt (siehe Seite 112). Diese Menge umfasst alle Knoten, die höchstens r Längeneinheiten von Knoten u entfernt sind.

Eine erweiterte Definition für gerichtete Graphen nennen Abraham u. a. [2010b, S. 785] ebenfalls. Für die Details zur Highway-Dimension sei an dieser Stelle auf die Originalquelle verwiesen.

Abraham u. a. [2010b, S. 783 ff.] argumentieren, dass Graphen, die aus digitalen Straßenkarten interpretiert wurden, eine geringe Highway-Dimension besitzen und nennen dies als Grund für den Erfolg mehrerer Wegsucheverfahren. Auf diesen Erkenntnissen aufbauend entwickeln Abraham u. a. [2011] ihr Verfahren: *Hub-Based Labeling (HL)*. Es basiert auf den namensgebenden *Labels* für Knoten des Graphen. Analog zu den Arc-Flags beim Arc-Flag-Verfahren[44] sind bei HL in den Knoten-Labels Informationen über die kürzesten Wege kodiert. In gerichteten Graphen sind die Labels in Vorwärts- und Rückwärtslabels aufgeteilt. Für einen Knoten $u \in V$ umfasst sein Label zwei Knotenmengen $L^K_{u,\text{vor}} \subseteq V$ und $L^K_{u,\text{rück}} \subseteq V$ sowie die beiden Mengen der zugehörigen Distanzen $L^D_{u,\text{vor}}$ und $L^D_{u,\text{rück}}$ zu jedem dieser Knoten: $L^D_{u,\text{vor}}(v) = \text{dist}(u,v), v \in L^K_{u,\text{vor}}$ bzw. $L^D_{u,\text{rück}}(v) = \text{dist}(v,u), v \in L^K_{u,\text{rück}}$.

Abdeckungs-Eigenschaft

Durch eine geeignete Knotenauswahl für die Labels kann eine so genannte *Abdeckungs-Eigenschaft*[45] erreicht werden. Ist diese Eigenschaft erfüllt, beinhaltet $L^K_{s,\text{vor}} \cap L^K_{t,\text{rück}}$ für zwei Knoten $s \in V$ und $t \in V$ stets einen Knoten $v \in V$, der Teil eines kürzesten Weges zwischen s und t ist, und $L^D_{s,\text{vor}}(v) = \text{dist}(s,v)$ sowie $L^D_{t,\text{rück}}(v) = \text{dist}(v,t)$.

Wegsuche

Brückenknoten

Besitzen alle Knoten des Graphen Labels mit dieser Abdeckungs-Eigenschaft, ist die Suche nach der Länge des kürzesten Weges leicht. Für einen Startknoten s und einen Zielknoten t muss nur noch derjenige Brückenknoten $m \in L^K_{s,\text{vor}} \cap L^K_{t,\text{rück}}$ ermittelt werden, der $L^D_{s,\text{vor}}(m) + L^D_{t,\text{rück}}(m)$ und damit auch $\text{dist}(s,m) + \text{dist}(m,t)$ minimiert.

Abbildung 4.17, S. 125, zeigt ein Beispiel, das der Übersichtlichkeit halber auf einem ungerichteten Graphen basiert. Daher wird nicht zwischen Vorwärts- und Rückwärtslabels unterschieden. Bei der Wegsuche von Knoten u nach Knoten z_2 ist $L^K_u \cap L^K_{z_2} = \{x, z\}$ und somit kommen x und z als Brückenknoten im kürzesten Weg in Frage.

[44] siehe Abschnitt 4.1.2.3, S. 108 ff.
[45] bei Abraham u. a. [2010a, S. 6]: *cover property*

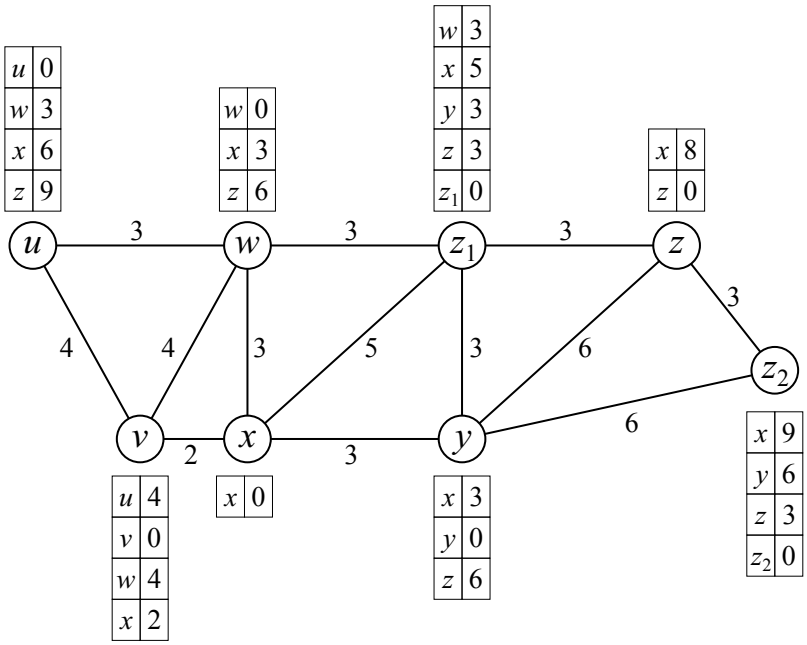


Abbildung 4.17: Beispiel für Labels beim Hub-Based-Labeling-Verfahren
Der Beispielgraph ist ungerichtet, daher muss nicht zwischen Vorwärts- und Rückwärtslabels unterschieden werden. Pro Knoten $v \in V$ sind die Labels als Tabelle dargestellt, in deren linker Spalte die Elemente aus L_v^K stehen. In den rechten Spalten sind jeweils die zugehörigen Werte aus L_v^D angegeben. Da für jeden Knoten $v \in V$ sowohl $v \in L_v^K$ als auch $L_v^D(v) = 0$ gilt, muss dies in einer konkreten Implementierung des Verfahrens selbstverständlich nicht explizit gespeichert werden.

Da $L_u^D(z) + L_{z_2}^D(z) = 9 + 3 = 12 < L_u^D(x) + L_{z_2}^D(x) = 6 + 9 = 15$, steht z als Brückenknoten gleichzeitig mit der Länge des kürzesten Weges, nämlich $\text{dist}(u, z_2) = 12$, fest. Sämtliche paarweise kürzesten Verbindungen können auf diese Weise im angegebenen Beispielgraphen ermittelt werden.

Auflösung des Pfads

Der implizierte Pfad muss bei Bedarf anschließend noch aufgelöst werden. Dies geschieht ähnlich wie beim TNR-Verfahren im Rahmen der nicht-lokalen Suche[46]. In einem kürzesten Weg zwischen den Knoten s und t folgt auf s derjenige Knoten $s' \in \mathcal{N}(s)$, für den gilt: $c(s, s') + \text{dist}(s', t) = \text{dist}(s, t)$[47]. Der noch unbekannte Wert $\text{dist}(s', t)$ ist für alle $s' \in \mathcal{N}(s)$ anhand des HL-Verfahrens sehr schnell zu berechnen, so dass der Pfad sukzessiv rekonstruiert werden kann.

[46] siehe Abschnitt 4.2.3 auf Seite 122, Stichwort „Auflösung des Pfads“

[47] Bei mehrdeutigen kürzesten Wegen reicht es, den *ersten* solchen Nachfolgerknoten $s' \in \mathcal{N}(s)$ auszuwählen, um *einen* kürzesten Weg zu erhalten.

Aufwandsabschätzung

Wie auch beim TNR-Verfahren fällt der theoretische Vergleich mit dem Dijkstra-Verfahren schwer, da die Suche nicht mit Dijkstra-Iterationen arbeitet. Die Lösung wird im Wesentlichen anhand der Labels von Start- und Zielknoten gefunden. Abraham u. a. [2010a, S. 6] weisen aber darauf hin, dass ein in Bezug auf die Größe der Labels, d. h. den Wert $|L^K_{u,\text{vor}} \cup L^K_{u,\text{rück}}|$, *linearer* Aufwand der Suche durch eine geschickte Implementierung erreichbar ist. Die Idee wird nachfolgend skizziert.

Bei der Suche nach den Knoten, die sowohl in den Labels des Startknotens s als auch in denen des Zielknotens t vorkommen, d. h. bei der Suche nach den Brückenknoten, muss für jedes Element aus $L^K_{s,\text{vor}}$ geprüft werden, ob es auch in $L^K_{t,\text{rück}}$ vorhanden ist, um die Schnittmenge dieser beiden Knotenmengen zu bilden. Werden die Knoten aus den Labels jedoch sortiert nach Knoten-Id vorgehalten, muss jedes Element aus $L^K_{s,\text{vor}}$ und $L^K_{t,\text{rück}}$ nur höchstens einmal betrachtet werden, da es ausreicht beide Mengen jeweils einmalig gemäß dieser Sortierung zu durchwandern. Im Beispiel in Abbildung 4.17, S. 125, sind die Labels bereits nach Knoten-Id sortiert. Für n Einträge pro Label ist der Aufwand für die Suche nach der Länge eines kürzesten Weges dann also $\mathcal{O}(n)$.

Beschleunigung

Das Verfahren erreichte zum Zeitpunkt seiner Veröffentlichung die bislang schnellsten Geschwindigkeiten bei der Suche nach kürzesten Distanzen und der Dauer von schnellsten Wegen und löste damit das TNR-Verfahren ab. Abraham u. a. [2011, S. 239] berichten von durchschnittlichen Suchdauern auf der auch bei Schultes [2008] verwendeten Europa-Straßenkarte von unter 300 Nanosekunden. Nach ihren Messungen sind sie damit mehr als 3,5mal so schnell wie die schnellste bis dato bekannte TNR-Variante.

Overhead

Eine Aussage, wie viel zusätzlicher Speicher pro Knoten des Graphen anfällt, geben Abraham u. a. [2011] nicht an, sondern nur absolute Werte. Allerdings führen sie für das TNR-Verfahren ebenfalls absolute Werte auf, was eine indirekte Ermittlung ermöglicht. Demnach ist für die schnellste HL-Variante mehr als der 5,7-fache Speicher notwendig als für die bei Bauer u. a. [2008] aufgeführte TNR-Variante (vgl. [Abraham u. a., 2011, S. 239]). Da diese TNR-Variante einen Overhead von 204 Bytes pro Knoten des Graphen hat und dieselbe Europa-Straßenkarte bei beiden Autorengruppen zugrunde liegt, kann daraus ein Overhead für das HL-Verfahren abgeschätzt werden: $5,7 \times 204$ Bytes pro Knoten $= 1162,8$ Bytes pro Knoten.

Vermutlich von diesem vergleichsweise großen Wert ausgehend, entwickeln Abraham u. a. [2011] eine HL-Variante, die auf geringeren Speicherbedarf abzielt. Sie ist zwar nur gut halb so schnell bei der Weg(längen)suche, benötigt aber um etwa Faktor 3,7 weniger Speicher (vgl. [Abraham u. a., 2011, S. 239]).

Bislang wurde noch nicht begründet, warum das HL-Verfahren in dieser Arbeit den *hierarchischen* Wegsuchen zugeordnet wurde. Der Grund liegt in den Vorberechnungen zu den Labels, die auf dem im nachfolgenden Kapitel 5 näher beschrieben Contraction-Hierarchies-Verfahren basieren, das seinerseits hierarchisch ist. Abschnitt 5.2.4, S. 157, erläutert diesen Zusammenhang zwischen Contraction Hierarchies und HL.

4.3 Zusammenfassung der vorgestellten Wegsucheverfahren

Tabelle 4.2, S. 128, zeigt noch einmal zusammenfassend einen Vergleich zwischen den vorgestellten Verfahren. Dargestellt sind der Aufwand für allgemeine Graphen, die durchschnittliche Beschleunigung bei zufälligen Suchanfragen im Vergleich zum unidirektionalen Dijkstra-Verfahren, die höchste denkbare Suchbeschleunigung in einem beliebigen Graphen sowie der gegenteilige und somit ungünstigste Fall und der zusätzliche Speicherbedarf für eine sinnvolle Konfiguration des jeweiligen Verfahrens. Als Grundlage der Geschwindigkeit dient wie zuvor die Anzahl der benötigten Dijkstra-Iterationen bei der Suche. Für Verfahren, die keine solchen Iterationen durchführen, wurde auf absolute Zeitwerte zurückgegriffen, wie sie bei den Originalquellen angegeben sind.

Best Case

Die Aussagen zur maximalen Beschleunigung beziehen sich meist auf Szenarien, die für das unidirektionale Dijkstra-Verfahren besonders ungünstig sind. Ein Beispiel hierfür ist in Abbildung 4.3a, S. 96, dargestellt.

Worst Case

Der im direkten Vergleich ungünstigste Fall dagegen tritt für die meisten hier beschriebenen Wegsucheverfahren ein, wenn Start- und Zielknoten der Wegsuche direkt über einen Pfeil verbunden sind. Das HL-Verfahren etwa iteriert *stets* über alle Labels von Start- und Zielknoten, so dass die Distanz zwischen den beiden Knoten für das Verfahren unerheblich ist.[48] Da die Anzahl der Labels mit der Größe des Graphen wächst, lässt sich (künstlich) der Spezialfall erzeugen, für den die unidirektionale Dijkstra-Suche beliebig schneller verläuft als die HL-Suche.

Nicht dargestellt ist das PA*-Verfahren[49], da dieses bei Werten von $\alpha > 1$ nur noch heuristisch arbeitet. Ansonsten, d. h. für $0 \leq \alpha \leq 1$, bewegt es sich gewissermaßen „zwischen“ dem Dijkstra- und dem Standard-A*-Verfahren.

[48] Abraham u. a. [2011, S. 237 f.] geben auch eine Variante an, bei der dies nicht der Fall ist. Sie verwenden hier so genannte *Partition Oracles*, ein dem Arc-Flags-Verfahren ähnliches Vorgehen, mit dem die Karte in Gebiete eingeteilt wird und anschließend kürzeste Wege zwischen den Gebieten vorberechnet werden. Diese Vorberechnungen dienen nachfolgenden Wegsuchen als Schranken bei der Iteration über die Labels von Start- und Zielknoten. Die Technik erfordert allerdings wiederum einen höheren Speicherbedarf und die nachfolgende Aussage über das Verfahren gilt weiterhin.

[49] siehe Abschnitt 4.1.1.3, S. 103

Verfahren	Aufwand	Beschleunigung[I] durchschnittlich	Beschleunigung[I] maximal	Beschleunigung[I] minimal	Overhead[II]
unidirektionale Dijkstra-Suche	$\mathcal{O}(\lvert V\rvert^2)$	-	-	-	-
bidirektionale Dijkstra-Suche	$\mathcal{O}(\lvert V\rvert^2)$	~2×	beliebig[III]	halb so schnell	-
A*	$\mathcal{O}(\lvert V\rvert^2)$	~4×	beliebig[III]	mindestens so schnell	-
Trennlinien	$\mathcal{O}(\lvert V\rvert^2)$	> A*	beliebig[X]	mindestens so schnell[X]	pro Trennlinie
ALT	$\mathcal{O}(\lvert V\rvert^2)$	> A*	beliebig[IV]	mindestens so schnell[X]	pro Landmarke und Pfeil
Arc-Flags	$\mathcal{O}(\lvert V\rvert^2)$	≫ A*	beliebig[IV]	mindestens so schnell[X]	pro Region und Pfeil: 1 Bit
HH	$\mathcal{O}(\lvert V\rvert^2)^{V,IX}$	~10 000×	beliebig[IX]	halb so schnell[IX]	~70 Bytes pro Knoten
HNR	$\mathcal{O}(\lvert V\rvert^2)^{V,IX}$	~9 000×	beliebig[IX]	halb so schnell[IX]	~9.5 Bytes pro Knoten
TNR	$\mathcal{O}(\lvert \mathcal{T}\rvert^2)^{VI}$	~1 800 000×[VII]	beliebig[XI]	variabel[VIII]	~21–301 Bytes p. Knoten[VIII]
HL	$\mathcal{O}(\lvert L^K_{\text{vor}} \cup L^K_{\text{rück}}\rvert)^{XII}$	~11 520 000×[XIII]	beliebig[III]	beliebig[XIV]	~1163 Bytes pro Knoten[XV]

[I] gegenüber einer unidirektionalen Dijkstra-Suche nach schnellsten Wegen auf einer Straßenkarte Europas
[II] für sinnvolle Konfigurationen
[III] Graphen und Wegsuchen mit beliebiger Beschleunigung denkbar
[IV] entsprechend A* mit perfektem Schätzer
[V] allerdings etwa logarithmisch geringerer Aufwand pro Hierarchie-Ebene
[VI] mit $\lvert\mathcal{T}\rvert$ = Menge an Zugangsknoten pro Knoten im Graph, daher: $\lvert\mathcal{T}\rvert \ll \lvert V\rvert$
[VII] bei *schnellsten* Wegen ~200× schneller als HNR (vgl. [Schultes, 2008, S. 199])
[VIII] in Abhängigkeit des eingesetzten Verfahrens für lokale Suche
[IX] da das Verfahren auf einer bidirektionalen Dijkstra-Suche basiert: siehe bidirektionale Dijkstra-Suche
[X] zur Begründung: siehe A*-Suche in Abschnitt 4.1.1.3 und insbesondere der Beweis auf Seite 102
[XI] für Wege über Transitknoten
[XII] mit $\lvert L^K_{\text{vor}} \cup L^K_{\text{rück}}\rvert$ = durchschnittliche Anzahl der Label-Einträge pro Knoten im Graph
[XIII] Abraham u. a. [2010a, S. 22] geben für HL um Faktor 6,4 schnellere Ergebnisse an als TNR bei Bauer u. a. [2008]
[XIV] da bei HL die Suche über die Labels erfolgt, ist theoretisch für sehr kurze Wege auf sehr großen Graphen eine Verlangsamung möglich
[XV] Abraham u. a. [2011, S. 239] geben einen um mehr als Faktor 5,7 größeren Speicherbedarf an als ihre TNR-Referenz Bauer u. a. [2008]

Tabelle 4.2: Überblick und Vergleich der vorgestellten Wegsucheverfahren für Graphen $G = (V, E, c)$

4.4 Kombinierte Wegsucheverfahren

Grundsätzlich sind alle der hier vorgestellten Verfahren um entweder eine zielgerichtete oder eine hierarchische Komponente erweiterbar – je nachdem, welche ihnen fehlt. Wie zuvor gilt für die Kombinationen, dass beschleunigte Wegsuchen in den meisten Fällen auch einen höheren Speicherbedarf für die Vorberechnungen mit sich bringen.

Eine ausführliche Übersicht über die Kombination erfolgreicher hierarchischer Verfahren mit zielgerichteten Verfahren findet sich bei Delling u. a. [2009a] und speziell auch bei Bauer u. a. [2008].[50] Besonders hervorzuheben ist hier die Erweiterung des TNR-Verfahrens um Arc-Flags. Mit einem um etwa 1,5-fachen Mehrbedarf an Speicher für die Vorberechnungen der Arc-Flags können so etwa doppelt so schnelle Wegsuchen erreicht werden als mit dem TNR-Verfahren an sich (vgl. [Bauer u. a., 2008, S. 313]).

Das in den nachfolgenden beiden Kapiteln vorgestellte hierarchische Verfahren wird seinerseits erfolgreich um die zielgerichtete A*-Suche erweitert. Ausführliche Benchmarks hierzu finden sich in den Abschnitten 5.6 und 6.6.

[50] Das HL-Verfahren fehlt allerdings in dem Überblick, da es erst später veröffentlicht wurde.

5 Contraction Hierarchies

Inhalt

Auf ein spezielles, hierarchisches Wegsucheverfahren soll in dieser Arbeit im Detail eingegangen werden: die *Contraction Hierarchies* (CH) von Geisberger u. a. [2008]. Das Verfahren besitzt drei wesentliche Eigenschaften:

1. Alle Knoten eines Graphen haben einen eindeutigen Hierarchie-Wert bzw. *Rang*.

2. Die Wegsuche erfolgt bidirektional nach dem Dijkstra-Verfahren, wobei nach jeder Iteration zwischen Vorwärts- und Rückwärtssuche abgewechselt wird und für jede Suchrichtung nur Pfeile expandiert werden, die zu einem *höherrangigen* Knoten führen.

3. Im Rahmen eines einmaligen Präprozesses wird der Graph um Shortcut-Pfeile angereichert, die sicherstellen, dass später alle kürzesten Wege gefunden werden können.

Wie auch die in Abschnitt 4.2 vorgestellten Verfahren, bestehen CH ebenfalls aus einem Präprozess-Schritt – der Erzeugung der Contraction Hierarchy im engeren Sinne – und der eigentlichen Wegsuche auf dieser Hierarchie. Der folgende Abschnitt erläutert zunächst die Grundzüge der Erzeugung der Hierarchie. Danach wird in Abschnitt 5.2 die Wegsuche genauer vorgestellt und die in Abschnitt 4.2.2, S. 115, angedeutete Technik des Blockierens von Knoten erklärt. Zusätzlich werden mehrere Möglichkeiten der Suchbeschleunigung zunächst in der Theorie vorgestellt. Abschnitte 5.3 und 5.4 bieten einen vertiefenden Einblick in die Details der Hierarchie-Erzeugung. Nachdem wesentliche Erkenntnisse hinsichtlich der Implementierung der beschriebenen Verfahren in Abschnitt 5.5 vorgestellt werden, schließt das Kapitel in Abschnitt 5.6 mit systematischen Benchmarks zur empirischen Überprüfung der vorab beschriebenen Überlegungen.

Notation der Knotenränge

Die Knotenränge in diesem Kapitel beginnen bei 1. Ihre Notation im Pseudocode wird als Attribut eines Knotens modelliert. Daher bedeutet $v.rang = 3$, dass Knoten v den Rang 3 besitzt. In der mathematischen Darstellung erscheint es jedoch lesbarer, den Rang als Funktion zu betrachten und deshalb zu schreiben: $\text{rang}(v) = 3$.

Contraction Hierarchies können als Spezialfall des in Abschnitt 4.2.2, S. 114 ff., vorgestellten Highway Node Routing verstanden werden. Für jeden Knoten des Originalgraphen existiert dabei eine eigene Hierarchie-Ebene. Die Wegsuche verläuft jedoch sowohl schneller als auch speichereffizienter (vgl. [Geisberger, 2008, S. 10]).

5.1 Hierarchie-Erzeugung durch Knotenkontraktion

Wie auch das Dijkstra-Verfahren[1] benötigt das CH-Verfahren einen nicht-negativ bewerteten, gerichteten Graphen als Eingabe. Während des Präprozess-Schrittes wird eine hierarchische Ordnung über die Knoten des Graphen anhand eines Verfahrens erzeugt, das als *Knotenkontraktion* bezeichnet wird. Aus diesem grundlegenden Verfahren leitet sich der Begriff *Contraction Hierarchies* ab.

[1] siehe Abschnitt 4.1.1.1, S. 89 ff.

Wie erwähnt, besitzt jeder Knoten einen eindeutigen Rang, so dass sich gemäß diesem Rang eine Sortierung vornehmen lässt. Die eigentliche *Ermittlung* der Knotenränge wird in Abschnitt 5.3, S. 158 ff., ausführlich erläutert. Sie sei an dieser Stelle als gegeben anzunehmen.

Die Knoten $v \in V_0$ des nicht-negativ bewerteten Originalgraphen $G_0 = (V_0, E_0, c_0)$ werden nacheinander, aufsteigend nach ihrem jeweiligen Rang *kontrahiert*. Das bedeutet, sie werden zusammen mit allen anliegenden Pfeilen aus dem Graphen entfernt. Da am Ende nur noch ein einzelner Knoten übrig ist, entstehen sukzessive insgesamt $|V_0|-1$ neue Graphen $G_l = (V_l, E_l, c_0)$ für $1 < l \leq |V_0|$. O. B. d. A. seien die Knoten aus G_0 aufsteigend nach Rang nummeriert, so dass $V_0 = \{v_1, v_2, \ldots, v_{|V_0|}\}$ und $\text{rang}(v_l) < \text{rang}(v_{l+1})$ für alle $1 \leq l < |V_0|$. Dann gilt:

$$V_l = V_{l-1} \setminus \{v_l\}, \tag{5.1}$$

$$E_l = \{(u, v) \mid (u, v) \in E_{l-1} \text{ und } u, v \in V_l\}.$$

Es ist leicht zu erkennen, dass auf diese Weise kürzeste Wege zwischen zwei übrig gebliebenen, zu v adjazenten Knoten zerstört werden können. Deshalb sind die Graphen G_l zumeist keine Overlay-Graphen[2]. Mit ihrer Hilfe lassen sich jedoch Overlay-Graphen erzeugen, wenn zusätzliche Shortcut-Pfeile[3] eingefügt werden: Wird Knoten v_l kontrahiert und führt der einzige kürzeste Weg zwischen zwei zu v_l adjazenten Knoten $u \in \mathcal{V}(v_l)$ und $w \in \mathcal{N}(v_l)$ über v_l, wird ein Shortcut-Pfeil $(u \ldots w)$ in den Overlay-Graphen eingefügt. Die Bewertung von $(u \ldots w)$ ergibt sich als Summe der beiden überbrückten Pfeile (u, v_l) und (v_l, w):

Shortcut-Pfeile

$$c(u \ldots w) = c(u, v_l) + c(v_l, w). \tag{5.2}$$

Sollte zu diesem Zeitpunkt ein Original-Pfeil $(u \to w) \in E_l$ existieren, muss offensichtlich gelten: $c(u \ldots w) < c(u \to w)$, da ansonsten bei der oben beschriebenen Kontraktion von v nicht der einzige kürzeste Weg betroffen gewesen wäre. Der Pfeil $(u \to w)$ kann also nicht Teil eines kürzesten Weges sein und ist für knotenbasierte Wegsuchen somit obsolet. Er wird daher aus dem Graphen entfernt.

Obsolete Original-Pfeile

Durch den Einbezug von Shortcut-Pfeilen kann dann ein tatsächlicher Overlay-Graph $G'_l = (V_l, E'_l, c_l)$ angegeben werden. Seien E_l^{neu} die Menge der bei der l-ten Knotenkontraktion erzeugten Shortcut-Pfeile, $E_l^{\text{ob}} \subset E_0$ die Menge der bei der l-ten Knotenkontraktion als obsolet erkannten Original-Pfeile und formal $E'_0 = E_0$, dann gilt für $1 \leq l < |V_0|$:

$$E'_l = (E_l \setminus E_l^{\text{ob}}) \cup E_l^{\text{neu}} \text{ und} \tag{5.3}$$

$$c_l : E'_l \to \mathbb{R} \text{ mit } c_l(u, w) = \begin{cases} c_{l-1}(u, w), & \text{falls } (u, w) \in E'_{l-1} \setminus E_l^{\text{ob}} \\ c_{l-1}(u, v_l) + c_{l-1}(v_l, w), & \text{sonst} \end{cases}$$

[2] zur Definition siehe Seite 111

[3] zur Notation von Shortcut-Kanten und Shortcut-Pfeilen siehe Seite 110

Nach Kontraktion *aller* Knoten lässt sich der Digraph einer vollständigen Contraction Hierarchy erzeugen. Er besteht aus allen Knoten und Pfeilen des originalen Graphen, abzüglich der als obsolet identifizierten Original-Pfeile, und zusätzlich aus allen Shortcut-Pfeilen, die durch Knotenkontraktionen erzeugt wurden. Zusätzlich werden die beiden Funktionen e_1 und e_2 eingeführt, die jeden Shortcut-Pfeil des Graphen auf den ersten bzw. zweiten von ihm überbrückten Pfeil abbilden und jeden Original-Pfeil auf sich selbst. Für eine Contraction Hierarchy $G_{\text{ch}} = (V_0, E_{\text{ch}}, c_{\text{ch}}, e_1, e_2)$ gilt daher:

$$E_{\text{ch}} = \left(\bigcup_{l=0}^{|V_0|-1} E'_l \right) \setminus \left(\bigcup_{l=1}^{|V_0|-1} E_l^{\text{ob}} \right), \tag{5.4}$$

$$c_{\text{ch}} : E_{\text{ch}} \to \mathbb{R} \text{ mit } c_{\text{ch}}(u,v) = c_l(u,v) \text{ und } l = \max_{0 \le l' < |V_0|} \left\{ l' \mid (u,v) \in E'_{l'} \right\},$$

$$e_1 : E_{\text{ch}} \to E_{\text{ch}} \text{ mit}$$
$$e_1((u,w)) = \begin{cases} (u,v), & \text{falls } (u,w) \text{ durch Kontraktion von Knoten } v \text{ entstand} \\ (u,w), & \text{sonst, d. h., } (u,w) \in E_0 \end{cases} \quad \text{und}$$

$$e_2 : E_{\text{ch}} \to E_{\text{ch}} \text{ mit}$$
$$e_2((u,w)) = \begin{cases} (v,w), & \text{falls } (u,w) \text{ durch Kontraktion von Knoten } v \text{ entstand} \\ (u,w), & \text{sonst, d. h., } (u,w) \in E_0 \end{cases}$$

Im Rahmen eines iterativen Verfahrens zur Erzeugung einer Contraction Hierarchy kann die Kontraktion des Knotens $v_{|V_0|-1}$, d. h. des Knoten mit zweithöchstem Rang, bereits keine Shortcut-Pfeile mehr erfordern. Denn ist $v_{|V_0|-1}$ an der Reihe, besitzt der aktuelle Overlay-Graph nur noch die beiden Knoten $v_{|V_0|-1}$ und $v_{|V_0|}$. Die Kontraktion eines dieser beiden Knoten kann somit keine kürzesten Wege zerstören. Streng genommen kann daher der Hierarchie-Aufbau an dieser Stelle abgebrochen werden, so dass der Laufindex l in Gleichung (5.1), S. 133, Gleichung (5.3), S. 133, sowie Gleichung (5.4) bis inklusive $|V_0| - 2$ ausreicht.

Die Pseudocode-Notation einer Knotenkontraktion ist in Verfahren 10, S. 135, von Zeile 6 bis einschließlich Zeile 17 angegeben. Im Pseudocode wird der Einfachheit halber nicht zwischen den einzelnen Overlay-Graphen unterschieden, sondern die Pfeilmenge E des Eingangsgraphen G sukzessive aktualisiert. Zusätzlich sind die Funktionen c_{ch}, e_1 sowie e_2 mit angegeben.

Verfahren 10 Erzeugung einer Contraction Hierarchy

Gegeben:

Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$,
Prioritätswarteschlange P mit allen nach Wichtigkeit sortierten Knoten aus G

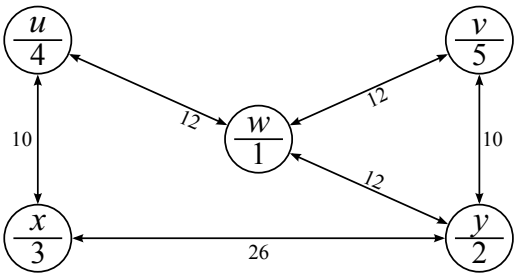
1: Setze $e_1((u,v)) \coloneqq e_2((u,v)) \coloneqq (u,v) \quad \forall (u,v) \in E$ ▸ Initialisierung
2: Setze $c_{\text{ch}}(u,v) \coloneqq c(u,v) \quad \forall (u,v) \in E$
3: **while** $|P| > 2$ **do** ▸ Die beiden höchstrangigen Knoten müssen nicht explizit kontrahiert werden
4: $v \coloneqq \min\{P\}$ ▸ Wähle das kleinste Element aus der Prioritätswarteschlange
5: $P \coloneqq P \setminus v$
6: **for all** $u \in \mathcal{V}(v)$ **do** ▸ Beginn der Kontraktion von v
7: **for all** $w \in \mathcal{N}(v)$ **do**
8: **if** $\langle u, v, w \rangle$ ist der einzige kürzeste Weg von u nach w **then**
9: **if** Es existiert ein Original-Pfeil $(u \to w) \in E$ **then**
10: Lösche $(u \to w)$ endgültig ▸ Denn $(u \to w)$ repräsentiert nicht den kürzesten Weg von u nach w
11: **end if**
12: Erzeuge neuen Shortcut-Pfeil $(u \dots w)$ mit
$e_1((u \dots w)) = (u, v)$,
$e_2((u \dots w)) = (v, w)$ sowie
$c_{\text{ch}}(u \dots w) = c_{\text{ch}}(u, v) + c_{\text{ch}}(v, w)$ und füge ihn zu E hinzu
13: **end if**
14: **end for**
15: **end for**
16: Entferne alle zu v adjazenten (Shortcut-)Pfeile temporär aus E
17: Entferne v temporär aus V ▸ Ende der Kontraktion von v
18: **end while**
19: Füge alle temporär entfernten Knoten zu V und alle temporär entfernten (Shortcut-)Pfeile wieder zu E hinzu.

Ergebnis: $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$ beschreibt eine vollständige Contraction Hierarchy

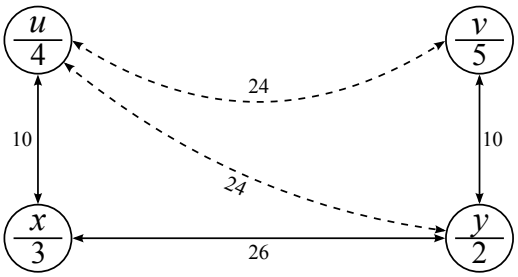
Da von einem zu kontrahierenden Knoten sämtliche Paare aus dessen Vorgänger- und Nachfolgerknoten untersucht werden müssen (siehe Zeilen 6 und 7 in Verfahren 10), besitzt eine einzelne Knotenkontraktion in einem Graphen mit Knotenmenge V eine Aufwandsabschätzung von $\mathcal{O}(|V|^2)$. Mit Ausnahme der letzten beiden Knoten muss jeder Knoten des Graphen kontrahiert werden. Für den ersten zu kontrahierenden Knoten müssen daher maximal $(|V|-1)^2$ Verbindungen geprüft werden. Für den zweiten zu kontrahieren Knoten sind es dann höchstens $(|V|-2)^2$, für den dritten $(|V|-3)^2$, usw. Das bedeutet der gesamte Aufwand des Verfahrens liegt[4] bei $\mathcal{O}(\sum_{i=1}^{|V|-3} i^2) = \mathcal{O}(\frac{(|V|-3)(|V|-2)(2|V|-5)}{6})$.

[4] gemäß der Summenformel für Quadratzahlen: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

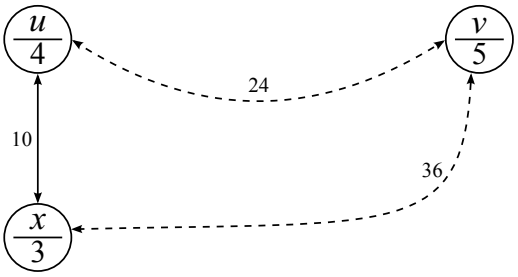
Abbildung 5.1 zeigt die Einzelschritte für einen Beispiel-Graphen. Der Übersichtlichkeit halber sind dort entgegen gerichtete Pfeile zwischen denselben Knoten durch eine einzige Linie mit zwei Pfeilköpfen dargestellt. Im Beispiel beinhaltet die vollständige Hierarchie (Abbildung 5.1e) sechs zusätzliche Shortcut-Pfeile.



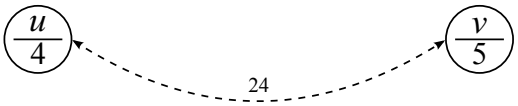
(a) Originalgraph
Knoten w wird nun kontrahiert.



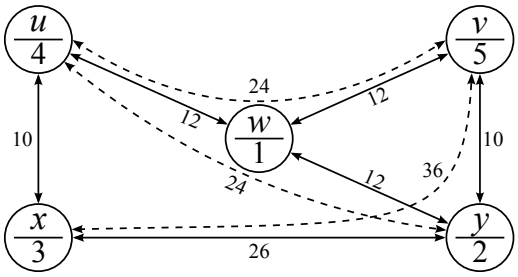
(b) Overlay-Graph nach der 1. Iteration
Knoten y wird nun kontrahiert.



(c) Overlay-Graph nach der 2. Iteration
Knoten x wird nun kontrahiert.



(d) Overlay-Graph nach der 3. Iteration
Abbruch, da nur noch zwei Knoten vorhanden sind



(e) Vollständige Hierarchie inklusive aller Shortcut-Pfeile

Abbildung 5.1: Beispielhafte Erzeugung einer Contraction Hierarchy
Eine Linie mit zwei Pfeilköpfen steht für zwei Pfeile in entgegengesetzte Richtungen. Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Zahlen an Pfeilen geben deren Länge an. Gestrichelte Linien symbolisieren Shortcut-Pfeile.

Beobachtungen für Shortcut-Pfeile

Es lassen sich für Shortcut-Pfeile innerhalb von Contraction Hierarchies folgende Beobachtungen festhalten:

- Sie repräsentieren den einzigen kürzesten Weg zwischen ihren Endknoten.
- Sie überbrücken stets genau einen Knoten sowie zwei Pfeile aus einer Ebene, wobei diese wiederum Shortcut-Pfeile sein können.
- Ihre Kosten ergeben sich direkt aus den beiden überbrückten Pfeilen.
- Ihr überbrückter Knoten besitzt einen geringeren Rang als ihre Start- und Endknoten.

5.2 Wegsuche in Contraction Hierarchies

Für die Suche nach einem optimalen Pfad zwischen zwei Knoten einer Contraction Hierarchy verwenden Geisberger u. a. [2008] eine alternierende, bidirektionale Variante der Dijkstra-Suche, die sehr ähnlich ist zur Wegsuche beim Highway-Node-Routing-Verfahren[5], das seinerseits auf dem Highway-Hierarchy-Verfahren[6] basiert. Wie beim HNR-Verfahren werden je Suchrichtung nur solche Pfeile berücksichtigt, die zu Knoten führen, die einen *höheren* Rang besitzen als der Ausgangsknoten. Nach jeder Iteration wird geprüft, ob der zuletzt in die Lösungsmenge einer Suchrichtung aufgenommene Knoten ein Brückenknoten[7] ist, und der kürzeste, aktuell bekannte Weg ggf. aktualisiert.

Die Suche wird für eine Suchrichtung beendet, sobald die Distanz zum aktuell in die Lösungsmenge aufzunehmenden Knoten nicht mehr kleiner ist als die Länge des kürzesten bekannten Weges vom Start- zum Zielknoten. Sind beide Suchrichtungen auf diese Weise beendet, ist der aktuell kürzeste bekannte Weg die Lösung der Wegsuche. Verfahren 11, S. 138, stellt die Wegsuche als Pseudocode dar.

[5] zur Wegsuche beim HNR-Verfahren siehe Seite 115, Stichwort „Wegsuche“
[6] zur Wegsuche beim HH-Verfahren siehe Seite 112, Stichwort „Wegsuche“
[7] zur Definition siehe Seite 98, Stichwort „Brückenknoten“

Verfahren 11 Wegsuche bei Contraction Hierarchies

Gegeben:
Contraction Hierarchy $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$,
Startknoten s, Zielknoten t mit $s, t \in V$

1: $s.distanz_{\text{vor}} \coloneqq 0$, $t.distanz_{\text{rück}} \coloneqq 0$ ▸ Initialisierung
2: $v.distanz_{\text{vor}} \coloneqq \infty$, $v.vorgänger_{\text{vor}} \coloneqq NULL$ $\forall v \in V \setminus \{s\}$
3: $v.distanz_{\text{rück}} \coloneqq \infty$, $v.vorgänger_{\text{rück}} \coloneqq NULL$ $\forall v \in V \setminus \{t\}$
4: $Lösungsmenge_{\text{vor}} \coloneqq \emptyset$, $Lösungsmenge_{\text{rück}} \coloneqq \emptyset$
5: $Horizontmenge_{\text{vor}} \coloneqq \{s\}$, $Horizontmenge_{\text{rück}} \coloneqq \{t\}$
6: $m \coloneqq NULL$ ▸ Brückenknoten des kürzesten bekannten Weges
7: $best \coloneqq \infty$ ▸ Länge des kürzesten bekannten Weges
8: $fertig_{\text{vor}} \coloneqq$ `falsch` , $fertig_{\text{rück}} \coloneqq$ `falsch` ▸ Abbruchkriterien
9: **while not** ($fertig_{\text{vor}}$ **und** $fertig_{\text{rück}}$) **do**
10: **if not** $fertig_{\text{vor}}$ **then**
11: Wähle v mit $v.distanz = \min\{v'.distanz \mid v' \in Horizontmenge_{\text{vor}}\}$
12: Entferne v aus $Horizontmenge_{\text{vor}}$
13: Nimm v in $Lösungsmenge_{\text{vor}}$ auf
14: **if** $v.distanz_{\text{vor}} \geq best$ **then**
15: $fertig_{\text{vor}} \coloneqq$ `wahr` ▸ Die Vorwärtssuche terminiert
16: **end if**
17: Expandiere v in Vorwärtsrichtung, verwende dabei nur solche Pfeile $(v, v') \in E$ mit $v.rang < v'.rang$ ▸ Dies aktualisiert ggf. die $v'.distanz_{\text{vor}}$
18: **if** $v.distanz_{\text{vor}} + v.distanz_{\text{rück}} < best$ **then** ▸ Wurde ein neuer bester Brückenknoten gefunden?
19: $best \coloneqq v.distanz_{\text{vor}} + v.distanz_{\text{rück}}$; $m \coloneqq v$
20: **end if**
21: **end if**
22: **if not** $fertig_{\text{rück}}$ **then**
23: Wähle v mit $v.distanz = \min\{v'.distanz \mid v' \in Horizontmenge_{\text{rück}}\}$
24: Entferne v aus $Horizontmenge_{\text{rück}}$
25: Nimm v in $Lösungsmenge_{\text{rück}}$ auf
26: **if** $v.distanz_{\text{rück}} \geq best$ **then**
27: $fertig_{\text{rück}} \coloneqq$ `wahr` ▸ Die Rückwärtssuche terminiert
28: **end if**
29: Expandiere v in Rückwärtsrichtung, verwende dabei nur solche Pfeile $(v', v) \in E$ mit $v'.rang > v.rang$ ▸ Dies aktualisiert ggf. die $v'.distanz_{\text{rück}}$
30: **if** $v.distanz_{\text{vor}} + v.distanz_{\text{rück}} < best$ **then** ▸ Wurde ein neuer bester Brückenknoten gefunden?
31: $best \coloneqq v.distanz_{\text{vor}} + v.distanz_{\text{rück}}$; $m \coloneqq v$
32: **end if**
33: **end if**
34: **end while**

Ergebnis: Ein kürzester Weg von s zu t über m ist für die erste Teilstrecke rekursiv konstruierbar aus $m.vorgänger_{\text{vor}}$ und für die zweite Teilstrecke aus $m.vorgänger_{\text{rück}}$. Für den Weg gilt: $m.distanz_{\text{vor}} + m.distanz_{\text{rück}} = \text{dist}(s, t) = best$. Wurde kein Brückenknoten gefunden, existiert kein Weg zwischen s und t.

Über die Suche nach Brückenknoten Geisberger u. a. [2008, S. 324] heben in ihrer Beschreibung der Wegsuche hervor, dass ein Knoten, der sowohl in der Lösungsmenge der Vorwärts- als auch der Rückwärtssuche aufgenommen wurde, ein Brückenknoten ist.[8] Dies ist zwar korrekt, aber unnötig einschränkend. Denn auch Knoten, die in den *Horizontmengen* beider Suchrichtungen vorkommen, sind Brückenknoten[9]. Zwar ist in diesem Fall noch nicht gesichert, dass pro Suchrichtung ein *kürzester* Weg zum betreffenden Brückenknoten gefunden wurde. Doch die Information kann dennoch zur Bildung einer besseren oberen Schranke für die Länge des gesuchten Weges verwendet werden und damit im Verfahren Iterationen einsparen. Dieser Aspekt ist in Verfahren 11, S. 138, in den Prüfungen in Zeilen 18 und 30 implizit berücksichtigt.

Zur Verdeutlichung der Wegsuche sei im Folgenden eine beispielhafte Wegsuche in der in Abbildung 5.1e, S. 136, gezeigten Contraction Hierarchy gegeben. Gesucht sei der kürzeste Weg von Knoten u nach Knoten y. Tabelle 5.1 zeigt getrennt für Vorwärts- und Rückwärtssuchrichtung die Horizont- und Lösungsmengen. Die Zahlen in eckigen Klammern geben die Länge des kürzesten bekannten Weges an, über die ein Knoten erreicht wurde.

Iteration	$Horizontm._{\text{vor}}$	$Lösungsm._{\text{vor}}$	$Horizontm._{\text{rück}}$	$Lösungsm._{\text{rück}}$
0	$\{u[0]\}$	$\emptyset$	$\{y[0]\}$	$\emptyset$
1	$\{v[24]\}$	$\{u\}$	$\{y[0]\}$	$\emptyset$
2	$\{v[24]\}$	$\{u\}$	$\{v[10], u[24], x[26]\}$	$\{y\}$
3	$\emptyset^{I}$	$\{u, v\}$	$\{v[10], u[24], x[26]\}$	$\{y\}$
4	$\emptyset$	$\{u, v\}$	$\{u[24], x[26]\}$	$\{y, v\}^{II}$
5	$\emptyset$	$\{u, v\}$	$\{x[26]\}$	$\{y, v, u\}^{III}$

[I] Die Vorwärtssuche terminiert, da ihre Horizontmenge leer ist.
[II] Knoten v ist ein Brückenknoten; der implizierte Weg ist $p_1 = \langle u, v, y\rangle$ mit $c(p_1) = 34$.
[III] Knoten u ist ein Brückenknoten; der implizierte Weg ist $p_2 = \langle u, y\rangle$ mit $c(p_2) = 24$ und damit kürzer als p_1. Knoten x wird in der Rückwärtssuche nicht mehr expandiert, da der zu ihm führende Weg länger als 24 ist und somit zu keiner Verbesserung des bislang kürzesten Weges beitragen kann. Die Suche ist damit beendet.

Tabelle 5.1: Horizont- und Lösungsmengen bei der Wegsuche gemäß Verfahren 11

Es ist nicht ohne weiteres ersichtlich, dass dieses Verfahren stets einen kürzesten Weg findet. Daher wird im Folgenden der Beweis in Anlehnung an Geisberger [2008, S. 30 f.] angeführt.

[8] Im englischen Original: „Whenever we settle a node in one direction that is already settled in the other direction, we get a new candidate for a shortest path“ [Geisberger u. a., 2008, S. 324].
[9] zur Definition von Brückenknoten im Rahmen des Dijkstra-Verfahrens siehe Seite 98, Stichwort „Brückenknoten“

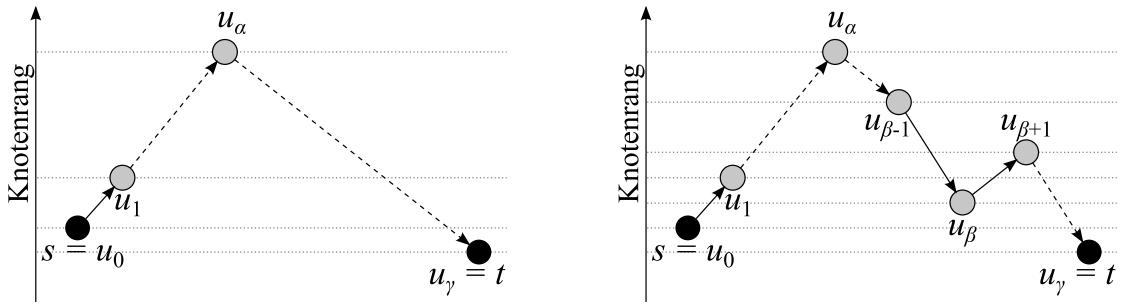
Beweis für die Korrektheit der Wegsuche bei Contraction Hierarchies. Gemäß der oben beschriebenen Wegsuche werden von einem Startknoten s zu einem Zielknoten t nur Wege p der Form

$$\begin{aligned} p = \langle s = u_0, u_1, \ldots, u_\alpha, \ldots, u_\gamma = t \rangle & \quad \text{mit } \alpha, \gamma \in \mathbb{N} \\ \alpha \leq \gamma & \quad \text{und} \\ \text{rang}(u_i) < \text{rang}(u_{i+1}),\ 0 \leq i < \alpha & \quad \text{und} \\ \text{rang}(u_j) > \text{rang}(u_{j+1}), \alpha \leq j < \gamma & \quad \text{und} \\ i, j \in \mathbb{N} & \end{aligned} \tag{5.5}$$

gefunden. Hierbei ist u_α der Brückenknoten, der sowohl von der Vorwärts- als auch von der Rückwärtssuche in die Lösungsmenge aufgenommen wurde. Umgekehrt wird kein Weg p' gefunden, der die nachfolgende Form besitzt:

$$\begin{aligned} p' = \langle s = u_0, u_1, \ldots, u_\alpha, \ldots, u_{\beta-1}, u_\beta, u_{\beta+1}, \ldots, u_\gamma = t \rangle & \quad \text{mit } \alpha, \beta, \gamma \in \mathbb{N} \\ \alpha \leq \beta \leq \gamma & \quad \text{und} \\ \text{rang}(u_h) < \text{rang}(u_{h+1}),\ 0 \leq h < \alpha & \quad \text{und} \\ \text{rang}(u_i) > \text{rang}(u_{i+1}),\ \alpha \leq i < \beta & \quad \text{und} \\ \text{rang}(u_\beta) < \text{rang}(u_{\beta+1}) & \quad \text{und} \\ \text{rang}(u_j) > \text{rang}(u_{j+1}), \beta + 1 \leq j < \gamma & \quad \text{und} \\ h, i, j \in \mathbb{N}. & \end{aligned} \tag{5.6}$$

In Abbildung 5.2 ist schematisch für beide Formen je ein Weg dargestellt. Der nachfolgende Beweis zeigt, dass für Wege der Form aus Gleichung (5.6) stets ein Alternativweg mit gleichen Kosten gefunden werden kann oder aber eine Repräsentation mithilfe von Shortcut-Pfeilen existiert, welche die Form aus Gleichung (5.5) besitzt.



(a) Form gefundener Wege

(b) Form nicht gefundener Wege

Abbildung 5.2: Beispiele für Formen von kürzesten Wegen in einer Contraction Hierarchy
Durchgezogene Pfeile symbolisieren reale Pfeile im dargestellten Weg, d. h. ggf. auch Shortcut-Pfeile. Gestrichelte Pfeile symbolisieren einen oder *mehrere* im dargestellten Weg aufeinander folgende Pfeile zwischen den entsprechenden Knoten.
Quelle: in Anlehnung an Geisberger [2008, S. 30]

Für einen kürzesten Weg in einem Graphen gilt, dass alle Teil-Wege ebenfalls kürzeste Wege sind. Angewendet auf Wege mit der Form aus Gleichung (5.6), S. 140, gilt daher insbesondere, dass auch $p'' = \langle u_{\beta-1}, u_\beta, u_{\beta+1} \rangle$ ein kürzester Weg sein muss. Nun kann eine Fallunterscheidung vorgenommen werden:

Fall 1 Es existiert ein Pfeil $(u_{\beta-1}, u_{\beta+1})$ mit $c(u_{\beta-1}, u_{\beta+1}) = c(u_{\beta-1}, u_\beta) + c(u_\beta, u_{\beta+1})$.

Fall 2 Es existiert kein Pfeil nach Fall 1 und es existiert ein weiterer Knoten u_δ mit $c(u_{\beta-1}, u_\beta) + c(u_\beta, u_{\beta+1}) = c(u_{\beta-1}, u_\delta) + c(u_\delta, u_{\beta+1})$.

Fall 2a Es gilt $\text{rang}(u_{\beta-1}) > \text{rang}(u_\delta) > \text{rang}(u_{\beta+1})$.

Fall 2b Es gilt $\text{rang}(u_{\beta-1}) < \text{rang}(u_\delta)$.

Fall 2c Sonst.

Fall 3 Sonst, d. h., es existiert weder ein Pfeil $(u_{\beta-1}, u_{\beta+1})$ nach Fall 1, noch ein Knoten u_δ nach Fall 2.

Für Fall 1 ist die Transformation des Weges p' in die Form aus Gleichung (5.5), S. 140, offensichtlich: Durch Austausch der Pfeile $(u_{\beta-1}, u_\beta)$ und $(u_\beta, u_{\beta+1})$ durch den Pfeil $(u_{\beta-1}, u_{\beta+1})$ entsteht ein Weg mit selbem Start- und Zielknoten sowie gleich hohen Kosten. Fall 1

Für Fall 2 existiert ein alternativer kürzester Weg vom Startknoten zum Zielknoten, der anstatt über Knoten u_β über Knoten u_δ führt. Für Fall 2a besitzt dieser alternative Weg die Form aus Gleichung (5.5), S. 140, die von der Wegsuche gefunden wird. Fall 2a

Fall 2b ist gleichbedeutend mit der Existenz eines alternativen kürzesten Weges, der *nicht* von der beschriebenen Wegsuche gefunden werden kann. Dieser Fall „verschiebt“ die Argumentation dieses Beweises aber lediglich um einen Knoten innerhalb des kürzesten Weges in Richtung Startknoten. Abbildung 5.3 verdeutlicht, dass bei diesem Alternativweg das Knotentripel $\{u_{\beta-2}, u_{\beta-1}, u_\delta\}$ anstelle des Knotentripels $\{u_{\beta-1}, u_\beta, u_{\beta+1}\}$ in analoger Weise untersucht werden muss. Fall 2b

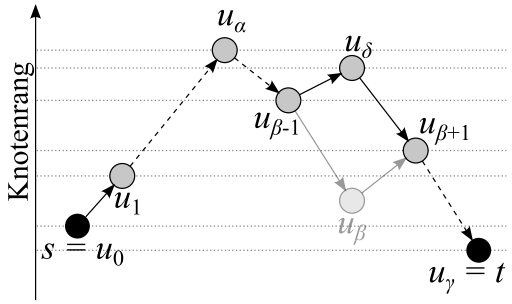


Abbildung 5.3: Alternativer kürzester Weg nach Fall 2b
Offensichtlich besitzt dieser Weg dieselbe Form wie der Weg in Abbildung 5.2b, S. 140.

Abschließend wird dargelegt, dass Fall 2c und Fall 3 nicht auftreten können. Betrachtet wird der Einfachheit halber zuerst Fall 3:

Fall 3

Knoten u_β ist ein direkter Nachbarknoten von $u_{\beta-1}$ und $u_{\beta+1}$ und er besitzt von diesen drei Knoten den geringsten Rang. Beim Erzeugen der CH wurde u_β also vor $u_{\beta-1}$ und $u_{\beta+1}$ kontrahiert. Hierbei wurde nach Fall 3 also der einzige kürzeste Weg zwischen Knoten $u_{\beta-1}$ nach Knoten $u_{\beta+1}$ zerstört, so dass ein Shortcut-Pfeil $(u_{\beta-1} \dots u_{\beta+1})$ erzeugt wurde, der eben diesen Weg repräsentiert. Die Existenz dieses Shortcut-Pfeils steht jedoch im direkten Widerspruch zu Fall 3, da er eine Verbindung zwischen Knoten $u_{\beta-1}$ und Knoten $u_{\beta+1}$ (gemäß Fall 1) darstellt. Fall 3 kann daher nicht auftreten.

Fall 2c

Wenn Fall 2c in Betracht kommt, ist Fall 2b nicht eingetreten und daher gilt stattdessen: $\text{rang}(u_{\beta-1}) > \text{rang}(u_\delta)$.[10] Da auch Fall 2a nicht eingetreten ist, lässt sich zusätzlich $\text{rang}(u_\delta) < \text{rang}(u_{\beta+1})$ schlussfolgern, d. h., von den drei Knoten $u_{\beta-1}$, u_δ und $u_{\beta+1}$ besitzt u_δ den kleinsten Rang. Für alle Knoten u_δ, die nach Fall 2c existieren, gilt deshalb, dass sie vor $u_{\beta-1}$ und $u_{\beta+1}$ kontrahiert wurden. Bei der Kontraktion des *letzten* dieser Knoten musste wie schon bei Fall 3 ein Shortcut-Pfeil zwischen Knoten $u_{\beta-1}$ nach Knoten $u_{\beta+1}$ eingefügt werden. Analog zur Argumentation bei Fall 3 steht die Existenz dieses Shortcut-Pfeils im direkten Widerspruch zu Fall 2c, der damit ebenfalls nicht auftreten kann. □

Auflösung des Pfads

Verfahren 11, S. 138, liefert kürzeste Wege, die neben originalen Pfeilen auch Shortcut-Pfeile beinhalten können. Die Überführung solcher Wege in Wege ohne Shortcut-Pfeile lässt sich anhand der Beobachtungen auf Seite 137 einfach rekursiv durchführen, da jeder Shortcut-Pfeil $(u \dots w)$ genau zwei andere Pfeile überbrückt, die allerdings ihrerseits Shortcut-Pfeile sein können. Verfahren 12 skizziert die Vorgehensweise.

Verfahren 12 Rekursive Auflösung eines Shortcut-Pfeils in Original-Pfeile

Gegeben: $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$, Shortcut-Pfeil $(u, w) \in E$

1: *Liste* $:= \emptyset$ ▸ Sortierte Liste der von $(u \dots w)$ überbrückten Pfeile
2: **if** $e_1((u, w)) = (u, w)$ **then** ▸ d. h., (u, w) ist ein Original-Pfeil
3: Hänge $(u \rightarrow w)$ an *Liste* an
4: **else**
5: Rufe das Verfahren rekursiv mit Parameter $e_1((u, w))$ auf und hänge das Ergebnis an *Liste* an
6: Rufe das Verfahren rekursiv mit Parameter $e_2((u, w))$ auf und hänge das Ergebnis an *Liste* an
7: **end if**
8: **return** *Liste*

Ergebnis: *Liste* beinhaltet sortiert alle von (u, w) überbrückten Original-Pfeile.

[10] Da die Knotenränge eindeutig sind, kann auch nicht $\text{rang}(u_{\beta-1}) = \text{rang}(u_\delta)$ gelten.

Abbruchkriterium der „flachen“ bidirektionalen Dijkstra-Suche für CH ungeeignet

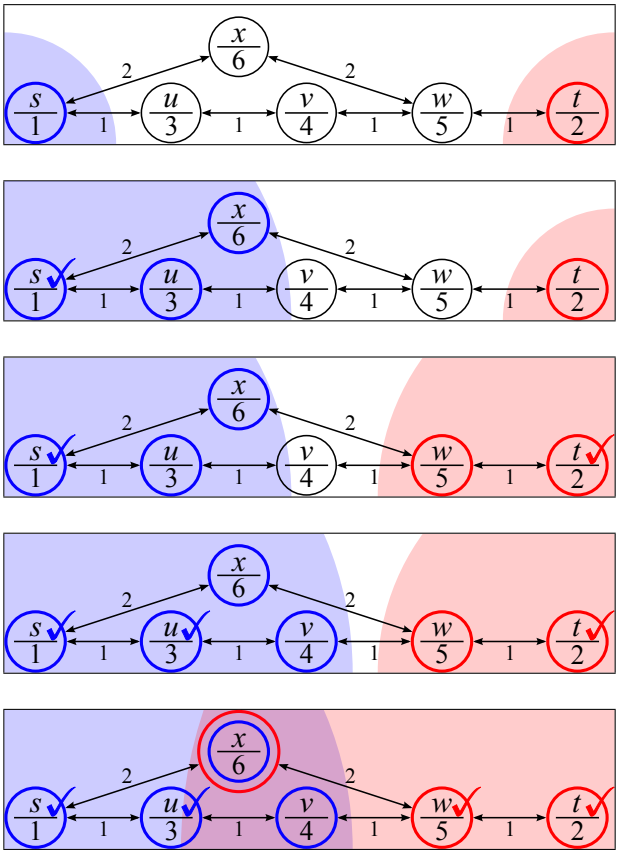
Über das Abbruchkriterium der Wegsuche An dieser Stelle wird erläutert, warum die Dijkstra-Wegsuche in einer Contraction Hierarchy erst dann terminiert, wenn sowohl der Suchbaum der Vorwärts- als auch der Rückwärtssuche die Weglänge des aktuell kürzesten bekannten Weges überschritten hat. Denn dies geschieht im Gegensatz zu der in Abschnitt 4.1.1.2, S. 95 ff., vorgestellten bidirektionalen Dijkstra-Suche, die mit Gleichung (4.3), S. 98, ein Abbruchkriterium besitzt, das erwartungsgemäß früher eintritt.

Der Grund liegt in der Art, wie bei der Suche in Contraction Hierarchies der Suchraum eingeschränkt wird. Bei der bidirektionalen Dijkstra-Suche *ohne* Hierarchie bewegen sich die Ränder der Suchräume aus Vorwärts- und Rückwärtssuche gewissermaßen aufeinander zu. Knoten, an denen sie sich „berühren“, sind die Brückenknoten. Bei dem Verfahren können ausschließlich die Knoten der *Horizontmenge* zu Brückenknoten werden.

Anders verhält es sich dagegen, wenn der Suchraum dadurch eingeschränkt wird, dass Knoten ausschließlich in Richtung höherrangiger Knoten expandiert werden. Dadurch entsteht eine Situation, bei der die beiden Suchrichtungen gewissermaßen in unterschiedlichen Lösungsräumen arbeiten: Nicht alle von der Vorwärtssuche erreichbaren Knoten sind in der Rückwärtssuche ebenfalls erreichbar und umgekehrt.

Dies kann dazu führen, dass die Vorwärts- und Rückwärtssuche in manchen Bereichen des Graphen geometrisch zwar überlappen und vorher dennoch kein Knoten in beiden Horizontmengen gleichzeitig aufgetaucht ist. Brückenknoten können daher „übersehen“ werden, wenn sie in einer der Suchrichtungen bereits Teil der Lösungsmenge sind, sobald sie in der anderen Richtung erst in die Horizontmenge aufgenommen werden.

Das einfache Beispiel in Abbildung 5.4, S. 144, zeigt ein entsprechendes Szenario, bei dem die Anwendung der „klassischen“ bidirektionalen Dijkstra-Suche in einer Contraction Hierarchy fehlschlägt. Die dargestellte Hierarchie kommt aufgrund der Knotenränge ohne Shortcut-Pfeile aus. Bei der Wegsuche von Knoten s nach Knoten t bricht das Verfahren nach der vierten Iteration ab, da mit Knoten x ein Brückenknoten gefunden wurde, der Gleichung (4.3), S. 98, erfüllt. Die als Kreisausschnitt schematisch dargestellten Horizontmengen überlappen in der vierten Iteration zwar, aber Knoten v befindet sich zu diesem Zeitpunkt nur in der Horizontmenge der Vorwärtssuche.



Start- und Zielknoten (blau bzw. rot)

Nach der 1. Iteration

Nach der 2. Iteration

Nach der 3. Iteration

Nach der 4. Iteration
Das Abbruchkriterium gemäß Gleichung (4.3), S. 98, ist erreicht.
x ist Brückenknoten.

Abbildung 5.4: Fehlschlagende, klassische bidirektionale Dijkstra-Suche in einer Contraction Hierarchy von Knoten s nach Knoten t
Der kürzeste Weg $p = \langle s, u, v, w, t \rangle$ wird nicht gefunden; stattdessen wird nur $p' = \langle s, x, w, t \rangle$ erkannt.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Häkchen kennzeichnen Knoten der jeweiligen Lösungsmenge. Eine Linie mit zwei Pfeilköpfen steht für zwei Pfeile in entgegengesetzte Richtungen. Die Horizontmengen der Suchrichtungen sind schematisch als Kreisausschnitt dargestellt.
Blau markiert: Vorwärtssuche
Rot markiert: Rückwärtssuche

Aufwandsabschätzung der Wegsuche Weil das Verfahren auf einer bidirektionalen Dijkstra-Wegsuche basiert, gilt im Graphen $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$ grundsätzlich dieselbe Aufwandsabschätzung von $\mathcal{O}(|V|^2)$.[11] Ein bedeutender Unterschied ergibt sich allerdings bei der Betrachtung des *durchschnittlichen* Aufwands: Aufgrund der Eingrenzung des Suchraums bei der Knotenexpansion eines Knotens v wird durchschnittlich nur die *Hälfte der Nachbarknoten* von v berücksichtigt. Denn im Mittel besitzt nur die Hälfte aller Knoten einen höheren Rang als v. Dieser Effekt tritt bei jeder Dijkstra-Iteration

[11] siehe auch Abschnitt 4.1.1.2 und insbesondere Seite 100, Stichwort „Aufwandsabschätzung“

der Vorwärts- und Rückwärtssuche auf und wird durch Shortcut-Pfeile – je nach deren Anzahl und Verteilung – nicht wesentlich konterkariert.

Dies führt dazu, dass bei Wegsuchen in einer Contraction Hierarchy jeder Knoten in den Suchbäumen der Vorwärts- und Rückwärtssuche durchschnittlich nur halb so viele Kinderknoten[12] besitzt wie bei der klassischen bidirektionalen Dijkstra-Wegsuche. Bei einer abgeschlossenen Wegsuche von Knoten $s \in V$ nach $t \in V$ für die bidirektionale Wegsuche *ohne* Hierarchie (*flach*) und die Wegsuche in einer Contraction Hierarchy (*ch*) lässt sich damit vergleichend erwarten[13]:

$$\left|Horizontmenge^{\text{flach}}_{\text{vor}}\right| \approx 2^{r_s(t)} \left|Horizontmenge^{\text{ch}}_{\text{vor}}\right|. \tag{5.7}$$

Eine spiegelbildliche Aussage gilt natürlich auch für die Horizontmengen der Rückwärtssuchen. Im direkten Vergleich mit der klassischen Dijkstra-Wegsuche wachsen beide Suchbäume daher durchschnittlich nur logarithmisch an. Da sich diese Aussage auch auf die Anzahl der notwendigen Dijkstra-Iterationen übertragen lässt, erreicht die Wegsuche in Contraction Hierarchies – in hinreichend großen Graphen – im Vergleich mit der klassischen bidirektionalen Wegsuche demnach im Mittel *beliebig* große Beschleunigungen. Dies hebt sie von den in Abschnitt 4.1.1 vorgestellten Verfahren ab.

Gleichwohl kommt der Knotensortierung im Rahmen der Vorberechnung der Hierarchie eine tragende Rolle zu. Denn sowohl eine zu hohe Anzahl künstlicher Shortcut-Pfeile als auch eine ungünstige lokale Verteilung der Knotenränge können die Suchbeschleunigung zunichte machen. Die Details und beispielhafte Szenarios werden in Abschnitt 5.3, S. 158 ff., vorgestellt.

Negativer Overhead möglich Auf einen weiteren Aspekt, der sich aus der Art der Wegsuche ergibt, weisen Geisberger u. a. [2008, S. 325] explizit hin: Contraction Hierarchies ermöglichen in manchen Fällen sogar einen *geringeren* Speicherplatzbedarf für den Graphen als das bidirektionale Dijkstra-Verfahren. Sie argumentieren, dass für eine effiziente Implementierung des letzteren Pfeile stets doppelt vorgehalten werden müssen; je einmal pro adjazentem Knoten. Als Konsequenz geben Geisberger u. a. [2008, S. 328] einen negativen Overhead für das Verfahren an.

Im Gegensatz dazu reicht es bei Contraction Hierarchies aus, jeden Pfeil allein bei dem jeweils niederrangigen Knoten zu speichern. Denn bei der Wegsuche wird ohnehin nur in Richtung des höherrangigen Knotens expandiert. Trotz der zusätzlichen Shortcut-Pfeile kann daher in manchen Fällen Speicherplatz eingespart werden.

Einige der in dieser Arbeit vorgestellten Techniken erfordern für einen effizienten Einsatz jedoch, dass an jedem Knoten *alle* adjazenten Pfeile vermerkt sind. Ein Beispiel hierfür ist das im nachfolgenden Abschnitt vorgestellte Verfahren der *Weitergabe der Knotenblockierung*. Dennoch bleibt ein ausgesprochen gutes Verhältnis von Suchgeschwindigkeit zu Overhead für Contraction Hierarchies festzuhalten.

[12] zur Definition des Suchbaums und der Kinderknoten im Suchbaum siehe auch Seite 90, Stichwort „Bildung des Suchbaums“

[13] Dabei beschreibt $r_s(t)$ den Dijkstra-Rang von Knoten t zu Knoten s; zu dessen Definition siehe Seite 94.

5.2.1 Blockieren von Knoten

Für eine zusätzliche Beschleunigung der Wegsuche beschreiben Geisberger u. a. [2008] die bereits in Abschnitt 4.2.2, S. 115, erwähnte Technik des *Blockierens von Knoten*. Die Überlegung wird im Folgenden für die Vorwärtssuche formuliert, gilt aber o. B. d. A. spiegelbildlich auch für die Rückwärtssuche.

Aufgrund der Art der Knotenexpansion – nur in nach Rang „aufsteigende Richtung“ – kann es bei einer Wegsuche von einem Startknoten s vorkommen, dass, im Gegensatz zum Standard-Dijkstra-Verfahren, ein Knoten v in die Vorwärts-Lösungsmenge aufgenommen wird, ohne dass der hierdurch induzierte Teilweg von s nach v ein kürzester Weg ist. Alle im Anschluss daran expandierten Knoten der Vorwärtssuche, deren induzierter Teilweg ebenfalls über v führt, können gleichermaßen nicht zur Lösung beitragen. Falls einer dieser Knoten jedoch zum tatsächlichen Lösungsweg gehören sollte, wird er stattdessen entweder durch die Rückwärtssuche „entdeckt“ oder implizit über einen Shortcut-Pfeil in eine der Lösungsmengen mit aufgenommen.

Solche Knoten führen demnach zu unnötigen Iterationen des Verfahrens. Geisberger u. a. [2008, S. 324 f.] entdecken einige von ihnen, indem sie bei der Vorwärtssuche jeden Knoten v vor dessen Expansion zunächst überprüfen. Falls ein höherrangiger Nachbarknoten $u \in \mathcal{V}(v)$ im Rahmen der Suche bereits erreicht wurde und der Weg vom Startknoten über u nach v kürzer ist als der bisher bekannte Weg zu v, wird v als *blockiert* gekennzeichnet. Formal wird v also genau dann blockiert, wenn gilt:

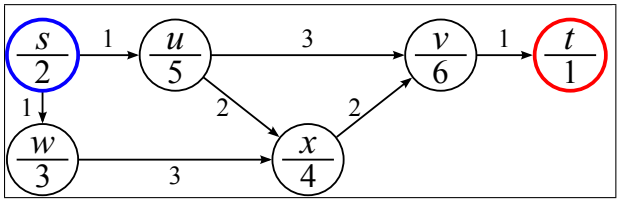
$$\begin{aligned} v.\mathit{distanz} = \min\{v'.\mathit{distanz} \mid v' \in \mathit{Horizontmenge}_{\text{vor}}\} \\ \wedge\, u \in \mathcal{V}(v) \\ \wedge\, v.\mathit{distanz}_{\text{vor}} > u.\mathit{distanz}_{\text{vor}} + c(u,v) \\ \wedge\, \text{rang}(v) < \text{rang}(u) \end{aligned} \tag{5.8}$$

Denn dann gilt für die Wegsuche vom Startknoten s:

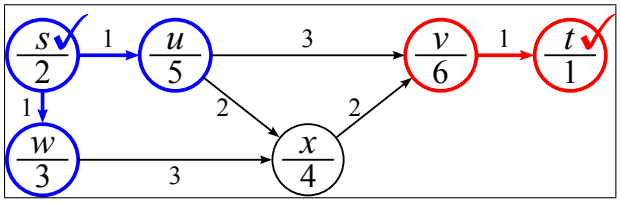
$$\begin{aligned} c(p_{s,v}) > c(p'_{s,v}) \text{ mit} \\ p_{s,v} = \langle s, \ldots, v\rangle \text{ und} \\ p'_{s,v} = \langle s, \ldots, u, v\rangle. \end{aligned} \tag{5.9}$$

Blockierte Knoten werden zwar in die Vorwärts-Lösungsmenge aufgenommen, aber *nicht expandiert*, wodurch unnötige Suchiterationen vermieden werden.

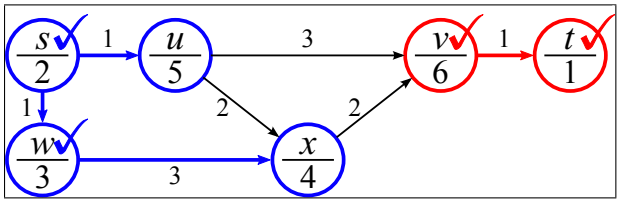
Abbildung 5.5, S. 147, zeigt ein einfaches Beispiel anhand einer Contraction Hierarchie, die aufgrund ihrer Knotenränge ohne Shortcut-Pfeile auskommt. Hier wird bei der Wegsuche von Knoten s nach t Knoten x im Rahmen der Vorwärtssuche blockiert. Um die Ersparnis im Aufwand der Wegsuche zu verdeutlichen, sind in der Darstellung nicht nur die Knoten entsprechend der Suchrichtung markiert, sondern zusätzlich auch die Pfeile, entlang derer expandiert wurde. Der Augenmerk liegt daher darauf, dass Pfeil (x, v) bei der Vorwärtssuche *nicht* expandiert wird.



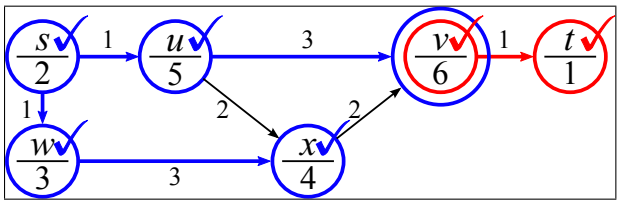
Start- und Zielknoten (blau bzw. rot)



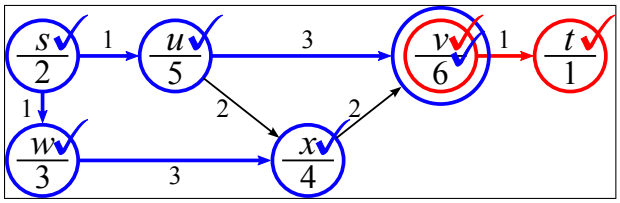
Nach der 2. Iteration



Nach der 4. Iteration



Nach der 6. Iteration: Knoten x wurde nach Prüfung des Pfeils (u, x) blockiert, da über diesen ein kürzerer Weg zu x existiert als der bisher bekannte bei der Vorwärtssuche. Deshalb wird x nicht entlang (x, v) expandiert.



Nach der 7. Iteration: Die Vorwärts- und Rückwärtssuche sind als „fertig“ markiert und die Suche ist beendet. Knoten v ist Brückenknoten.

Abbildung 5.5: Beispiel für Blockieren von Knoten in einer Contraction Hierarchy
Knoten x wird bei der Wegsuche von Knoten s nach t blockiert. Dadurch muss entlang Pfeil (x, v) nicht mehr expandiert werden.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Häkchen kennzeichnen Knoten der jeweiligen Lösungsmenge.
Blau markiert: Vorwärtssuche
Rot markiert: Rückwärtssuche

Weitergabe der Knotenblockierung Wird v aufgrund von u erfolgreich blockiert, kann diese Information „weitergereicht“ werden, um noch weitere Knoten auf Grundlage von u zu blockieren (vgl. [Geisberger u. a., 2008, S. 324 f.]). Die Weitergabe im Rahmen der Vorwärtssuche kann auf zwei Arten geschehen:

1. Für den aktuell blockierten Knoten v wird für alle ausgehenden Pfeile (v, w) untersucht, ob Knoten w seinerseits via v blockiert werden kann. Hierbei ist es unerheblich, ob v einen höheren Rang besitzt als w oder nicht.

2. Für den zur Blockierung von v verwendeten Knoten u wird für alle ausgehenden Pfeile (u, w') mit $\text{rang}(u) > \text{rang}(w')$ geprüft, ob w' aufgrund von u blockiert werden kann.

Verfahren 13, S. 149, beschreibt im Pseudocode das Vorgehen, das im Folgenden auch *erweitertes Blockieren* genannt wird. Das Verfahren für das *einfache Blockieren* – also ohne Weitergabe – ergibt sich, wenn Zeilen 6–19 übersprungen werden. Die Weitergabe bei der Rückwärtssuche ist spiegelbildlich zu verstehen, was insbesondere für die Richtungen der zu untersuchenden Pfeile gilt.

Beim zweiten Teil des erweiterten Blockierens (ab Zeile 13) muss das zugrunde liegende Datenmodell den Zugriff von Knoten auf Pfeile gewährleisten, die zu einem Knoten mit geringerem Rang führen. Dies hat Auswirkungen auf die Speichereffizienz.[14]

Aufwandsabschätzung

Für das Verfahren gilt eine Aufwandsabschätzung von $\mathcal{O}(|V|)$. Zwar werden zwei verschachtelte Schleifen, nämlich über die Vorgänger und Nachfolger zu Knoten v, abgearbeitet, die auf $\mathcal{O}(|V|^2)$ hindeuten, doch der vorzeitige Abbruch in Zeile 20 reduziert den Aufwand, so dass im ungünstigsten Fall zunächst alle bis auf den letzten Pfeil $(u, v) \in E$ untersucht werden, ohne dass eine Blockierung stattfindet, und erst der letzte untersuchte Pfeil zu einer Blockierung und damit erst zur Prüfung der Weitergabe der Blockierung (Zeilen 6–19) führt. Nach erstmalig erfolgreichem Blockieren bricht das Verfahren ab.

In allen Fällen ist es jedoch wichtig, die Blockierung eines Knotens v aus der Horizontmenge stets wieder aufzuheben, sobald bei einer Expansion ein kürzerer Weg zu v gefunden wird[15]. Denn die Verbesserung kann bewirken, dass der Grund, der ursprünglich zur Blockierung von v gemäß Gleichung (5.8), S. 146, führte, nicht mehr gilt (vgl. [Geisberger u. a., 2008, S. 325]).

In Abschnitt 5.6.5 wird empirisch untersucht, wie stark das Blockieren von Knoten die Wegsuche beschleunigt. Dort zeigt sich, dass ein einfaches Blockieren, d. h. ohne „Weiterreichen“ der Information an Nachfolgerknoten, am effizientesten arbeitet.

[14] siehe Seite 145, Stichwort: „Negativer Overhead möglich“
[15] siehe Zeile 12 bei Verfahren 5, S. 91

Verfahren 13 Erweiterte Knotenblockierung b. Wegsuchen i. einer CH (Vorwärtssuche)

Gegeben: Der nächste in Verfahren 11, S. 138, Zeile 17, zu expandierende Knoten v

```
1:  for all u ∈ V(v) mit u.rang > v.rang do
2:    if u.distanz_vor + c(u,v) < v.distanz_vor then
3:      Blockiere Knoten v aufgrund von Knoten u
4:      v.distanz_vor := u.distanz_vor + c(u,v)
5:      v.vorgänger_vor := u
6:      for all w ∈ N(v) do                ▸ Prüfe Weitergabe der Blockierung von v nach w
7:        if v.distanz_vor + c(v,w) < w.distanz_vor then
8:          Blockiere Knoten w aufgrund von Knoten v
9:          w.distanz_vor := v.distanz_vor + c(v,w)
10:         w.vorgänger_vor := v
11:       end if
12:     end for
13:     for all w' ∈ N(u) mit u.rang > w'.rang do ▸ Prüfe Weitergabe der Blockierung von u nach w'
14:       if u.distanz_vor + c(u,w') < w'.distanz_vor then
15:         Blockiere Knoten w' aufgrund von Knoten u
16:         w'.distanz_vor := u.distanz_vor + c(u,w')
17:         w'.vorgänger_vor := u
18:       end if
19:     end for                            ▸ Ende der Weitergabe der Blockierung
20:     return                             ▸ Mehrmaliges Blockieren desselben Knotens v sinnlos
21:   end if
22: end for
```

Ergebnis: Die Blockierung des als nächstes zu expandierenden Knotens v wurde festgelegt. Im Falle einer Blockierung wurde diese an die Nachfolgerknoten von v und desjenigen Knotens u, der zur Blockierung von v führte, weitergereicht.

5.2.2 Zielgerichtete Wegsuche in Contraction Hierarchies

Dieser Abschnitt beschreibt zwei Strategien zur weiteren Beschleunigung der Wegsuche in Contraction Hierarchies. Sie basieren auf den dem A*-Verfahren zugrunde liegenden Ideen und wurden zuerst von Nowak u. a. [2012] veröffentlicht. Der erste Ansatz beschränkt den Suchraum der beiden Suchrichtungen durch Zuhilfenahme einer Restkosten-Schätzung und der zweite wendet das A*-Verfahren für die Suche nach Brückenknoten an. Die Beschleunigungen lassen sich sehr gut miteinander kombinieren. Auch die im vorigen Abschnitt beschriebene Methode des Knotenblockierens kann gleichzeitig angewendet werden. Empirische Messungen zu den Verfahren dieses Abschnitts befinden sich in Abschnitt 5.6.6, S. 188 ff.

5.2.2.1 A*-Eingrenzung des Suchraums

Besteht – wie beim in Abschnitt 4.1.1.3, S. 101 ff., vorgestellten A*-Verfahren – die Möglichkeit, eine untere Schranke für die Restdistanz von einem Knoten zu einem anderen anzugeben, kann diese Information dafür genutzt werden, den Suchraum erheblich zu beschränken. Im Folgenden wird die Idee aus der Perspektive der Vorwärtssuche beschrieben. Sämtliche Erkenntnisse lassen sich jedoch leicht auf die Rückwärtssuche übertragen.

Wie im Kontext des A*-Verfahrens erläutert[16], lässt sich für jeden Knoten v aus der Horizontmenge der Term $v.\mathit{distanz} + \mathrm{rd}(v)$ als Abschätzung für die Weglänge vom Startknoten s via v zum Zielknoten t interpretieren. Gleichzeitig gilt beim Dijkstra-Verfahren, dass ein kürzester Weg zu v gefunden wurde, sobald v die Horizontmenge verlässt. Zu diesem Zeitpunkt steht somit $v.\mathit{distanz}$ fest und $v.\mathit{distanz} + \mathrm{rd}(v)$ wird zu einer *unteren Schranke* für die Länge eines kürzesten Weges vom Startknoten via v zum Zielknoten.

Bei der Wegsuche in Contraction Hierarchies ist das Auffinden eines Brückenknotens m gleichbedeutend mit dem Finden *eines* Weges $p_m = \langle s, \ldots, m, \ldots, t \rangle$ vom Startknoten s zum Zielknoten t. Die Länge $c(p_m)$ ist eine *obere Schranke* für den gesuchten Weg.

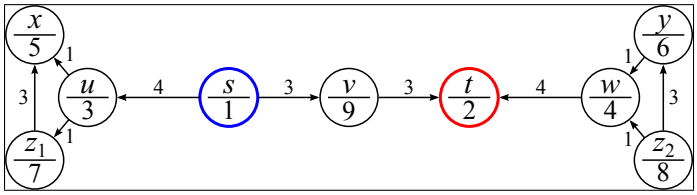
Als Konsequenz können daher diejenigen Knoten v', die in einer Dijkstra-Iteration zuletzt ausgewählt wurden (Zeile 11 in Verfahren 11, S. 138), *nicht* zum gesuchten Weg gehören, für welche die untere Schranke der Weglänge bereits mindestens so groß ist wie die ermittelte obere Schranke für den gesuchten Weg, wenn also gilt:

$$v'.\mathit{distanz} + \mathrm{rd}(v') \geq c(p_m). \tag{5.10}$$

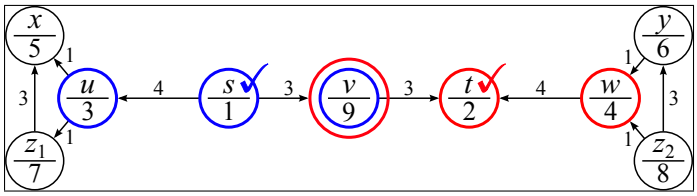
Solche Knoten werden zwar in die Lösungsmenge aufgenommen, aber nicht expandiert. Das Prinzip ähnelt in diesem Aspekt dem im vorigen Abschnitt beschriebenen Blockieren. Betroffen sind bei der Vorwärtssuche im Wesentlichen Knoten, die vom Startknoten ausgehend in die dem Zielknoten entgegengesetzte Richtung führen.

Abbildung 5.6, S. 151, verdeutlicht das Prinzip der Suchraum-Eingrenzung an einem einfachen Beispiel, das aufgrund der vergebenen Knotenränge ohne Shortcut-Pfeile auskommt. Hier werden auf diese Weise pro Suchrichtung jeweils zwei Knoten nicht erreicht, die unmöglich zu einem kürzeren Weg führen können. Das Beispiel verzichtet aus Gründen der Übersichtlichkeit auf die explizite Angabe des Restdistanz-Schätzers, der in diesem Fall euklidisch ist.

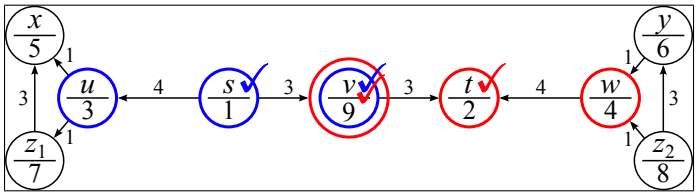
[16] siehe Seite 101



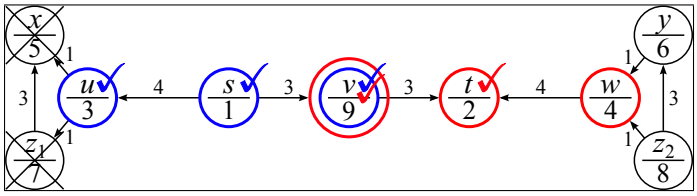
Start- und Zielknoten (blau bzw. rot)



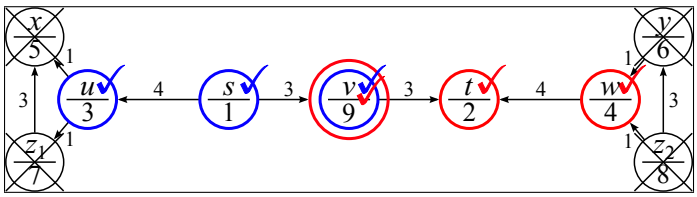
Nach der 2. Iteration



Nach der 4. Iteration: Knoten v ist Brückenknoten.



Nach der 5. Iteration: Knoten u wird nicht mehr expandiert.



Nach der 6. Iteration: Knoten w wird nicht mehr expandiert.

Abbildung 5.6: Beispiel für A*-Eingrenzung des Suchraums
Dargestellt ist die Wegsuche von Knoten s nach t.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Häkchen kennzeichnen Knoten der jeweiligen Lösungsmenge. Der Restdistanz-Schätzer ist euklidisch und der Übersichtlichkeit halber nicht explizit angegeben. Der Graph beinhaltet keine Shortcut-Pfeile.
Blau markiert: Vorwärtssuche
Rot markiert: Rückwärtssuche

5.2.2.2 Bidirektionale A*-Suche

Die Technik der A*-Eingrenzung des Suchraums kann offensichtlich erst dann zu greifen beginnen, nachdem überhaupt ein Brückenknoten identifiziert wurde. Der Brückenknoten des *kürzesten Weges* befindet sich geometrisch im Allgemeinen innerhalb eines Korridors zwischen Start- und Zielknoten. Nur kürzeste Wege, die um Hindernisse wie beispielsweise Seen, Flüsse oder Berge herum führen, bilden hier die Ausnahme.

Um diesen Korridor bevorzugt zu untersuchen, bietet es sich an, die alternierende bidirektionale Dijkstra-Suche in Contraction Hierarchies um die Zielrichtungs-Komponente des A*-Verfahrens zu erweitern. Der hieraus resultierende alternierende bidirektionale A*-Algorithmus findet den Brückenknoten des kürzesten Weges erwartungsgemäß erheblich schneller.

Nowak u. a. [2012] schlagen vor, nach erstmaliger Identifikation eines Brückenknotens mit dem A*-Verfahren zum originalen bidirektionalen Dijkstra-Verfahren der Contraction Hierarchies zu wechseln. Für das Dijkstra-Verfahren liegen die Knoten der Horizontmengen aus Vorwärts- und Rückwärtssuche nun jedoch ungeeignet sortiert vor und müssen daher einmalig gemäß ihrer Entfernung zum Start- bzw. Zielknoten umsortiert werden. Erst dann beginnen die weiteren Iterationen.

Eine genauere Analyse zeigt jedoch, dass dieser Ansatz unnötig kompliziert ist. Aufbauend auf einer digitalen Straßenkarte Niedersachsens[17] wurden hierfür 10 000 zufällige kürzeste Wegsuchen durchgeführt und die Lage des dem kürzesten Weg zugrunde liegenden Brückenknotens bezüglich des Start- und Zielknotens untersucht. Die Vermutung ist, dass sich in den meisten Fällen dieser Brückenknoten innerhalb einer Ellipse um den Start- und Zielknoten befindet. Anschaulich formuliert gilt dabei: Je flacher diese Ellipse ist, desto schneller findet eine bidirektionale A*-Suche[18] den Brückenknoten.

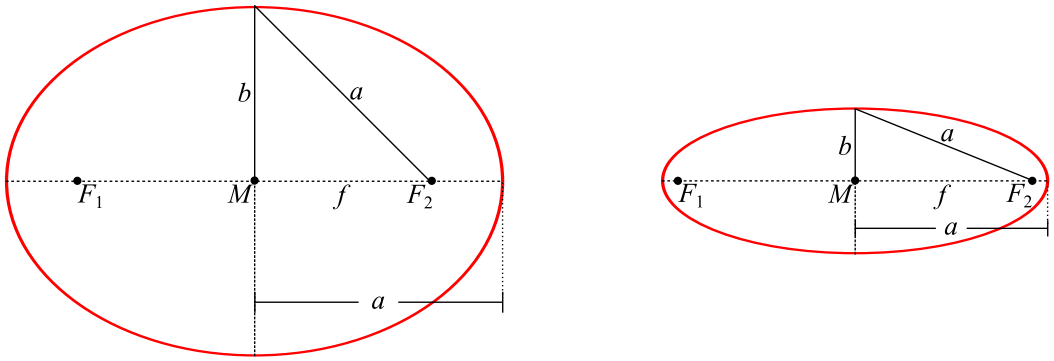
Einschub: Kenngrößen einer Ellipse Eine Ellipse bezeichnet eine Kurve um zwei so genannte *Brennpunkte* F_1 und F_2. Für jeden Punkt auf der Ellipse gilt, dass die Summe der euklidischen Distanzen zu den beiden Brennpunkten gleich groß ist. Die Achse durch die beiden Brennpunkte wird *Hauptachse* genannt; der Ellipsen-Mittelpunkt M trennt sie in die beiden *großen Halbachsen*. Senkrecht zur Hauptachse verläuft die *Nebenachse* durch den Mittelpunkt, der sie in die beiden *kleinen Halbachsen* teilt.

Exzentrizität

Für die präzise Beschreibung der Form der Ellipse reicht die Kenntnis der Lage ihrer Brennpunkte nicht aus. Sie kann aber mithilfe der *Exzentrizität* ε ausgedrückt werden. Dieser einheitenlose Wert berechnet sich aus dem Verhältnis des Abstands zwischen Brenn- und Mittelpunkt f zur Länge der großen Halbachse a und nimmt Werte zwischen 0 inklusive und 1 exklusive an. Für $f = 0 \Rightarrow \varepsilon = 0$ liegen die beiden Brennpunkte aufeinander; es ergibt sich als Sonderfall ein Kreis. Je größer ε ist, desto flacher verläuft die Ellipse. Abbildung 5.7, S. 153, zeigt beispielhaft zwei Ellipsen mit gleichem Abstand zwischen den Brennpunkten aber verschiedenen Exzentrizitäten.

[17] für genauere Informationen zur Niedersachsen-Karte: siehe Kapitel 3 und insbesondere Tabelle 3.6, S. 85

[18] siehe hierzu auch Abschnitt 4.1.1.3, S. 101 ff., und insbesondere Abbildung 4.6, S. 102, zur Ausbreitung des A*-Suchraums



(a) Ellipse mit $\varepsilon = 0,714$ (b) Ellipse mit $\varepsilon = 0,925$

Abbildung 5.7: Kenngrößen einer Ellipse
Je größer $\varepsilon = f/a$, desto flacher verläuft die Ellipse.

Für jede der 10 000 Wegsuchen wurde eine Ellipse mittels euklidischer Distanzen gebildet, deren Brennpunkte der Start- und der Zielknoten sind. Die Form dieser Ellipsen, d. h. ihre Exzentrizität, ergab sich, indem definiert wurde, dass der Brückenknoten, der von der Wegsuche zwischen Start- und Zielknoten identifiziert wurde, *auf* der Ellipse liegt. Für diese Ellipsen gilt in Anlehnung an Abbildung 5.7:

- Die Summe der beiden Luftdistanzen vom Start- und Zielknoten zum Brückenknoten entspricht genau $2a$, denn der Brückenknoten liegt auf der Ellipse.
- Die halbe Luftdistanz zwischen Start- und Zielknoten entspricht f.

Daraus ergibt sich direkt ε als das Verhältnis von f zu a. Die in Abbildung 5.9, S. 154, zusammengefasste Analyse zeigt ein deutliches Ergebnis: Die überwiegende Mehrheit der untersuchten Brückenknoten liegt in einer extrem flachen Ellipse um den Start- und den Zielknoten der zugehörigen Wegsuche. In einem Viertel der Wegsuchen besaß die oben beschriebene Ellipse eine Exzentrizität von mindestens 0,999; der Median aller Exzentrizitäten lag bei etwa 0,995. Abbildung 5.8 zeigt die Form einer derartigen Ellipse.

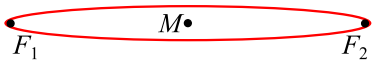


Abbildung 5.8: Ellipse mit $\varepsilon = 0,995$

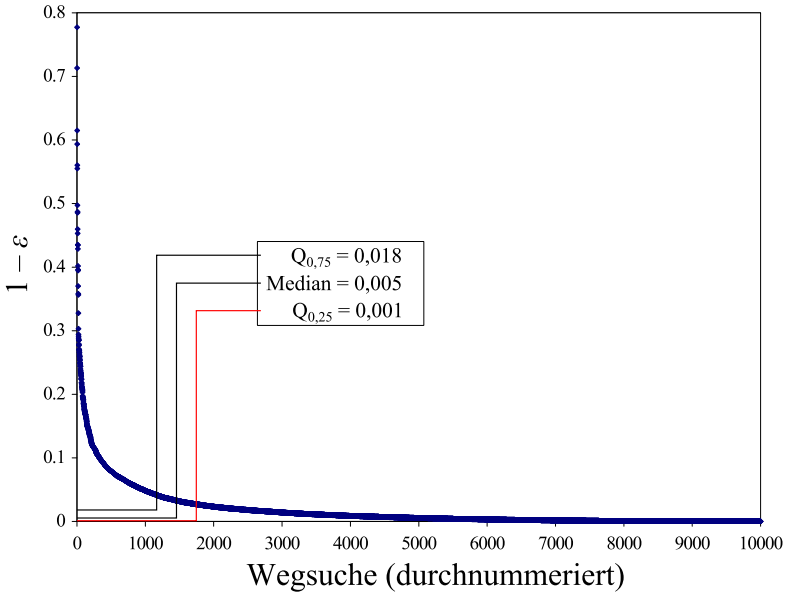


Abbildung 5.9: Auswertung der Exzentrizitätsmessung

Der untersuchte Weg mit der größten Abweichung ist in Abbildung 5.10 dargestellt. Start- und Zielknoten liegen hier auf verschiedenen Seiten der Weser. Der Weg führt über die nächstgelegene Brücke, wodurch sich der weit entfernte Brückenknoten erklärt.

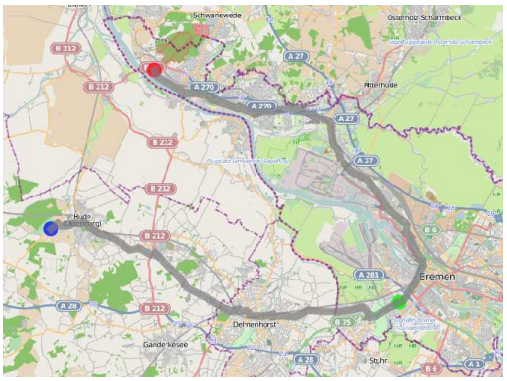


Abbildung 5.10: Ausreißer bei der Auswertung der Exzentrizitätsmessung
Der kürzeste Weg zwischen Startknoten (Mitte links, blau) und Zielknoten (oben links, rot) überquert die Weser und macht einen Bogen. Der Brückenknoten (unten rechts, grün) liegt daher ungewöhnlich positioniert zu den anderen beiden.

Als Konsequenz der Analyse ist eine bidirektionale A*-Suche auf einer Contraction Hierarchy der bei Geisberger u. a. [2008] vorgeschlagenen bidirektionalen Dijkstra-Suche vorzuziehen. Voraussetzung ist natürlich, dass eine Abschätzung der Restdistanz – beispielsweise per Luftdistanz – möglich ist. Die empirischen Untersuchungen in Abschnitt 5.6.6 unterstützen diese Aussage. Verfahren 14, S. 155, vereint die Erkenntnisse aus Abschnitt 5.2.2.1 und aus diesem Abschnitt in einem zielgerichteten Wegsuche-Algorithmus. Zusätzlich wird dort in Zeile 17 auch die Knotenblockierung aus Abschnitt 5.2.1 durchgeführt.

Verfahren 14 Zielgerichtete Wegsuche bei Contraction Hierarchies

Gegeben: Contraction Hierarchy $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$,
Startknoten s, Zielknoten t mit $s, t \in V$

1: $s.\mathit{distanz}_{\text{vor}} \coloneqq 0$, $t.\mathit{distanz}_{\text{rück}} \coloneqq 0$ ▸ Initialisierung
2: $v.\mathit{distanz}_{\text{vor}} \coloneqq \infty$, $v.\mathit{vorgänger}_{\text{vor}} \coloneqq \mathit{NULL}$ $\forall v \in V \setminus \{s\}$
3: $v.\mathit{distanz}_{\text{rück}} \coloneqq \infty$, $v.\mathit{vorgänger}_{\text{rück}} \coloneqq \mathit{NULL}$ $\forall v \in V \setminus \{t\}$
4: $\mathit{Lösungsmenge}_{\text{vor}} \coloneqq \emptyset$, $\mathit{Lösungsmenge}_{\text{rück}} \coloneqq \emptyset$
5: $\mathit{Horizontmenge}_{\text{vor}} \coloneqq \{s\}$, $\mathit{Horizontmenge}_{\text{rück}} \coloneqq \{t\}$
6: $m \coloneqq \mathit{NULL}$ ▸ Brückenknoten des kürzesten bekannten Weges
7: $\mathit{best} \coloneqq \infty$ ▸ Länge des kürzesten bekannten Weges
8: $\mathit{fertig}_{\text{vor}} \coloneqq$ `falsch` , $\mathit{fertig}_{\text{rück}} \coloneqq$ `falsch` ▸ Abbruchkriterien
9: **while not** ($\mathit{fertig}_{\text{vor}}$ **und** $\mathit{fertig}_{\text{rück}}$) **do**
10: **if not** $\mathit{fertig}_{\text{vor}}$ **then**
11: Wähle v mit $v.\mathit{distanz}+\text{rd}(v) = \min\{v'.\mathit{distanz}+\text{rd}(v') \mid v' \in \mathit{Horizontmenge}_{\text{vor}}\}$
12: Entferne v aus $\mathit{Horizontmenge}_{\text{vor}}$
13: Nimm v in $\mathit{Lösungsmenge}_{\text{vor}}$ auf
14: **if** $v.\mathit{distanz}_{\text{vor}} + \text{rd}(v) \geq \mathit{best}$ **then**
15: $\mathit{fertig}_{\text{vor}} \coloneqq$ `wahr` ▸ Die Vorwärtssuche treminiert
16: **end if**
17: **if** v kann nicht gemäß Gleichung (5.8), S. 146, blockiert werden **then**
18: Expandiere v in Vorwärtsrichtung,
verwende dabei nur solche Pfeile $(v, v') \in E$ mit $v.\mathit{rang} < v'.\mathit{rang}$
19: **end if**
20: **if** $v.\mathit{distanz}_{\text{vor}} + v.\mathit{distanz}_{\text{rück}} < \mathit{best}$ **then**
21: $\mathit{best} \coloneqq v.\mathit{distanz}_{\text{vor}} + v.\mathit{distanz}_{\text{rück}}$
22: $m \coloneqq v$
23: **end if**
24: **end if**
25: **if not** $\mathit{fertig}_{\text{rück}}$ **then**
26: Führe analog zu Zeilen 11–23 eine A*-Iteration der Rückwärtssuche aus
27: **end if**
28: **end while**

Ergebnis: Ein kürzester Weg von s zu t über m ist für die erste Teilstrecke rekursiv konstruierbar aus $m.\mathit{vorgänger}_{\text{vor}}$ und für die zweite Teilstrecke aus $m.\mathit{vorgänger}_{\text{rück}}$. Es gilt: $m.\mathit{distanz}_{\text{vor}} + m.\mathit{distanz}_{\text{rück}} = \text{dist}(s, t) = \mathit{best}$. Wurde kein Brückenknoten gefunden, existiert kein Weg zwischen s und t.

5.2.3 Parallele Wegsuche in Contraction Hierarchies

Die in diesem Kapitel vorgestellten bidirektionalen Varianten der Wegsuche lassen sich alle auf effiziente Weise parallelisieren. Der Grundgedanke ist dabei jeweils, die *Vorwärts-* und die *Rückwärtssuche* unabhängig voneinander parallel auszuführen. Nach n unabhängigen Iterationen muss für beide Suchrichtungen geprüft werden, ob einer der letzten n Knoten, die in die Lösungsmenge aufgenommen wurden, bereits Teil der Horizont- oder Lösungsmenge der jeweils anderen Suchrichtung ist und somit als Brückenknoten dienen kann. Wie auch bei der nicht-parallelen Wegsuche, muss nur derjenige Brückenknoten gespeichert werden, der zum kürzesten, bisher bekannten Weg gehört.

Stellgröße: Schrittweite

Zentrale Stellgröße der parallelisierten Wegsuchen ist die Anzahl der Iterationen, die gleichzeitig in Vorwärts- und Rückwärtsrichtung erfolgen, bevor geprüft wird, ob ein Brückenknoten gefunden wurde. Dies wird hier mit dem Begriff *Schrittweite* bezeichnet. Offensichtlich kann das Verfahren durch zu große Schrittweiten nahezu beliebig *verlangsamt* werden. Andererseits wird der positive Effekt einer Parallelisierung bei zu wenigen nebenläufigen Iterationen vom zusätzlich notwendigen Verwaltungsaufwand, wie etwa dem Speichermanagement der parallelen Prozesse, konterkariert.

Die optimale Anzahl paralleler Iterationen ist im Allgemeinen abhängig vom zugrunde liegenden Graphen sowie der Implementierung der Nebenläufigkeit. Messungen und Optimalwerte für die in dieser Arbeit verwendeten Straßenkarten und Implementierung finden sich in Abschnitt 5.6.7, S. 193 ff.

Eine Optimierung der parallelen Wegsuche besteht darin, nicht stets nach Brückenknoten zu suchen. Eine einfache Überlegung hilft, hier unnötige Prüfungen zu vermeiden: Da der Brückenknoten m, mit dessen Hilfe der kürzeste Weg letztlich rekonstruiert werden kann, selbst Teil des kürzesten Weges zwischen Start- und Zielknoten s und t ist, muss für ihn selbstverständlich gelten:

$$m.\mathit{distanz}_{\text{vor}} + m.\mathit{distanz}_{\text{rück}} \geq \text{rd}(s). \tag{5.11}$$

Hierbei steht $\text{rd}(s)$ wie zuvor schon beim A*-Verfahren[19] für die nicht-überschätzende Restdistanz-Schätzung vom Startknoten s zum Zielknoten t. Die Suche nach einem Brückenknoten ist erst dann sinnvoll, wenn die Vorwärts- und Rückwärtssuche weit genug vorangeschritten sind.

Seien im Laufe einer s-t-Wegsuche $u \in \mathit{Horizontmenge}_{\text{vor}}$ und $v \in \mathit{Horizontmenge}_{\text{rück}}$. Ferner gelte $u.\mathit{distanz}_{\text{vor}} = \max\{u'.\mathit{distanz}_{\text{vor}} \mid u' \in \mathit{Horizontmenge}_{\text{vor}}\}$ und spiegelbildlich $v.\mathit{distanz}_{\text{rück}} = \max\{v'.\mathit{distanz}_{\text{rück}} \mid v' \in \mathit{Horizontmenge}_{\text{rück}}\}$. Dann kann *kein* Brückenknoten erreicht worden sein, solange noch gilt:

$$u.\mathit{distanz}_{\text{vor}} + v.\mathit{distanz}_{\text{rück}} < \text{rd}(s). \tag{5.12}$$

[19] siehe Abschnitt 4.1.1.3, S. 101 ff.

Diese Optimierung gilt nicht nur für die parallele Wegsuche – hier ist sie jedoch etwas effizienter. Denn nach den n nebenläufigen Vorwärts- und Rückwärts-Iterationen kann im Idealfall die Überprüfung für alle n zuletzt zu den Lösungsmengen hinzugefügten Knoten auf einmal übersprungen werden.

Darüber hinaus sind Varianten der parallelen Wegsuche denkbar, die adaptiv die Anzahl der unabhängigen Vorwärts- und Rückwärtsiterationen regulieren. Auch hier bietet sich ein kontinuierlicher Abgleich zwischen den Distanzen der zuletzt erreichten Knoten und der Restdistanz-Schätzung an. Allerdings muss der Aufwand zur Ermittlung der Iterationen-Anzahl gering bleiben, um nicht den Nutzen der Parallelisierung zu übersteigen.

5.2.4 Zusammenhang zwischen Contraction Hierarchies und Hub-Based Labeling

In diesem Abschnitt wird die Berechnung der Labels des in Abschnitt 4.2.4, S. 123 ff., vorgestellten HL-Verfahren erläutert. Es wird gezeigt, inwiefern HL als eine Erweiterung von Contraction Hierarchies verstanden werden kann.

Zur Erinnerung: Bei HL erfolgt die Suche nach dem kürzesten Weg zwischen einem Startknoten s und einem Zielknoten t mithilfe der Labels, die an jedem Knoten $v \in V$ hinterlegt sind. Diese Labels bestehen aus einer Knotenmenge $L^K_{v,\text{vor}} \subseteq V$ und einer zugehörigen Distanzmenge $L^D_{v,\text{vor}}$ für bei v beginnende Vorwärtssuchen. Analog existieren $L^K_{v,\text{rück}} \subseteq V$ und $L^D_{v,\text{rück}}$ für Rückwärtssuchen. Der kürzeste Weg zwischen $s \in V$ und $t \in V$ führt dann über denjenigen Brückenknoten $m \in L^K_{s,\text{vor}} \cap L^K_{t,\text{rück}}$, der $L^D_{s,\text{vor}}(m) + L^D_{t,\text{rück}}(m)$ minimiert. Außerdem stellt das Verfahren die so genannte *Abdeckungs-Eigenschaft*[20] sicher, d. h., für alle kürzesten Wege kann ein derartiger Brückenknoten innerhalb der Labels gefunden werden.

In Abschnitt 4.2.4 wurde allerdings noch nicht erläutert, wie Labels ermittelt werden. Dies soll mit dem Wissen um die Wegsuche bei Contraction Hierarchies an dieser Stelle nachgeholt werden.

In dem hier vorgestellten Verfahren bildet eine vorberechnete Contraction Hierarchy $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2)$ die Grundlage für die Berechnung der Labels des HL-Verfahrens. Für jeden Knoten aus G_{ch} wird dazu gemäß der beim CH-Verfahren eingesetzten Dijkstra-Variante der vollständige Suchbaum zunächst für die Vorwärtssuche ermittelt, d. h., Knoten werden nur in Richtung derjenigen Pfeile expandiert, die zu Knoten mit höherem Rang führen. Für einen Knoten $v \in V$ bilden die so erreichten Knoten die Knotenmenge $L^K_{v,\text{vor}}$ und die zugehörigen Distanzen werden in $L^D_{v,\text{vor}}$ gespeichert. Anschließend wird für die Rückwärtslabels in analoger Weise der vollständige Suchbaum der Rückwärtssuche aufgebaut und ausgewertet. Dies erklärt den hohen Speicherbedarf des Verfahrens und ist vermutlich auch der Grund dafür, dass bei HL nicht auch noch die Vorgänger- und Nachfolger-Beziehungen zu den Knoten aus den Suchbäumen mitgeführt werden, wie es beim bidirektionalen Dijkstra-Verfahren ansonsten der Fall wäre[21], sondern nur die Distanzen der Wege, die zu den Knoten führen.

[20] siehe Seite 124, Stichwort „Abdeckungs-Eigenschaft“
[21] siehe etwa Verfahren 11, S. 138

Im Anschluss kann für Wegsuchen eine sehr stark verkürzte Variante des modifizierten Dijkstra-Algorithmus nach Verfahren 11, S. 138, durchgeführt werden. Denn die Suchbäume von Start- und Zielknoten stehen bereits zur Verfügung. Es muss lediglich der Brückenknoten gefunden werden, der gemäß des Beweises von Seite 140 ff. in beiden Suchbäumen vorhanden ist. Da in vielen Fällen mehrere Knoten für diese Rolle in Frage kommen, wird derjenige gewählt, der die Gesamtdistanz des von ihm induzierten Weges minimiert.

5.3 Knotensortierung bei Contraction Hierarchies

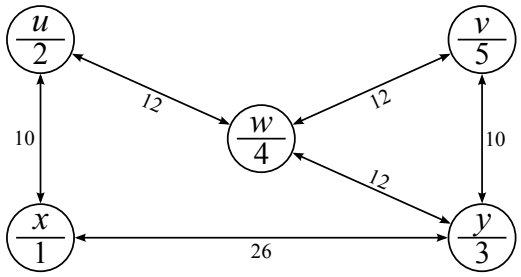
In diesem Abschnitt wird der Aspekt der Knotensortierung, d. h. der Rangvergabe an die Knoten eines Eingangsgraphen, während der Erzeugung einer Contraction Hierarchy, genauer beleuchtet. Dazu werden ausgewählte Vorschläge aus der Literatur[22] sowie einige neue vorgestellt. Die Bedeutung einer geschickten Knotensortierung lässt sich erahnen, wenn das Beispiel in Abbildung 5.1, S. 136, erneut betrachtet wird. Abbildung 5.11, S. 159, zeigt, wie sich hier durch eine andere Verteilung der Knotenränge eine CH erzeugen lässt, die gänzlich *ohne* Shortcut-Pfeile auskommt.

Dass eine CH möglichst wenig zusätzliche Shortcut-Pfeile besitzen sollte, um die Anzahl der Iterationen bei der Wegsuche möglichst gering zu halten, ist einerseits naheliegend. Andererseits ist der Zusammenhang zwischen diesen beiden Größen – Anzahl der Shortcut-Pfeile im Graph und mittlere Anzahl Dijkstra-Iterationen bei einer Wegsuche – auch nicht streng proportional, wie auf Seite 163 an einem Gegenbeispiel gezeigt wird.

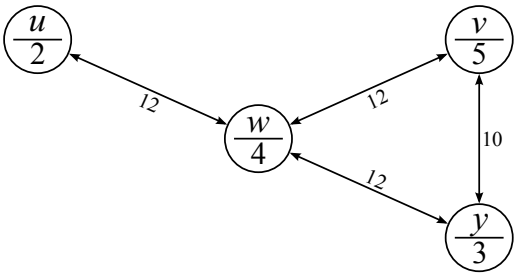
Das grundsätzliche Vorgehen ist bei allen in diesem Abschnitt vorgestellten Ansätzen zur Knotensortierung identisch und wird nachfolgend erläutert. Bei der Erzeugung einer CH werden die Knoten des Eingangsgraphen der Reihe nach kontrahiert[23]. Anders jedoch, als in Abschnitt 5.1, S. 132 ff., vereinfacht dargestellt, sind die Ränge der Knoten während der Hierarchie-Erzeugung typischerweise weder statisch, noch bereits *vor* der Kontraktion festgelegt. Denn vorab ist nur schwer abzusehen, welche Auswirkungen eine vollständige Zuweisung von Knotenrängen sowohl auf die Notwendigkeit von Shortcut-Pfeilen als auch auf die Wegsuchen in der entstehenden CH besitzen.

Stattdessen sollten alle während der Hierarchie-Erzeugung bislang noch nicht kontrahierten Knoten einer dynamischen *Vorsortierung* unterliegen, die sich im Verlauf der Hierarchie-Erzeugung noch ändern kann. Erst durch die tatsächliche Kontraktion eines Knotens v, d. h., sobald auch alle notwendigen Shortcut-Pfeile in den Graphen eingefügt werden, die v überbrücken, steht der Rang von v endgültig fest: Ist Knoten v der i-te kontrahierte Knoten, erhält v den Rang i.

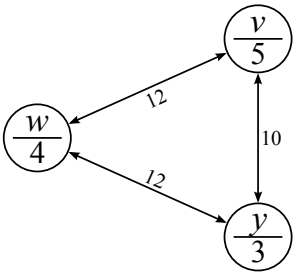
[22] Geisberger u. a. [2008, S. 322 f.], Geisberger [2008, S. 14. ff.] sowie Geisberger u. Vetter [2011, S. 106]
[23] siehe Abschnitt 5.1, S. 132 ff.



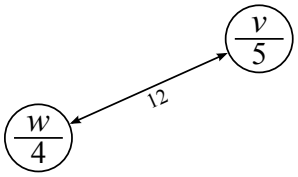
(a) Originalgraph
Knoten x wird nun kontrahiert



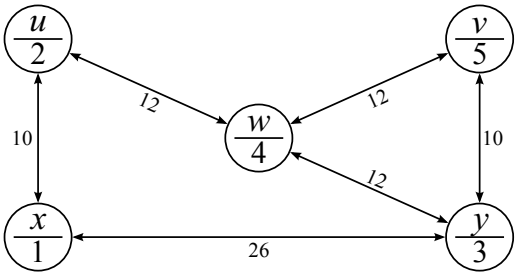
(b) Overlay-Graph nach der 1. Iteration
Knoten u wird nun kontrahiert



(c) Overlay-Graph nach der 2. Iteration
Knoten y wird nun kontrahiert



(d) Overlay-Graph nach der 3. Iteration
Abbruch, da nur noch zwei Knoten vorhanden sind



(e) Vollständige Hierarchie, ohne dass ein Shortcut-Pfeil notwendig ist

Abbildung 5.11: CH-Erzeugung mit anderer Knotensortierung
Durch diese Knotensortierung entstehen in der vollständigen CH keine Shortcut-Pfeile.
Eine Linie mit zwei Pfeilköpfen steht für zwei Pfeile in entgegengesetzte Richtungen. Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Zahlen an Pfeilen geben deren Länge an.

Die dynamische Vorsortierung der Knoten bei der Hierarchie-Erzeugung erfolgt mittels einer Bewertungsfunktion $\psi : V \to \mathbb{R}$, die jedem Knoten des Graphen eine aktuelle *Wichtigkeit*[24] zuweist. Es wird immer der aktuell *unwichtigste* Knoten kontrahiert, d. h. Knotenwichtigkeit

[24] im englischen Original bei Geisberger u. a. [2008]: *node importance*

derjenige Knoten v, für den gilt:

$$\psi(v) = \min\{\psi(v') \mid v' \in V\} \text{ und} \quad v \text{ wurde noch nicht kontrahiert.} \tag{5.13}$$

Notation der Knotensortierung

Formal sind die in diesem Abschnitt vorgestellten Ansätze unterschiedliche *Bewertungsfunktionen* für die bislang noch nicht kontrahierten Knoten. Die Ansätze lassen sich darüber hinaus auch kombinieren, indem $\psi(v)$ als gewichtete Summe aus den Einzelbewertungen der beteiligten Ansätze berechnet wird. Zu jedem Ansatz ist daher nachfolgend ein Kürzel angegeben, mit dessen Hilfe sich die Kombination der Ansätze ausdrücken lässt. Beispielweise bedeutet eine Sortierung nach $1ED+2R$, dass die im Folgenden vorgestellte *Pfeildifferenz* (ED) einfach und die *Anzahl notwendiger Shortcut-Pfeile* (R) doppelt in die Wichtigkeit eines Knotens einfließen. Es steht damit verkürzend für den Ausdruck $\psi(v) = ED(v) + 2R(v) \quad \forall v \in V$.

Die Kontraktion eines Knotens ändert in den meisten Fällen die Wichtigkeit anderer Knoten. Im Rahmen einer konkreten Implementierung ist es deshalb für Graphen mit vielen Knoten unpraktisch das Minimum aus Gleichung (5.13) nach jeder Knotenkontraktion für alle im Graphen verbliebenen Knoten neu zu berechnen. Dies gilt besonders für Graphen, die auf großen Straßenkarten basieren, wie beispielsweise der in dieser Arbeit verwendeten Karte von Europa[25] mit mehreren Millionen Knoten.

Geisberger u. a. [2008, S. 322] schlagen daher vor, die Wichtigkeit aller Knoten des Eingangsgraphen einmalig zu berechnen und anhand des Ergebnisses eine Prioritätswarteschlange P mit den Knoten zu füllen, an deren Anfang der aktuell unwichtigste Knoten steht. Bevor der aktuell unwichtigste Knoten v aus P *möglicherweise* kontrahiert wird, berechnen sie $\psi(v)$ erneut. Ist der neu ermittelte Wert kleiner oder gleich dem alten Wert, wird v *tatsächlich* kontrahiert, so dass sein Rang feststeht, und er wird aus P entfernt. Ansonsten wird v mit dem neuen Wert wieder in P einsortiert, so dass im Anschluss ein anderer Knoten am Anfang von P steht. Geisberger u. a. [2008, S. 322] sprechen in diesem Zusammenhang von einem *Lazy-Update* der Knoten.

Nach Kontraktion: Update aller Nachbarn

Zusätzlich sollte direkt im Anschluss an die Kontraktion eines Knotens v die Wichtigkeit aller seiner Nachbarknoten aktualisiert werden (vgl. [Geisberger u. a., 2008, S. 322]). Dadurch wird deren Position in P aktualisiert. Es entsteht eine „genauere“ Sortierung, weil die Wichtigkeit der Nachbarknoten erwartungsgemäß von der Kontraktion von v beeinflusst wird.

Die für diese Arbeit verwendete Implementierung arbeitet sowohl mit Lazy-Update der Knoten als auch mit der Aktualisierung aller Nachbarknoten des zuletzt kontrahierten Knotens. Verfahren 15, S. 161, zeigt den Ablauf einer CH-Erzeugung mit dynamischer Knotensortierung als Pseudocode. Dort ist die letzte Wichtigkeit eines Knotens v, d. h. das Ergebnis der jüngsten Berechnung von $\psi(v)$, als Knotenattribut `wichtigkeit` modelliert.

[25] siehe Tabelle 3.6, S. 85

Verfahren 15 Erzeugung einer Contraction Hierarchy mit Lazy-Update

Gegeben: Nicht-negativ bewerteter, gerichteter Graph $G = (V, E, c)$, Prioritätswarteschlange P

1: Befülle P mit allen Knoten aus V ▸ wahlweise willkürlich oder gemäß der Kriterien aus diesem Abschnitt
2: **while** $P \neq \emptyset$ **do**
3: $v \coloneqq \min\{P\}$ ▸ Wähle das kleinste Element aus der Prioritätswarteschlange
4: $P \coloneqq P \setminus v$
5: $\mathit{Wichtigkeit}_{\text{alt}} \coloneqq v.\mathit{wichtigkeit}$ ▸ Zwischenspeichern der „alten“ Wichtigkeit
6: $v.\mathit{wichtigkeit} \coloneqq \psi(v)$ ▸ Aktualisiere die Wichtigkeit gemäß der Kriterien aus diesem Abschnitt
7: **if** $v.\mathit{wichtigkeit} \leq \mathit{Wichtigkeit}_{\text{alt}}$ **then**
8: Kontrahiere v ▸ vgl. Zeile 6 - 17 in Verfahren 10, S. 135
9: **for all** $v' \in \mathcal{NB}(v)$ **do** ▸ Update aller Nachbarknoten von v
10: $P \coloneqq P \setminus v'$
11: $v'.\mathit{wichtigkeit} \coloneqq \psi(v')$ ▸ Aktualisiere die Wichtigkeit von v'
12: Sortiere v' erneut in P ein
13: **end for**
14: **else**
15: Sortiere v erneut in P ein
16: **end if**
17: **end while**

Ergebnis: Eine vollständige Contraction Hierarchy

Es folgt ein kurzer Überblick über mögliche Bewertungsfunktionen. Die Auswahl ist beliebig erweiterbar, denn das grundsätzliche Verfahren liefert mit *allen* Knotensortierungen korrekte Ergebnisse (vgl. [Geisberger u. a., 2008, S. 325]). Die nachfolgenden vier Sortierkriterien (*ED*, *D*, *C* und *Q*) stammen aus Geisberger u. a. [2008, S. 322 ff.].

ED

Pfeildifferenz Die Bewertungsfunktion *Pfeildifferenz*[26] *ED* zielt direkt auf die oben genannte Überlegung ab: Je weniger Shortcut-Pfeile die CH benötigt, desto schneller verläuft grundsätzlich eine durchschnittliche Wegsuche. Die Pfeildifferenz ergibt sich für einen Knoten v aus der Zahl der durch eine Kontraktion von v neu zu erzeugenden Shortcut-Pfeile abzüglich der Zahl der vor der Kontraktion von v zu v adjazenten Pfeile. Formal gilt daher:

$$ED(v) = |SC_v| - |\mathcal{V}(v)| - |\mathcal{N}(v)| \text{ mit} \tag{5.14}$$
SC_v = Menge der neu zu erzeugenden Shortcut-Pfeile, sollte v kontrahiert werden.

Simulierte Knotenkontraktion

Offensichtlich kann $ED(v)$ nur ermittelt werden, indem die Kontraktion von v zwar nicht vollständig, jedoch bis inklusive der Ermittlung der notwendigen Shortcut-Pfeile durchgeführt wird. Dies bezeichnet Geisberger [2008, S. 23] als *simulierte Knotenkontraktion*.

[26] im englischen Original bei Geisberger u. a. [2008]: *edge difference*

D **Anzahl kontrahierter Nachbarknoten** Die Wegsuche erreicht ihre starke Beschleunigung gegenüber dem Standard-Dijkstra-Verfahren, indem viele Knoten „übersprungen“ werden. Insofern ist es wünschenswert, dass die Knotenränge in etwa gleichmäßig über den gesamten Graphen verteilt werden. Andernfalls droht eine Degeneration, wie sie als Extremfall in der CH in Abbildung 5.12 dargestellt ist, die aufgrund der vergebenen Knotenränge keine Shortcut-Pfeile besitzt.

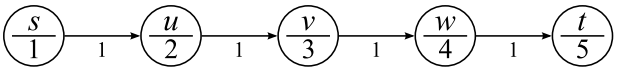


Abbildung 5.12: Beispiel für Degeneration einer CH-Wegsuche
Die Vorwärtssuche von Knoten s nach t besucht sämtliche Knoten des Beispiel-Graphen.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Zahlen an Pfeilen geben deren Länge an.

Wird hier der kürzeste Weg von Knoten s nach Knoten t gesucht, bricht die Rückwärtssuche sofort ab, nachdem Knoten t zu Lösungsmenge$_{\text{rück}}$ hinzugefügt wurde, da kein anderer Knoten einen höheren Rang besitzt. Gleichzeitig iteriert die Vorwärtssuche über jeden Knoten des kürzesten Weges, so dass sich gegenüber dem Standard-Dijkstra-Verfahren keine Beschleunigung ergibt. Im Gegenteil: Während letzteres nach fünf Iterationen den kürzesten Weg findet, benötigt die Wegsuche nach dem CH-Verfahren sogar sechs; fünf für die Vorwärtssuche und eine für die Rückwärtssuche.

Tatsächlich ist die dargestellte Knotensortierung für den obigen Graphen die ungünstigste. Denn es existiert keine andere Knotensortierung, bei der irgendeine andere Wegsuche noch mehr Iterationen erfordert. Im Gegensatz dazu zeigt Abbildung 5.13 beispielhaft zwei für die Wegsuche von Knoten s nach Knoten t günstigere Sortierungen.

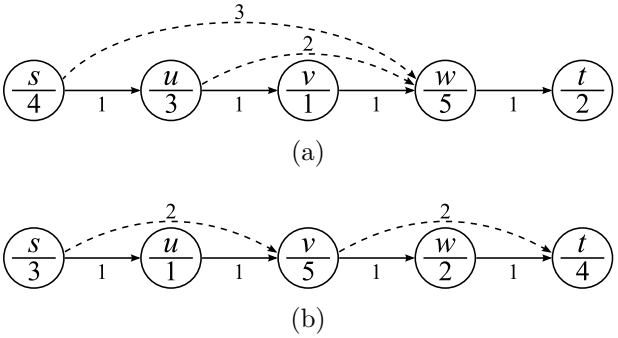


Abbildung 5.13: Beispiele für CH-Wegsuchen ohne Degeneration
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Zahlen an Pfeilen geben deren Länge an. Gestrichelte Linien symbolisieren Shortcut-Pfeile.

Für die CH in Abbildung 5.13a wird als Lösungsweg $p_{s,t} = \langle (s,w),(w,t) \rangle$ gefunden, und in Abbildung 5.13b ist es $p'_{s,t} = \langle (s,v),(v,t) \rangle$. Beide Wege führen über Shortcut-Pfeile und implizieren damit denselben, nur auf Original-Pfeilen basierenden, kürzesten Weg.

Anzahl Pfeile vs. Anzahl Iterationen der Suche

Das obige Beispiel verdeutlicht gleichzeitig die Erkenntnis, dass eine geringe Anzahl zusätzlicher Shortcut-Pfeile in einer Contraction Hierarchy nicht notwendigerweise zu schnellen Wegsuchen führt. Tatsächlich wächst die Anzahl benötigter Dijkstra-Iterationen der Wegsuche nicht monoton mit der Anzahl der Pfeile im Graphen.

Geisberger u. a. [2008, S. 323] vermerken für jeden Knoten des Graphen, wie viele seiner Nachbarknoten bereits kontrahiert wurden[27]. Je größer dieser Wert ist, desto *wichtiger* wird der Knoten. Als Konsequenz wird es unwahrscheinlicher, dass zwei benachbarte Knoten in direkter Folge kontrahiert werden. Denn die Kontraktion des einen erhöht die Wichtigkeit des anderen, und wichtige Knoten werden später kontrahiert.

C

Kontraktionskosten Bei einer Knotenkontraktion ist die Untersuchung, ob Shortcut-Pfeile eingefügt werden müssen, der aufwendigste Teil. Die Kontraktionskosten[28] zielen daher darauf ab, Knoten möglichst spät zu kontrahieren, für welche diese Suche besonders aufwendig ist. Als Messgröße hierfür dient die Anzahl der Knoten in den Dijkstra-Lösungsmengen bei der Suche nach notwendigen Shortcut-Pfeilen. Je größer die Lösungsmengen sind, desto *wichtiger* wird der Knoten. Wie auch die Pfeildifferenz können Kontraktionskosten für einen Knoten v also ausschließlich ermittelt werden, indem die Kontraktion von v nur *simuliert* wird.

Q

Wegsuchekosten Das nächste Sortierkriterium ist auf die Wegsuche in der vollständigen CH ausgerichtet. Der Wert der Wegsuchekosten[29] ist zunächst für jeden Knoten des Graphen null. Wird ein Knoten v kontrahiert, werden die Wegsuchekosten $Q(u)$ aller seine Nachbarknoten $u \in \mathcal{NB}(v)$ wie folgt aktualisiert: $Q(u) = \max\{Q(u), Q(v) + 1\}$. Geisberger [2008, S. 21 f.] zeigt, dass dieser Wert eine obere Schranke für die Anzahl der Dijkstra-Iterationen einer Wegsuche von einem beliebigen Startknoten zu u darstellt. Die Motivation ist daher, einen Knoten mit hohen Wegsuchekosten möglichst spät zu kontrahieren, damit dessen Nachbarknoten nicht die hohen Kosten „erben“.

Bei Geisberger u. Vetter [2011, S. 106] finden sich zwei weitere Sortierkriterien, die auch in dieser Arbeit verwendet werden. Beide setzen jeweils neu zu erzeugende Pfeile mit zu löschenden Pfeilen in Bezug.

EQ

Pfeilquotient In Analogie zur Pfeildifferenz (s. o.) wird für den Pfeilquotienten[30] eines Knotens v die Anzahl neu zu erzeugender Shortcut-Pfeile bei Kontraktion von v durch die Anzahl der zu v adjazenten Pfeile inklusive möglicherweise bereits erzeugter Shortcut-Pfeile geteilt.

[27] im englischen Original bei Geisberger u. a. [2008]: *deleted neighbors*; bei Geisberger [2008]: *contracted neighbors*

[28] im englischen Original bei Geisberger u. a. [2008]: *cost of contraction*

[29] im englischen Original bei Geisberger u. a. [2008]: *cost of queries*; bei Geisberger u. Vetter [2011]: *hierarchy depth*

[30] im englischen Original bei Geisberger u. Vetter [2011]: *edge quotient*

OEQ **Originaler Pfeilquotient** Der originale Pfeilquotient[31] teilt für einen Knoten v die Anzahl der originalen Pfeile des Graphen, die bei Kontraktion von v durch neue Shortcut-Pfeile überbrückt werden, durch die Anzahl der zu v adjazenten Original-Pfeile.

Abschließend werden noch drei neue Sortierkriterien vorgestellt. Die Idee zu diesen stammt aus eigenen Experimenten mit Kartenmaterial des OpenStreetMap-Projekts.

RAND **Zufälliger Bias** Um Strukturenbildungen in der zu erzeugenden Hierarchie zu vermeiden, kann jeder Knoten mit einem zufälligen Bias initialisiert werden.[32] Insbesondere dient dieser Ansatz der verbesserten Verteilung der (un-)wichtigen Knoten über die Karte. Damit der Wertebereich der Bewertungsfunktion transparent bleibt, empfiehlt es sich, den Bias zwischen null und eins zu wählen.

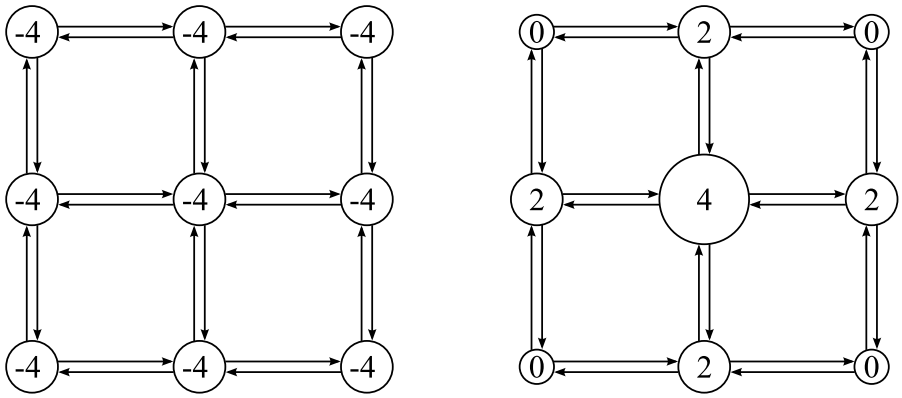
Von einer Sortierung, die *ausschließlich* auf diesem Kriterium beruht, ist allerdings abzuraten. In eigenen Tests zeigte sich, dass sowohl die Dauer für die Erzeugung dieser Hierarchie als auch deren Such-Performance nicht annähernd vergleichbar ist mit den anderen hier vorgestellten Kriterien. Stattdessen sollte der Bias immer in Kombination mit anderen Bewertungsfunktionen verwendet werden, um dort bei ansonsten gleich wichtigen Knoten den Ausschlag zu geben. Denn gerade bei der CH-Erzeugung für den Graphen einer digitalen Straßenkarte kommt es zu Beginn, wenn noch wenige Shortcut-Pfeile erzeugt wurden, bei allen anderen hier vorgestellten Bewertungsfunktionen dazu, dass mehrere Knoten des Graphen eine gleich große Wichtigkeit erhalten. Auch bei einer Linearkombination der Bewertungsfunktionen ohne Bias passiert dies häufig.

AB **Abbiegebeschränkungs-Flag** Abbiegebeschränkungs-Flags weisen Knoten, an denen Abbiegebeschränkungen vorliegen, den Wert eins zu und allen anderen Knoten den Wert null. Das Sortierkriterium kann verwendet werden, wenn die in Kapitel 6 beschriebenen Wegsuche-Methoden zur Berücksichtigung von Abbiegebeschränkungen innerhalb von Contraction Hierarchies zum Einsatz kommen. Es erlaubt, diese Kreuzungsknoten auf besondere Weise hervorzuheben.

R **Anzahl notwendiger Shortcut-Pfeile** Ein Knoten wird hier umso wichtiger, je mehr Shortcut-Pfeile bei seiner Kontraktion erzeugt werden müssen. Dieses Kriterium bildet gewissermaßen die zweite Komponente der oben beschriebenen Pfeildifferenz.

Die Stärke dieser Kennzahl gegenüber der Pfeildifferenz liegt in der differenzierteren Bewertung der Knoten. Hierzu sei auf Abbildung 5.14, S. 165, verwiesen, in der die beiden Sortierkriterien im direkten Vergleich dargestellt sind. Während die Pfeildifferenz (Abbildung 5.14a) jedem Knoten des Graphen die gleiche Wichtigkeit zuweist, kommt die Bewertung anhand der Anzahl notwendiger Shortcut-Pfeile (Abbildung 5.14b) näher an eine intuitive Beurteilung.

[31] im englischen Original bei Geisberger u. Vetter [2011]: *original edge quotient*
[32] In Abschnitt 5.5.2, S. 168 f., findet sich ein konkretes Beispiel für eine derartige Strukturenbildung.



(a) Bewertung nach Pfeildifferenz (*ED*)

(b) Bewertung nach Anzahl notwendiger Shortcut-Pfeile (*R*)

Abbildung 5.14: Vergleich der Knotensortierungen nach Pfeildifferenz und nach Anzahl notwendiger Shortcut-Pfeile
Die Zahlen in den Knoten geben die aktuelle Wichtigkeit nach der jeweiligen Bewertungsfunktion an; die Knotengröße dient lediglich der visuellen Unterstützung. Die Länge aller Pfeile sei gleich und ist daher nicht explizit mit angegeben.

Eine systematische Untersuchung unterschiedlicher Linearfaktorkombinationen der hier vorgestellten Sortierkriterien erfolgt nicht im Rahmen dieser Arbeit. Soweit nicht anders angeführt, liegt den für diese Arbeit generierten Contraction Hierarchies eine Sortierung nach dem Schema $2R + 8EQ + 4OEQ + 1Q + 1RAND$ zugrunde. Sie basiert auf der Sortierung, die Geisberger u. Vetter [2011, S. 106] verwenden. Jedoch verwendet sie anstelle der *Pfeildifferenz* die *Anzahl notwendiger Shortcut-Pfeile* und zusätzlich kommt ein *Bias* zum Einsatz, um Clusterbildungen zu vermeiden.

5.4 Eigenschaften des Graphen während einer Hierarchie-Erzeugung

Auch wenn sie in weiten Teilen von der Knotensortierung abhängig ist[33], existieren bei der Hierarchie-Erzeugung gewisse Charakteristika, die sich stets wiederholen. Insbesondere zählt hierzu der überproportional zur Anzahl der bereits kontrahierten Knoten wachsende durchschnittliche Knotengrad. Das Entfernen der ersten Knoten erfordert in den meisten Fällen noch kein Einfügen von Shortcut-Pfeilen. Erst später und insbesondere bei den letzten Prozenten bzw. Bruchteilen von Prozenten der zu kontrahierenden Knoten sind zunehmend neue Shortcut-Pfeile notwendig. In Abbildung 5.15, S. 166, ist das überproportionale Wachstum des durchschnittlichen Knotengrads bei der Hierarchie-Erzeugung deutlich zu erkennen.

Auffällig ist hier auch der Unterschied zwischen den verwendeten Metriken. Der durchschnittliche Knotengrad bei der Kürzeste-Wege-Metrik liegt durchweg deutlich über dem

[33] siehe hierzu Abschnitt 5.3

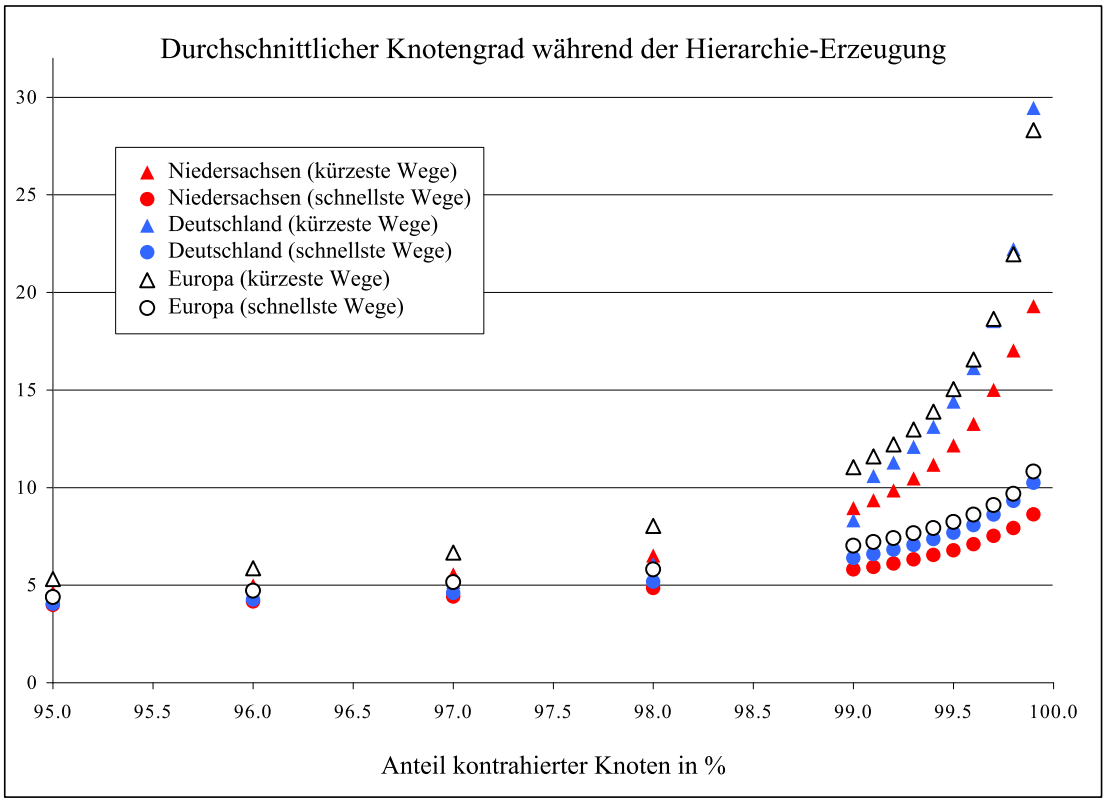


Abbildung 5.15: Durchschnittlicher Knotengrad während der Hierarchie-Erzeugung
Deutlich erkennbar ist der überproportionale Anstieg gegen Ende der Hierarchie-Erzeugung.
Zugrunde liegende Knotensortierung: $2R{+}8EQ{+}4OEQ{+}1Q{+}1RAND$

der Schnellste-Wege-Metrik. Dies erklärt sich dadurch, dass für schnellste Wege seltener neue Shortcut-Pfeile eingefügt werden müssen. Denn erwartungsgemäß besitzen die optimalen Wege hier eine höhere Überdeckung – nämlich entlang der Autobahnen und Schnellstraßen.

Das Verhalten beeinflusst natürlich auch die Laufzeit. Es ergibt sich ein überproportionaler Anteil der Gesamtzeit, der auf das letzte Prozent der zu kontrahierenden Knoten entfällt. Da diese Knoten einen besonders hohen Knotengrad besitzen, müssen für sie besonders viele Wegsuchen während ihrer simulierten Kontraktion erfolgen.

Darüber hinaus wird nach der Kontraktion eines Knotens auch die Wichtigkeit aller seiner Nachbarknoten neu berechnet[34]. Die Auswirkungen auf die Laufzeit potenzieren sich daher messbar: Abbildung 5.16, S. 167, stellt den Anteil der Laufzeit, die für die Kontraktion des letzten Prozents der Knoten anfällt, an der Gesamtlaufzeit der Hierarchie-Erzeugungen verschiedener Karten und Metriken vergleichend dar. Wie auch bei den durchschnittlichen Knotengraden in Abbildung 5.15 macht es einen deutlichen Unterschied, ob eine Contraction Hierarchy für schnellste oder für kürzeste Wege berechnet wird.

[34] siehe Seite 160

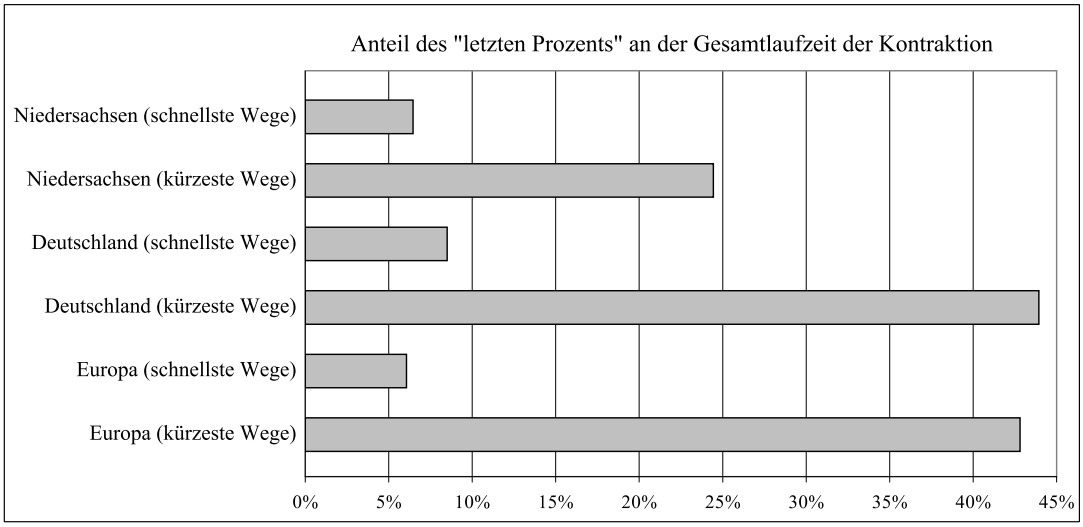


Abbildung 5.16: Anteil des „letzten Prozents“ an der Gesamtlaufzeit der Kontraktion
Zugrunde liegende Knotensortierung: $2R+8EQ+4OEQ+1Q+1RAND$
Für Details zu den Karten: siehe Kapitel 3, S. 61 ff., und insbesondere Tabelle 3.6, S. 85.

5.5 Implementierungsdetails der Verfahren

In diesem Abschnitt werden einige der Besonderheiten bei der Implementierung der in diesem Kapitel vorgestellten Verfahren näher erläutert. Im Wesentlichen betrifft dies Überlegungen zur Performance.

Abweichend von den in Kapitel 3, S. 61 ff., beschriebenen Präprozess-Schritten zum Import und zur Aufbereitung der OSM-Daten wurden die in dieser Arbeit erstellten Contraction Hierarchies nicht in einer Datenbank, sondern in einer Binärdatei gespeichert. Ausschlaggebend für diese Entscheidung sind zwei Hauptgründe:

1. Die Hierarchien sollen besonders effizient zu laden sein. Ein binäres Dateiformat ist hierfür einer Datenbank vorzuziehen.

2. Die Datenbank der Europa-Straßenkarte umfasst ohne Hierarchie-Informationen bereits über 37 GB. Aus Gründen der einfacheren Handhabbarkeit entschied sich der Autor, diese zentrale Datei nicht noch weiter zu vergrößern.

Das Format der CH -Binärdateien ist in Listing A.1, S. 317 f., als Referenz dokumentiert. Variable Ausdrücke sind durch spitze Klammern kenntlich gemacht; Zeilenumbrüche dienen lediglich der Lesbarkeit. Kommentaren innerhalb des Listing ist eine Raute vorangestellt.

5.5.1 Caching der Kosten eines Shortcut-Kantenobjekts

Wie bereits erwähnt[35], überbrücken Shortcut-Kantenobjekte immer genau zwei andere Kantenobjekte, die ihrerseits ebenfalls Shortcut-Kantenobjekte sein können. Sie lassen sich daher gemäß Verfahren 12, S. 142, rekursiv auflösen, um die ihnen zugrunde liegenden Original-Kantenobjekte zu ermitteln. Dies ist notwendig, wenn – je nach Metrik – die Länge eines Shortcut-Kantenobjekts oder die Fahrtdauer über dieses ermittelt werden muss.

Im Zuge der Erzeugung einer Contraction Hierarchy kommt es zu vielen Anfragen dieser Art an ein Kantenobjekt. Dies geschieht mindestens bei der Kontraktion jeder seiner beiden Endknoten, wenn ermittelt wird, ob ein Shortcut-Kantenobjekt über diesen Knoten erzeugt werden muss.

Tatsächlich aber werden fast alle Knoten im Rahmen einer *simulierten Kontraktion*[36] sehr häufig wiederholt betrachtet. Wird ein Shortcut-Kantenobjekt dabei jedes Mal von neuem rekursiv in seine Original-Kantenobjekte zerlegt, ist dies ineffizient. Stattdessen wird bei jedem Shortcut-Kantenobjekt[37] gespeichert, mit welcher Metrik er zuletzt bewertet wurde und welcher Wert sich dabei ergab. Beim nächsten Mal kann dieser Wert direkt erneut zurückgegeben werden, wenn dieselbe Metrik verwendet wird. Dieselbe Technik – im Prinzip eine Art „Ein-Eintrag-Cache“ – ist im Übrigen auch bei der Wegsuche auf einer vollständigen Contraction Hierarchy sehr sinnvoll und kommt daher ebenfalls bei allen Implementierungen dieser Arbeit zum Einsatz.

5.5.2 Besonderheit bei der Knotensortierung für OpenStreetMap-Karten

Beim Erzeugen einer Contraction Hierarchy müssen die Knoten des zugrunde liegenden Graphen aufsteigend nach ihrer Wichtigkeit sortiert kontrahiert werden[38]. Mit den meisten in Abschnitt 5.3 vorgestellten Sortierkriterien kann es aber immer wieder passieren, dass zwei verschiedene Knoten eine gleich große Wichtigkeit besitzen. Die zugrunde liegende Implementierung löste eine solche Mehrdeutigkeit zunächst, indem zwei Knoten bei identischer Wichtigkeit gemäß ihrer Knoten-Id unterschieden wurden. Dies führte jedoch bei Sortierungen mit besonders vielen Mehrdeutigkeiten, etwa bei ausschließlicher Sortierung nach der *Pfeildifferenz*, und eines zugrunde liegenden OSM-Datensatzes zu auffälligen Ergebnissen.

Korrelation zwischen Knoten-Ids und Straßenkategorie

Denn bei OSM sind die Knoten-Ids keineswegs zufällig verteilt, sondern werden chronologisch mit dem „Alter“, d. h. dem Zeitpunkt der erstmaligen Erzeugung, eines Knotens automatisch und aufsteigend vergeben. Gleichzeitig wachsen OSM-Karten zunächst entlang der Autobahnen und Hauptstraßen. Schließlich dienen diese nicht zuletzt auch als Orientierungspunkt, und auch ortsfremde Benutzer fühlen sich in der Lage sie einzutragen. Erst nach und nach werden kleinere Straßen in höheren Detailgraden eingepflegt. Somit wird gewissermaßen die Korrelation zwischen Knoten-Alter und -Id auch an die Straßenkategorien weitergegeben: Die Wahrscheinlichkeit, dass ein Knoten mit verhältnismäßig kleiner Id auf einer Autobahn oder Straße vergleichbaren Ranges liegt, ist ungleich höher als die, dass dieser Knoten in einem Wohngebiet liegt.

[35] siehe Seite 137, Stichwort „Beobachtungen für Shortcut-Pfeile“
[36] siehe Seite 161, Stichwort „simulierte Knotenkontraktion“
[37] also auch bei Shortcut-Schlingen, die allerdings erst in Kapitel 6, S. 199 ff., relevant werden
[38] siehe Verfahren 10, S. 135, sowie Abschnitt 5.3

Abbildung 5.17 verdeutlicht diesen Zusammenhang graphisch. Dargestellt ist der Anteil der Knoten, die zu einem OSM-Highway der Kategorie „Motorway“ oder „Trunk“ gehört – sinngemäß also Autobahn oder Autobahnzubringer[39] – für verschiedene Anteile der Knoten mit geringsten Id. Werden nur die ersten 10 % der Knoten mit kleinsten Ids betrachtet, gehören von diesen fast 17 % zu einer Autobahn oder einem Autobahnzubringer. Vom ersten Prozent der Knoten mit kleinsten Ids gehören sogar fast 33 % zu diesen Straßenkategorien.

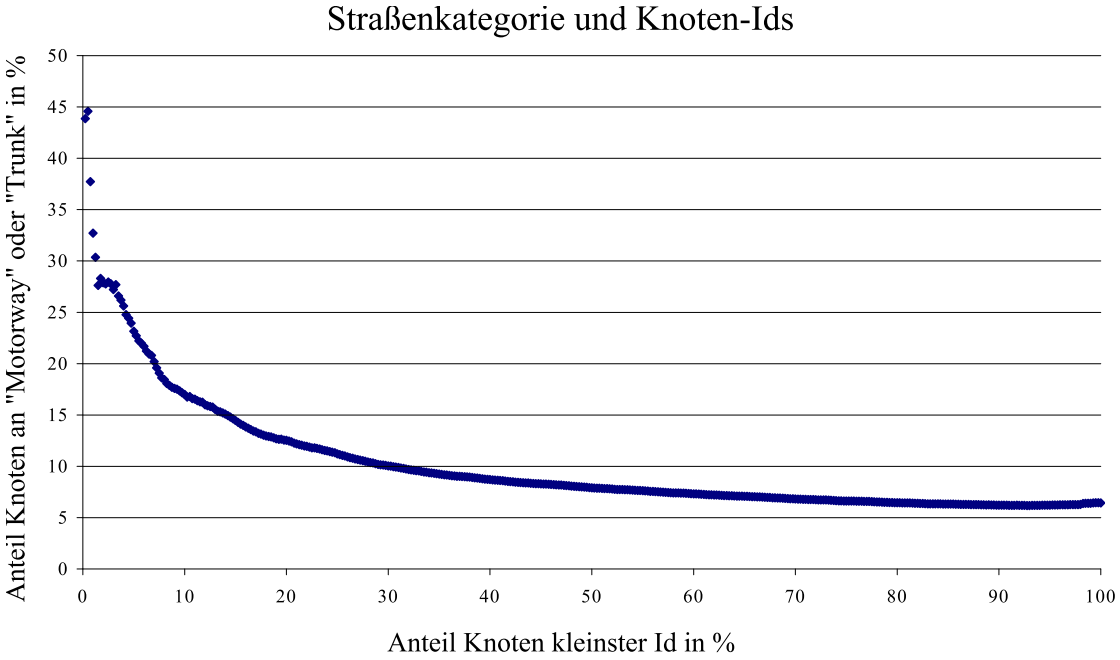


Abbildung 5.17: Zusammenhang zwischen Knoten-Id und Straßenkategorie bei OSM Datengrundlage: Niedersachsenkarte, die bis inklusive Schritt 6 importiert wurde (siehe Abbildung 3.1, S. 61, bzw. Abschnitt 3.6, S. 78 ff.)

Werden bei der Erzeugung einer Contraction Hierarchy zwei Knoten gleicher Wichtigkeit miteinander verglichen, und die Unterscheidung erfolgt dann auf Grundlage der Knoten-Id, führt dies teils zu einer ungewollten Sortierung nach der Straßenkategorie. Von den hier vorgestellten Sortierkriterien neutralisiert einzig die Verwendung des *zufälligen Bias* (*RAND*)[40] diesen ungewünschten Effekt.

5.5.3 Möglichkeiten der Parallelisierung bei der Erzeugung einer Contraction Hierarchy

Moderne Computer sind immer häufiger mit mehr als einem Rechenkern ausgestattet, was die Möglichkeit eröffnet, manche Arbeitsschritte parallel ausführen zu lassen. Für die Erzeugung von Contraction Hierarchies untersucht Vetter [2009] hierzu zwei Ansätze:

[39] siehe Listing 3.1, S. 62, zur Auflistung aller verwendeten Highwaykategorien bzw. zum Verständnis des OSM-Datenmodells auch Abschnitt 2.2.1, S. 41 ff.
[40] siehe Seite 164

1. eine parallele Knotenkontraktion[41] und als Teilaufgabe davon
2. eine parallele Überprüfung, ob bei einer Knotenkontraktion ein Shortcut-Pfeil erzeugt werden muss[42].

Nach Vorstellung dieser beiden Ansätze wird ein dritter Ansatz präsentiert, der sich beim Einsatz im Rahmen dieser Arbeit bewährt hat.

Parallele Knotenkontraktion Bei der parallelen Knotenkontraktion muss besonders darauf geachtet werden, dass die Kontraktion eines Knotens v bei fast allen hier vorgestellten Sortierkriterien[43] in der Folge die Wichtigkeit eines anderen Knotens v' in seiner Umgebung beeinflusst. Werden v und v' gleichzeitig kontrahiert, kann dies daher zu Konflikten bei der Knotensortierung führen. Vetter [2009, S. 11 ff.] umgeht dies, indem er nur Knoten aus *unabhängigen Mengen*[44] parallel kontrahiert. Hierzu verwendet er die *n-Nachbarschaft* $\mathcal{N}'_n(v)$, welche bereits im Rahmen des Highway-Hierarchies-Verfahren in Abschnitt 4.2.1, S. 112 ff., erwähnt wurde. Die unabhängige Menge $\bar{V}$ der Knoten, die parallel kontrahiert werden können, ist definiert als

$$\bar{V} = \left\{ v \in V_i \mid \operatorname{rang}(v) < \min\left\{ \operatorname{rang}(v') \mid v' \in \mathcal{N}'_n(v) \right\} \right\}. \tag{5.15}$$

Dabei beschreibt V_i gemäß Gleichung (5.1), S. 133, während der Hierarchie-Erzeugung die Menge derjenigen Knoten, die noch nicht kontrahiert wurden.

Eine Parallelisierung ist allerdings nur so lange möglich, wie die verwendeten Knotensortierungskriterien[45] stets allenfalls Einfluss auf die direkte Nachbarschaft *eines* kontrahierten Knotens ausüben und nicht darüber hinaus. Ansonsten trifft die Grundannahme der *parallelen Knotenkontraktion* nicht mehr zu.

Parallele Prüfung der Notwendigkeit neuer Shortcut-Pfeile Wenn bei der tatsächlichen oder simulierten Kontraktion eines Knotens überprüft wird, ob Shortcut-Pfeile infolge der Kontraktion neu erzeugt werden müssen bzw. müssten, finden keine Änderungen am, sondern ausschließlich Wegsuchen innerhalb des zugrunde liegenden Graphen statt. Vetter [2009, S. 11] merkt hier an, dass viele Knoten bei Graphen, die auf Straßenkarten beruhen, nur wenige Nachbarn besitzen. Daher sind die Wegsuchen bei der Prüfung der Notwendigkeit neuer Shortcut-Pfeile im Wesentlichen sehr lokale und damit kurze Suchen über wenige Pfeile. Vetter [2009, S. 11] erwartet daher bei einer Parallelisierung an dieser Stelle nur wenig Beschleunigungspotential, da die parallelen Arbeitsschritte zu kurz sind. Die Zeitersparnis ginge beim Zusammenführen der Ergebnisse wieder verloren.

Wie in Abschnitt 5.4 erläutert und Abbildung 5.15, S. 166, dargestellt wurde, wächst der durchschnittliche Knotengrad gegen Ende der Hierarchie-Erzeugung jedoch exponentiell. In dieser Phase greift die Argumentation gegen die Parallelisierung deshalb nicht mehr. Nach eigenen Tests – basierend auf der hier vorliegenden Implementierung – kann zumindest während der Kontraktion des letzten Prozents der Knoten eine entsprechende Parallelisierung eine Beschleunigung von etwa 20 % bewirken.

[41] siehe Zeilen 3–18 in Verfahren 10, S. 135
[42] siehe Zeilen 6–15 in Verfahren 10, S. 135
[43] Die Ausnahme bilden hier nur der zufällige Bias (*RAND*) und die Abbiegebeschränkungs-Flags (*AB*).
[44] im englischen Original: *Independent Node Sets*
[45] siehe Abschnitt 5.3, S. 158 ff.

Parallele simulierte Knotenkontraktion Ein dritter und verhältnismäßig leicht zu verwirklichender Ansatz für die Parallelisierung soll im Folgenden vorgestellt werden: die parallele *simulierte* Knotenkontraktion. Sie besitzt die Vorzüge der oben beschriebenen parallelen Knotenkontraktion und vermeidet gleichzeitig deren Konflikt einer wechselseitigen Beeinflussung der Arbeitsschritte.

Wie in Abschnitt 5.3 beschrieben[46], ist es empfehlenswert, nach der Kontraktion eines Knotens die Wichtigkeit seiner Nachbarknoten neu zu berechnen. Dies geschieht, indem für die Nachbarknoten eine Kontraktion simuliert wird. Da in diesem Schritt keine Veränderung am zugrunde liegenden Graphen stattfindet, eignet er sich sehr für die Parallelisierung. Nachdem alle simulierten Knotenkontraktionen fertig berechnet sind, endet die Nebenläufigkeit, die neuen Wichtigkeits-Werte werden den Knoten zugewiesen und die Knoten auf dieser Grundlage neu in die Prioritätswarteschlange eingefügt.

Da eine simulierte Kontraktion mehrere Berechnungen umfasst, ist der Effekt der Parallelisierung deutlicher als bei der parallelen Prüfung der Notwendigkeit neuer Shortcut-Pfeile. Besonders während der Endphase der Hierarchie-Erzeugung, wenn der durchschnittliche Knotengrad rapide ansteigt[47], lassen sich auf diese Weise Laufzeiteinsparungen erzielen. Allerdings skaliert das Verfahren in der vorliegenden Implementierung nur bis zu einem gewissen Grad, wie Abbildung 5.18 verdeutlicht.

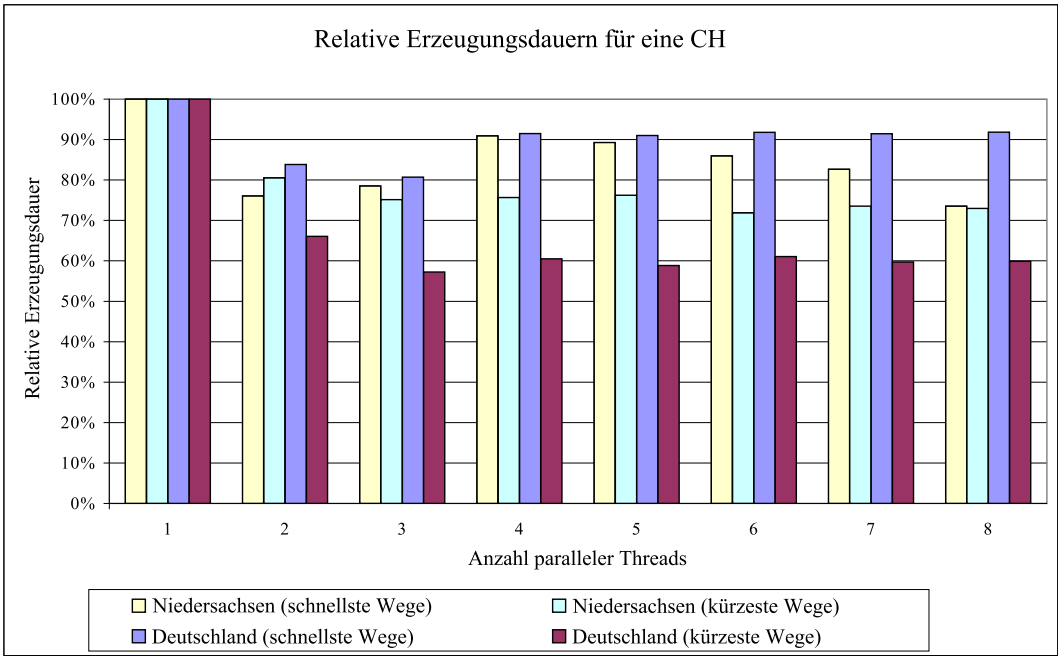


Abbildung 5.18: Parallele CH-Erzeugung
Die Parallelisierung skaliert nicht gut, weil die gleichzeitig bearbeiteten Teilaufgaben klein sind.

Der größere Effekt wird bei der Berechnung einer Contraction Hierarchy für *kürzeste* Wege erzielt. Dies liegt daran, dass auf einer Straßenkarte im Allgemeinen für kürzeste

[46] Stichwort „Update aller Nachbarn“ auf Seite 160
[47] siehe hierzu besonders Abbildung 5.15, S. 166

Wege alle Pfeile eine ähnliche Bedeutung besitzen. Bei der Suche nach *schnellsten* Wegen werden dagegen Autobahnen erheblich bevorzugt. Dies führt bei der CH-Erzeugung für kürzeste Wege zu deutlich mehr annähernd gleich kurzen Alternativen, die im Rahmen der Knotenkontraktion berücksichtigt werden müssen.[48]

5.5.4 Abschätzung der Fahrtdauer entlang eines Pfeils

Die vorliegende Implementierung versucht auf der einen Seite, eine möglichst hohe Flexibilität zu gewährleisten, damit nicht für jede Metrik eine neue Hierarchie erzeugt werden muss. Auf der anderen Seite versucht sie sparsam im Speicherverbrauch zu sein. Bei großen Straßenkarten, wie der von Europa, ist dies besonders relevant.

Die einfachste Möglichkeit, die Fahrtdauer oder -geschwindigkeit entlang eines Pfeils zu speichern, läge offensichtlich darin, diesen Wert, etwa in Minuten, beim in der Implementierung zugrunde liegenden Kantenobjekt zu hinterlegen. Offensichtlich hätte dies jedoch nur für ein spezielles Fahrzeug, bzw. eine Fahrzeugkategorie wie Pkw oder Lkw, Gültigkeit. Eine solche Karte lieferte für andere Fahrzeugkategorien keine sinnvollen Ergebnisse.

Die Erweiterung um eine Angabe der Fahrtdauer oder -geschwindigkeit pro Fahrzeugkategorie wäre ebenfalls eine ungünstige Modellierung. Denn erstens vergrößert sich hierdurch der Speicherbedarf der Karte. Zweitens ist dieses Vorgehen unflexibel. Sollte einmal eine neue Fahrzeugkategorie hinzukommen, müsste das Datenmodell der Karte um diese erweitert und die Hierarchie von Neuem berechnet werden.

Erweiterte Kantenkategorien

Die vorgeschlagene Lösung des Problems besteht darin, pro Kantenobjekt lediglich eine *erweiterte Kantenkategorie* zu hinterlegen. In dieser ist die Straßenkategorie, das Land, in dem der Pfeil liegt, sowie die explizit angegebene Höchstgeschwindigkeit – sofern vorhanden – zusammengefasst. Die Kantenkategorien wurden ausführlicher in Abschnitt 3.4 ab Seite 73 erläutert.

Auf diese Weise wird der Speicherbedarf gering gehalten, da es erheblich weniger Kantenkategorien als Kantenobjekte in den Karten gibt. Darüber hinaus muss im Falle einer neuen Fahrzeugkategorie für diese lediglich die Fahrtgeschwindigkeit je Kantenkategorie festgelegt werden.

Heuristischer Ansatz für verschiedene Fahrzeugkategorien

Für die Verwendung einer Contraction Hierarchy mit verschiedenen Fahrzeugtypen ergibt sich dann ein Kompromiss. Auf Hierarchien, die mithilfe der Schnellste-Wege-Metrik auf Grundlage einer Fahrzeugkategorie f_1 berechnet wurden, lassen sich nicht in jedem Fall auch sämtliche schnellsten Wege für einen anderen Fahrzeugtyp f_2 ermitteln. Für f_2 können vielmehr nur *heuristische* Lösungswege gefunden werden. Diese weichen jedoch erwartungsgemäß – wenn überhaupt – nur wenig von der Optimallösung ab. Denn im Allgemeinen beinhaltet das Schnellste-Wege-Netz einer Straßenkarte beispielsweise alle Autobahnen und Schnellstraßen. Zusätzlich gilt für die meisten Fahrzeugkategorien, dass die Reihenfolge der für sie nach Höchstgeschwindigkeit sortierten Kantenkategorien weitgehend übereinstimmt: Sowohl Pkw, Kleintransporter, Lkw als auch Sattelzüge fahren ihre jeweilige Höchstgeschwindigkeit auf Autobahnen und nur annähernd 30 km/h in entsprechend ausgeschilderten Zonen.

[48] siehe hierzu auch Abschnitt 5.4 sowie insbesondere Abbildung 5.15, S. 166

Erzeugung einer exakten Contraction Hierarchy für mehrere Fahrzeugkategorien Im Folgenden wird dargestellt, wie eine Contraction Hierarchy erzeugt werden kann, auf der *sämtliche* schnellsten Wege für n Fahrzeugkategorien $FK = f_1, \ldots, f_n$ gefunden werden können. Das Verfahren besteht aus drei Schritten[49]:

1. Die Berechnung einer vollständigen Hierarchie $G_{\text{ch}}^{f_1} = (V_0, E_{\text{ch}}^{f_1}, c_{\text{ch}}^{f_1}, e_1^{f_1}, e_2^{f_1})$ auf Basis eines nicht-negativ bewerteten, gerichteten Graphen $G_{\text{ch}} = (V_0, E_0, c_0)$ für eine der Fahrzeugkategorien. Dies sei o. B. d. A. f_1. Hierbei entsteht explizit eine Knotensortierung nach Rängen.

2. Die Berechnung einer vollständigen Hierarchie für jede weitere Fahrzeugkategorie $f_2, \ldots, f_n$, wobei die Knotensortierung aus dem ersten Schritt exakt übernommen werden muss. Es entstehen die Hierarchien $G_{\text{ch}}^{f_2} = (V_0, E_{\text{ch}}^{f_2}, c_{\text{ch}}^{f_2}, e_1^{f_2}, e_2^{f_2})$, …, $G_{\text{ch}}^{f_n} = (V_0, E_{\text{ch}}^{f_n}, c_{\text{ch}}^{f_n}, e_1^{f_n}, e_2^{f_n})$.

3. Die Vereinigung von $G_{\text{ch}}^{f_1}$ mit $G_{\text{ch}}^{f_2}, \ldots, G_{\text{ch}}^{f_n}$ zu einer Ergebnis-Hierarchie $G_{\text{ch}}^{*} = (V_0, E_{\text{ch}}^{*}, c_{\text{ch}}^{*}, e_1^{*}, e_2^{*})$, in der die Shortcut-Pfeile aller n Hierarchien vorhanden sind.

   Die Kostenfunktion c_{ch}^{*} benötigt als zusätzlichen Parameter eine Fahrzeugkategorie und es gilt[50]:

$$E_{\text{ch}}^{*} = \bigcup_{i=1}^{|FK|} E_{\text{ch}}^{f_i}, \tag{5.16}$$

$$c_{\text{ch}}^{*} = FK \times E_{\text{ch}}^{*} \to \mathbb{R} \text{ mit}$$

$$c_{\text{ch}}^{*}(f_k, (u,v)) = \begin{cases} c_{\text{ch}}^{f_k}(u,v), & \text{falls } (u,v) \in E_{\text{ch}}^{f_k} \\ \sum\limits_{(u' \to v') \in (u \ldots v)} c_{\text{ch}}^{f_k}(u' \to v'), & \text{sonst} \end{cases} \quad \text{für } f_k \in FK,$$

$$e_1^{*} : E_{\text{ch}}^{*} \to E_{\text{ch}}^{*} \text{ mit}$$

$$e_1^{*}((u,v)) = e_1^{f_k}((u,v)) \text{ und } (u,v) \in E_{\text{ch}}^{f_k} \quad \text{für } f_k \in FK,$$

$$e_2^{*} : E_{\text{ch}}^{*} \to E_{\text{ch}}^{*} \text{ mit}$$

$$e_2^{*}((u,v)) = e_2^{f_k}((u,v)) \text{ und } (u,v) \in E_{\text{ch}}^{f_k} \quad \text{für } f_k \in FK.$$

Ein Nachteil dieses Vorgehens besteht in der starren Vorgabe der Knotensortierung bei der Berechnung der einzelnen Hierarchien. Hierdurch entstehen erwartungsgemäß sehr viele zusätzliche Shortcut-Pfeile. Aus Speicherplatzgründen ist den Hierarchien, die in dieser Arbeit anhand der Schnellste-Wege-Metrik berechnet wurden, daher ausschließlich der Fahrzeugtyp „Transporter“ zugrunde gelegt.

hier: ausschließlich CH auf Basis von „Transporter“

[49] Für Details und die Notation siehe auch Abschnitt 5.1, S. 132 ff.

[50] zur Erinnerung: $(u' \to v')$ steht für den Original-Pfeil von Knoten u' nach v'; zur Notation von Shortcut-Pfeilen siehe auch Seite 110

5.5.5 Knotensortierung während der Wegsuche

Einen wesentlichen Einfluss auf die Suchdauer besitzt die Sortierung der Knoten aus den Horizontmengen[51]. Dieser Abschnitt hebt vier zweckmäßige Datenstrukturen hervor, mit deren Hilfe diese Mengen in einer Sortierung vorgehalten werden können.

Die in dieser Arbeit verwendete Programmiersprache Java[52] bietet viele Möglichkeiten einer Implementierung der Horizontmenge. Besonders geeignet erscheint dabei die Verwendung einer stets sortierten Liste bzw. Menge. In Java kommen hier unter anderem die Klassen `java.util.PriorityQueue` und `java.util.TreeSet` häufig zum Einsatz. Diese beiden unterscheiden sich konzeptionell erheblich voneinander.

`Priority Queue`

Eine `Priority Queue` oder auch *Prioritätswarteschlange* verwaltet ein Objekt-Array der Länge n, in dem eine Sortierung der Elemente erfolgt, so dass eine virtuelle Heap-Struktur entsteht.[53] Die Dokumentation der Klasse beschreibt den Aufwand für die Entferne- und Einfüge-Funktionen des kleinsten Elements mit $\mathcal{O}(\log(n))$ und den Aufwand für das gezielte Entfernen eines Elements mit $\mathcal{O}(n)$.[54]

`Tree Set`

Ein `Tree Set` implementiert eine zuerst von Bayer [1972] beschriebene Variante des balancierten Binärbaums mit n Elementen. Der Aufwand für das Entfernen, Einfügen und Suchen von Elementen ist jeweils $\mathcal{O}(\log(n))$[55]. Im Gegensatz zur oben genannten Implementierung der `PriorityQueue` liegt einem `TreeSet` kein Daten-Array zugrunde. Es erzeugt stattdessen für jedes zu speichernde Element ein eigenes Knotenobjekt. Dies hat einen tendenziell verlangsamenden Effekt auf die Gesamtperformance, da ein konstanter zeitlicher Overhead für das Erzeugen eines solchen Knotenobjekts bei jeder Einfüge-Operation veranschlagt werden muss.

`Skip-Liste`

Eine interessante und verhältnismäßig einfach zu implementierende Datenstruktur ist darüber hinaus die zuerst von Pugh [1990] beschriebene *Skip-Liste*. Sie erweitert das Konzept einer einfachen verketteten Liste um mehrere *Levels* von Verkettungen. Als Ergebnis kann ein Listenelement nicht nur auf das direkt folgende Element verweisen, sondern – ähnlich dem Konzept der Shortcut-Pfeile des hier vorgestellten Wegsucheverfahrens – durch Verwendung einer Verkettung eines höheren Levels auch auf weiter hinten liegende Elemente. Im Gegensatz zu (Binär-)Bäumen oder Heaps entfällt beim Einsatz einer Skip-Liste die Neu-Sortierung eines Teils ihrer Elemente beim Einfügen neuer Daten. Abbildung 5.19, S. 175, zeigt den schematischen Aufbau von Skip-Listen mit einem (a) bis vier Level (d).

[51] siehe Zeilen 11 und 23 in Verfahren 11, S. 138

[52] in der API-Version 1.6

[53] Aus dem von Oracle veröffentlichten Quellcode von `java.util.PriorityQueue`: "Priority queue represented as a balanced binary heap: the two children of *queue*[n] are *queue*[$2\times n+1$] and *queue*[$2\times(n+1)$]. (...) For each node n in the heap and each descendant d of n, $n \leq d$. The element with the lowest value is in *queue*[0], assuming the queue is nonempty."

[54] Siehe Javadoc der Klasse `java.util.PriorityQueue`: "Implementation note: this implementation provides $\mathcal{O}(\log(n))$ time for the enqueing and dequeing methods (`offer`, `poll`, `remove()`[sic] and `add`); linear time for the `remove(Object)` and `contains(Object)` methods; and constant time for the retrieval methods (`peek`, `element`, and `size`)."

[55] Aus dem Javadoc der Klasse `java.util.TreeMap`, die ihrerseits die Grundlage für `java.util.TreeSet` bildet: "A Red-Black tree based [...] implementation. [...] This implementation provides guaranteed $\log(n)$ time cost for the containsKey, get, put and remove operations."

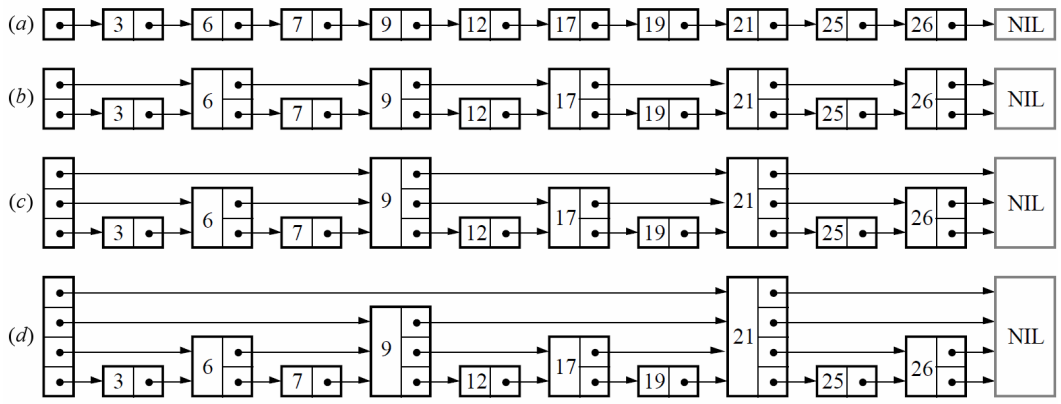


Abbildung 5.19: Schematische Darstellung von Skip-Listen mit unterschiedlich vielen Levels bei identischem Inhalt
Eine einfache verkettete Liste ist (a); (d) besitzt insgesamt vier Levels.
Quelle: Pugh [1990, S. 669]

Eine Stellgröße für Skip-Listen ist die zu definierende Maximal-Anzahl ihrer Level. Beim Einfügen eines neuen Elements in die Liste wird diesem ein zufälliges Level zugewiesen. Für das Suchen oder sortierte Einfügen eines Elements werden die Level der Skip-Liste in absteigender Reihenfolge vom Startelement aus betrachtet: Sobald das nächste Element auf einem bestimmten Level größer ist als das gesuchte, wechselt die Suche ins nächst-niedrigere Level und sucht dort weiter. Auf diese Weise wächst der *durchschnittliche* Suchaufwand nicht linear, sondern nur logarithmisch mit der Menge der Elemente in einer Skip-Liste.

In der Landau-Notation für den Aufwand für das Einfügen oder Entfernen eines Elements ändert dies jedoch im Vergleich mit einer normalen verketteten Liste nichts. Denn noch immer können Fälle auftreten, in denen die Skip-Liste nahezu ausschließlich aus Elementen mit nur einem Level besteht. Der Aufwand für diese Operationen ist daher $\mathcal{O}(n)$. Andererseits ist der Aufwand für das Entfernen des kleinsten Elements einer sortierten Skip-Liste gänzlich unabhängig von der Menge der Elemente und damit $\mathcal{O}(1)$.

`Fibonacci Heap`

Die vierte und letzte hier untersuchte Datenstruktur ist der so genannte *Fibonacci Heap*. Dieser wurde in Abschnitt 4.1.1.1 auf Seite 91 bereits kurz erwähnt: Mithilfe von Fibonacci Heaps kann das Dijkstra-Verfahren – bei dünn besetzten Graphen $G = (V, E, c)$[56] – einen durchschnittlichen Aufwand von $\mathcal{O}(|V| \log |V| + |E|)$ erreichen. Seine Eigenschaften als Heap klassifizieren ihn ansonsten wie die oben beschriebene `Priority Queue`.

Da die Landau-Notation nur ein *asymptotisches* Laufzeit-Verhalten angibt, kann nicht a priori entschieden werden, welche Datenstruktur zu bevorzugen ist.[57] Tatsächlich kommt es entscheidend auf die Größe des Graphen an, in dem eine Dijkstra-Suche stattfindet; genauer gesagt: auf die erwartete Anzahl der Elemente in der Horizontmenge. Die Benchmarks in Abschnitt 5.6.3, S. 181 ff., umfassen daher auch Implementierungen mit allen vier hier genannten Datenstrukturen.

[56] $|V| \ll |E| \ll |V|^2$, siehe Seite 91
[57] siehe auch Abschnitt 1.3.4, S. 33

5.5.6 Indizierung der Knoten während der Wegsuche

Bei jeder Iteration einer Dijkstra-Wegsuche muss nicht nur entschieden werden, welcher Knoten als nächstes expandiert werden soll, sondern bei der Expansion muss auch effizient feststellbar sein, ob ein Knoten bereits Teil der Horizont- oder Lösungsmenge ist[58]. Die im vorangegangenen Abschnitt vorgestellten Datenstrukturen zur Sortierung der Knoten können für die Beantwortung dieser Frage herangezogen werden, doch wäre dies nicht effizient. Ein zweiter Index über die besuchten Knoten der Wegsuche ist stattdessen sinnvoll.

Index-freie Modellierung

Ein nahe liegender und sehr schneller Ansatz ist es, diese Information direkt als Knoten-Attribut zu modellieren. So können bei jedem Knoten zwei zusätzliche boolesche Werte – so genannte *Flags* – gespeichert werden, um diese Informationen darzustellen. Während der in dieser Arbeit verwendete Pseudocode diese Darstellung der einfachen Lesbarkeit halber verwendet, besitzt der Ansatz jedoch zwei gravierende Nachteile:

Speicher- und Effizienzprobleme Die meisten Wegsuchen betreffen nur einen kleinen Bruchteil aller Knoten. Um aber auf den Index zu verzichten müssten an jedem Knoten des Graphen die zusätzlichen Informationen für Vorwärts- und Rückwärtssuchen angelegt werden. Dies würde den benötigten Speicher dauerhaft um ein konstantes Vielfaches der Knotenanzahl vergrößern.

Keine Parallelisierbarkeit Liegen die zusätzlichen Such-Informationen an jedem Knoten vor, so sind sie nur für genau eine Suche gültig, nämlich für die zwischen dem aktuellen Start- und dem aktuellen Zielknoten. Eine Parallelisierung unterschiedlicher Wegsuchen ist damit ausgeschlossen.

Temporäre, erweiterte Knotenobjekte

Der Einsatz temporärer (Knoten-)Objekte, welche die Knoten des Graphen um die für die Suche nötigen Attribute erweitern, behebt beide Nachteile. Allerdings entsteht hierdurch Bedarf für einen zusätzlichen Index über ebendiese Objekte.

Index per Hashtabelle

Ein einfacher Index für diesen Zweck ist eine Hashtabelle, in der als Schlüssel wahlweise die Knoten-Id oder auch der Knotenrang verwendet wird; beide sind eindeutig. Basierend auf einem Datenarray wird hierbei die Position, an der ein temporäres Knotenobjekt gespeichert wird, mithilfe einer Hash-Funktion berechnet, die als Eingabe den Hash-Schlüssel entgegennimmt. Auf diese Weise können zwar Kollisionen, also identische Positions-Ergebnisse bei unterschiedlichen Hash-Schlüsseln, entstehen. Doch diese sind bei geeigneter Wahl der Hash-Schlüssel und -Funktion – je nach Größe des zugrunde liegenden Datenarrays – erstens selten und zweitens auf mehrere Arten behandelbar, indem die Daten im Array beispielsweise zusätzlich als verkettete Liste implementiert sind, so dass implizit mehrere Objekte im selben Array-Index abgelegt werden können.

Dies lässt nach der Landau-Notation die Frage nach dem Vorhandensein eines Knoten in der Horizont- oder Lösungsmenge durchschnittlich mit einem Aufwand von $\mathcal{O}(1)$ beantworten (vgl. [Cormen u. a., 2001, S. 234]). Allerdings muss jedes Mal[59] das Ergebnis der Hash-Funktion neu berechnet und innerhalb der Hashtabelle auf Kollisionen geprüft werden.

Index per Ränge-Array

Die eindeutigen Knotenränge bei Contraction Hierarchies ermöglichen einen zweiten

[58] siehe Zeilen 9 und 15 in Verfahren 5, S. 91

[59] sofern es für diesen Zweck nicht extra gespeichert wird

Ansatz für die Indizierung: ein einfaches Array, das exakt so lang ist, dass es jeden Knoten der Karte aufnehmen kann. Sei *minRang* der kleinste Knotenrang des Graphen und *maxRang* der höchste. Der Array-Index eines Knotens kann dann einfach aus der Differenz seines Knotenrangs zu *minRang* errechnet werden. Anders als bei einer Hashtabelle können so nie Kollisionen auftreten. Ein Nachteil entsteht dadurch, dass das Array groß genug sein muss, um alle Knoten des zu untersuchenden Graphen aufzunehmen. Bei großen Graphen kann dies offensichtlich zu Speicherproblemen führen.

Die Eigenschaft der Wegsuche innerhalb einer Contraction Hierarchy, dass stets nur in nach Knotenrang aufsteigender Richtung gesucht wird, kann allerdings ausgenutzt werden, um Speicher zu sparen. Denn bei jeder s-t-Wegsuche steht sofort fest, dass die Vorwärts- bzw. Rückwärtssuche nur höchstens $(maxRang - s.rang)$ bzw. $(maxRang - t.rang)$ Knoten erreichen kann, so dass der Rang von s bzw. t gewissermaßen als *Offset* bei der Index-Zuweisung für einen Knoten im oben beschriebenen Array verwendet werden kann. Im Mittel halbiert dies offensichtlich den Speicherbedarf für das Index-Array. Doch sobald der Start- oder Zielknoten einen sehr geringen Rang besitzen, entfällt die Ersparnis.

Die zu diesen Überlegungen gehörigen Benchmarks befinden sich in Abschnitt 5.6.4, S. 183 ff. Dort zeigt sich kein Performance-Gewinn durch die Verwendung von Index-Arrays. Offenbar konterkariert das Erzeugen der Index-Arrays mit teils mehreren Millionen Einträgen die Zeitersparnis. Daher setzen alle im Folgenden analysierten Wegsuchen-Implementierungen dieser Arbeit auf eine Hashtabelle als Knotenindex.

Gemeinsame Datenhaltung für Horizont- und Lösungsmenge Zwar wird im Pseudocode stets zwischen der Horizont- und der Lösungsmenge unterschieden. Eine effiziente Implementierung, die einen zusätzlichen Index verwendet, wie er hier beschrieben wurde, kann jedoch darauf verzichten, die Lösungsmenge gesondert zu speichern. Dazu reicht es aus, die temporären Knotenobjekte bei der Wegsuche um ein weiteres Attribut zu erweitern, aus dem hervorgeht, ob der entsprechende Knoten bereits Teil der Lösungsmenge ist; beispielsweise ein Flag namens `inLösung`. Dann muss ein Knoten, nachdem er expandiert wurde, zwar immer noch aus der Horizontmenge entfernt werden. Das Einfügen in eine Lösungsmenge entfällt aber, indem `inLösung` auf `wahr` gesetzt und das Knotenobjekt explizit *nicht* aus dem Index gelöscht wird. So bleiben alle Informationen über den Knoten vorhanden und es kann – dank des Indexes – effizient ermittelt werden, ob ein Knoten bereits Teil der Horizont- oder Lösungsmenge ist.

5.6 Benchmarks für Contraction Hierarchies

In diesem Abschnitt werden die untersuchten Varianten der Wegsuche innerhalb einer Contraction Hierarchy einander gegenüber gestellt und verglichen. Ein Fazit zu den Messungen wird in Abschnitt 5.6.8, S. 197, präsentiert.

Der Aufbau des Benchmarks orientiert sich an dem von Sanders u. Schultes [2005, S. 577 f.] beschriebenen Vorgehen. Die Grundüberlegung ist hier, dass Wegsuchen zwischen rein zufällig ausgewählten Start- und Zielknoten allein als Benchmark eine verhältnismäßig geringe Aussagekraft besitzen, da zusätzliche Informationen beispielsweise über die Weglänge notwendig sind.

Dijkstra-Rang

Daher gruppiert der Benchmark zufällige Wegsuchen gemäß ihrer *Dijkstra-Ränge*[60]. Der Dijkstra-Rang eines Weges p korreliert zwar positiv mit der Länge von p, aber eine konkrete Aussage über die Weglänge lässt sich nicht extrahieren, wenn keine Informationen bezüglich der Kantenlängen des zugrunde liegenden Graphen vorliegen.

5.6.1 Aufbau der Benchmarks

Für die nachfolgenden Benchmarks wurden pro zugrunde liegender Karte 1 000 zufällige Startknoten ausgewählt und Standard-Dijkstra-Suchen ausgehend von diesen Knoten begonnen. Bei jeder 2^n-ten Iteration mit $n \in \mathbb{N}$ wurde derjenige Knoten, welcher zuletzt in die Lösungsmenge aufgenommen wurde, als Zielknoten für die Benchmarks gespeichert. Dadurch können später gezielt Wegsuchen mit vorgegebenem Dijkstra-Rang – nämlich zwischen dem Startknoten und den gespeicherten Zielknoten – wiederholt werden.

Jede untersuchte Dijkstra-Variante absolvierte sämtliche 1 000 gespeicherten Wegsuchen für alle verschiedenen Dijkstra-Ränge. Dadurch entstanden pro Dijkstra-Rang und Wegsucheverfahren 1 000 Ergebnisse. Die Visualisierung erfolgt aggregiert mithilfe eines Boxplot-Diagramms, auf dem neben dem Median der Messwerte auch das so genannte *untere* und *obere Quartil* ablesbar sind.[61] Üblicherweise wird die Darstellung der namensgebende Box um diese Quartile noch um *Antennen* ergänzt. Dies sind zwei Linien an gegenüber liegenden Seiten einer Box, die in den Bereich der ersten und des vierten Quartils der Messwerte hinein ragen. Nach Tukey [1977] besitzen die Antennen auf beiden Seiten der Box eine Länge, die höchstens dem 1,5-fachen des Interquartilsabstands, d. h. des Abstands zwischen den Quartilen, entspricht. Eine Antenne endet dabei exakt auf dem letzten Messwert, der noch im Interquartilsabstand liegt. Werte außerhalb der Antennen können als Ausreißer interpretiert und wiederum gesondert dargestellt werden. Anhand der Größen sowohl der Box als auch der Antennen lässt sich die Verteilung der Messwerte auf einen Blick abschätzen.

Boxplot-Diagramme

In diesem Abschnitt wurde zugunsten der Übersichtlichkeit auf die Darstellung der Antennen und Ausreißer-Werte verzichtet. Die vollständigen Abbildungen befinden sich jedoch in Anhang B, S. 319 ff. Dort sind die Antennen der Boxplots so gewählt, dass sie zusammen mit der Box diejenigen 99 % aller Messwerte umfassen, die am dichtesten beim Median liegen. Das übrige Prozent ist als Ausreißer dargestellt.

Die x-Achse der Diagramme verläuft entlang der Dijkstra-Ränge in der Form 2^n mit $n \in \mathbb{N}$. Bei der Interpretation der Ergebnisse darf daher nicht vergessen werden, dass diese Achse logarithmiert dargestellt ist.

Verwendete Contraction Hierarchies Die in diesem Abschnitt vorgestellten Benchmarks basieren auf den Straßenkarten von Deutschland und Europa. Die Details zu ihrer Herkunft sind ausführlich in Kapitel 3, S. 61 ff., beschrieben[62].

Auf Grundlage beider Straßenkarten wurden je zwei Contraction Hierarchies erstellt: eine für die Suche nach kürzesten Wegen und eine für schnellste Wege. Die Schnellste-

[60] siehe Abschnitt 4.1.1.1 auf Seite 94 zur Definition

[61] Bei einer eindimensionalen Datenreihe, wie den hier vorliegenden Messungen, umfasst das erste Quartil den Wertebereich, in dem das kleinste Viertel aller Daten liegt, das zweite Quartil den Wertebereich des nächsten Viertels usw. Der Median der Werte liegt daher zwischen dem zweiten und dritten Quartil. Diese beiden werden auch als *unteres* und *oberes* Quartil bezeichnet.

[62] siehe insbesondere Tabelle 3.6, S. 85

Wege-Metrik basierte auf der Fahrzeugkategorie „Transporter“.[63] In beiden Fällen erfolgte die Knotensortierung nach dem Schema $2R + 8EQ + 4OEQ + 1Q + 1RAND$.[64] Tabelle 5.2 listet deren wesentlichen Merkmale.

	Deutschland		Europa	
Wege:	schnellste	kürzeste	schnellste	kürzeste
Anzahl Knoten	3 282 166	3 282 166	15 462 474	15 462 474
Anzahl Original-Pfeile	7 833 668	7 833 585	37 249 828	37 249 559
Anzahl Shortcut-Pfeile	4 346 138	5 508 283	22 075 112	27 975 617
Summe der Pfeile	12 179 806	13 341 868	59 324 940	65 225 176

Tabelle 5.2: Kennzahlen der verwendeten Straßenkarten

Verwendete Hardware Die hier angegebenen Benchmarks liefen auf den Rechenknoten einer Rocks Cluster Distribution[65]. Jeder der verwendeten Rechenknoten ist ausgestattet mit 47 GB Hauptspeicher und 24 Intel®-Xeon®-Prozessoren, die jeweils mit 2,40 GHz getaktet sind. Als Betriebssystem kommt eine Linux-Distribution zum Einsatz: CentOS release 6.2 bzw. Rocks release 6.0.

Verwendete Java-Umgebung Für alle Benchmarks dieses Kapitels wurde die Java Virtual Machine (JVM) in der Version 1.7.0_03 in der Implementierung der Firma Oracle eingesetzt. Abgesehen von der Zuweisung einer ausreichenden Menge Arbeitsspeicher per Parameter `-Xmx` wurden keine weiteren Veränderungen an der Standardeinstellung des `java`-Befehls vorgenommen.

5.6.2 Über die Angabe absoluter Suchdauern

In diesem Kapitel werden mitunter auch die absoluten Dauern der Wegsuchen genannt. Diese Angabe ist allerdings von vielen, teilweise auch externen Einflussgrößen abhängig. Nachfolgend werden ein paar wesentliche Einflussgrößen diskutiert:

Die Struktur des zugrunde liegenden Graphen Die in dieser Arbeit zugrunde liegenden Graphen besitzen aufgrund der in Kapitel 3 beschriebenen Extraktion aus den OSM-Daten eine deutlich andere Struktur als diejenigen Graphen, die für die in Kapitel 4 vorgestellten Verfahren zur Geschwindigkeitsmessung verwendet wurden. Der wesentliche Unterschied liegt im höheren mittleren Knotengrad der hier untersuchten Originalgraphen, d. h. ohne Contraction Hierarchy, der sich durch die in Abschnitt 3.6 beschriebene *Kantenobjekt-Verbindung* ergibt. Der höhere durchschnittliche Knotengrad führt zu einem ebenfalls höheren durchschnittlichen Knotengrad in einer darauf aufbauenden Contraction Hierarchy. Dies bewirkt letztendlich, dass die Wegsuchen hier innerhalb jeder Iteration durchschnittlich aufwendiger sind, da beim Expandieren eines Knotens im Mittel mehr Nachfolge- bzw. Vorgängerknoten erreicht werden.

[63] für das Geschwindigkeitsprofil siehe Tabelle 3.5, S. 83
[64] Zur Notation: siehe Abschnitt 5.3 auf Seite 160, Stichwort „Notation der Knotensortierung“
[65] `http://www.rocksclusters.org/wordpress/`

Die Programmiersprache Java Eine wesentliche Eigenschaft der in dieser Arbeit verwendeten Programmiersprache Java ist, dass Quellcode zunächst in so genannten *Bytecode*, anstatt *Maschinencode*, kompiliert wird. Der Bytecode wird erst bei der Ausführung eines Java-Programms von der JVM, einem weiteren Programm, in Maschinencode übersetzt. Dies geschieht im vorliegenden Fall durch den Einsatz eines so genannten Just-in-time-Compilers. Auf diese Weise kann ein Java-Programm unverändert auf allen (Betriebs-)Systemen laufen, auf denen eine plattformspezifische JVM vorhanden ist. Das Vorgehen wirkt sich jedoch tendenziell negativ auf die Laufzeit aus.[66]

Die Garbage Collection Eine weitere grundlegende Eigenschaft der Java-Technologie liegt im Speichermanagement. Die so genannte *Garbage Collection* ist ein Hintergrundprozess, der mit jedem Java-Programm gestartet wird. Wie ihr Name andeutet, kümmert sie sich um nicht mehr referenzierte Objekte und gibt nicht mehr benutzte Speicherbereiche frei. Zwar kann sie auch manuell angestoßen werden, grundsätzlich startet sie jedoch automatisch. Dabei ist explizit weder spezifiziert, *wann* dies geschieht, noch *wie*[67]. Dies bedeutet für die zeitlichen Messungen in dieser Arbeit, dass kein Einfluss darauf genommen werden konnte, ob die Garbage Collection während der Benchmarks einsetzte. Das Testsystem verfügt allerdings über mehrere Prozessoren und die Garbage Collection läuft parallel zum eigentlichen Java-Programm, so dass durch sie die Messungen deutlich weniger verzerrt werden als auf einem System mit nur einem Prozessor.

Hardware Maßgeblichen Einfluss auf die Zeitmessungen besitzt die eingesetzte Hardware. Dies darf bei Betrachtung jedweder zeitlicher Benchmarks nie vernachlässigt werden.

Aktive Prozesse Jeder parallel zur Zeitmessung aktive Prozess kann sich negativ auf das Ergebnis auswirken. Zwar erfordern die Entscheidungen des Prozess-Schedulers aus dem Betriebssystem, welcher Prozess aktuell den bzw. einen Prozessor verwenden darf, nur sehr wenig Zeit. Doch da die hier vorgestellten Messergebnisse ohnehin im Millisekundenbereich liegen, kann der relative Einfluss dieser Prozesse relevant sein.

Tuning für Benchmarks Um besonders schnelle Wegsuchen zu erzielen, kann der Quellcode speziell für diese ausgerichtet werden. Beispielsweise wurde für die Angaben der Suchdauern in Kapitel 4 in der dort gegebenen Literatur auf die Rekonstruktion der tatsächlichen *Pfade* verzichtet; Anforderung war in den meisten Fällen lediglich die Weglänge bzw. Fahrtdauer. In diesen Fällen kann auf ein Mitführen und Verwalten des `vorgänger`-Attributs gemäß Verfahren 5, S. 91, verzichtet werden.

[66] Im Internet finden sich sehr viele Vergleiche zwischen Programmiersprachen. Gerade in Bezug auf die Programmiersprache Java muss dabei jedoch die Sprach-Version berücksichtigt werden. Einen guten Überblick mittels vergleichender Benchmarks bietet unter anderem `http://benchmarksgame.alioth.debian.org/` (zuletzt geprüft am 4.7.2013). Dort wird ebenfalls auf die beschränkte Interpretierbarkeit solcher Benchmarks hingewiesen.

[67] Sun Microsystems [2006, S. 4] beschreibt dies im Detail: „The precise algorithm used to organize memory and allocate and deallocate space is handled by the garbage collector and hidden from the programmer“ sowie „The timing of garbage collection is up to the garbage collector“ [ebd.].

Eine weitere Strategie beim Tuning von Quellcode für Zeitmessungen besteht im so genannten *Inlining*. Hierbei werden unter anderem bewusst große Teile des Programms in einem Stück geschrieben um beispielsweise auf Schleifen oder Code-Auslagerungen in Methoden zu verzichten, da diese beiden ihrerseits einen gewissen zusätzlichen Aufwand erfordern. So muss bei der Ausführung von Schleifen eine Sprungmarke im Maschinencode gesetzt und die Schleifen-Abbruchbedingung regelmäßig überprüft werden; Methoden-Aufrufe kopieren im Maschinencode ebenfalls die Rücksprung-Adresse und die an sie übergebenen Parameter sowie eventuelle Rückgabe-Werte.

Im Allgemeinen ergeben sich jedoch Nachteile bei anderen Anforderungen an eine effiziente Wegsuche. Als Beispiele sind schwerer wartbarer oder schlecht portierbarer Code zu nennen[68]. Das zu Beginn von Abschnitt 5.5.6, S. 176, angedeutete Vorgehen zur Knotenindizierung besitzt beispielsweise den erstgenannten Nachteil.

Benchmark-Fokus: Iterationen vor Suchdauer

Die für diese Arbeit geschriebenen Programme und Wegsuche-Algorithmen sind in diesem Sinne nicht auf eine schnelle Ausführung optimiert. Daher sollte bei der Betrachtung der Benchmarks der Anzahl der Iterationen mehr Beachtung geschenkt werden als der Suchdauer, denn erstere ist in diesem Sinne eine „objektive“ Größe und unabhängig von den hier angedeuteten Programmier-Optimierungen und Messeinflüssen.

5.6.3 Benchmarks zur Knotensortierung während der Wegsuche

Die Benchmarks in diesem Abschnitt vergleichen die in Abschnitt 5.5.5, S. 174 f., vorgestellten Datentypen zur Sortierung der Horizontmengen. Alle nachfolgenden Benchmarks verwenden die hier gewonnen Erkenntnisse. In Tabelle 5.3 sind die Quellen der verwendeten Implementierungen als Übersicht angegeben:

Datenstruktur	Implementierung
Priority Queue	Klasse `java.util.PriorityQueue` aus dem Oracle-JDK[†]
TreeSet	Klasse `java.util.TreeSet` aus dem Oracle-JDK[†]
Skip-Liste	angepasste Version von LiteratePrograms [2009]
Fibonacci Heap	Java-Implementierung[※] aus dem quelloffenen *graphmaker*-Projekt[‡]

[†] Version 1.6.0_35

[‡] `http://code.google.com/p/graphmaker/`

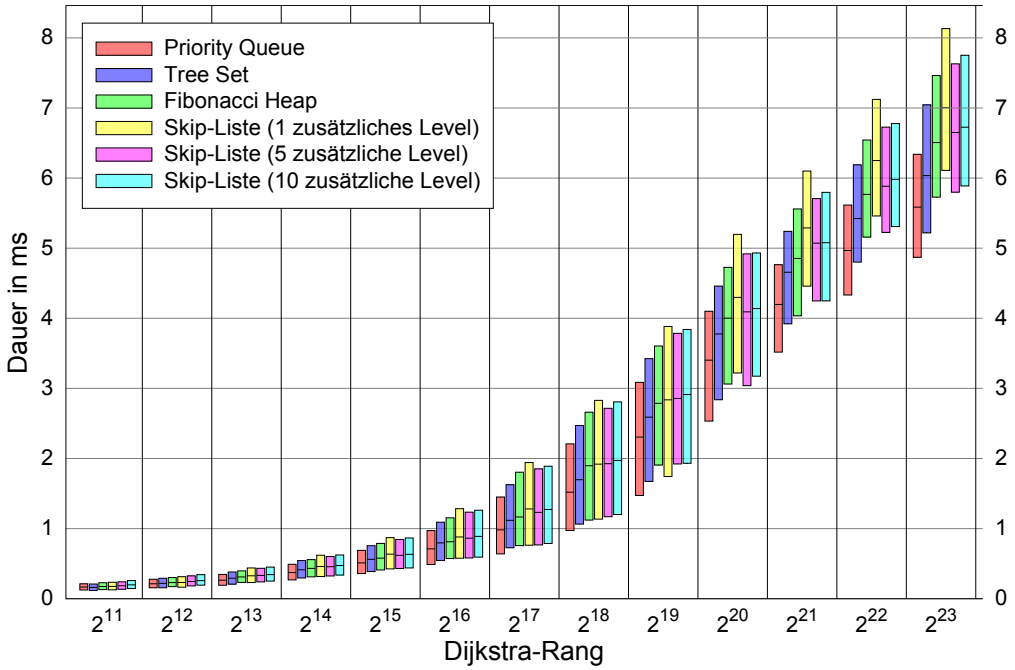
[※] `http://code.google.com/p/graphmaker/source/browse/core/src/com/bluemarsh/graphmaker/core/util/FibonacciHeap.java`; Stand vom 5.12.2008

Tabelle 5.3: Implementierungen der untersuchten Datenstrukturen

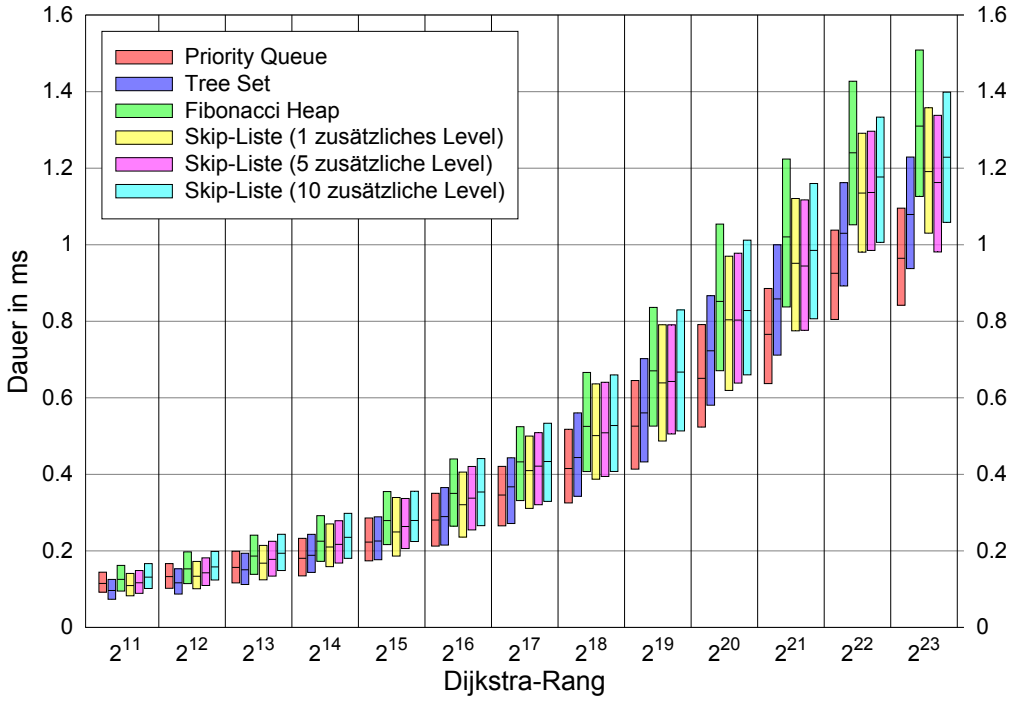
Abbildung 5.20, S. 182, zeigt die Auswertung der Messungen auf der Straßenkarte Europas, die in Kapitel 3, S. 61 ff., vorgestellt wurden[69]. Je Dijkstra-Rang erfolgten 1 000 zufällige Wegsuchen. Das vollständige Diagramm inklusive Antennen zeigt Abbildung B.1, S. 320, im Anhang.

[68] Diese Erkenntnisse meint Knuth [1974, S. 368] mit seiner mittlerweile berühmten Aussage: „Premature optimization is the root of all evil“. (In etwa: „Voreilige Optimierung ist die Wurzel allen Übels.“.)

[69] siehe insbesondere Tabelle 3.6, S. 85



(a) Suche nach kürzesten Wegen



(b) Suche nach schnellsten Wegen

Abbildung 5.20: Vergleich verschiedener Datenstrukturen bei der Knotensortierung Zugrunde liegendes Verfahren ist die Standard-Wegsuche in Contraction Hierarchies (Verfahren 11, S. 138).

Interpretation der Messergebnisse Die Unterschiede bei den Laufzeiten unter Verwendung der verschiedenen Datenstrukturen sind eher gering. Dies liegt vor allem an der vergleichsweise kleinen Zahl notwendiger Iterationen für eine Wegsuche in einer Contraction Hierarchie[70]. Allerdings bewirkt die Verwendung der Priority Queue in allen Fällen leichte Geschwindigkeitsvorteile. Am auffälligsten sind diese bei den Suchen nach *kürzesten* Wegen, da diese durch Contraction Hierarchies weniger stark beschleunigt werden als die Suche nach *schnellsten* Wegen. Als Konsequenz dieser Untersuchung werden für den Rest dieser Arbeit daher – sofern nicht anders angegeben – ausschließlich Implementierungen verwendet, die für die Sortierung der Horizontmengen auf Priority Queues basieren.

Beschränkung durch Kartenränder

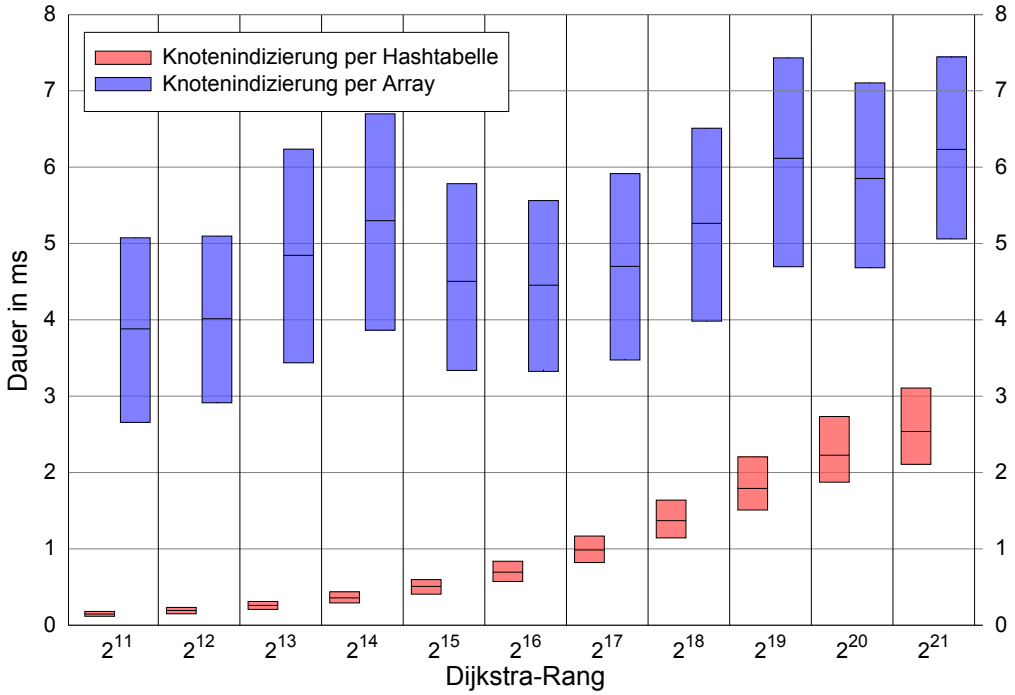
Ebenfalls in Abbildung 5.20, S. 182, erkennbar ist ein leicht S-förmiger Verlauf aller Messungen. Besonders in den letzten beiden angegebenen Dijkstra-Rängen nehmen die Anzahl der benötigten Iterationen weniger schnell zu. Der Grund liegt jedoch nicht im Verfahren, sondern an der zugrunde liegenden Karte: Da die Suche nach Wegen mit hohen Dijkstra-Rängen den größten Teil der Karte abarbeitet, stößt sie auch auf die *Kartenränder*. Diese beschränken für diese Suchen den Suchraum und bewirken den gemessenen Effekt, der auch bei anderen Messungen dieses Kapitels auftritt.

5.6.4 Benchmarks zur Indizierung der Knoten während der Wegsuche

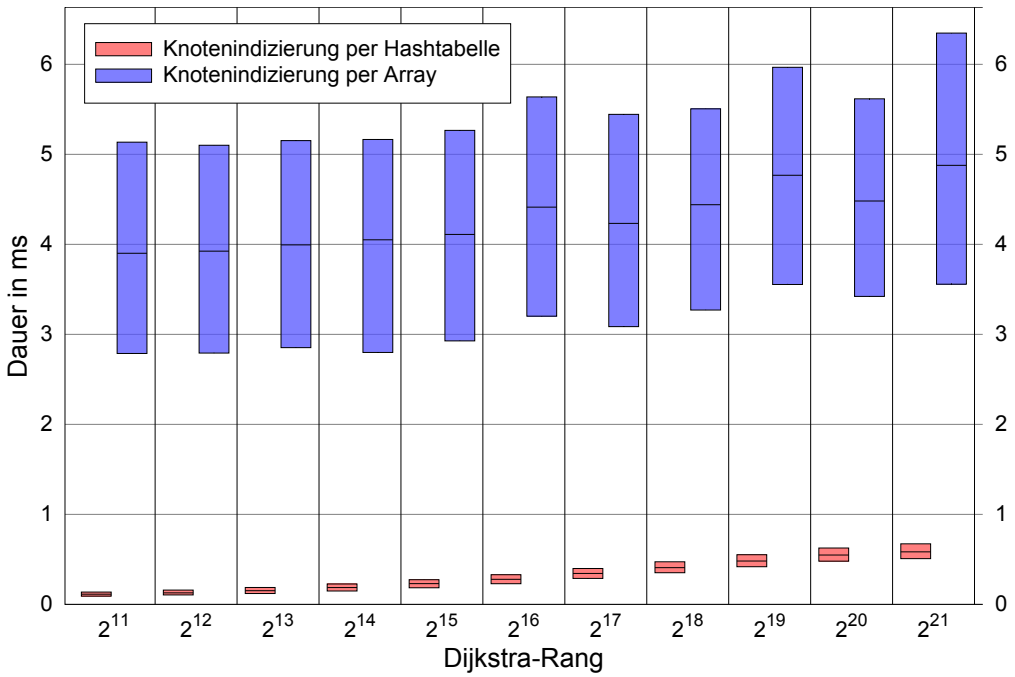
In diesem Abschnitt wird die in Abschnitt 5.5.6, S. 176 ff., vorgestellte Knotenindizierung per Hashtabelle mit der Indizierung per Ränge-Array verglichen. Die Index-freie Modellierung wird aufgrund ihrer mangelnden praktischen Relevanz nicht näher beleuchtet. Die im vorangegangenen Abschnitt durchgeführten Messungen basierten auf einem Hashtabellen-Index.

Da der Index per Ränge-Array deutlich mehr Speicher benötigt als der Hashtabellen-Index, wurden die hier aufgeführten Benchmarks auf dem Straßennetz der Deutschlandkarte anstelle der Europakarte durchgeführt[71]. Abbildung 5.21, S. 184, fasst die Messergebnisse zusammen.

[70] siehe hierzu beispielsweise auch Abbildungen 5.24a, S. 189 und 5.25a, S. 190
[71] siehe Kapitel 3, S. 61 ff., und insbesondere Tabelle 3.6, S. 85



(a) Suche nach *kürzesten* Wegen



(b) Suche nach *schnellsten* Wegen

Abbildung 5.21: Benchmark für die Indizierung der Knoten
Zugrunde liegendes Verfahren ist die Standard-Wegsuche in Contraction Hierarchies (Verfahren 11, S. 138).

Interpretation der Messergebnisse Die Performance der Knotenindizierung per Array wird dominiert durch die Zeit, die für das Anlegen des Index-Arrays benötigt wird. Dies erklärt auch die in etwa gleiche Größe der Boxes über alle Dijkstra-Ränge hinweg in den Boxplot-Diagrammen. Während sich bei der Suche nach schnellsten Wegen tendenziell ein ähnlicher Anstieg der mittleren Suchdauer beider Indizierungs-Arten erkennen lässt, ist dies bei der Suche nach kürzesten Wegen nicht der Fall. Die starke Schwankung ist hier wahrscheinlich auf den Einsatz der Garbage Collection zurückzuführen. Da die Zeitdauern jedoch derart stark von denen der Hashtabellen-Indizierung nach oben abweichen, wird im Folgenden dieser Frage nicht weiter nachgegangen. Ferner wird in allen noch folgenden Verfahren – sofern nicht anders hervorgehoben – nur noch eine Indizierung per Hashtabelle verwendet.

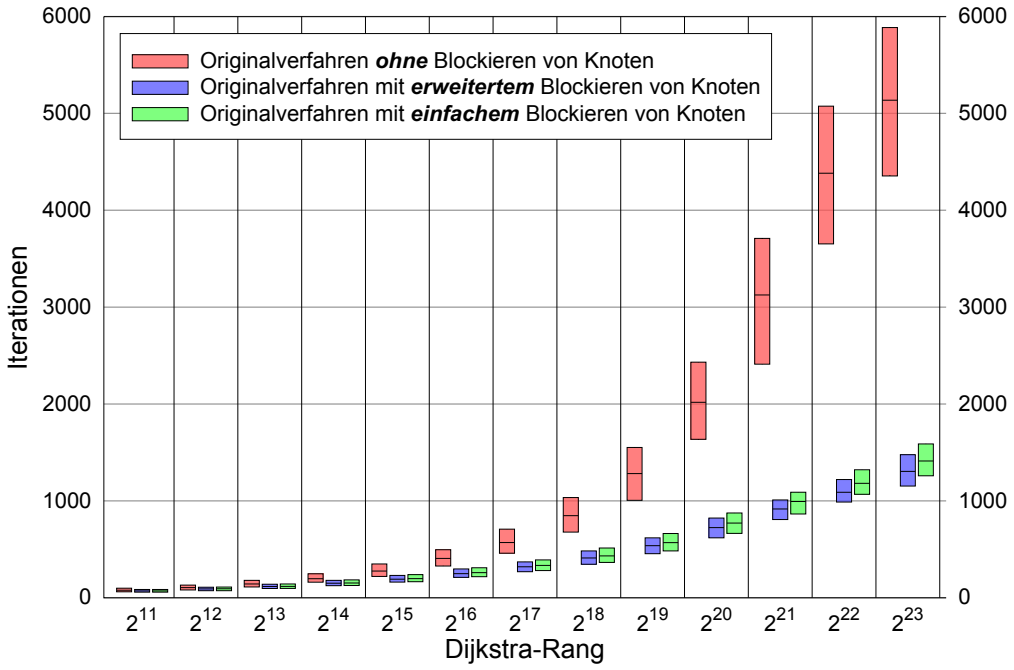
5.6.5 Benchmarks zum Blockieren von Knoten

Diese Messung beschäftigt sich mit dem Einfluss des Blockierens von Knoten bei der Wegsuche. Drei Szenarien werden betrachtet:

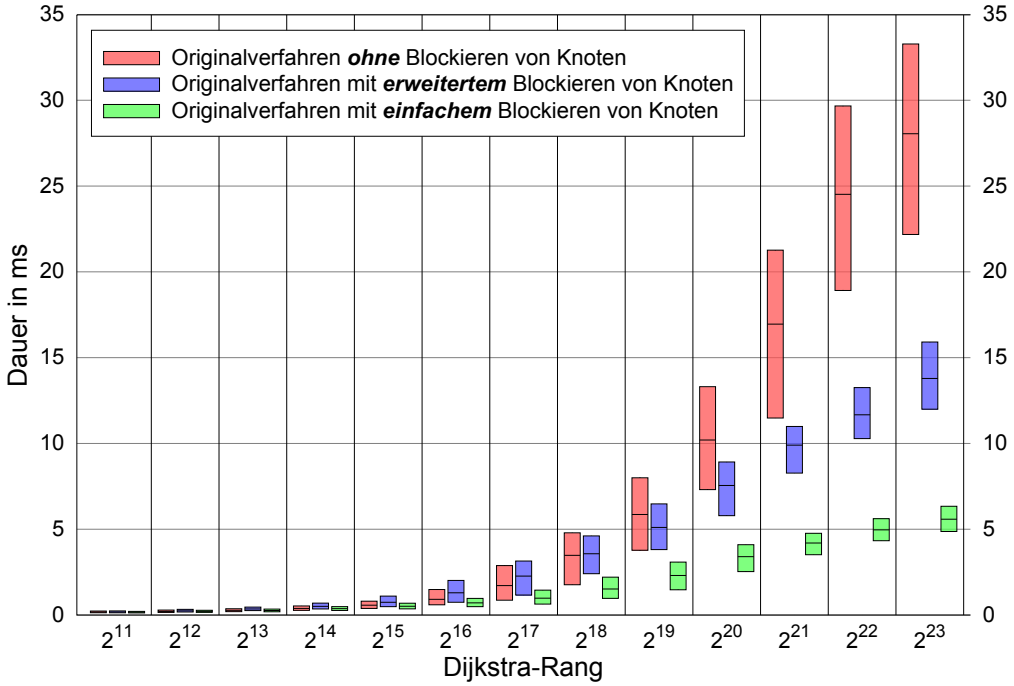
1. kein Blockieren von Knoten,
2. einfaches Blockieren von Knoten sowie
3. erweitertes Blockieren von Knoten.

Der zweite Fall bezieht sich auf das in Abschnitt 5.2.1 vorgestellte Prinzip der Knotenblockierung; der zweite auf die dort ebenfalls[72] beschriebene *Weitergabe der Blockierung*. Abbildungen 5.22, S. 186 und 5.23, S. 187 zeigen die Messergebnisse für Suchen nach kürzesten und nach schnellsten Wegen.

[72] siehe Seite 148

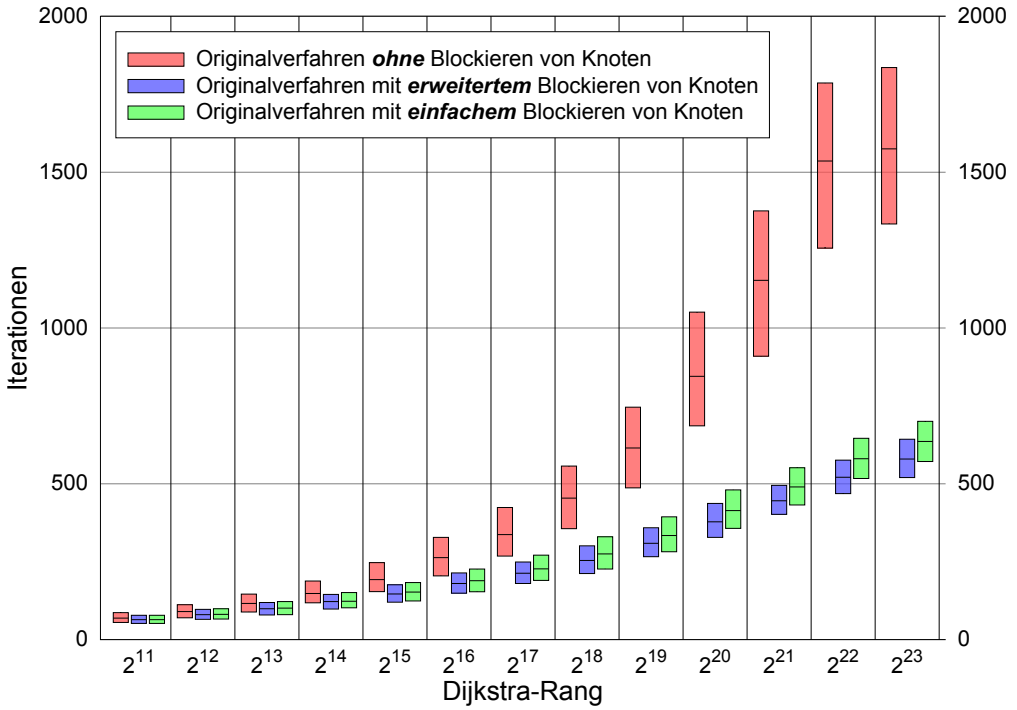


(a) Benötigte Iterationen

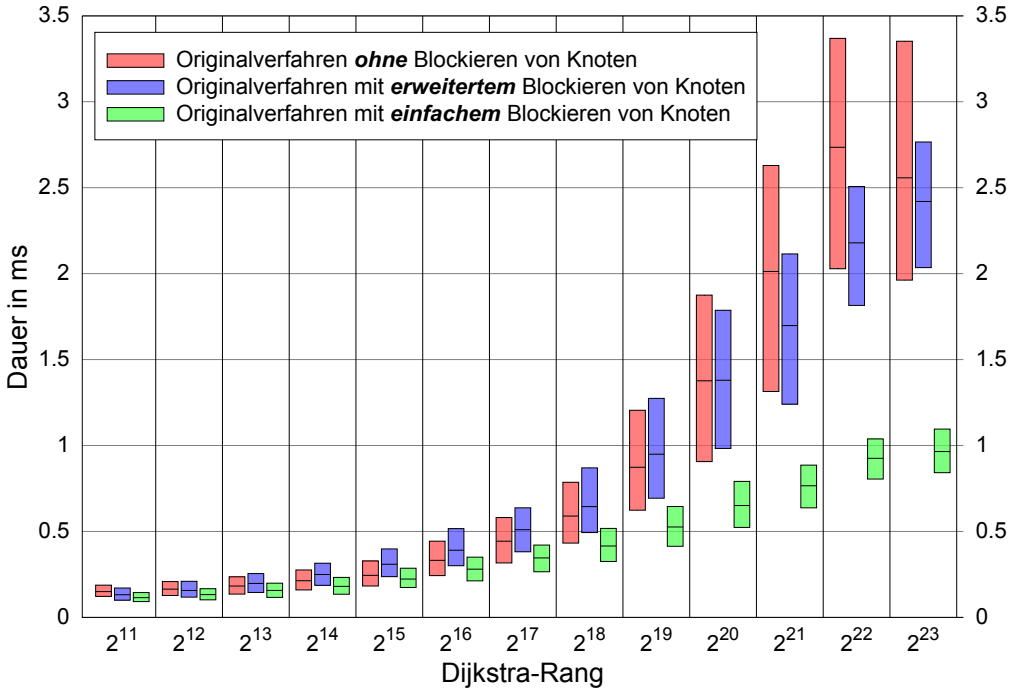


(b) Suchdauer

Abbildung 5.22: Benchmark für Knotenblockierung bei Suchen nach *kürzesten* Wegen Zugrunde liegendes Verfahren ist die Standard-Wegsuche in Contraction Hierarchies (Verfahren 11, S. 138).



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 5.23: Benchmark für Knotenblockierung bei Suchen nach *schnellsten* Wegen Zugrunde liegendes Verfahren ist die Standard-Wegsuche in Contraction Hierarchies (Verfahren 11, S. 138).

Interpretation der Messergebnisse Der Effekt, der durch den Einsatz der beiden Varianten *mit* Knotenblockierung entsteht, ist deutlich. Allerdings spart das *erweiterte Knotenblockieren* nur verhältnismäßig wenig zusätzliche Iterationen im Vergleich zum *einfachen Knotenblockieren* ein. Der erhöhte Aufwand, der bei der Untersuchung der Nachbarknoten des erstmalig blockierten Knotens v und des „blockierenden“ Knotens u in Verfahren 13, S. 149, entsteht, wird durch die dadurch eingesparten Iterationen nicht egalisiert: Das Verfahren mit *einfacher Knotenblockierung* ist mit Abstand das schnellste der drei Varianten und wird in den nachfolgenden Messungen – sofern nicht anders erwähnt – durchgehend eingesetzt.

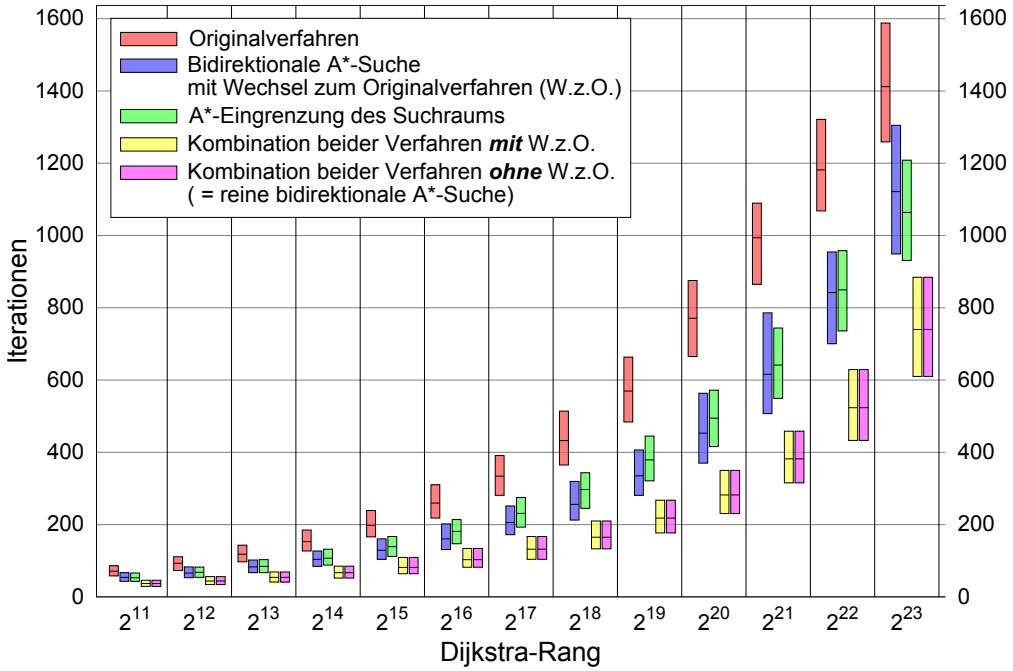
5.6.6 Benchmarks zur zielgerichteten Wegsuche

Im Folgenden werden die in Abschnitt 5.2.2, S. 149 ff., formulierten Überlegungen zur zielgerichteten Wegsuche in einer Contraction Hierarchy anhand der Benchmark-Systematik untersucht. Hierzu gehören

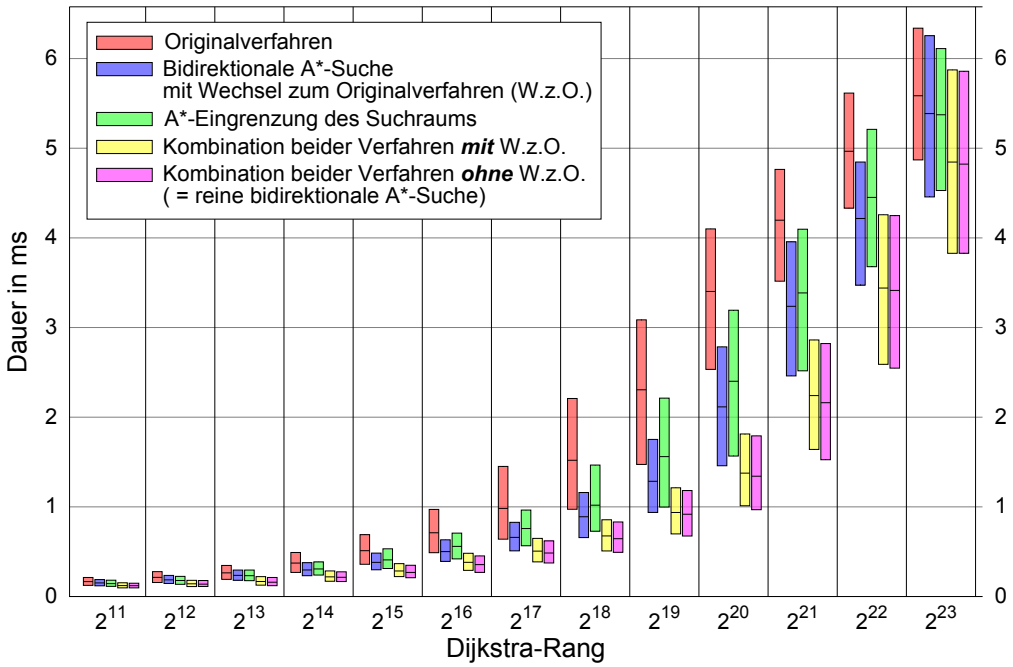
1. die A*-Eingrenzung des Suchraums[73],
2. die bidirektionale A*-Suche mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens gemäß Nowak u. a. [2012][74],
3. die Kombination aus 1. und 2. mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens gemäß Nowak u. a. [2012] und
4. die Kombination aus 1. und 2. als reine bidirektionale A*-Suche.

Nach den Erkenntnissen des vorangegangenen Abschnitts verwenden alle vier Verfahren das *einfache Blockieren von Knoten*. Ferner basieren sie auf Priority Queues zum Sortieren der Horizontmengen. Abbildungen 5.24 und 5.25 zeigen die Ergebnisse der Messungen bei Verwendung der Kürzeste-Wege- und der Schnellste-Wege-Metrik.

[73] siehe Abschnitt 5.2.2.1, S. 150
[74] siehe Abschnitt 5.2.2.2, S. 152

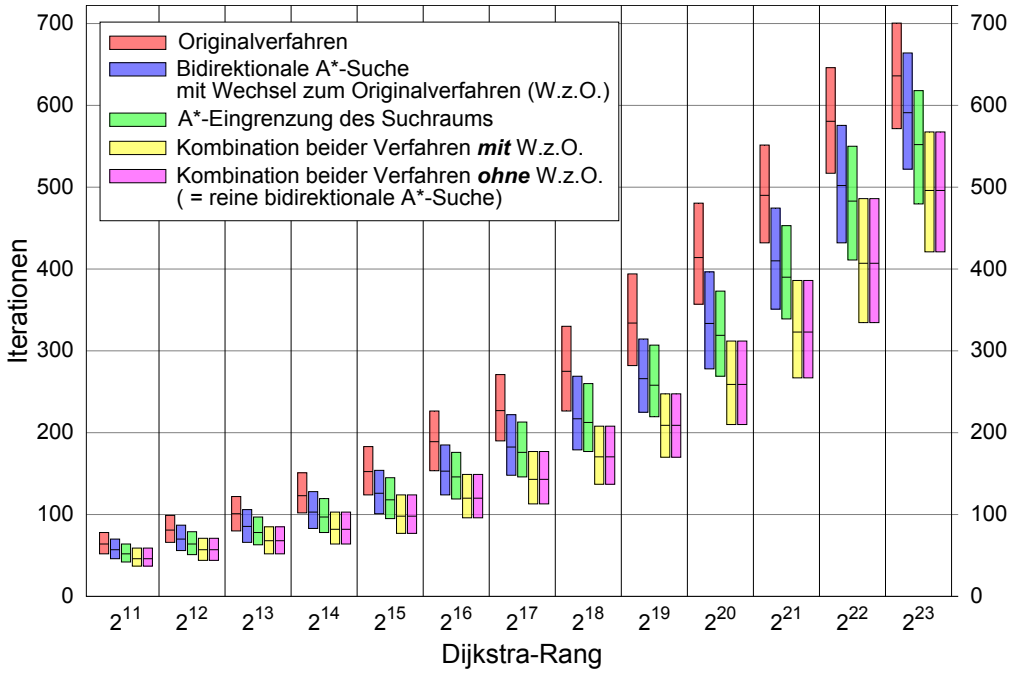


(a) Benötigte Iterationen

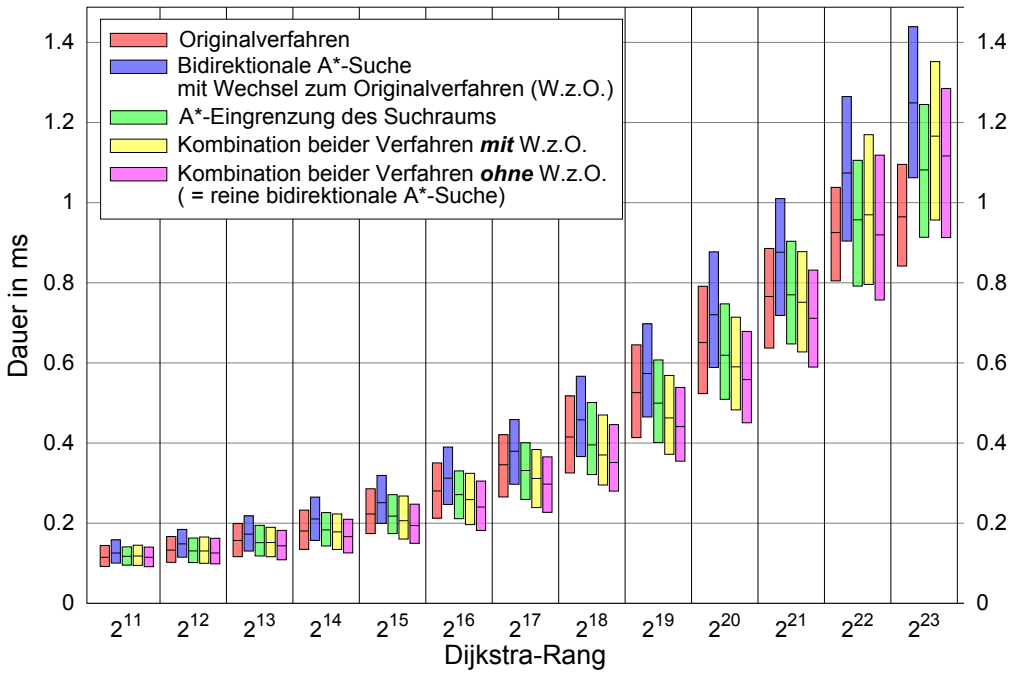


(b) Suchdauer

Abbildung 5.24: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 5.25: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen

Interpretation der Messergebnisse Die *A*-Eingrenzung des Suchraums* allein bewirkt sowohl eine deutliche Verringerung der Iterationen als auch der Suchdauer. Ihre Vorteile

sind deutlicher bei der Suche nach *kürzesten* Wegen. Dies liegt hauptsächlich daran, dass die Restkosten-Schätzung für schnellste Wege eine schlechtere untere Schranke liefert als für kürzeste Wege.

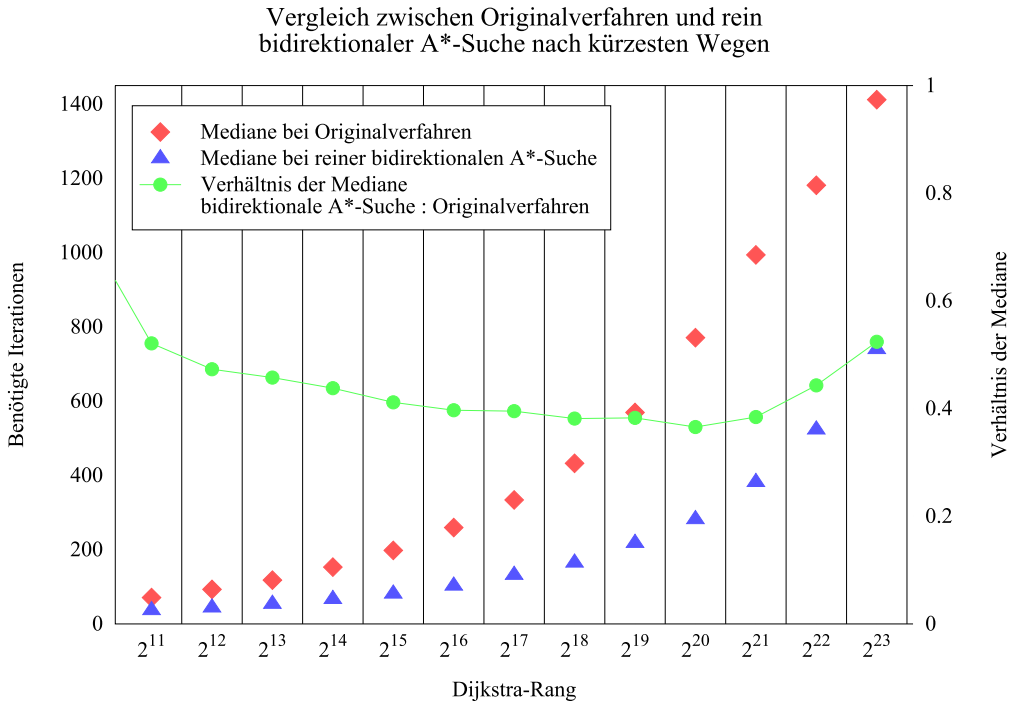
Die Variante der bidirektionalen A*-Suche, die nach dem Finden des ersten Brückenknotens zum Originalverfahren zurückkehrt, ist für sich genommen ebenfalls vorteilhaft. Einzig bei ihrem Einsatz zur Suche nach schnellsten Wegen mit hohem Dijkstra-Rang zeigt sich eine Verschlechterung bei der Suchdauer gegenüber dem Originalverfahren. Beide Verfahren benötigen in diesem Fall ähnlich viele Iterationen. Da aber bei der bidirektionalen A*-Suche für alle erreichten Knoten zusätzlich die Restluftdistanz berechnet wird, ergibt sich hierdurch die Verlangsamung der Suche. Der oben angesprochene Effekt der *Beschränkung durch die Kartenränder* kommt dem Originalverfahren dabei stärker zugute.

Die Kombination aus *bidirektionaler A*-Suche* und *A*-Eingrenzung des Suchraums* ist lohnenswert. Besonders gilt dies aufrund der Qualität des Restkosten-Schätzers bei der Suche nach *kürzesten* Wegen. Bezogen auf die Suchdauer ist es fast unerheblich, ob nach dem Auffinden eines Brückenknotens eine Rückkehr zum Originalverfahren vollzogen (I) oder die A*-Suche beibehalten wird (II). Besonders interessant ist, dass beide Suchvarianten exakt die gleichen Anzahlen an Iterationen benötigen.[75] Theoretisch könnten allerdings diesbezüglich durchaus Unterschiede auftreten, wie im Folgenden dargelegt wird.

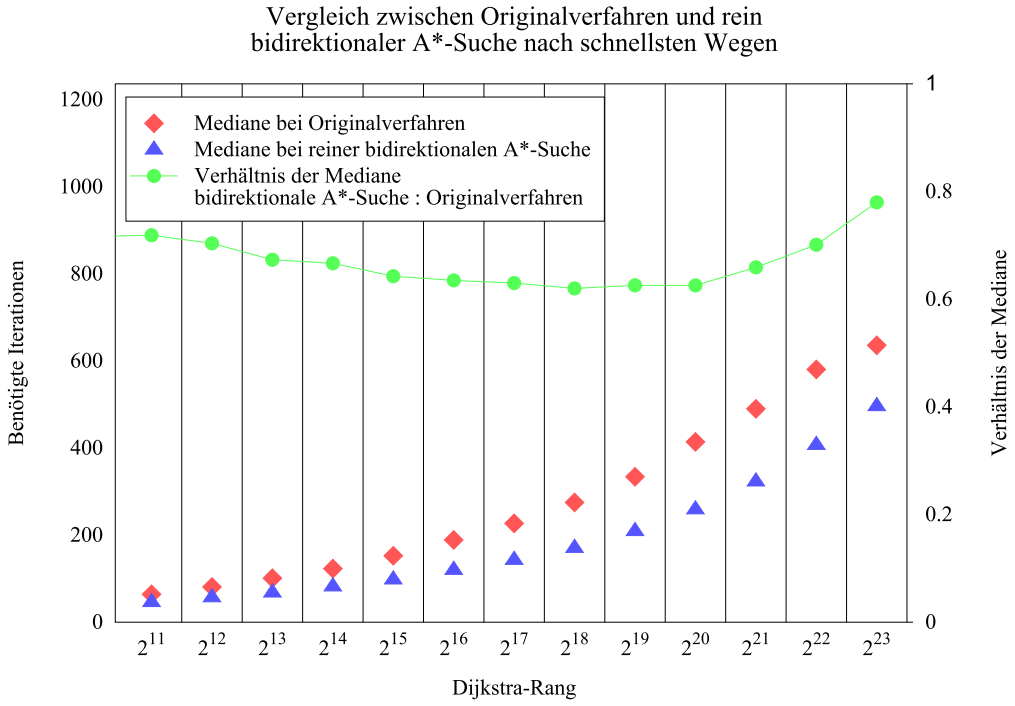
Beide kombinierten Verfahren unterscheiden sich erst, nachdem der erste Brückenknoten gefunden wurde: Während (I) zur bidirektionalen Dijkstra-Suche mit A*-Eingrenzung des Suchraums wechselt, bleibt (II) bei der bidirektionalen A*-Suche mit gleicher Eingrenzung des Suchraums. (II) kann dann mit weniger Iterationen den Lösungsweg finden, wenn es aufgrund der A*-Suche schnell einen weiteren Brückenknoten findet, der zudem einen deutlich kürzeren Weg impliziert als der erste Brückenknoten. Dies ermöglicht nämlich der A*-Eingrenzung des Suchraums, solche Knoten zu verwerfen, die möglicherweise noch bei (I) untersucht werden müssten. Bei den Wegsuchen für die Benchmarks tritt dieser Fall offenbar nicht ein. Dies legt den Schluss nahe, dass der durch A*-Suche gefundene erste Brückenknoten bereits durch die anschließende A*-Eingrenzung des Suchraums bei (II) einen derart starken Effekt besitzt, das nachfolgend gefundene Brückenknoten zwar eine Verkürzung des Weges liefern. Diese Verkürzung ist dann jedoch nicht deutlich genug, als dass Knoten aus den Horizontmengen durch die A*-Eingrenzung des Suchraums verworfen werden können.

Die Standardsuche profitiert deutlich, wenn sie um die bidirektionale A*-Suche und die A*-Eingrenzung des Suchraums erweitert wird. Für die Suche nach kürzesten Wegen lässt sich die Anzahl notwendiger Dijkstra-Iterationen mehr als halbieren. Für die Suche nach schnellsten Wegen wird etwa ein Drittel der Iterationen eingespart. Abbildung 5.26, S. 192, fasst beide Erkenntnisse zusammen.

[75] Diese Erkenntnis lässt sich nicht aus der Grafik, sondern nur dem Vergleich der Einzelmessungen entnehmen.



(a) Suche nach *kürzesten* Wegen



(b) Suche nach *schnellsten* Wegen

Abbildung 5.26: Relative Einsparungen bei zielgerichteter Wegsuche in Contraction Hierarchies
Bezugsgröße für die Mediane ist die Anzahl der benötigten Iterationen.

Beschleunigung

Im Vergleich mit dem Standard-Dijkstra-Verfahren zeigt sich zudem: Je mehr Iterationen die Standard-Dijkstra-Suche benötigt, desto größer ist die Suchbeschleunigung mit dem CH-Verfahren. Bei der Suche nach kürzesten Wegen werden Beschleunigungen um mehr als den Faktor 11 000 erreicht; bei der Suche nach schnellsten Wegen sogar um mehr als Faktor 16 700[76]. Diese Beobachtungen bestätigen im Wesentlichen die theoretischen Überlegungen auf Seite 144 f[77].

Die notwendigen Iterationen in Abbildungen 5.24a und 5.25a wachsen allerdings nicht *streng* logarithmisch mit dem Dijkstra-Rang der zugrunde liegenden Wegsuchen. Dies liegt an der Existenz von Shortcut-Pfeilen, die im Vergleich mit dem der Contraction Hierarchy zugrunde liegenden Originalgraphen zu einem höheren durchschnittlichen Knotengrad führen.

5.6.7 Benchmarks zur Schrittweite bei paralleler Wegsuche

Der hier vorgestellte Benchmark untersucht das Laufzeitverhalten der in Abschnitt 5.2.3, S. 156, vorgestellten parallelen Wegsuche. Wie dort erwähnt, ist die Stellgröße des Verfahrens die Anzahl unabhängiger Iterationen der Vorwärts- und Rückwärtssuche, bis geprüft wird, ob ein Brückenknoten identifiziert werden kann. Dieser Wert wird hier als *Schrittweite* bezeichnet, um Mehrdeutigkeiten mit dem Begriff der *Iterationen* zu vermeiden. Abbildung 5.27, S. 194, zeigt das Boxplot-Diagramm für parallele Suchen nach schnellsten Wegen mit unterschiedlichen Schrittweiten auf der Europa-Straßenkarte[78][79]. Abbildung 5.28, S. 195, zeigt zum Vergleich die Ergebnisse einer parallelisierten reinen bidirektionalen Wegsuche mit A*-Eingrenzung des Suchraums[80]. Beide Implementierungen verwenden das *einfache Blockieren von Knoten* und basieren auf Priority Queues zum Sortieren der Horizontmengen.

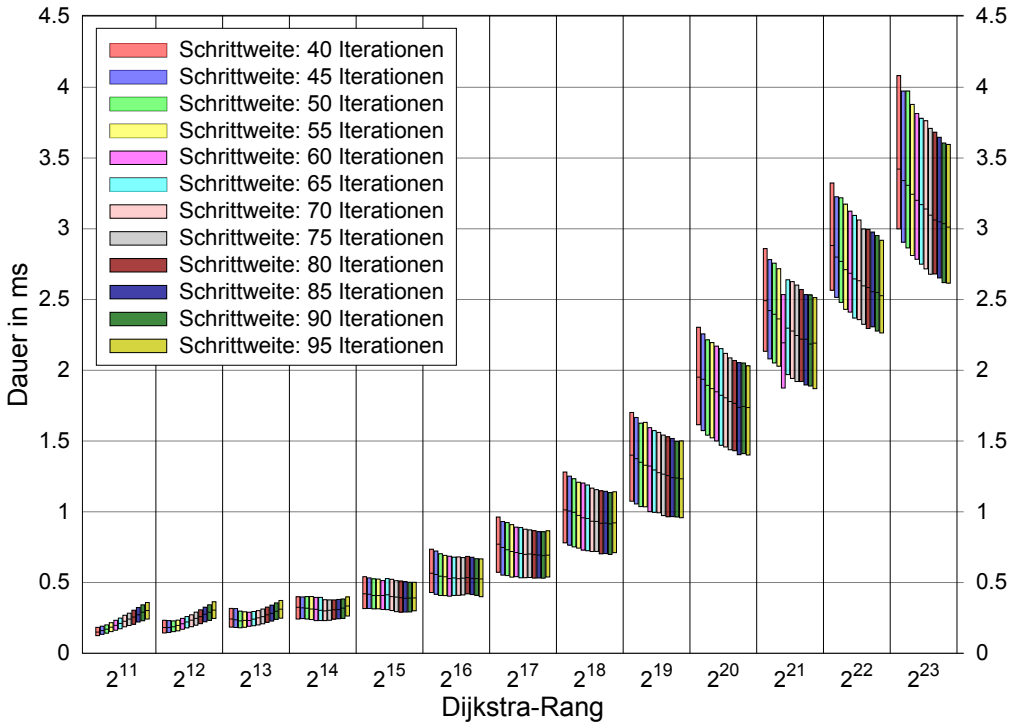
[76] siehe Abbildungen 5.24a und 5.25a jeweils bei Dijkstra-Rang 2^{23}

[77] siehe Stichwort: „Aufwandsabschätzung der Wegsuche“

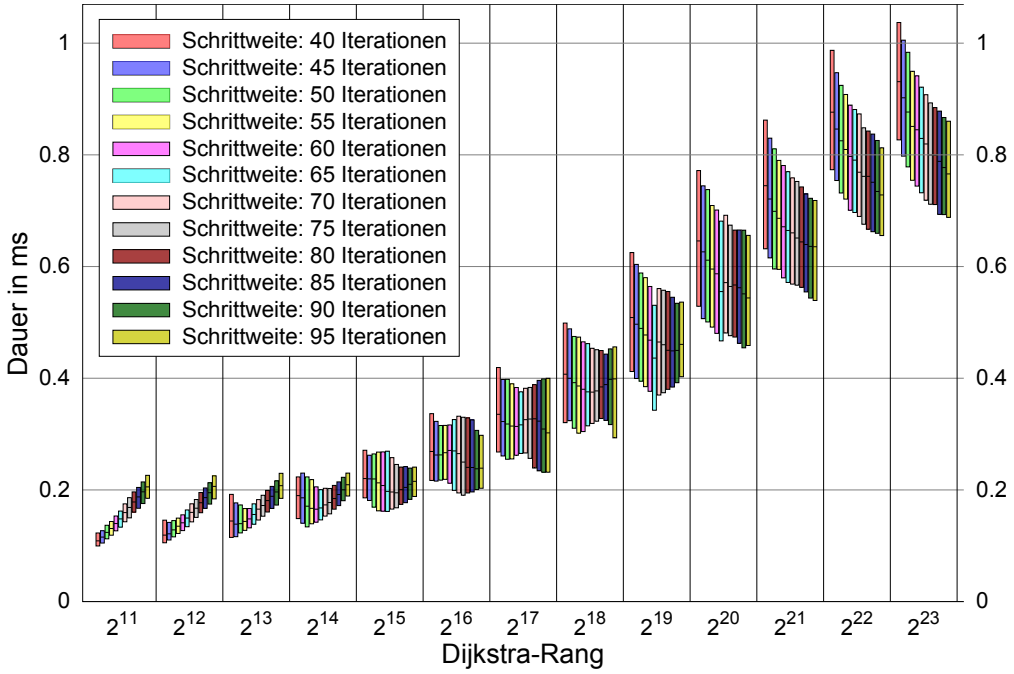
[78] siehe Tabelle 3.6, S. 85

[79] Die Versuchsgruppen sind pro Dijkstra-Rang aufsteigend nach Schrittweite sortiert: Links stehen damit die Wegsuchen mit einer Schrittweite von 40 Iterationen; rechts diejenigen mit 95. Das vollständige Diagramm inklusive Antennen zeigt Abbildung B.7, S. 326, im Anhang.

[80] siehe Abschnitt 5.2.2, S. 149 ff.

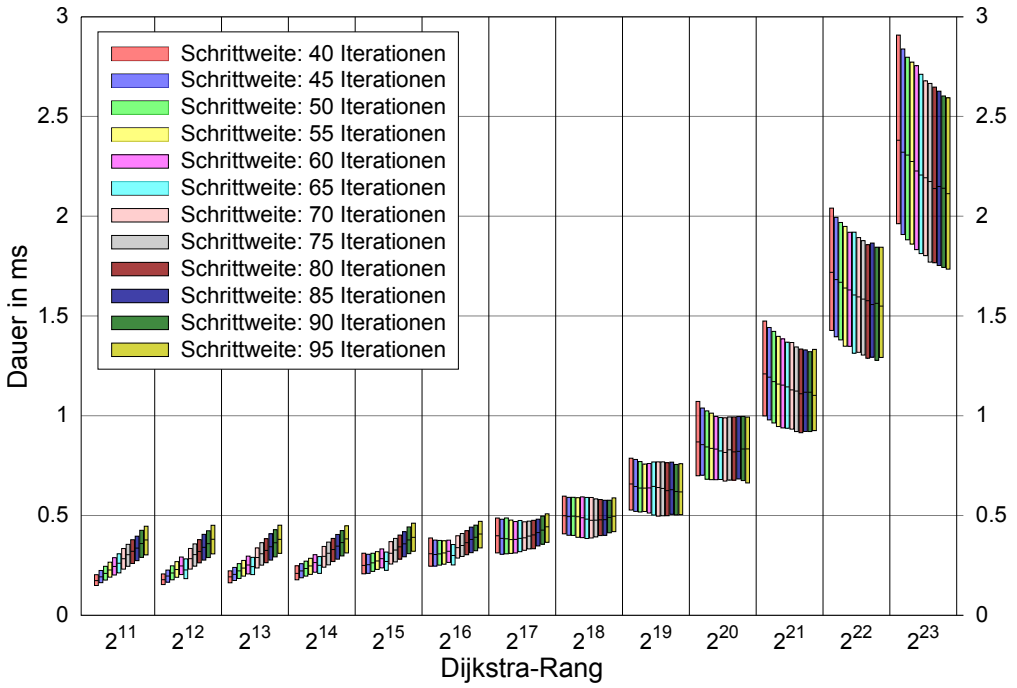


(a) Suche nach *kürzesten* Wegen

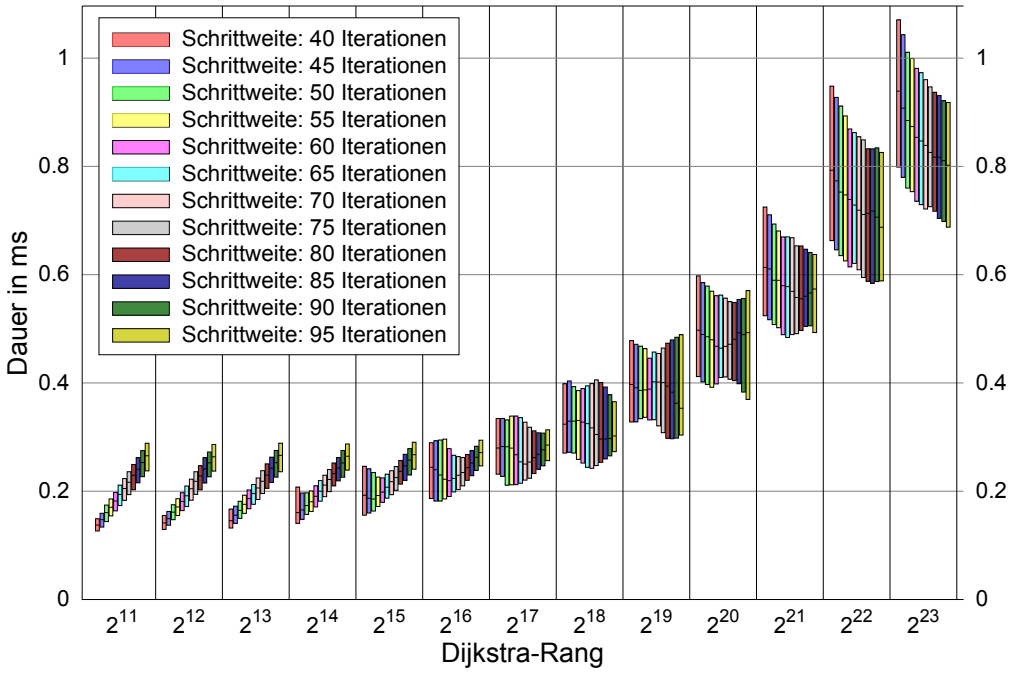


(b) Suche nach *schnellsten* Wegen

Abbildung 5.27: Benchmark für parallele Standard-Wegsuchen in Contraction Hierarchies (Verfahren 11, S. 138).



(a) Suche nach *kürzesten* Wegen



(b) Suche nach *schnellsten* Wegen

Abbildung 5.28: Benchmark für parallele, zielgerichtete Wegsuchen
Zugrunde liegendes Verfahren ist die bidirektionale A*-Wegsuche in Contraction Hierarchies (Verfahren 14, S. 155).

Interpretation der Messergebnisse Deutlich erkennbar ist der Zusammenhang zwischen dem Dijkstra-Rang und der Schrittweite der Wegsuche. Tendenziell gilt: Je größer der Dijkstra-Rang, desto vorteilhafter ist eine größere Schrittweite. Allerdings ist letztere – wie zu erwarten – für Wegsuchen mit geringerem Dijkstra-Rang nachteilig. Zwei Hauptgründe sind hierfür verantwortlich:

1. Wegsuchen mit kleinen Dijkstra-Rängen benötigen in Contraction Hierarchies im Allgemeinen sehr wenige Iterationen. In diesem Fall erzwingen zu hohe Schrittweiten unnötige Iterationen in Vorwärts- und Rückwärtsrichtung.
2. Durch zu große Schrittweiten und damit einhergehende Iterationen werden mehr Knoten als notwendig in die Horizont- und Lösungsmengen aufgenommen. Dies macht die Suche nach einem Brückenknoten für Wegsuchen mit kleinem Dijkstra-Rang übermäßig aufwendig.

Die Wahl der optimalen Schrittweite bei paralleler Wegsuche hängt demnach sehr vom Anwendungsfall ab. Liegen Informationen bezüglich der zu erwartenden Dijkstra-Ränge der Wegsuchen vor, sollten sie für die Wahl der Schrittweite verwendet werden. Die nachfolgende Aufzählung nennt beispielhaft Quellen, aus denen diese Informationen stammen können:

Vergangenheitsdaten Liegen Daten über vorangegangene Wegsuchen vor, kann aus diesen der durchschnittliche bzw. häufigste Dijkstra-Rang ermittelt werden, anhand dessen die Schrittweite gewählt werden sollte.

Kartendaten Offensichtlich stellt die Kartengröße eine obere Schranke für die überhaupt erreichbaren Dijkstra-Ränge der Wegsuchen dar: Auf einer Karte mit $|V|$ Knoten, kann keine Standard-Dijkstra-Wegsuche mehr als $|V|$ Iterationen benötigen.

Externe Beschränkungen Es sind Szenarien denkbar, in denen aus anderen Gründen Schranken für die zu erwartenden Dijkstra-Ränge gegeben sind. Als Beispiel können hier Verkehrssimulationen genannt werden, bei denen die Wegstrecken aufgrund der Berücksichtigung von Lenk- und Ruhezeiten[81] eine maximale Länge nicht überschreiten dürfen. Aus dieser Weglänge kann wiederum der maximal zu erwartende Dijkstra-Rang berechnet werden.

Mitunter kann es sinnvoll sein, auf parallele Wegsuchen gänzlich zu verzichten: Für Wegsuchen, die mit wenigen Iterationen gelöst werden können, bedeutet die Parallelisierung einen nachteiligen Verwaltungsaufwand, der sich in der Laufzeit leicht negativ niederschlägt. Dies zeigt sich im Vergleich mit den Messergebnissen in Abbildungen 5.24 und 5.25 auf den Seiten 189 und 190. Diese Erkenntnis hängt zusätzlich davon ab, wie effizient parallele Prozesse auf dem ausführenden System implementierbar sind. Hier unterscheiden sich Desktop-Computer erwartungsgemäß von mobilen Plattformen wie beispielsweise Navigationsgeräten in Kraftfahrzeugen.

Annähernd doppelte Suchgeschwindigkeit für kürzeste Wege

Auf dem verwendeten Benchmarksystem zeigt sich allerdings im Allgemeinen eine Beschleunigung der Wegsuche. Zwischen der Suche nach *kürzesten* und der Suche nach *schnellsten* Wegen existiert dabei ein deutlicher Unterschied. Während für kürzeste Wege fast die theoretisch optimale Verdopplung der Suchgeschwindigkeit erreicht wird, ist die Beschleunigung bei der Suche nach schnellsten Wegen nur gering. Grund hierfür ist die verschieden hohe Anzahl notwendiger Dijkstra-Iterationen bei den Suchen.

[81] siehe hierzu auch Abschnitt 2.3, S. 49 ff.

5.6.8 Zusammenfassung der Benchmarkergebnisse

Ausgehend von den Messungen der vorangegangenen Abschnitte können die nachfolgenden wesentlichen Erkenntnisse festgehalten werden:

- Von den hier untersuchten Datentypen zur Knotensortierung während der Wegsuchen lieferte die Priority Queue die schnellsten Ergebnisse. Ein wesentlicher Unterschied im Vergleich zu den ebenfalls getesteten TreeSet-, Skip-Liste- und Fibonacci-Heap-Implementierungen war allerdings erst bei Wegsuchen mit hohen Dijkstra-Rängen zu beobachten.
- Als gänzlich ungeeignet erwies sich die zusätzliche Knotenindizierung per Array. Die Dauer für das Anlegen der Index-Arrays dominiert die Dauer der Gesamtsuche.
- Eine intelligente Knotenblockierung kann viele Iterationen bei der Wegsuche einsparen. Die wenigsten Iterationen benötigt die *erweiterte Knotenblockierungen* (Verfahren 13, S. 149), aber deutlich am schnellsten arbeitet die *einfache Knotenblockierung* (Verfahren 13 ohne die Zeilen 6–19).
- Der Einsatz der zielgerichteten Wegsuche ist in Contraction Hierarchies sehr vorteilhaft. Verfahren 14, S. 155, benötigt gegenüber der Standard-Wegsuche in Contraction Hierarchies deutlich weniger Dijkstra-Iterationen. Bei der Suche nach kürzesten Wegen können mehr als die Hälfte der Iterationen eingespart werden und bei der Suche nach schnellsten Wegen ist die Ersparnis etwa ein Drittel der Iterationen (siehe Abbildung 5.26, S. 192).
- Aufgrund der bidirektionalen Vorgehensweise lässt sich die Vorwärts-Wegsuche gut mit der Rückwärts-Wegsuche parallelisieren. Die Schrittweite, d. h. die Anzahl unabhängiger Iterationen in beide Suchrichtungen, sollte hierbei jedoch auf die Kartengröße angepasst sein, um unnötige Iterationen zu vermeiden.

6 Contraction Hierarchies mit Abbiegebeschränkungen

Inhalt

Dieses Kapitel widmet sich der Erweiterung der Contraction Hierarchies um die Berücksichtigung von Abbiege*verboten* und Abbiege*geboten*, wobei *Abbiegebeschränkungen* hier als Sammelbegriff dient. Es wird gezeigt, welche Unterschiede sich bei der Erzeugung der Hierarchie ergeben, sowohl bei der Herangehensweise als auch beim Ergebnis. Darüber hinaus werden zwei angepasste Wegsucheverfahren vorgestellt und miteinander verglichen.

Das Kapitel ist wie folgt gegliedert: Im nachfolgenden Abschnitt werden Abbiegebeschränkungen formal eingeführt. Wie diese im Rahmen von Wegsuchen berücksichtigt werden können, ist in Abschnitt 6.2 detailliert zunächst für das Dijkstra-Verfahren dargelegt. Diese Erkenntnisse werden danach in Abschnitt 6.3 auf Wegsuchen in Contraction Hierarchies übertragen. Deren *Erzeugung* unter Berücksichtigung von Abbiegebeschränkungen beschreibt Abschnitt 6.4. Der Rest des Kapitels ist analog zu Kapitel 5 aufgebaut: Nach einer Diskussion der Implementierungsdetails in Abschnitt 6.5 folgen systematische Benchmarks für die Wegsuche in Abschnitt 6.6.

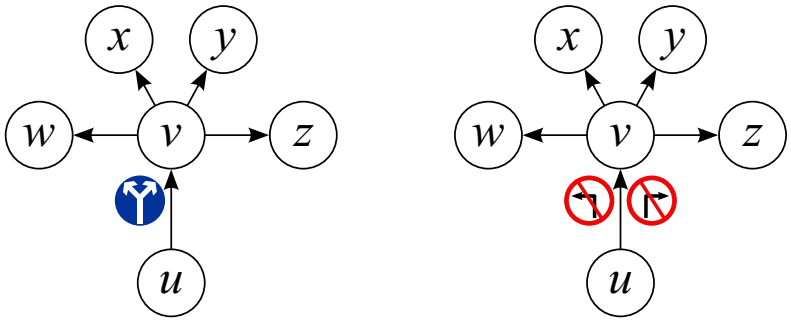
6.1 Notation und Modellierung von Abbiegebeschränkungen

Abbiegebeschränkungen beziehen sich in dieser Arbeit stets auf den Übergang zweier durch einen Knoten verbundener Pfeile.[1] Dabei muss der Endknoten des ersten Pfeils mit dem Startknoten des zweiten Pfeils übereinstimmen. Der Begriff *Abbiegung* umfasst alle solche Pfeilpaare, selbst wenn die beiden Pfeile zusammen einen 180°-Winkel bilden und somit, auf den Graphen einer Straßenkarte übertragen, eine „Geradeaus-Fahrt“ beschreiben.

Die Darstellung erfolgt folglich pfeilbasiert. $\ddot{U}((u,v),(v,w))$ symbolisiere den Übergang, d. h. das „Abbiegen“, von Pfeil (u,v) nach Pfeil (v,w) mit $(u,v),(v,w) \in E$.

In digitalen Straßenkarten finden sich sowohl Abbiege*gebote* als auch Abbiege*verbote*. Beide Arten der Abbiegebeschränkungen lassen sich ineinander überführen. Dies wird am Beispiel aus Abbildung 6.1, S. 201, verdeutlicht. Im dargestellten Graphen sind direkt nach Pfeil (u,v) ausschließlich die Pfeile (v,x) und (v,y) zulässig; die Pfeile (v,w) und (v,z) sind dagegen verboten. In Abbildung 6.1a ist der Sachverhalt durch Abbiege*gebote* formuliert und in Abbildung 6.1b dagegen mithilfe von Abbiege*verboten*.

[1] In sehr seltenen Fällen kann eine Abbiegebeschränkung auf diese Weise nicht modelliert werden. Ein Beispiel aus den USA ist in `http://wiki.openstreetmap.org/w/index.php?title=Relation:restriction&oldid=745411` (Stand vom 7. März 2012) beschrieben. Hier dürfen Autofahrer, die einen bestimmten Highway verlassen und dadurch auf einen Freeway einbiegen, nach kurzer Fahrt über den Freeway dessen direkt nachfolgende Abfahrt nicht verwenden.



(a) Modellierung durch Abbiegegebote

(b) Modellierung durch Abbiegeverbote

Abbildung 6.1: Zusammenhang zwischen Abbiegegeboten und -verboten
Beide Teilabbildungen stellen denselben Sachverhalt dar.

Der sprachlichen Einfachheit halber werden Abbiegebeschränkungen in diesem Kapitel daher – sofern nicht explizit anders angegeben – behandelt, als seien sie ausschließlich durch Abbiege*verbote* modelliert. Die Menge aller Abbiegeverbote in einem Graphen werde mit TR ausgedrückt[2], so dass ein gerichteter Graph mit Abbiegebeschränkungen und Kantenbewertungsfunktion c in der Form $G = (V, E, c, TR)$ geschrieben werden kann. Die Existenz eines Abbiegeverbots von Pfeil (u, v) nach Pfeil (v, w) in G lässt sich dann mit der Formulierung: $\ddot{U}((u, v), (v, w)) \in TR$ ausdrücken. Übertragen auf den Beispielgraphen in Abbildung 6.1 gilt also $TR = \{\ddot{U}((u, v), (v, w)), \ddot{U}((u, v), (v, z))\}$.

Die bisherige Definition von Wegen[3] beinhaltet noch nicht, dass diese Abbiegebeschränkungen berücksichtigen. Es werden daher *Abbiegebeschränkungen berücksichtigende Wege* wie folgt definiert:

Ein Weg $p = \left\langle (u_0, u_1), (u_1, u_2), \ldots, (u_{|p|-1}, u_{|p|}) \right\rangle$ mit $(u_i, u_{i+1}) \in E$ für $0 \leq i < |p|$ ist ein *Abbiegebeschränkungen berücksichtigender Weg*, wenn für ihn gilt:

$$\ddot{U}((u_i, u_{i+1}), (u_{i+1}, u_{i+2})) \notin TR \text{ für } 0 \leq i < |p| - 1.$$

6.2 Wegsucheverfahren für Wege mit Abbiegebeschränkungen

Der Einbezug von Abbiegebeschränkungen auf die Wegsuche mag auf den ersten Blick recht einfach erscheinen: Ein Wegsucheverfahren wie das Dijkstra-Verfahren müsste lediglich darauf achten, bei der Knoten-Expansion keine Abbiegeverbote zu verletzen. Verfahren 16, S. 202, skizziert die hierfür notwendige Veränderung des Dijkstra-Verfahrens.

[2] vom englischen *turn restrictions*

[3] siehe Abschnitt 1.3.1 auf Seite 30; dort wurde $|p|$ als die Anzahl der in p beinhalteten Pfeile definiert

Verfahren 16 Naive Anpassung des Dijkstra-Algorithmus für Abbiegebeschränkungen (1:1-Suche)

Gegeben: Nicht-negativ bewerteter Digraph $G = (V, E, c, TR)$, Startknoten $s \in V$

```
1:  ...                                              ▸ Initialisierung identisch zu Verfahren 5, S. 91,
2:  while Horizontmenge ≠ ∅ do
3:    ...                                            ▸ Schritt 1 aus Verfahren 5, S. 91,
4:    for all v' ∈ 𝒩(v) do                           ▸ Schritt 2: Expansion von v
5:      if Es existiert ein Ü((v.vorgänger, v), (v, v')) ∈ TR then
6:        goto: 4
7:      end if
8:      ...                                          ▸ Rest identisch zu Verfahren 5, S. 91,
9:    end for
10: end while
```

Ergebnis: Wie bei Verfahren 5, S. 91, mit dem Unterschied, dass kein Abbiegeverbot verletzt wurde.

Zwar findet das veränderte Dijkstra-Verfahren in manchen Fällen tatsächlich den kürzesten Weg, wie etwa in Abbildung 6.2 dargestellt. Hervorzuheben ist für dieses Beispiel, dass der kürzeste Weg nicht mehr aus lauter kürzesten Teilwegen besteht: Der Teilweg $\langle v, w, x \rangle$ ist kein kürzester Weg von v nach x.

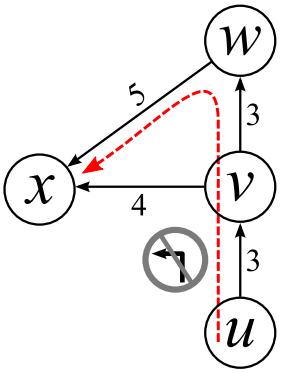


Abbildung 6.2: Beispiel für kürzesten Weg mit nicht-kürzestem Teilweg
Betroffen ist der Teilweg zwischen Knoten v und Knoten x. Rot eingezeichnet ist der von Verfahren 16 gelieferte Weg von Knoten u nach Knoten x.

Schnell wird jedoch deutlich, dass sich so nicht *alle* kürzesten Wege finden lassen. Ein Gegenbeispiel liefert die Suche nach dem kürzesten Weg von Knoten u nach Knoten x auf dem leicht veränderten Graphen in Abbildung 6.3, S. 203. Verfahren 16 liefert hier *keinen* Weg.

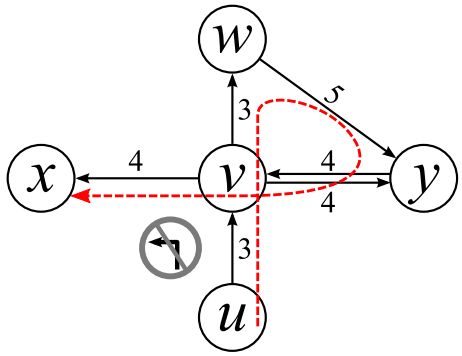


Abbildung 6.3: Beispiel für optimalen Weg mit Zyklus
Betroffen ist der Weg zwischen Knoten u und Knoten x.
Rot eingezeichnet ist der kürzeste zulässige Weg zwischen diesen beiden Knoten. Verfahren 16 findet diesen Weg nicht.

Der genauerer Blick auf den Verlauf von Horizont- und Lösungsmenge während des Verfahrens in Tabelle 6.1 verrät den Grund: Knoten v wird nach der zweiten Iteration Teil der Lösungsmenge. Die Expansion findet danach korrekterweise nicht in Richtung Knoten x statt. Doch anschließend berücksichtigt das Verfahren Knoten v nicht mehr, so dass Knoten x nie erreicht wird und die Wegsuche aufgrund der leeren Horizontmenge erfolglos abbricht.

Iteration	Horizontmenge	Lösungsmenge
0	$\{u\}$	$\emptyset$
1	$\{v\}$	$\{u\}$
2	$\{w\}$	$\{u, v\}$
3	$\{y\}$	$\{u, v, w\}$
4	$\emptyset$	$\{u, v, w, y\}$

Tabelle 6.1: Horizont- und Lösungsmenge bei der Wegsuche von Knoten u nach x im Graphen in Abbildung 6.3 gemäß Verfahren 16, S. 202,

Offensichtlich können Abbiegebeschränkungen dazu führen, dass aufgrund von Zyklen in den gesuchten Wegen manche Knoten *mehrfach* expandiert werden müssen. In extremen Fällen kann der optimale Weg sogar mehrere Zyklen um *denselben* Knoten besitzen. Abbildung C.1, S. 339, im Anhang zeigt diesen äußerst seltenen Fall.

Ein leicht verändertes Dijkstra-Verfahren, das die Mehrfachaufnahme von Knoten erlaubt, fände jedoch immer noch nicht den korrekten Weg $p^* = \langle u, v, w, y, v, x\rangle$. Stattdessen lieferte es den Weg $p' = \langle u, v, y, v, x\rangle$. Dieser ist zwar kürzer, beinhaltet aber einen Fahrtrichtungswechsel von Pfeil (v, y) bei Knoten y nach (y, v). Derartige Wege sollten aus verschiedenen Gründen vermieden werden:

Wenden vermeiden

- Möglicherweise sind die Fahrspuren baulich getrennt, so dass eine Wende überhaupt nicht möglich ist.
- Je nach Fließgeschwindigkeit des Verkehrs stellt eine Wende ein Risiko für andere Verkehrsteilnehmer dar.
- Die Fahrbahn kann in einigen Fällen zu schmal für eine zügige Wende sein. Als Beispiel seien hier Serpentinen in Gebirgen genannt.

Zulässige Wege

Für den Rest dieser Arbeit – und im Speziellen für dieses Kapitel – werden Wege daher als *zulässig* bezeichnet, wenn sie

1. alle Abbiegebeschränkungen des Graphen berücksichtigen[4] und
2. keine Wenden beinhalten.

Grundsätzlich stellen kürzeste Wege mit Zyklen für die bisher in dieser Arbeit vorgestellten Wegsucheverfahren eine Herausforderung dar. Dennoch genießt das Thema Abbiegebeschränkungen in der Literatur nur geringe Aufmerksamkeit. Die nachfolgenden drei Abschnitte beschreiben die am weitesten verbreiteten Ansätze. Der daran anschließende Abschnitt 6.2.4 stellt einen neuen adaptiven Ansatz vor. In jedem der Abschnitte wird zunächst die Idee des Verfahrens vorgestellt, dann folgt eine kurze Diskussion der Vor- und Nachteile des Verfahren und schließlich Überlegungen zur Eignung des präsentierten Verfahrens für den Einsatz bei Contraction Hierarchies.

6.2.1 Graph-Transformation per Knotensplitting

Ein pragmatischer Ansatz für eine Wegsuche mit Abbiegebeschränkungen findet sich bei Schaechterle u. Braun [1977, S. 11]. Hierfür wird nicht das Wegsucheverfahren, sondern der zugrunde liegende Graph G transformiert.

Jeder Knoten $v \in V$ des nicht-negativ bewerteten, gerichteten Ursprungsgraphen $G = (V, E, c, TR)$ wird dafür im veränderten Graphen $G_{KS} = (V_{KS}, E_{KS}, c_{KS})$ gesplittet. Hierbei steht das Kürzel *KS* für *Knotensplitting*. Es entsteht pro zu v adjazenten Pfeil ein neuer Knoten und v wird durch die neuen Knoten ersetzt.[5]

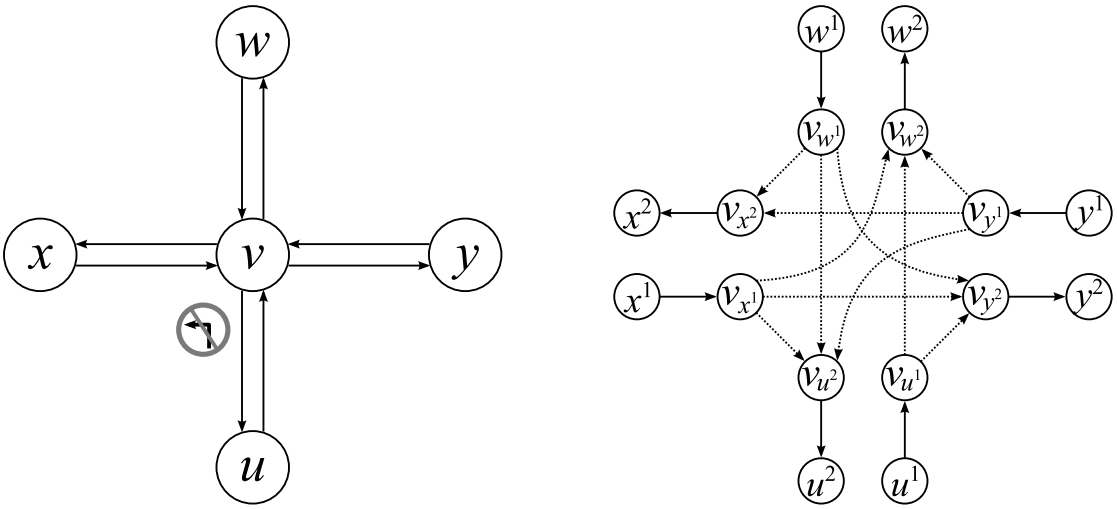
Neue Knoten, die aus demselben Knoten des Ursprungsgraphen hervorgingen, werden paarweise mit neuen künstlichen Pfeilen verbunden. Diese Pfeile lassen sich als Abbiegevorgang innerhalb eines Weges verstehen. Daher werden in G_{KS} nur solche künstliche Pfeile erzeugt, die für erlaubte Abbiegevorgänge stehen. Auch Wendeverbote (siehe S. 203) lassen sich so einfach erreichen, indem keine künstlichen Pfeile erzeugt werden, die Wenden repräsentieren.

Abbiegekosten

Beinhaltet G Informationen zu *Abbiegekosten* als Verallgemeinerung von Abbiegebeschränkungen, dann werden diese Kosten den künstlichen Pfeilen zugeordnet. Ansonsten erhalten sie eine Null-Bewertung. Abbildung 6.4, S. 205, zeigt ein schematisches Beispiel, an dem deutlich wird, dass dieses Vorgehen anschaulich auch als „Hineinzoomen“ in Kreuzungen interpretiert werden kann. Im Beispiel gibt die Knotenbezeichnung im veränderten Graphen G_{KS} Aufschluss darüber, aus welchem Pfeil des Ursprungsgraphen G der Knoten entstanden ist. Beispielhaft existiert Knoten $v_{u^1} \in V_{KS}$ aufgrund von Pfeil $(u, v) \in E$. Der zusätzliche Zahlenindex unterscheidet für die „äußeren“ Knoten der dargestellten Kreuzung in G die durch Splitting entstandenen Knoten in G_{KS}.

[4] zur Definition von Abbiegebeschränkungen berücksichtigenden Wegen siehe Seite 201

[5] *Splitting* bedeutet hier also nicht ausschließlich das Aufteilen in exakt zwei neue Knoten. Stattdessen können beim Splitting aus einem Knoten *mehrere* neue Knoten hervorgehen.



(a) Ursprungsgraph vor dem Knotensplitting

(b) Veränderter Graph nach dem Knotensplitting
Die neuen künstlichen Pfeile innerhalb der Kreuzung sind gepunktet dargestellt.

Abbildung 6.4: Idee beim Knotensplitting
Die verbotene Abbiegung von Knoten u über v nach x ist im veränderten Graphen nicht mehr möglich, denn Pfeil (v_{u^1}, v_{x^2}) existiert nicht. Wege mit Wenden sind ebenfalls ausgeschlossen, da die Pfeile (v_{u^1}, v_{u^2}), (v_{w^1}, v_{w^2}), (v_{x^1}, v_{x^2}) und (v_{y^1}, v_{y^2}) nicht existieren. Bezogen auf das Beispiel in Abbildung 6.3, S. 203, ist hier das Fehlen von Pfeil (v_{y^1}, v_{y^2}) hervorzuheben.

Vorteile der Graph-Transformation per Knotensplitting Auf den veränderten Graphen G_{KS} können knotenbasierte Wegsucheverfahren uneingeschränkt eingesetzt werden. Die gefundenen Wege berücksichtigen aufgrund der Konstruktion von G_{KS} alle Abbiegebeschränkungen. Die Wege können – übertragen auf den Ursprungsgraphen G – sogar Zyklen beinhalten, denn in G_{KS} enthalten sie keinen Knoten doppelt.

Eine Implementierung, die den Originalgraphen unverändert lässt, muss die Abbiegebeschränkungen gesondert speichern, um auf sie Zugriff zu haben. Bei einer Wegsuche muss dann zu jedem erreichten Knoten – etwa in einem Index – nachgeschlagen werden, ob an diesem Knoten eine Abbiegebeschränkung existiert. Bei der Graph-Transformation per Knotensplitting entfällt dieser zusätzliche Aufwand, was sich beschleunigend auf die Wegsuche auswirkt. Zusätzlich können Abbiegekosten als Verallgemeinerung von Abbiegebeschränkungen auf einfache Art und Weise in den Graphen über die neuen künstlichen Pfeile integriert werden.

Nachteile der Graph-Transformation per Knotensplitting Da für jeden Knoten v des Ursprungsgraphen G im veränderten Graphen G_{KS} neue Knoten mit *denselben* Koordinaten wie v existieren, entsteht eine Mehrdeutigkeit für Wegsuchen, die bei v beginnen. Schmid [2001, S. 55] umreißt hierzu eine Lösung durch Einfügen eines weiteren Knotens in G_{KS}. Dieser müsste mit allen Startknoten der aus v führenden Pfeile verbunden werden und wird dann als Startknoten derartiger Wegsuchen verwendet. Spiegelbildlich verhält es sich mit Wegsuchen, die bei v enden.

Weiterhin vergrößert das Knotensplitting den Graphen G_{KS} im Vergleich mit G ganz erheblich. Denn an jedem Knoten u aus G mit g_u^- eingehenden und g_u^+ ausgehenden Pfeilen[6] existieren in G_{KS} $g_u^- + g_u^+$ Knoten, die ihrerseits – abzüglich der Abbiegebeschränkungen und Wendeverbote – durch $g_u^- \times g_u^+$ Pfeile zu verbinden sind.

Hiervon beeinflusst sind natürlich auch die Wegsucheverfahren, die ihrerseits im veränderten Graphen langsamer zum Ziel führen. Wurde bei der Standard-Dijkstra-Suche in G ein Knoten u nur höchstens einmal in die Lösungsmenge aufgenommen, können in G_{KS} bei einer Suche in Vorwärtsrichtung g_u^- der durch Splitting von Knoten u entstandenen Knoten aufgenommen werden. Bei einer Rückwärtssuche sind es entsprechend bis zu g_u^+. Für große Graphen ist dieser Ansatz deshalb grundsätzlich schlechter geeignet.

Eignung für Contraction Hierarchies Die Graph-Transformation per Knotensplitting lässt sich methodisch problemlos mit Contraction Hierarchies kombinieren. Hierzu muss lediglich der Ausgangsgraph G wie beschrieben erweitert werden, um anschließend aus G_{KS} die Hierarchie zu erzeugen. Wegsuchen in der CH berücksichtigen dann automatisch Abbiegebeschränkungen und sogar allgemein Abbiegekosten.

Allerdings wirkt sich die erhöhte Anzahl von Knoten und Pfeilen in G_{KS} ebenfalls auf den um Shortcut-Pfeile angereicherten Graphen G_{CH} der Contraction Hierarchy aus. Die Anzahl von Shortcut-Pfeilen in G_{CH} erhöht sich um ein Vielfaches im Vergleich zu einer Contraction Hierarchy, die auf G aufbaut.

6.2.2 Pfeilbasierte Suche für Wege mit Abbiegebeschränkungen

Eine Möglichkeit, optimale Wege in einem Graphen mit Abbiegebeschränkungen zu finden, ohne den Graphen dabei zu verändern, ist die Umstellung von knotenbasierten Wegsuchen auf pfeilbasierte.

Werden bei der Dijkstra-Suche in der Horizont- und Lösungsmenge *Pfeile* anstelle von Knoten aufgenommen, ermöglicht dies implizit eine mehrfache Expansion ein und desselben Knotens. Die Suche erfolgt dann ebenfalls nicht zwischen Knoten, sondern zwischen einem Start- und einem Endpfeil. Ein optimaler Weg zwischen einem Startpfeil (s, s') und einem Zielpfeil (t', t) ist dann ein kostenminimaler, zulässiger Weg, der mit (s, s') beginnt und mit (t', t) endet.

Optimale Wege zwischen Pfeilen

[6] zur Definition des positiven und negativen Knotengrads siehe Seite 30

Die Sortierung der Pfeile in der Horizontmenge erfolgt nach den Kosten des implizierten Weges bis inklusive ihrer Endknoten. Ansonsten ist Verfahren 17 strukturell gleich zu Verfahren 6, S. 93.

Verfahren 17 Pfeilbasierter Dijkstra-Algorithmus (1:1-Suche)

Gegeben: Nicht-negativ bewerteter, gerichteter $G = (V, E, c, TR)$,
Startpfeil (s, s'), Zielpfeil (t', t) mit $(s, s'), (t', t) \in E$

1: $(s, s').\mathit{distanz} \coloneqq c((s, s'))$ ▸ Initialisierung
2: $(u, v).\mathit{distanz} \coloneqq \infty$, $(u, v).\mathit{vorgänger} \coloneqq \mathit{NULL}$ $\forall (u, v) \in E \setminus \{(s, s')\}$
3: $\mathit{Lösungsmenge} \coloneqq \emptyset$, $\mathit{Horizontmenge} \coloneqq \{(s, s')\}$
4: **while** $\mathit{Horizontmenge} \neq \emptyset$ **do**
5: Wähle (u, v) mit $(u, v).\mathit{distanz} = \min\{(u', v').\mathit{distanz} \mid (u', v') \in \mathit{Horizontmenge}\}$
6: **if** $(u, v) = (t', t)$ **then**
7: Abbruch des Verfahrens
8: **end if**
9: Entferne (u, v) aus *Horizontmenge*
10: Nimm (u, v) in *Lösungsmenge* auf
11: **for all** $(v, w) \in E \mid w \neq u$ **do** ▸ Expansion von (u, v)
12: **if** Es existiert ein $\ddot{U}((u, v), (v, w)) \in TR$ **oder** $(v, w) \in \mathit{Lösungsmenge}$ **then**
13: **goto:** 11
14: **end if**
15: **if** $(u, v).\mathit{distanz} + c((v, w)) < (v, w).\mathit{distanz}$ **then**
16: $(v, w).\mathit{vorgänger} \coloneqq (u, v)$ ▸ (besserer) Weg zu (v, w) gefunden
17: $(v, w).\mathit{distanz} \coloneqq (u, v).\mathit{distanz} + c((v, w))$
18: **if** $(v, w) \notin \mathit{Horizontmenge}$ **then**
19: Nimm (v, w) in *Horizontmenge* auf
20: **end if**
21: **end if**
22: **end for**
23: **end while**

Ergebnis: Ein zulässiger, kürzester mit (s, s') beginnender und mit (t', t) endender Weg ist rekursiv konstruierbar aus $(t', t).\mathit{vorgänger}$. Es gilt: $(t', t).\mathit{distanz} = c((s, s')) + \mathrm{dist}(s', t) + c((t, t'))$. Ist $(t', t).\mathit{vorgänger}$ nicht gesetzt, existiert kein solcher Weg.

Knoten als Start oder Ziel

Eine leichte Anpassung erlaubt darüber hinaus auch Knoten als Start und Ziel. Dazu müssen in der Initialisierungsphase (Zeile 3 in Verfahren 17, S. 207) anstatt des Startknotens s alle von s wegführenden Pfeile in die Horizontmenge aufgenommen werden. Ein kürzester Weg zum Zielknoten t ist dann gefunden, sobald t Endknoten desjenigen Pfeils ist, der zuletzt in die Lösungsmenge aufgenommen wurde.

Vorteile der pfeilbasierten Suche Die pfeilbasierte Wegsuche arbeitet auf dem Ursprungsgraphen G, so dass kein zusätzlicher statischer Speicherplatz benötigt wird. Lediglich während der Suche enthalten Horizont- und Lösungsmenge der Dijkstra-Suche mehr Elemente als bei einer knotenbasierten Suche.

Im Vergleich zur Graph-Transformation per Knotensplitting aus dem vorangegangenen Abschnitt benötigt die pfeilbasierte Dijkstra-Wegsuche weniger Iterationen. Zwar kann in beiden Ansätzen ein Knoten implizit so oft Teil der Lösungsmenge werden, wie Pfeile in ihn hineinführen. Doch im Gegensatz zum durch Knotensplitting erweiterten Graphen G_{KS} enthalten die Wege in G keine künstlichen Pfeile. Die Dijkstra-Iterationen, welche die Endknoten dieser künstlichen Pfeile expandieren, entfallen daher.

Nachteile der pfeilbasierten Suche Im Gegensatz zur Graph-Transformation per Knotensplitting können allgemeine Abbiegekosten nicht so elegant in die Wegsuche integriert werden. Sie müssten stattdessen entweder bei den Knoten oder „graphextern“ beispielsweise in einer Kostentabelle hinterlegt werden (vgl. hierfür bspw. [Geisberger u. Vetter, 2011, S. 107] oder [Winter, 2002, S. 345]). Verglichen mit der Bewertung der künstlichen Pfeile bei der Graph-Transformation per Knotensplitting erfordern beide Varianten in der Implementierung einen zusätzlichen Aufwand für das Nachschlagen der Abbiegekosten.

Des Weiteren werden bei der Wegsuche viele Endknoten von Pfeilen mehrfach aufgenommen und expandiert, ohne dass der Weg zu ihnen Abbiegebeschränkungen beinhaltet. Dadurch entstehen zusätzliche Dijkstra-Iterationen. Wie sich diese zu einem gewissen Teil verhindern lassen, wird in Abschnitt 6.2.4, S. 211 ff., gezeigt.

Eignung für Contraction Hierarchies Der pfeilbasierte Ansatz ist grundsätzlich auf Contraction Hierarchies übertragbar. Sowohl die Hierarchie-Erzeugung als auch die Wegsuche müssen hierfür allerdings umgestellt werden. Geisberger u. Vetter [2011] beschreiben die hierfür notwendigen Änderungen. Die Details werden ausführlich in Abschnitt 6.3, S. 218 ff., und Abschnitt 6.4, S. 230 ff., erläutert.

6.2.3 Knotenbasierte Suche im Pseudographen

Nur zwei Jahre nach Dijkstras Veröffentlichung seiner Wegsuche beschrieb Caldwell [1961] seinen Ansatz zur Integration von Abbiegekosten. Diese Idee setzt wiederum auf eine Veränderung des ursprünglichen Graphen.

Pseudograph

Caldwell [1961, S. 107 f.] überführt den Ursprungsgraphen in einen neuen Graphen, den er *Pseudograph*[7] nennt. Dieser besitzt für jeden Pfeil des Ursprungsgraphen einen Pseudoknoten und für jede direkte Folge von je zwei Pfeilen des Ursprungsgraphen einen

[7] im englischen Original: *pseudo network*

Pseudopfeil. Das Besondere am Pseudograph ist, dass sämtliche knotenbasierten Wegsucheverfahren in ihm Pseudopfade finden, die – übertragen auf den Ursprungsgraphen – für Pfade stehen, die Abbiegebeschränkungen berücksichtigen.

Eine einfache formale Definition für den Pseudographen findet sich bei Winter [2002, S. 348 ff.], der das Konstrukt jedoch einen *pseudo-dualen Graphen* nennt. Er hebt damit hervor, dass die nachfolgend beschriebene Überführung eines Ursprungsgraphen in einen Pseudographen keine bijektive Funktion ist (vgl. [Winter, 2002, S. 350]).

Definition eines Pseudographen[8] Gegeben sei ein nicht-negativ bewerteter, gerichteter Ursprungsgraph $G = (V, E, c)$ mit Abbiegekostenfunktion $\theta : E \times E \rightarrow \mathbb{R} \cup \{\infty\}$. Abbiegeverbote sind durch unendlich hohe Abbiegekosten modelliert. Der gerichtete Graph $G_P = (V_P, E_P, c_P)$ ist der Pseudograph zu G, wenn gilt:

- Für jeden Pfeil $(u, v) \in E$ existiert ein Knoten $v \in V_P$ und ϕ sei eine bijektive Funktion, die beide ineinander überführt, d. h., $\phi((u, v)) = v$ und $\phi^{-1}(v) = (u, v)$.

- Für alle paarweise verbundenen Pfeile $(u, v), (v, w) \in E$ aus dem Ursprungsgraphen existiert im Pseudograph zwischen den entsprechenden Knoten ein Pfeil $(\phi((u, v)), \phi((v, w)))$. Eine Ausnahme bilden Pfeilpaare, für die ein Abbiegeverbot existiert. Für Pfeile $(u, v), (v, w) \in E$ mit Abbiegekosten $\theta((u, v), (v, w)) = \infty$ besitzt der Pseudograph explizit keinen Pfeil.

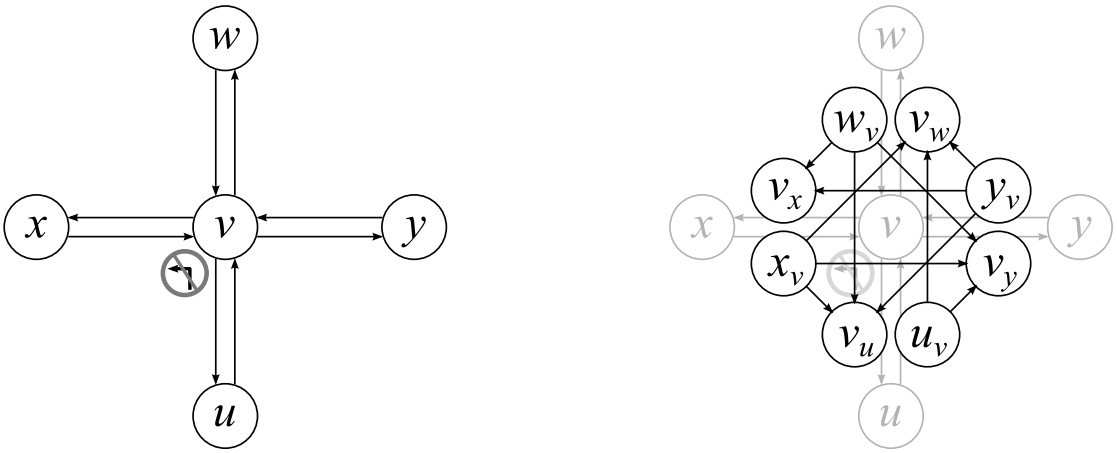
- Darüber hinaus existieren keine weiteren Pfeile im Pseudographen.

- Die Kosten eines Pfeils des Pseudographen ergeben sich aus der Summe der Kosten der zugrunde liegenden Pfeile des Ursprungsgraphen und der Abbiegekosten an ihrem Verbindungsknoten: $c_P(\phi((u, v)), \phi((v, w))) = c((u, v)) + c((v, w)) + \theta((u, v), (v, w))$ mit $(u, v), (v, w) \in E$.

Pfade auf dem Ursprungsgraphen mit Fahrtrichtungswechseln[9] lassen sich im Übrigen ausschließen, indem im Pseudographen bestimmte Pseudopfeile eliminiert werden. Dazu sind alle Pseudopfeile zu löschen, die für eine solche Wende im Ursprungsgraphen stehen. Die sich dadurch ergebende Pfeilmenge $E'_P \subseteq E_P$ lässt sich wie folgt definieren: $E'_P = E_P \setminus \{(\phi((u, v)), \phi((v, u))) \mid (u, v), (v, u) \in E\}$.

Abbildung 6.5, S. 210, zeigt anhand eines einfachen Beispiels den Zusammenhang zwischen Ursprungsgraphen und Pseudographen. Die Bezeichnung der Knoten im Pseudographen ist dabei an die Bezeichnung der Pfeile des Ursprungsgraphen angelehnt, so dass aus Pfeil (u, v) des Ursprungsgraphen der Pseudoknoten u_v wurde, d. h., es gilt $\theta((u, v)) = u_v$. Das Beispiel beinhaltet ein Abbiegeverbot von Pfeil (u, v) nach (v, x), so dass im Pseudographen der entsprechende Pfeil (u_v, v_x) fehlt.

[8] in Anlehnung an Winter [2002, S. 348 ff.]
[9] siehe S. 203, Stichwort „Wenden vermeiden“



(a) Ursprungsgraph

(b) Ursprungsgraph (grau) mit überlagertem Pseudograph (schwarz)

Abbildung 6.5: Ursprungsgraph und Pseudograph
Die Bezeichnung der Pseudoknoten erfolgt nach dem Schema $u_v = \theta((u, v))$. Die verbotene Abbiegung von Knoten u über v nach x ist im Pseudographen nicht mehr möglich, da die hier kein Pfeil zwischen den Pseudoknoten u_v und v_x existiert.

Vorteile der knotenbasierten Suche im Pseudographen Der Ansatz hat dieselben Vorteile wie die Graph-Transformation per Knotensplitting[10]. Darüber hinaus ist der Zusammenhang zwischen den Knoten und Pfeilen des Ursprungsgraphen und denen des Pseudographen einfacher nachzuvollziehen als bei der Graph-Transformation per Knotensplitting. Das Modell erscheint dadurch kognitiv leichter zugänglich.

Nachteile der knotenbasierten Suche im Pseudographen Wie auch bei der Graph-Transformation per Knotensplitting benötigt der Pseudograph deutlich mehr Speicherplatz als der Ursprungsgraph aufgrund der größeren Knoten- und Pfeilanzahl.[11] Dies führt wiederum zu langsameren (Dijkstra-)Wegsuchen im Pseudographen.

Da knotenbasierte Wegsuchen auf dem Pseudographen $G_P = (V_P, E_P, c_P)$ pfeilbasierte Wegsuchen auf dem Ursprungsgraphen $G = (V, E, c)$ repräsentieren, ist eine Wegsuche von Knoten s nach Knoten t $(s, t \in V)$ nur bedingt möglich. Schließlich müssen im Pseudographen hierfür sämtliche Knoten $\phi((s, u))$ als Startknoten und sämtliche Knoten $\phi((v, t))$ als Endknoten mit $(s, u), (v, t) \in E$ berücksichtigt werden. Der gesuchte kürzeste Weg ist dann der kürzeste unter allen diese Startknoten mit diesen Endknoten verbindenden Wegen (vgl. [Winter, 2002, S. 350]).

Winter [2002, S. 350] schlägt alternativ vor, für jede s-t-Wegsuche das Problem mithilfe zweier zusätzlicher, temporärer Pseudoknoten v_s und v_t im Pseudographen zu beheben. Hierzu werden v_s bzw. v_t vor der Suche über zusätzliche, temporäre und wegkostenneutrale Pseudopfeile mit allen Pseudoknoten $\phi((s, u))$ bzw. $\phi((v, t))$ aus V_P verbunden.

[10] siehe S. 205 f.

[11] Eine detaillierte Analyse der Speichereffizienz eines Pseudographen findet sich bei Winter [2002, S. 352 ff.].

Die Lösung der Wegsuche von v_s nach v_t im Pseudographen repräsentiert dann den kürzesten Weg von s nach t im Ursprungsgraphen und die temporären Pseudoknoten und Pseudopfeile können wieder entfernt werden. Offensichtlich entsteht aber auch bei dieser Lösung für jede Wegsuche ein Mehraufwand durch die Identifikation der notwendigen temporären Pseudoknoten und Pseudopfeile.

Eignung für Contraction Hierarchies Die knotenbasierte Suche kann grundsätzlich auch für Contraction Hierarchies eingesetzt werden, indem diese auf einem Pseudographen aufbauen. Durch die stark erhöhte Anzahl von Pfeilen im Pseudographen im Vergleich zum Ursprungsgraphen entstehen allerdings wiederum erheblich mehr Shortcut-Pfeile in der Hierarchie. Zusätzlich bleibt bei einer einen Pseudographen erweiternden Contraction Hierarchy das oben beschriebene Problem bei der Wegsuche von einem Start- zu einem Zielknoten.

6.2.4 Adaptive Suche für Wege mit Abbiegebeschränkungen

Eine Möglichkeit, Abbiegebeschränkungen bei der Wegsuche zu berücksichtigen, ohne dabei jedoch den Graphen zu verändern, ist die oben erwähnte pfeilbasierte Suche. Deren größter Nachteil besteht darin, dass im Rahmen einer Dijkstra-Wegsuche jeder Knoten des Graphen grundsätzlich mehrfach implizit in die Lösungsmenge aufgenommen werden kann, was sich negativ auf die Suchgeschwindigkeit auswirkt.

Diesen Mangel greift die adaptive Wegsuche auf (vgl. [Nowak u. a., 2014, S. 569 ff.]). Sie basiert auf der grundlegenden Erkenntnis, dass bei einer Wegsuche nur solche Knoten mehrfach aufgenommen werden müssen, zu denen der Weg – vom Startknoten aus betrachtet – über eine Kreuzung führt, die eine Abbiegebeschränkung besitzt, bzw. präziser formuliert: bei denen dieser Weg einen Pfeil enthält, *von dem ausgehend* eine Abbiegebeschränkung zu einem anderen Pfeil existiert.

Schmid [2001, S. 34 ff.] skizziert die Idee der adaptiven Suche zwar als *Methode der mehrfachen Knotenaufnahme*. Doch er macht einen entscheidenden Fehler, da er ein Verfahren beschreibt, bei dem ein Knoten nur dann mehrfach expandiert werden darf, wenn dieser „*über eine Kante* aufgenommen [wird], von welcher aus ein Abbiegeverbot existiert“ ([Schmid, 2001, S. 35]). Der sprachliche Unterschied zu einem Knoten, der „*über einen Weg erreicht* wird“ (ebd.), legt nahe, dass das dort vorgestellte Verfahren *keine* Knoten mehrfach aufnimmt, die nicht Teil eines Abbiegeverbots sind. Auch das Beispiel bei Schmid [2001, S. 35 f.] enthält keinen solchen denkbaren Fall.

Zur Verdeutlichung des Unterschieds werde dazu Abbildung 6.3, S. 203, ergänzt, so dass der Graph aus Abbildung 6.6, S. 212, entsteht. Der kürzeste Weg von Knoten u nach x ist dann $p = \langle u, v, w, z_1, z_2, w, v, x\rangle$ mit $c(p) = 17$. Offensichtlich muss in diesem Fall nicht nur Knoten v, sondern auch Knoten w zweifach in die Lösungsmenge einer Dijkstra-Wegsuche aufgenommen werden, obwohl Knoten w nicht Teil einer Abbiegebeschränkung ist.

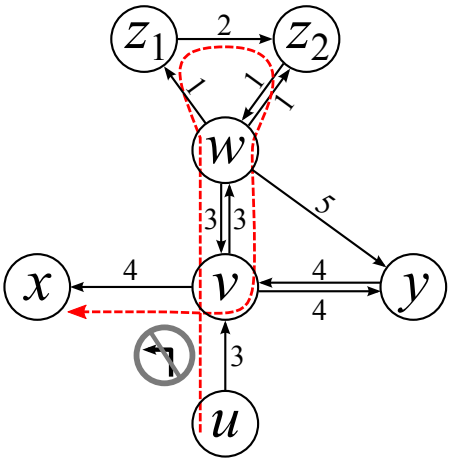


Abbildung 6.6: Optimaler Weg mit Zyklus an „beschränkungsfreiem“ Knoten
Betroffen ist der kürzeste Weg zwischen Knoten u und Knoten x. Sowohl Knoten v als auch Knoten w sind darin zweifach vorhanden.

Eine Dijkstra-Wegsuche muss also speichern, ob ein Knoten v über einen Weg erreicht wurde, der einen Pfeil enthält, von der aus eine Abbiegebeschränkung existiert. In diesem Fall darf v mehrfach in die Horizont- und Lösungsmenge aufgenommen werden.

Damit v aber nicht öfter als nötig in diese Mengen aufgenommen wird, muss unterschieden werden, über welchen Pfeil v erreicht wurde. Dieser Aspekt des Verfahrens ist daher pfeilbasiert und stellt die namensgebende, adaptive Eigenschaft der Wegsuche dar. Im Folgenden wird dargelegt, wie die Wegsuche korrekte Ergebnisse liefern kann und gleichzeitig zu häufiges Expandieren desselben Knotens vermieden wird.

Überlegungen zur mehrfachen Knotenaufnahme Je nachdem, über welchen Weg ein Knoten v erreicht wird, der in die Horizont- oder Lösungsmenge aufgenommen werden soll, muss entschieden werden, ob v mehrfach aufgenommen werden muss. Der sprachlichen Einfachheit halber wird im Folgenden ein Weg $p = \langle s, \ldots, t \rangle$ als *verbotsgefährdet* bezeichnet, wenn er einen Pfeil beinhaltet, von dem *ausgehend* eine Abbiegebeschränkung existiert. Schließlich steht in diesem Fall noch nicht fest, ob es einen kürzeren Weg von s nach t gibt, der jedoch Abbiegebeschränkungen missachtet. Andernfalls sei p *verbotsfrei*.

Verbotsgefährdet vs. verbotsfrei

Im Verlaufe der Dijkstra-Wegsuche „vererbt“ ein Knoten v, der über einen verbotsgefährdeten Weg erreicht wurde, diese Eigenschaft bei seiner Expansion an die von ihm erreichten Knoten. Bezogen auf die Horizont- und Lösungsmenge gilt die Verbotsgefährdung daher für den gesamten bei v beginnenden Teilbaum der Suche.

Nachfolgend wird gezeigt, unter welchen Umständen eine Mehrfachaufnahme eines Knotens v erfolgen muss. Dies ist nur der Fall, wenn v bereits über einen Weg p_{alt} in die Horizontmenge aufgenommen wurde. Nun werde v erneut durch einen zweiten Weg p_{neu} erreicht. Über zwei Szenarien lässt sich die vollständige Fallunterscheidung vornehmen:

Szenario 1 p_{alt} ist *verbotsfrei* (siehe Abbildung 6.7a, S. 213)

Szenario 2 p_{alt} ist *verbotsgefährdet* (siehe Abbildung 6.7b, S. 213)

(a) Szenario 1: v wurde vorab über einen verbots*freien* Weg p_{alt} erreicht

	p_{neu} ist verbotsfrei	p_{neu} ist verbotsgefährdet und erreicht v über...	
		...denselben Pfeil wie p_{alt}	...einen anderen Pfeil als p_{alt}
$c(p_{\text{neu}}) < c(p_{\text{alt}})$	Fall (1) Aktualisiere v	Fall (2) Aktualisiere v	Fall (3) Füge v erneut der Horizontmenge hinzu
$c(p_{\text{neu}}) \geq c(p_{\text{alt}})$	Fall (4) Tue nichts		

(b) Szenario 2: v wurde vorab nur über einen verbots*gefährdeten* Weg p_{alt} erreicht

	p_{neu} ist verbotsfrei	p_{neu} ist verbotsgefährdet und erreicht v über...	
		...denselben Pfeil wie p_{alt}	...einen anderen Pfeil als p_{alt}
$c(p_{\text{neu}}) < c(p_{\text{alt}})$	Fall (5) Aktualisiere v Entferne alle weiteren Vorkommen von v aus der Horizontmenge, die über einen längeren Weg erreicht wurden	Fall (6) Aktualisiere v	Fall (7) Füge v erneut der Horizontmenge hinzu
$c(p_{\text{neu}}) \geq c(p_{\text{alt}})$	Fall (8) Füge v erneut der Horizontmenge hinzu, falls p_{neu} Knoten v über einen anderen Pfeil erreicht als p_{alt}	Fall (9) Tue nichts	

Abbildung 6.7: Fallunterscheidungen bei adaptiver Wegsuche und alternativen Wegen zu einem Knoten
Grün hinterlegt: knotenbasierte Wegsuche (bezogen auf Knoten v)
Weiß hinterlegt: pfeilbasierte Wegsuche (bezogen auf Knoten v)

Bei Szenario 1 sind nur diejenigen Fälle (1)–(3) relevant, für die gilt: $c(p_{\text{neu}}) < c(p_{\text{alt}})$. Ist p_{neu} nach Fall (1) ferner verbotsfrei, bleibt das Verfahren in Bezug auf v knotenbasiert: $v.\mathit{vorgänger}$ und $v.\mathit{distanz}$ werden aktualisiert. Ansonsten wird es bezogen auf v von nun an pfeilbasiert. Dabei ist in dieser Situation noch entscheidend, ob p_{neu} v über denselben Pfeil erreicht wie p_{alt}. Dann nämlich wird v nach Fall (2) nur aktualisiert. Ansonsten wird v bereits in dieser Situation gemäß Fall (3) zum zweiten Mal in die Horizontmenge aufgenommen. Die Fälle (2) und (3) stellen einen Übergang von der knotenbasierten zur pfeilbasierten Suche dar.

Szenario 2 lässt sich nur noch schwer rein textuell erläutern und ist daher stattdessen Abbildung 6.7, S. 213, zu entnehmen. Dort sind sämtliche Details als Übersicht gezeigt. Hervorzuheben ist, dass mit Fall (5) ein Fall existiert, der den Übergang von der pfeil- zur knotenbasierten Suche beschreibt.

Neue Attribute `verbotsfrei` und `erreichtVia`

Zusammenfassend darf v so lange mehrfach in die Horizont- und Lösungsmenge aufgenommen werden, bis v erstmalig über einen verbotsfreien Weg in die Lösungsmenge aufgenommen wird. Verfahren 18, S. 215, stellt die adaptive Wegsuche im Pseudocode dar. Die Eigenschaft, ob ein Knoten über einen verbotsfreien Weg erreicht wurde, ist hier als das boolesche Knotenattribut `verbotsfrei` modelliert. Ebenfalls ein Knotenattribut ist der Pfeil, über den ein Knoten erreicht wurde: `erreichtVia`.

Notation für Mehrfachaufnahme eines Knotens v

Die i-te Aufnahme eines Knotens v in die Horizont- oder Lösungsmenge wird im Pseudocode mit v_i ausgedrückt. Ist dieser Aspekt unbedeutend, wird nur v geschrieben.

Ausgelagert in Verfahren 19, S. 216, und Verfahren 20, S. 216, ist die Beschreibung der Pfeil-Expansionen. Verfahren 19 formuliert die Expansion eines Pfeils (v, v'), der zu keinem Abbiegeverbot gehört. Der bisher beste bekannte Weg zum Endknoten dieses Pfeils v' ist daher möglicherweise vollständig verbotsfrei. In diesem Fall erfolgt die Wegsuche mit Bezug auf v' knotenbasiert. Zu berücksichtigen sind damit alle Fälle, die in den Abbildungen 6.7a und 6.7b auf der linken Seite dargestellt sind.

In Verfahren 20 ist der zu expandierende Pfeil (v, v') dagegen Teil eines Abbiegeverbots. Dadurch müssen alle Fälle berücksichtigt werden, die in den Abbildungen 6.7a und 6.7b auf der rechten Seite dargestellt sind.

Zu beachten ist der Unterschied beim Vergleich der Längen der erreichten Wege gleich zu Beginn der beiden Verfahren (Verfahren 19: größer; Verfahren 20: größer-gleich). Dies ermöglicht im Spezialfall zweier gleich langer Wege zu Knoten v unter Umständen die Rückkehr zur knotenbasierten Suche nach Fall (5).[12]

Zur weiteren Verdeutlichung des Verfahrens diene die ausführlichere Wegsuche von Knoten s nach Knoten t im Beispielgraphen in Abbildung 6.8, S. 217.[13] Tabelle 6.2, S. 217, zeigt den Verlauf des Verfahrens und weist auf die Besonderheiten hin. Die Notation in der Tabelle kennzeichnet durch einen Index an Knoten, die über einen verbotsgefährdeten Weg erreicht wurden, den Pfeil, über den diese erreicht wurden. Somit steht etwa x_s für das verbotsgefährdete Erreichen von Knoten x via Pfeil (s, x). Umgekehrt bedeutet ein Knoten *ohne* Index in Tabelle 6.2, dass der betreffende Knoten über einen *verbotsfreien* Weg erreicht wurde.

Der Lösungsweg ist $\langle s, u, v, w, z_1, z_2, w, v, t\rangle$. Er beinhaltet einen Zyklus um den Knoten v.

[12] Dieser Spezialfall fehlt aus Gründen der Übersichtlichkeit in Abbildung 6.7b, S. 213.
[13] Dieser Graph ist unabhängig von den bisherigen Beispielen.

Verfahren 18 Adaptiver Dijkstra-Algorithmus

Gegeben: Nicht-negativ bewerteter Digraph $G = (V, E, c, TR)$,
Startknoten s, Zielknoten t mit $s, t \in V$

1: $s.distanz \coloneqq 0$ ▸ Initialisierung
2: $v.distanz \coloneqq \infty$, $v.vorgänger \coloneqq NULL$ $\forall v \in V \setminus \{s\}$
3: $Lösungsmenge \coloneqq \emptyset$, $Horizontmenge \coloneqq \{s\}$
4: **while** $Horizontmenge \neq \emptyset$ **do**
5: Wähle v mit $v.distanz = \min\{v'.distanz \mid v' \in Horizontmenge\}$
6: **if** $v = t$ **then**
7: Abbruch des Verfahrens
8: **end if**
9: Entferne v aus *Horizontmenge*
10: Nimm v in *Lösungsmenge* auf
11: **if** $v.verbotsfrei$ **then**
12: Entferne alle $v_i \in Horizontmenge$ ▸ v_i sind weitere Vorkommen desselben Knotens v
13: **end if**
14: **for all** $v' \in \mathcal{N}(v)$ **do** ▸ Expansion von v
15: **if** Es existiert ein $\ddot{U}((v.vorgänger, v), (v, v')) \in TR$ **then**
16: **goto:** 14
17: **end if**
18: **if** $v.verbotsfrei$ **und** es existiert kein $\ddot{U}((v, v'), (v', w)) \in TR$ mit $(v', w) \in E$ **then**
19: Nimm v' via (v, v') verbotsfrei auf (Verfahren 19, S. 216)
20: **else**
21: Nimm v' via (v, v') verbotsgefährdet auf (Verfahren 20, S. 216)
22: **end if**
23: **end for** ▸ Ende der Expansion von v
24: **end while**

Ergebnis: Ein zulässiger, kürzester Weg von s zu t ist rekursiv konstruierbar aus $t.vorgänger$. Es gilt: $t.distanz = \text{dist}(s, t)$. Ist $t.vorgänger$ nicht gesetzt, existiert kein solcher Weg zwischen s und t.

Verfahren 19 Verbotsfreie Aufnahme eines Pfeils in Verfahren 18

Gegeben:
Zuletzt expandierter Pfeil (v, v'), aktuelle *Horizontmenge* und *Lösungsmenge*

1: **if** Es existiert ein $v'_i \in$ *Horizontmenge* $\cup$ *Lösungsmenge*
und (v'_i.*verbotsfrei* **oder** v'_i.*erreichtVia* $= (v, v')$)
und v.*distanz* $+ c(v, v') > v'_i$.*distanz* **then**
2: **return** ▸ knotenbasiert nach Fall (4) bzw. pfeilbasiert nach Fall (8)
3: **end if**

4: Entferne alle Vorkommen von v'_j aus *Horizontmenge*, für die gilt:
v'_j.*distanz* $\geq v$.*distanz* $+ c(v, v')$ ▸ Fall (5)

5: v'.*vorgänger* $:= v$ ▸ Beinhaltet Aktualisierung nach Fall (1) bzw. (5), sowie Aufnahme nach Fall (8)
6: v'.*distanz* $:= v$.*distanz* $+ c(v, v')$
7: v'.*erreichtVia* $:= (v, v')$
8: v'.*verbotsfrei* $:=$ `wahr`
9: Nimm v' in *Horizontmenge* auf

Ergebnis: Der Endknoten v' des zuletzt expandierten Pfeils (v, v') wurde verarbeitet.

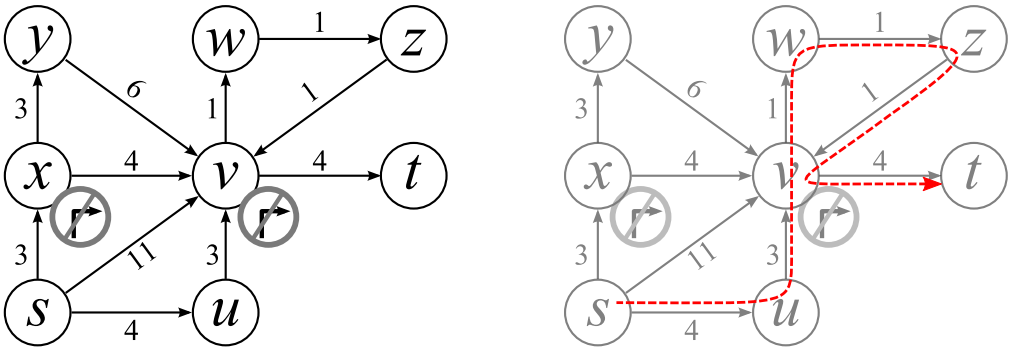
Verfahren 20 Verbotsgefährdete Aufnahme eines Pfeils in Verfahren 18

Gegeben:
Zuletzt expandierter Pfeil (v, v'), aktuelle *Horizontmenge* und *Lösungsmenge*

1: **if** Es existiert ein $v'_i \in$ *Horizontmenge* $\cup$ *Lösungsmenge*
und (v'_i.*verbotsfrei* **oder** v'_i.*erreichtVia* $= (v, v')$)
und v.*distanz* $+ c(v, v') \geq v'_i$.*distanz* **then**
2: **return** ▸ Fälle (4) und (9)
3: **end if**

4: **if** Es existiert ein $v'_j \in$ *Horizontmenge* **und** v'_j.*erreichtVia* $= (v, v')$ **then**
5: v'_j.*distanz* $:= v$.*distanz* $+ c(v, v')$ ▸ pfeilbasierte Aktualisierung nach Fall (2) bzw. (6)
6: v'_j.*verbotsfrei* $:=$ `falsch` ▸ überschreibt hier u.U. den Wert `wahr`
7: **else**
8: v'.*vorgänger* $:= v$ ▸ Fall (3) bzw. (7)
9: v'.*distanz* $:= v$.*distanz* $+ c(v, v')$
10: v'.*erreichtVia* $:= (v, v')$
11: v'.*verbotsfrei* $:=$ `falsch`
12: Nimm v' (ggf. zum wiederholten Mal) in *Horizontmenge* auf
13: **end if**

Ergebnis: Der Endknoten v' des zuletzt expandierten Pfeils (v, v') wurde verarbeitet.



(a) Graph mit Abbiegeverboten

(b) Kürzester Weg von s nach t.

Abbildung 6.8: Ausführliches Beispiel für die adaptive Wegsuche
Tabelle 6.2 zeigt den ausführlichen Suchverlauf von s nach t.

Iteration	Horizontmenge	Lösungsmenge
0	$\{s[0]\}$	$\{\}$
1	$\{x_s[3], u[4], v[11]\}^{I}$	$\{s[0]\}$
2	$\{u[4], y_x[6], v[11]\}^{II}$	$\{s, x_s[3]\}$
3	$\{y_x[6], v_u[7], v[11]\}^{III}$	$\{s, x_s, u[4]\}$
4	$\{v_u[7], v[11]\}^{IV}$	$\{s, x_s, u, y_x[6]\}$
5	$\{w_v[8], v[11]\}$	$\{s, x_s, u, y_x, v_u[7]\}$
6	$\{z_w[9], v[11]\}$	$\{s, x_s, u, y_x, v_u, w_v[8]\}$
7	$\{v_z[10], v[11]\}$	$\{s, x_s, u, y_x, v_u, w_v, z_w[9]\}$
8	$\{v[11], t_v[14]\}$	$\{s, x_s, u, y_x, v_u, w_v, z_w, v_z[10]\}$
9	$\{t_v[14]\}^{V}$	$\{s, x_s, u, y_x, v_u, w_v, z_w, v_z, v[11]\}^{VI}$
10	$\{\}$	$\{s, x_s, u, y_x, v_u, w_v, z_w, v_z, v, t_v[14]\}^{VII}$

[I] Knoten v wurde über einen verbotsfreien Weg in die Horizontmenge aufgenommen.
[II] Obwohl (x, y) nicht Teil eines Abbiegeverbots ist, „erbt“ Knoten y die Verbotsgefährdung von x.
[III] Es wurde über Knoten u ein kürzerer Weg zu v gefunden. Da dieser Weg aber verbotsgefährdet ist, wird v in der Horizontmenge nicht aktualisiert, sondern erneut aufgenommen.
[IV] v wird *nicht* erneut via Pfeil (y, v) mit Distanz 12 in die Horizontmenge aufgenommen, da dieser Weg länger ist als ein bereits bekannter *verbotsfreier* Weg zu v.
[V] t wird nicht erneut via Pfeil (v, t) aufgenommen, da bereits ein kürzerer Weg über *denselben* Pfeil bekannt ist.
[VI] v wurde über einen verbotsfreien Weg in die Lösungsmenge aufgenommen. Selbst wenn im Graph noch weitere Pfeile zu v existierten, würde v von nun an nicht mehr in die Horizontmenge aufgenommen werden.
[VII] Die Suche ist beendet, da der Zielknoten in die Lösungsmenge aufgenommen wurde.

Tabelle 6.2: Horizont- und Lösungsmenge bei der Wegsuche von Knoten s nach t im Graphen in Abbildung 6.8 gemäß Verfahren 18, S. 215, Zahlen in eckigen Klammern geben die Länge des kürzesten bekannten Weges wieder; Indizes heben für Knoten, die über einen verbotsgefährdeten Weg erreicht wurden, hervor, über welchen Pfeil diese Knoten erreicht wurden.

Vorteile der adaptiven Suche Es gelten die gleichen Vorteile wie bei der pfeilbasierten Suche in Abschnitt 6.2.2 auf S. 208. Zusätzlich jedoch werden bei der adaptiven Dijkstra-Wegsuche nur *höchstens genauso viele* Iterationen benötigt wie bei der pfeilbasierten Dijkstra-Suche. Grund hierfür ist das oben beschriebene Vorgehen, nach dem ein Knoten nicht mehr in die Horizontmenge aufgenommen wird, wenn er bereits über einen verbotsfreien Weg in die Lösungsmenge aufgenommen wurde.

Der adaptive Charakter der hier vorgestellten Wegsuche wirkt sich darüber hinaus vorteilhaft auf die Performance aus. Denn je nach „Dichte“ der Abbiegebeschränkungen arbeitet die adaptive Suche möglichst lange knotenbasiert. Insbesondere gilt als Extremfall daher, dass die Suche in einem abbiegebeschränkungsfreien Gebiet der Karte gemessen an den nötigen Dijkstra-Iterationen genau so schnell verläuft wie die knotenbasierte Suche ohne Berücksichtigung von Abbiegebeschränkungen. Dieser Vorteil ist im direkten Vergleich mit der Graph-Transformation per Knotensplitting sowie der pfeilbasierten und der knotenbasierten Suche aus den vorherigen Abschnitten besonders hervorzuheben.

Nachteile der adaptiven Suche Dieser Ansatz kann seine Vorteile nur für Abbiegebeschränkungen realisieren. Für eine verallgemeinernde Betrachtung von Abbiegekosten gelten sie nicht.

Eignung für Contraction Hierarchies Die adaptive Suche ist für den Einsatz während der *Erzeugung* einer Contraction Hierarchy ungeeignet. Bei der Knotenkontraktion reicht es nicht aus, mithilfe dieses Ansatzes die Nachbarknoten zu überprüfen, ob ein Shortcut-Pfeil zu erzeugen ist. Die Gründe hierfür sind in Abschnitt 6.4 detaillierter erläutert.

Sehr gut lässt sich dagegen die alternierende bidirektionale Dijkstra-Suche der Contraction Hierarchies mit der adaptiven Suche vereinen. Dies wird gesondert in Abschnitt 6.3.2 verdeutlicht.

6.3 Wegsuche in Contraction Hierarchies mit Abbiegebeschränkungen

Dieser Abschnitt beschreibt die notwendigen Anpassungen an die pfeilbasierte Wegsuche und an die adaptive Wegsuche, damit sie erfolgreich in einer Contraction Hierarchy eingesetzt werden können. Abschnitte 6.6.1 und 6.6.2 liefern später vergleichbare Benchmarkergebnisse für beide Verfahren.

Die Beispiele in diesem Abschnitt gehen stets davon aus, dass eine vollständige Hierarchie vorliegt, auf der sämtliche kürzesten Wege gefunden werden können. Dies beinhaltet im Speziellen zwei Forderungen:

1. Alle Knoten besitzen einen eindeutigen Rang.
2. Der Graph besitzt alle notwendigen Shortcut-Pfeile, so dass sich alle kürzesten Wege gemäß der Form nach Gleichung (5.5), S. 140, darstellen lassen. Dies gilt insbesondere auch für kürzeste Wege, die aufgrund von Abbiegebeschränkungen Teil-Wege beinhalten, die keine kürzesten Wege sind.[14] Infolge der Existenz der Shortcut-Pfeile liefern natürlich auch die beiden Funktionen[15] e_1 und e_2 den ersten bzw. zweiten überbrückten Pfeil zu jedem Shortcut-Pfeil.

Wie sich Contraction Hierarchies für abbiegebeschränkungssensible Wegsuchen *erzeugen* lassen, beschreibt Abschnitt 6.4, S. 230 ff., ausführlich. Da diese Wege teilweise Knoten mehrfach beinhalten können, ergibt sich, dass bei der Hierarchie-Erzeugung mitunter auch Shortcut-Pfeile notwendig werden, die *Schlingen* darstellen. Es sei daher bereits hier auf diesen Umstand hingewiesen.

Shortcut-Schlingen

Weiterhin können durch Abbiegebeschränkungen parallele Shortcut-Pfeile zwischen zwei Knoten notwendig werden (vgl. [Geisberger u. Vetter, 2011, S. 104]). Dieser Aspekt wird ebenfalls in Abschnitt 6.4, S. 230 ff., diskutiert. Zwecks einer Unterscheidung zwischen parallelen Shortcut-Pfeilen zwischen Knoten u und v wird daher zusätzlich auch in folgender Form notiert: $(u \to u' \ldots v' \to v)$. Dabei stehen $(u \to u')$ bzw. $(v' \to v)$ für den ersten bzw. letzten vom Shortcut-Pfeil überbrückten Original-Pfeil der zugrunde liegenden Graphen. Eine Mehrdeutigkeit ist durch diese Schreibweise ausgeschlossen (vgl. ebd.).

Darüber hinaus muss gefordert werden, dass sich Abbiegeverbote auch auf Shortcut-Pfeile beziehen: Ein Abbiegeverbot zwischen zwei Original-Pfeilen $(u \to v)$ und $(v \to w)$ gilt in gleicher Weise auch für zwei Shortcut-Pfeile der Form $(u'' \to u' \ldots u \to v)$ und $(v \to w \ldots w' \to w'')$. Gleichfalls gilt dasselbe Abbiegeverbot natürlich auch zwischen $(u \to v)$ und $(v \to w \ldots w' \to w'')$ sowie zwischen $(u'' \to u' \ldots u \to v)$ und $(v \to w)$.

Abbiegebeschränkungen zwischen Shortcut-Pfeilen

6.3.1 Pfeilbasierte Wegsuche in Contraction Hierarchies

Bei Geisberger u. Vetter [2011] werden viele der notwendigen Änderungen am Standard-Suchverfahren in Contraction Hierarchies für die pfeilbasierte Wegsuche ausführlich vorgestellt. Das Gerüst der in Abschnitt 5.2 beschriebenen, alternierenden bidirektionalen Dijkstra-Suche kann beibehalten werden; die Vorwärts- und Rückwärtssuche bedürfen allerdings, wie in Abschnitt 6.2.2 gezeigt, einer Anpassung. Konkret bedeutet dies, dass

- grundsätzlich von einem Startpfeil zu einem Zielpfeil gesucht wird,
- Pfeile bei der Vorwärtssuche bzw. Rückwärtssuche an ihren Endknoten bzw. Startknoten expandiert werden und
- beim Expandieren eines Pfeils bei einem seiner Knoten v nur solche Pfeile in die Horizontmenge aufgenommen werden, die zu einem Knoten mit einem höheren Rang als $\text{rang}(v)$ führen oder aber Shortcut-Schlingen sind.

[14] Hierzu gehören auch Wege mit Zyklen (siehe Abschnitt 6.2, S. 201 ff., sowie Abbildung 6.2, S. 202, und Abbildung 6.3, S. 203)

[15] siehe Seite 134

Fokus liegt auf Original-Pfeilen

Eine wesentliche Erkenntnis beim Einsatz der pfeilbasierten Wegsuche in einer Contraction Hierarchy besteht darin, dass ausschließlich *Original-Pfeile* (in Abgrenzung zu Shortcut-Pfeilen) in die Horizont- und Lösungsmengen aufgenommen werden müssen (vgl. [Geisberger u. Vetter, 2011, S. 103]). Die Betrachtung dieser Teilmenge aller Pfeile des Graphen ist hinreichend, da auf diese Weise sämtliche *Richtungen*, aus der eine Kreuzung, d. h. ein Knoten, erreicht werden kann, abgedeckt werden.

Expandiert die Vorwärtssuche daher entlang eines Shortcut-Pfeils $(u \to u' \dots v' \to v)$, wird nicht dieser, sondern $(v' \to v)$ in der Horizontmenge und später in der Lösungsmenge gespeichert. Spiegelbildlich speichert die Rückwärtssuche für denselben Shortcut-Pfeil den Original-Pfeil $(u \to u')$.

Damit sich Wege effizient rekonstruieren lassen, muss zu jedem Original-Pfeil, der auf diese Weise in die Lösungsmenge aufgenommen wurde, gespeichert werden, über welchen (Shortcut-)Pfeil er erreicht wurde. Dies ist notwendig, da Contraction Hierarchies mit Abbiegebeschränkungen aufgrund von Shortcut-Pfeilen auch Schlingen und parallele Pfeile enthalten können.[16] Im Pseudocode dieses Abschnitts wird dies über das Pfeil-Attribut `erreichtVia` ausgedrückt. Wenn bei der Vorwärtssuche daher der Original-Pfeil $(u'' \to u)$ über $(u \to u' \dots v' \to v)$ expandiert und damit $(v' \to v)$ erstmalig in die Horizontmenge aufgenommen wird, muss $(v' \to v).\mathit{erreichtVia} \coloneqq (u \to\ u' \dots v' \to\ v)$ gesetzt werden.

Adaption für *Knoten* als Start und Ziel Eine kleine Anpassung ermöglicht auch Knoten als Start und Ziel. Dafür müssen während der Initialisierung der Suche in Analogie zu Verfahren 17, S. 207, sämtliche vom Startknoten abgehenden Original-Pfeile in die Horizontmenge der Vorwärtssuche und sämtliche in den Zielknoten führenden Original-Pfeile in die Horizontmenge der Rückwärtssuche eingefügt werden.

6.3.1.1 Suche nach Brückenknoten

Brückenknoten-Suche schwerer

Zwar verringert die Fokussierung auf Original-Pfeile deutlich die Anzahl der möglichen Einträge in den Horizont- und Lösungsmengen, aber gleichzeitig wird die Suche nach Brückenknoten erschwert. Denn bei Aufnahme eines Original-Pfeils $(v' \to v)$ in die Lösungsmenge der Vorwärtssuche ist v genau dann ein Brückenknoten, wenn ein von v *ausgehender* Original-Pfeil $(v \to v'')$ bereits in der Lösungsmenge der Rückwärtssuche vorhanden ist und gleichzeitig kein Abbiegeverbot von $(v' \to v)$ zu $(v \to v'')$ existiert.

Genügt bei Verfahren 11, S. 138, noch ein einfaches Nachschlagen beispielsweise in einer Hashtabelle der Rückwärtssuche[17], muss bei der pfeilbasierten Suche für *jeden* von v ausgehenden Original-Pfeil überprüft werden, ob er bereits Element der Lösungsmenge der Rückwärtssuche ist. Wiederum versteht sich diese Aussage in angepasster Weise auch für die Rückwärts-Suchrichtung. Damit erhöht sich der Aufwand für diesen Aspekt der Wegsuche im Mittel um einen konstanten Faktor, nämlich den durchschnittlichen Knotengrad des Originalgraphen.

[16] Die Wegsuche in Contraction Hierarchies *ohne* Abbiegebeschränkungen kommt auch ohne dieses Attribut aus, da wegen der Definition der Knotenkontraktion keine parallelen Pfeile durch Shortcut-Pfeile entstehen können (siehe Verfahren 10, S. 135, und insbesondere Zeile 8 für das Kriterium zum Erzeugen neuer Shortcut-Pfeile).

[17] siehe auch Abschnitt 5.5.6, S. 176

Negativer Overhead nicht mehr möglich

Zusätzlich bedeutet diese Art der Brückenknoten-Suche, dass beim zuletzt erreichten Knoten v auch solche Original-Pfeile überprüft werden müssen, die zu einem Knoten mit *niedrigerem* Rang als rang(v) führen. Im Minimalbeispiel in Abbildung 6.9 wird v nur als Brückenknoten erkannt, wenn von v der ausgehende Pfeil (v, w) untersucht wird. Dies bedeutet insbesondere aber, dass bei den Knoten des Graphen *sämtliche* Pfeile gespeichert werden müssen und nicht mehr nur diejenigen, die zu Knoten mit höheren Rängen führen[18]. Die Möglichkeit, durch den Einsatz von Contraction Hierarchies sogar Speicherplatz einzusparen, ist dadurch genommen.

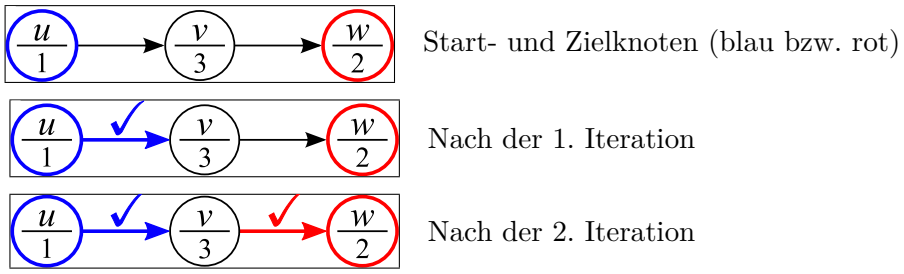


Abbildung 6.9: Brückenknoten-Suche im Rahmen der pfeilbasierten Wegsuche in einer Contraction Hierarchy. Nach der zweiten Iteration – einer Rückwärts-Iteration – muss überprüft werden, ob Pfeil (u, v) bereits in der Lösungsmenge der Vorwärtssuche ist.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Linien ohne Pfeilspitzen sind als zwei entgegengesetzte Pfeile zu interpretieren. Häkchen kennzeichnen Pfeile der jeweiligen Lösungsmenge.
Blau markiert: Vorwärtssuche
Rot markiert: Rückwärtssuche

6.3.1.2 Blockieren von Pfeilen

Wie auch die knotenbasierte Wegsuche in Contraction Hierarchies[19] lässt sich die pfeilbasierte Wegsuche beschleunigen, indem solche Original-Pfeile blockiert werden, die über einen offensichtlich nicht-optimalen Weg erreicht wurden (vgl. [Geisberger u. Vetter, 2011, S. 106]). Im Speziellen bedeutet dies, dass Original-Pfeil $(u \to v)$ aus $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2, TR)$ im Rahmen der Vorwärtssuche blockiert werden kann, wenn ein Shortcut-Pfeil $(y \to y' \ldots u \to v) \in E$ und ein weiterer Original-Pfeil $(y'' \to y) \in E$ existieren, so dass gilt:

$$\begin{gathered}(y''\to y).\mathit{distanz}_{\text{vor}} + c((y\to y' \ldots u\to v)) < (u\to v).\mathit{distanz}_{\text{vor}} \\ \text{und } \text{rang}(y) > \text{rang}(v) \\ \text{und } \ddot{U}((y''\to y),(y\to y')) \notin TR.\end{gathered} \tag{6.1}$$

[18] siehe Abschnitt 5.2, S. 145, (Stichwort „Negativer Overhead möglich“)
[19] siehe Abschnitt 5.2, S. 137 ff., und insbesondere Seite 146 zum Knoten-Blockieren

Abbildung 6.10 verdeutlicht den Zusammenhang grafisch. Gestrichelte Linien gleicher Farbe symbolisieren hier Shortcut-Pfeile. Es werde angenommen, dass Original-Pfeil $(u \to v)$ über den orangefarben markierten Shortcut-Pfeil $(z \to u \ldots u \to v)$ in die Horizontmenge aufgenommen wurde. Bevor $(u \to v)$ jedoch in die Lösungsmenge überführt wird, erfolgt die Blockierungsprüfung, bei der all jene zu v führenden (Shortcut-)Pfeile untersucht werden, die bei einem Knoten mit geringerem Rang als v beginnen. Im gezeigten Graphen-Ausschnitt trifft dies auf den grün dargestellten Shortcut-Pfeil $(y \to y' \ldots u \to v)$ zu. Sofern der Original-Pfeil $(y'' \to y)$ bereits Teil der Horizont- oder Lösungsmenge ist[20] und kein Abbiegeverbot von ihm zu Pfeil $(y \to y')$ besteht, kann geprüft werden, ob Original-Pfeil $(u \to v)$ aufgrund dieses alternativen Weges zu ihm blockierbar ist.

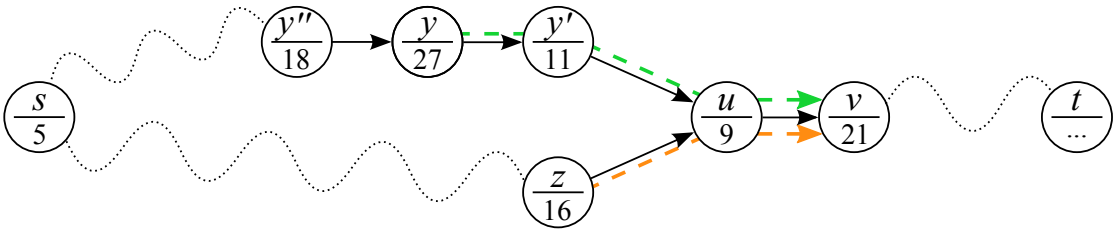


Abbildung 6.10: Blockierung bei pfeilbasierter Suche in Contraction Hierarchies
Dargestellt ist ein *Ausschnitt* eines Graphen und die Wegsuche von Knoten s nach Knoten t.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Gestrichelte Linien gleicher Farbe symbolisieren Shortcut-Pfeile. Gepunktete Linien stehen für nicht im Detail dargestellte Wege.

Weitergabe der Blockierung Wie auch bei der abbiegebeschränkungsfreien Wegsuche kann die Information über das Blockieren eines Pfeils an weitere Pfeile in der Nachbarschaft des blockierten Pfeils weitergegeben werden, um Iterationen bei der Wegsuche einzusparen[21]. Verfahren 21, S. 223, zeigt die Weitergabe im Pseudocode in Analogie zu Verfahren 13, S. 149. Werden Zeilen 9–25 ausgelassen, ergibt sich die Wegsuche mit *einfachem* Blockieren, wie sie zu Beginn dieses Abschnitts beschrieben wurde.

[20] Diese Aussage ist gleichbedeutend mit $(y'' \to y).\mathit{distanz} < \infty$, wodurch überhaupt erst die Möglichkeit besteht, dass Gleichung (6.1), S. 221, erfüllt werden kann.
[21] siehe Abschnitt 5.2.1 auf Seite 148

Verfahren 21 Erweiterte Pfeilblockierung b. Wegsuchen in einer CH (Vorwärtssuche)

Gegeben: Nächster zu expandierender Original-Pfeil $(u \to v)$

1: **for all** $(y \to y' \dots u \to v) \in E$ mit $y.rang > v.rang$ **do**
2: **for all** $(y'' \to y) \in E$ mit $\ddot{U}((y'' \to y), (y \to y')) \notin TR$ **do**
3: **if** $(y'' \to y).distanz_{\text{vor}} + c((y \to y' \dots u \to v)) < (u \to v).distanz_{\text{vor}}$ **then**
4: Blockiere $(u \to v)$ aufgrund von $(y'' \to y)$
5: $(u \to v).distanz_{\text{vor}} \coloneqq (y'' \to y).distanz_{\text{vor}} + c((y \to y' \dots u \to v))$
6: $(u \to v).vorgänger_{\text{vor}} \coloneqq (y'' \to y)$
7: $(u \to v).erreichtVia_{\text{vor}} \coloneqq (y \to y' \dots u \to v)$
8: ▸ Prüfe Weitergabe der Blockierung von $(u \to v)$ an $(w \to w')$
9: **for all** $(v \to v' \dots w \to w') \in E$ mit $\ddot{U}((u \to v), (v \to v')) \notin TR$ **do**
10: **if** $(u \to v).distanz_{\text{vor}} + c((v \to v' \dots w \to w')) < (w \to w').distanz_{\text{vor}}$ **then**
11: Blockiere Pfeil $(w \to w')$ aufgrund von Pfeil $(u \to v)$
12: $(w \to w').distanz_{\text{vor}} \coloneqq (u \to v).distanz_{\text{vor}} + c((v \to v' \dots w \to w'))$
13: $(w \to w').vorgänger_{\text{vor}} \coloneqq (u \to v)$
14: $(w \to w').erreichtVia_{\text{vor}} \coloneqq (v \to v' \dots w \to w')$
15: **end if**
16: **end for**
17: ▸ Prüfe Weitergabe der Blockierung von $(y'' \to y)$ an $(w \to w')$
18: **for all** $(y'' \to y \dots w \to w') \in E$ mit $\ddot{U}((y'' \to y), (w \to w')) \notin TR$ **do**
19: **if** $(y'' \to y).distanz_{\text{vor}} + c((y'' \to y \dots w \to w')) < (w \to w').distanz_{\text{vor}}$ **then**
20: Blockiere Pfeil $(w \to w')$ aufgrund von Pfeil $(y'' \to y)$
21: $(w \to w').distanz_{\text{vor}} \coloneqq (y'' \to y).distanz_{\text{vor}} + c((y'' \to y \dots w \to w'))$
22: $(w \to w').vorgänger_{\text{vor}} \coloneqq (y'' \to y)$
23: $(w \to w').erreichtVia_{\text{vor}} \coloneqq (y'' \to y \dots w \to w')$
24: **end if**
25: **end for** ▸ Ende der Weitergabe der Blockierung
26: **return** ▸ Mehrmaliges Blockieren desselben Pfeils $(u \to v)$ sinnlos
27: **end if**
28: **end for**
29: **end for**

Ergebnis: Die Blockierung des als nächstes zu expandierenden Pfeils $(u \to v)$ wurde festgelegt. Im Falle einer Blockierung wurde diese an die Nachfolgerpfeile von $(u \to v)$ und desjenigen Pfeils $(y'' \to y)$, der zur Blockierung von $(u \to v)$ führte, weitergereicht.

6.3.1.3 Zielgerichtete Wegsuche

Das Verfahren kann leicht um den Aspekt einer zielgerichteten Suche erweitert werden. Die in Abschnitt 5.2.2, S. 149 ff., vorgestellten Ideen dienen dabei als Grundlage und bedürfen nur kleinerer Anpassungen für die pfeilbasierte Wegsuche. Vorab sei an dieser Stelle daran erinnert, dass ausschließlich Original-Pfeile in die Horizont- und Lösungsmengen aufgenommen werden[22].

A*-Eingrenzung des Suchraums Bevor ein Pfeil $(u \to v)$ expandiert wird, wird überprüft, ob er überhaupt Teil eines Weges sein kann, der kürzer als der kürzeste bekannte Weg p_b vom Start- zum Zielknoten ist[23]. Da bei der Expansion von $(u \to v)$ ein kürzester Weg vom Startknoten der Wegsuche bis einschließlich $(u \to v)$ bekannt ist, muss die Restdistanz zum Zielknoten der Wegsuche noch abgeschätzt werden. Dies kann wiederum mithilfe der Luftdistanz erfolgen, die eine untere Schranke für diesen Zweck liefert. Pfeil $(u \to v)$ wird also genau dann nicht expandiert, wenn für ihn gilt:

$$(u \to v).\mathit{distanz} + \mathrm{rd}(v) > c(p_b). \tag{6.2}$$

Bidirektionale A*-Suche Auch die in Abschnitt 5.2.2.2, S. 152 ff., vorgestellte bidirektionale A*-Suche lässt sich mit der pfeilbasierten Wegsuche in Contraction Hierarchies kombinieren. Hierfür werden alle Original-Pfeile aus der Horizontmenge gemäß dem A*-Verfahren sortiert, so dass stets derjenige Original-Pfeil $(u \to v)$ als nächstes aus dieser Menge entfernt und expandiert wird, für den gilt:

$$(u \to v).\mathit{distanz} + \mathrm{rd}(v) = \min\{(u' \to v').\mathit{distanz} + \mathrm{rd}(v') \mid (u' \to v') \in \mathit{Horizontmenge}\}. \tag{6.3}$$

Die zu diesem Abschnitt gehörigen Benchmarks in Abschnitt 6.6.1.2, S. 243 ff., untersuchen in Analogie zu den Messungen aus Abschnitt 5.2.2.2 zwei Varianten der bidirektionale A*-Suche. Die erste kehrt zum Originalverfahren inklusive A*-Eingrenzung des Suchraums zurück, sobald der erste Brückenknoten identifiziert wurde. Die zweite bleibt eine reine bidirektionale A*-Wegsuche.

[22] siehe Seite 220
[23] siehe Abschnitt 5.2.2.1, S. 150, für eine ausführlichere Beschreibung der Idee

6.3.2 Adaptive Wegsuche in Contraction Hierarchies

Die Vorzüge der adaptiven Wegsuche lassen sich auch auf Contraction Hierarchies übertragen. Ihre Stärken gegenüber der pfeilbasierten Suche werden in diesem Kontext noch deutlicher, wie sich später zeigen wird.

Die Anpassung von Verfahren 11, S. 138, erfolgt als Verknüpfung mit Verfahren 18, S. 215,; es entsteht Verfahren 22, S. 226. Die Unterschiede zur Wegsuche in einer Contraction Hierarchy ohne die Berücksichtigung von Abbiegebeschränkungen sind wie folgt:

- Die mehrfache Aufnahme eines Knotens in Horizont- und Lösungsmenge beider Suchrichtungen ist erlaubt (Zeilen 11, 23 und 25).
- Bei der Knotenexpansion eines Knotens v zu einem Nachfolgerknoten v' können auch Shortcut-Schlingen vorkommen, d. h., möglicherweise ist $v = v'$. Daher gilt für die durch Expansion von v erreichbaren Knoten v': $\text{rang}(v') \geq \text{rang}(v)$ (Zeile 18).
- Bei der Suche nach einem Brückenknoten m darf kein Abbiegeverbot bei m die Verbindung von der Vorwärts- mit der Rückwärtssuche verhindern (Zeile 28).
- Viele Knoten werden im Verlauf des Verfahrens über Shortcut-Pfeile aufgenommen. Da diese mehrere andere Pfeile überbrücken, muss für jeden Shortcut-Pfeil ermittelt werden, ob von einem der von ihm überbrückten Original-Pfeile ein Abbiegeverbot ausgeht (Zeile 22). Hierfür eignet sich beispielsweise die in Verfahren 12 auf Seite 142 beschriebene rekursive Auflösung von Shortcut-Pfeilen. Für eine effizientere Abfrage sollte jedoch direkt bei den Shortcut-Kantenobjekten ein Flag gesetzt sein. Dies ist allerdings eine Anforderung an die Modellierung und wird in Abschnitt 6.5.1, S. 234 ff., näher beleuchtet.
- Die Berücksichtigung der Abbiegebeschränkungen muss bei der Rückwärtssuche ebenfalls angepasst werden. Während die Vorwärtssuche bei Expansion von Knoten v prüfen muss, ob ein $\ddot{U}(v.\mathit{erreichtVia}, (v, v')) \in \mathit{TR}$ existiert[24], ist im Rahmen der Rückwärtssuche stattdessen zu prüfen, ob ein $\ddot{U}((v', v), v.\mathit{erreichtVia}) \in \mathit{TR}$ existiert.

[24] siehe Verfahren 18, S. 215, Zeilen 15 und 18

Verfahren 22 Abbiegebeschränkungen berücksichtigende, adaptive Wegsuche in CH

Gegeben:

Contraction Hierarchy $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2, TR)$,
Startknoten s, Zielknoten t mit $s, t \in V$

1: $s.distanz_{\text{vor}} \coloneqq 0$, $t.distanz_{\text{rück}} \coloneqq 0$ ▸ Initialisierung
2: $v.distanz_{\text{vor}} \coloneqq \infty$, $v.vorgänger_{\text{vor}} \coloneqq NULL$ $\forall v \in V \setminus \{s\}$
3: $v.distanz_{\text{rück}} \coloneqq \infty$, $v.vorgänger_{\text{rück}} \coloneqq NULL$ $\forall v \in V \setminus \{t\}$
4: $Lösungsmenge_{\text{vor}} \coloneqq \emptyset$, $Lösungsmenge_{\text{rück}} \coloneqq \emptyset$
5: $Horizontmenge_{\text{vor}} \coloneqq \{s\}$, $Horizontmenge_{\text{rück}} \coloneqq \{t\}$
6: $m \coloneqq NULL$, $best \coloneqq \infty$ ▸ Brückenknoten und Länge des kürzesten bekannten Weges
7: $fertig_{\text{vor}} \coloneqq$ `falsch` , $fertig_{\text{rück}} \coloneqq$ `falsch` ▸ Abbruchkriterien
8: **while not** ($fertig_{\text{vor}}$ **und** $fertig_{\text{rück}}$) **do**
9: **if not** $fertig_{\text{vor}}$ **then**
10: Wähle v mit $v.distanz = \min\{v'.distanz \mid v' \in Horizontmenge_{\text{vor}}\}$ ▸ Vorw.-Iteration
11: Entferne v aus $Horizontmenge_{\text{vor}}$ und nimm v in $Lösungsmenge_{\text{vor}}$ auf
12: **if** $v.distanz_{\text{vor}} \geq best$ **then**
13: $fertig_{\text{vor}} \coloneqq$ `wahr`; **goto:** 8 ▸ Die Vorwärtssuche terminiert
14: **end if**
15: **if** v.verbotsfrei **then**
16: Entferne alle $v_i \in Horizontmenge_{\text{vor}}$ ▸ v_i sind weitere Vorkommen desselben Knotens v
17: **end if**
18: **for all** $v' \in \mathcal{N}(v)$ mit $v'.rang \geq v.rang$ **do** ▸ Vorwärts-Expansion von v
19: **if** Es existiert ein $\ddot{U}(v.erreichtVia, (v, v')) \in TR$ **then**
20: **goto:** 18
21: **end if**
22: **if** $v.verbotsfrei$ **und** es existiert kein $\ddot{U}((w, x), (x, y)) \in TR, (w, x) \in (v, v')$ **then**
23: Nimm v' via (v, v') verbotsfrei auf (Verfahren 19, S. 216)
24: **else**
25: Nimm v' via (v, v') verbotsgefährdet auf (Verfahren 20, S. 216)
26: **end if**
27: **end for** ▸ Ende der Expansion von v
28: **if** $v \in Lösungsmenge_{\text{rück}}$ ▸ Suche nach Brückenknoten
und $v.distanz_{\text{vor}} + v.distanz_{\text{rück}} < best$
und $\ddot{U}(v.erreichtVia_{\text{vor}}, v.erreichtVia_{\text{rück}}) \notin TR$ **then**
29: $best \coloneqq v.distanz_{\text{vor}} + v.distanz_{\text{rück}}$, $m \coloneqq v$
30: **end if**
31: **end if** ▸ Ende der Vorwärts-Iteration
32: **if not** $fertig_{\text{rück}}$ **then**
33: Führe spiegelbildlich zur Vorwärtssuche eine Rückwärts-Iteration durch
34: **end if**
35: **end while**

Ergebnis: Ein zulässiger, kürzester Weg von s zu t über m ist für die erste Teilstrecke rekursiv konstruierbar aus $m.vorgänger_{\text{vor}}$ und für die zweite Teilstrecke aus $m.vorgänger_{\text{rück}}$. Es gilt: $m.distanz_{\text{vor}} + m.distanz_{\text{rück}} = \text{dist}(s, t) = best$. Wurde kein Brückenknoten gefunden, existiert kein solcher Weg.

6.3.2.1 Suche nach Brückenknoten

Anders als bei der pfeilbasierten Suche verläuft die Suche nach potentiellen Brückenknoten wie gewohnt und damit effizient: Nach einer Iteration in eine Suchrichtung wird geprüft, ob der zuletzt in die Lösungsmenge aufgenommene Knoten v bereits Teil der Lösungsmenge der anderen Suchrichtung ist. Werden die Lösungsmengen beider Suchrichtungen in Hashtabellen hinterlegt, lässt sich diese Frage besonders schnell beantworten.

Die einzige Anpassung betrifft natürlich Abbiegebeschränkungen an Brückenknoten. Wie die Zusammenführung der Vorwärts- und Rückwärtssuche erfolgen muss, beschreibt Zeile 28 in Verfahren 22, S. 226, formal. Zu beachten ist hier die Möglichkeit, dass ein potentieller Brückenknoten von mindestens einer der beiden Suchrichtungen aus per Shortcut-Pfeil erreicht wurde.[25]

6.3.2.2 Blockieren von Knoten

Wiederum lässt sich die Wegsuche durch geschicktes Blockieren von Knoten beschleunigen. Die Vorgehensweise orientiert sich direkt an der in Abschnitt 5.2 auf Seite 146 beschriebenen, muss aber um die Berücksichtigung von Abbiegebeschränkungen ergänzt werden. Dabei darf bei der Vorwärtssuche ein Knoten u nur dann zum Blockieren eines anderen Knotens v herangezogen werden, wenn kein Abbiegeverbot ausgehend von Pfeil (u, v) existiert. Bei der Rückwärtssuche wird analog Pfeil (v, u) geprüft. Hier darf kein Abbiegeverbot in Richtung dieses Pfeils existieren. Andernfalls sind Szenarien denkbar, in denen der kürzeste Weg übersehen würde.

Abbildung 6.11, S. 228, zeigt ein Beispiel, das diese Überlegung illustriert. Betrachtet werde die Wegsuche von Knoten s nach Knoten t. Den Verlauf der Wegsuche zeigt Tabelle 6.3, S. 228, ausführlich. In Iteration 5 wird hier Knoten v in die Lösungsmenge der Vorwärtssuche aufgenommen. Bei der Prüfung, ob v blockiert werden kann, stößt die Suche auf Knoten u. Wie in Abschnitt 5.2 beschrieben, erfüllt u die Voraussetzungen für das Blockieren von Knoten v gemäß Gleichung (5.8), S. 146. Dennoch darf in diesem Fall v *nicht* blockiert werden, denn nicht alle von v aus erreichbaren Knoten lassen sich via Knoten u über einen kürzeren Weg erreichen. Insbesondere gilt dies für Knoten t, dem Zielknoten.

[25] Zu Abbiegebeschränkungen zwischen Shortcut-Pfeilen siehe auch Seite 219.

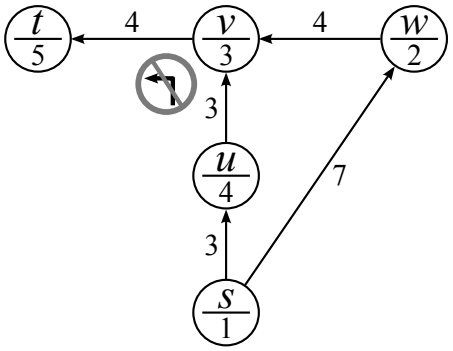


Abbildung 6.11: Vorwärts-Knotenblockierung bei adaptiver Suche in Contraction Hierarchies
Besonderheit: Bei der Wegsuche von Knoten s nach t darf Knoten v *nicht* aufgrund von Knoten u blockiert werden.
Detaillierter Verlauf der Suche: siehe Tabelle 6.3
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Die dargestellte Contraction Hierarchy benötigt aufgrund der gewählten Knotenränge keine Shortcut-Pfeile.

Iteration	Horizontm.$_{\text{vor}}$	Lösungsm.$_{\text{vor}}$	Horizontm.$_{\text{rück}}$	Lösungsm.$_{\text{rück}}$
0	$\{s[0]\}$	$\emptyset$	$\{t[0]\}$	$\emptyset$
1	$\{u[3], w[7]\}$	$\{s[0]\}$	$\{t[0]\}$	$\emptyset$
2	$\{u[3], w[7]\}$	$\{s[0]\}$	$\emptyset$	$\{t[0]\}^{I}$
3	$\{w[7]\}$	$\{s, u[3]\}$	$\emptyset$	$\{t[0]\}$
4	$\{v[11]\}$	$\{s, u, w[7]\}$	$\emptyset$	$\{t[0]\}$
5	$\{t[15]\}^{III}$	$\{s, u, w, v[11]\}^{II}$	$\emptyset$	$\{t[0]\}$
6	$\emptyset$	$\{s, u, w, v, t[15]\}$	$\emptyset$	$\{t[0]\}$

^{I} Die Rückwärtssuche terminert, da ihre Horizontmenge leer ist.
^{II} Knoten v darf an dieser Stelle explizit *nicht* blockiert werden, obwohl ein kürzerer Weg von s zu v via Knoten u existiert.
^{III} Knoten t ist ein Brückenknoten; der implizierte Weg ist $p = \langle s, w, v, t \rangle$.

Tabelle 6.3: Horizont- und Lösungsmengen zur Wegsuche in Abbildung 6.11
Während der Wegsuche von Knoten s nach t werden alle Knoten über verbotsfreie Wege erreicht.

Grundsätzlich ließe sich an dieser Stelle noch das Ausmaß des Blockierens genauer spezifizieren: Ein Knoten u, von dem ausgehend ein Abbiegeverbot besteht, kann einen anderen Knotens v nur bezüglich des Expandierens in Richtung derjenigen Pfeile blockieren, zu denen ausgehend von (u, v) kein Abbiegeverbot besteht. Spiegelbildlich kann u bei der Rückwärtssuche Knoten v nur bezüglich derjenigen Pfeile blockieren, von denen ausgehend nach (v, u) kein Abbiegeverbot besteht.

Der Prüf- und Verwaltungsaufwand dieser strengeren Formulierung führt in der Praxis jedoch zu langsameren Wegsuchen. Effizienter arbeitet die oben genannte, schwächere Blockierbedingung. Zwar werden hierdurch weniger Knoten blockiert, da aber die Wegsuche auch gänzlich ohne Knotenblockierung korrekte Lösungen findet, ist dies kein Fehler. Formal zusammengefasst wird in einer Contraction Hierarchy $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2, TR)$ ein Knoten $v \in V$ bei einer Iteration der Vorwärtssuche daher effizient blockiert, wenn gilt:

$$\begin{aligned} & v.\mathit{distanz} = \min\{v'.\mathit{distanz} \mid v' \in \mathit{Horizontmenge}_{\text{vor}}\} \\ & \wedge\, u \in \mathcal{V}(v) \\ & \wedge\, v.\mathit{distanz}_{\text{vor}} > u.\mathit{distanz}_{\text{vor}} + c_{\text{ch}}(u, v) \\ & \wedge\, \mathit{rang}(v) < \mathit{rang}(u) \\ & \wedge\, \ddot{U}((u, v)(v, w)) \notin TR \\ & \wedge\, (v, w) \in E. \end{aligned} \tag{6.4}$$

Für die Rückwärtssuche müssen die „vor“-Indizes selbstverständlich zu „rück“-Indizes verändert werden. Zusätzlich müssen die letzten beiden Zeilen aus Gleichung (6.4), S. 229, ausgetauscht werden. Sie lauten stattdessen:

$$\begin{aligned} & \wedge\, \ddot{U}((w, v), (v, u)) \notin TR \text{ und} \\ & \wedge\, (w, v) \in E. \end{aligned}$$

Blockierung nie via Schlinge

Aus der Beschreibung des Verfahrens ergibt sich, dass die zu einer Knotenblockierung herangezogenen Pfeile niemals (Shortcut-)Schlingen sein können. Schließlich bedeutet eine Blockierung von Knoten v inhaltlich, dass er via Knoten u über einen kürzeren Weg hätte erreicht werden können. Ist aber $v = u$, kann diese Überprüfung unmöglich zu einem kürzeren Weg zu v führen.

Benchmarks zur Blockierung von Knoten im Rahmen der adaptiven Wegsuche werden in Abschnitt 6.6.2.1, S. 249 ff., vorgestellt. Hier zeigt sich ein ähnliches Verhalten wie bei der knotenbasierten Wegsuche *ohne* Berücksichtigung von Abbiegebeschränkungen (siehe Abschnitt 5.6.5, S. 185 ff.).

6.3.2.3 Zielgerichtete Wegsuche

Das Einbinden der in Abschnitt 5.2.2.1, S. 150, vorgestellten A*-Eingrenzung erfolgt ohne Anpassung: Ist bereits ein Weg p vom Start- zum Zielknoten der Wegsuche bekannt, werden nur noch diejenigen Knoten v der Horizontmenge expandiert, für die gilt:

$$v.\mathit{distanz} + \text{rd}(v) < c(p). \tag{6.5}$$

Auch die in Abschnitt 5.2.2.2, S. 152 ff., beschriebene bidirektionale A*-Suche kann problemlos auf die adaptive Wegsuche in Contraction Hierarchies mit Abbiegebeschränkungen übertragen werden. Wie die Benchmark-Messungen in Abschnitt 6.6.2.2 zeigen, ergeben sich mithilfe der zielgerichteten Komponente auch für die adaptive Wegsuche in Contraction Hierarchies deutliche Beschleunigungen.

6.4 Hierarchie-Erzeugung mit Abbiegebeschränkungen

Nachdem in Abschnitt 6.3 zwei Möglichkeiten für die Anpassung der *Wegsuche* auf einer bereits vorliegenden Hierarchie für die Berücksichtigung von Abbiegebeschränkungen gezeigt wurden, wird in diesem Abschnitt erläutert, wie sich die Berücksichtigung von Abbiegebeschränkungen bei der *Erzeugung* einer Contraction Hierarchy auswirkt.

Bei der Kontraktion eines Knotens wird gemäß des in Abschnitt 5.1, S. 132 ff., beschriebenen Vorgehens standardmäßig nur untersucht, ob der kürzeste Weg zwischen je zwei Nachbarknoten zerstört wird. Nur in diesem Fall wird ein Shortcut-Pfeil erzeugt. Dies stellt ein Problem für kürzeste, Abbiegebeschränkungen berücksichtigende Wege dar. Denn diese können – wie oben gezeigt – Teil-Wege beinhalten, die keine kürzesten Wege sind. Die Prüfung nach notwendigen Shortcut-Pfeilen muss hierfür also weiter greifen.

Zur Verdeutlichung dieser Überlegung werde das Beispiel in Abbildung 6.12 betrachtet, das Abbildung 6.6, S. 212, um vorgegebene Knotenränge erweitert. Die Knotenränge sind in diesem Beispiel so gewählt, dass die Hierarchie bei der bisher beschriebenen Methode der Knotenkontraktion ohne Berücksichtigung von Abbiegebeschränkungen keinen Shortcut-Pfeil enthält; die Abbildung stellt also mutmaßlich eine vollständige Hierarchie dar. Wie sich schnell zeigt, ermöglicht die derart gewonnene Hierarchie jedoch nicht die Wegsuche mit Berücksichtigung von Abbiegebeschränkungen.

Der kürzeste Weg $p = \langle u, v, w, z_1, z_2, w, v, x \rangle$ mit $c(p) = 17$ zwischen Knoten u und x ist hier mit den im vorangegangenen Abschnitt vorgestellten Wegsucheverfahren nicht ermittelbar. Stattdessen liefern beide Verfahren *keinen* Weg. Bei der pfeilbasierten Wegsuche terminiert die Vorwärtssuche sofort nach Aufnahme von Pfeil (u, v) in die Lösungsmenge und die Rückwärtssuche nach Aufnahme von Pfeil (v, x). Analog endet auch die adaptive Wegsuche bei Knoten v sowohl in Vorwärts- als auch in Rückwärtsrichtung. Der Grund liegt im Fehlen einer Shortcut-Schlinge bei Knoten v, wie nachfolgend dargelegt wird.

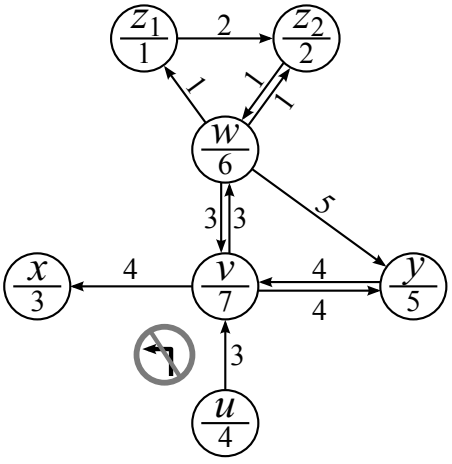


Abbildung 6.12: Fehlerhafte Contraction Hierarchy mit Abbiegebeschränkungen
Aufgrund der Knotenränge existieren hier gemäß Verfahren 10, S. 135, fälschlicherweise keine Shortcut-Pfeile. Eine Wegsuche von Knoten u nach Knoten x liefert daher nicht die korrekte Lösung.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge.

Eine korrekte Erzeugung von Contraction Hierarchies, die auch Abbiegebeschränkungen berücksichtigen, beschreiben Geisberger u. Vetter [2011]. Der Grundgedanke ist dabei eine pfeilbasierte Sicht auf die Hierarchie und betrifft die Entscheidung, wann bei der Kontraktion eines Knotens v ein Shortcut-Pfeil erzeugt werden muss. Die Idee wird im Folgenden näher erläutert und abschließend in Verfahren 23, S. 233, zusammengefasst.

Ohne Abbiegebeschränkungen wird ein Knoten v überbrückender Shortcut-Pfeil gemäß Abschnitt 5.1, S. 132 ff., immer dann erzeugt, wenn der einzige kürzeste Weg zwischen zwei zu v adjazenten Knoten $u \in \mathcal{V}(v)$ und $w \in \mathcal{N}(v)$ über v führt. Bei der pfeilbasierten Sichtweise werden die (Shortcut-)Pfeile von u nach v und von v nach w betrachtet. Ähnlich wie bei der pfeilbasierten Wegsuche in Contraction Hierarchies gilt die Erkenntnis, dass ausschließlich kürzeste Wege zwischen *Original-Pfeilen* relevant sind und nicht zerstört werden dürfen[26].

Fokus auf Original-Pfeile

Dabei sei gemäß der Beschreibung der Knotenkontraktion aus Abschnitt 5.1 daran erinnert, dass die in der Fallunterscheidung verwendeten Knoten u und w per Definition einen höheren Rang besitzen müssen als der zu kontrahierende Knoten v. Denn nach dessen Kontraktion werden alle seine adjazenten Pfeile temporär aus dem Graphen entfernt und es entsteht der nächste Overlay-Graph.

Im Vergleich zur Hierarchie-Erzeugung ohne Abbiegebeschränkungen sind durch dieses Vorgehen einerseits erheblich mehr Prüfungen nötig, um herauszufinden, ob ein Shortcut-Pfeil erstellt werden muss. Andererseits *entstehen* auch sehr viel mehr Shortcut-Pfeile.

Reduktion der zu erzeugenden Shortcut-Pfeile Bei der oben beschriebenen Variante der Shortcut-Pfeile-Erzeugung fällt auf, dass bei der Suche nach kürzesten, Abbiegebeschränkungen berücksichtigenden Wegen jeweils der erste und letzte Original-Pfeil bereits vorgegeben ist[27]. Geisberger u. Vetter [2011, S. 104] merken daher an, dass es ausreicht einen Shortcut-Pfeil $(u \dots w)$ nur dann zu erzeugen, wenn dieser einen kürzesten, zulässigen Weg zwischen einem in $(u \dots w)$ eingehenden und einem aus $(u \dots w)$ führenden Original-Pfeil darstellt. Anstatt also den Weg von $(u \to u')$ nach $(w' \to w)$ zu suchen, überprüfen sie sämtliche Wege von $(u'' \to u)$ nach $(w \to w'')$.[28] Offensichtlich erfordert dies erheblich mehr lokale Suchen für die Entscheidung, ob ein Shortcut-Pfeil erzeugt werden muss.

Im Rahmen dieser Arbeit kommt ein leicht adaptiertes Verfahren zum Einsatz. Hierfür wird zunächst die „schwächere“ Prüfung nach Notwendigkeit eines Shortcut-Pfeils durchgeführt. Denn repräsentiert der potentiell zu erzeugende Shortcut-Pfeil $((u \to u' \dots v' \to v), (v \to v'' \dots w' \to w))$ nicht einmal den einzigen kürzesten und zulässigen Weg zwischen $(u \to u')$ und $(w' \to w)$, dann muss er in keinem Fall erzeugt werden. Die aufwendige Prüfung aller paarweiser Verbindungen zwischen Original-Pfeilen der Form $(u'' \to u)$ und $(w \to w'')$ kann dann entfallen.

[26] Ein optimaler Weg zwischen einem Startpfeil (s, s') und einem Zielpfeil (t', t) ist ein kostenminimaler, zulässiger Weg, der mit (s, s') beginnt und mit (t', t) endet (siehe auch Seite 206).

[27] siehe auch Seite 206, Stichwort „Optimale Wege zwischen Pfeilen“

[28] Das Vorgehen bezeichnen Geisberger u. Vetter [2011, S. 104] im englischen Original als *aggressive local search*.

Verfahren 23, S. 233, gibt den Pseudocode für das adaptierte Vorgehen zusammengefasst wieder. Die gerade beschriebene Beschleunigung des Verfahrens mithilfe der „schwächeren“ Prüfung geschieht in Zeile 20. Die Darstellung der Erzeugung eines Shortcut-Pfeils, der eine oder mehrere Schlingen beinhaltet, erfolgt in Zeile 27 aus Platzgründen verkürzt.

Eine Hierarchie, die nach diesem Verfahren erzeugt wurde, zeigt Abbildung 6.13, S. 232. Im Gegensatz zum Graphen in Abbildung 6.12, S. 230, sind hier die notwendigen Shortcut-Pfeile beinhaltet, so dass sämtliche Abbiegebeschränkungen berücksichtigende kürzesten Wege von den Verfahren aus Abschnitt 6.3 gefunden werden können. Einzig die ebenfalls nach Verfahren 23, S. 233, zu generierende Shortcut-Schlinge über die Pfeile $(v \to w)$ und $(w \to y \ldots y \to v)$ ist aus Gründen der Übersichtlichkeit nicht mit angegeben.

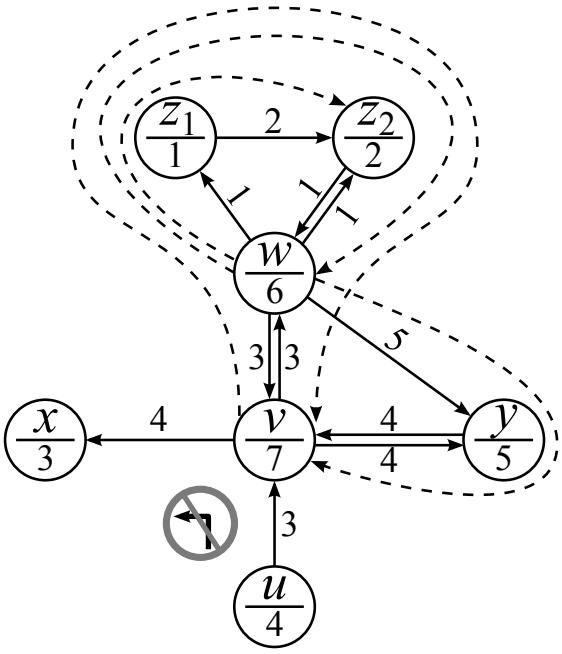


Abbildung 6.13: Korrekte Contraction Hierarchy mit Abbiegebeschränkungen
Auch die Wegsuche von Knoten u nach Knoten x liefert in dieser Hierarchie eine korrekte Lösung. In Pfeil-Notation lautet der von den Verfahren in Abschnitt 6.3 auffindbare kürzeste Weg p hier $p = \langle (u \to v), (v \to w \ldots w \to v), (v \to x) \rangle$.
Buchstaben in Knoten sind deren Bezeichner; die Zahlen darunter Knotenränge. Gestrichelte Linien symbolisieren Shortcut-Pfeile. Deren Länge ist nicht explizit angegeben.

Verfahren 23 Erzeugung einer Contraction Hierarchy mit Abbiegebeschränkungen

Gegeben:

Nicht-negativ bewerteter Digraph $G = (V, E, c, TR)$,
Prioritätswarteschlange P mit allen nach Wichtigkeit sortierten Knoten aus G

1: Setze $e_1((u,v)) \coloneqq e_2((u,v)) \coloneqq (u,v)$ und $c_{\text{ch}}(u,v) \coloneqq c(u,v)\ \forall (u,v) \in E$ ▸ Initialisierung
2: $p^* \coloneqq NULL$ ▸ Platzhalter für den Weg, den möglicherweise ein neuer Shortcut-Pfeil überbrücken muss
3: $p_{v,v} \coloneqq NULL$ ▸ Platzhalter für den kürzesten Teilweg, der ausschließlich aus Schlingen besteht
4: **while** $|P| > 1$ **do** ▸ Im Gegensatz zu Verfahren 10, S. 135, kann um den letzten Knoten eine Schlinge entstehen
5: $v \coloneqq \min\{P\}$; $P \coloneqq P \setminus v$ ▸ Wähle und entferne kleinstes Element aus der Prioritätswarteschlange
6: **for all** $(u,v),(v,w) \in E$ **do** ▸ (u,v) und (v,w) sind evtl. Shortcut-Pfeile; auch $u = w$ ist möglich
7: $p_{v,v} \coloneqq NULL$
8: **if** Weg $p = \langle (u,v),(v,w) \rangle$ ist nicht zulässig **then**
9: $p_{v,v}$:= kürzester, aus Schlingen bei v bestehender Weg, so dass $\langle (u,v), p_{v,v}, (v,w) \rangle$ ein zulässiger Weg ist
10: **if** $p_{v,v} = NULL$ **then goto:** 6 **end if** ▸ Abbruch dieser Shortcut-Suche
11: **end if**
12: **if** $p_{v,v} = NULL$ **then** ▸ Erzeugung des Weges p^*, über den ggf. ein Shortcut-Pfeil führen muss
13: $p^* \coloneqq \langle (u,v),(v,w) \rangle$
14: **else**
15: $p^* \coloneqq \langle (u,v), p_{v,v}, (v,w) \rangle$
16: **end if**
17: Sei $(u \to u')$ der erste von (u,v) überbrückte Original-Pfeil ▸ d. h., evtl. ist auch $u' = v$
18: Sei $(w' \to w)$ der letzte von (v,w) überbrückte Original-Pfeil ▸ d. h., evtl. ist auch $w' = v$
19: **if** p^* ist *nicht* einziger kürzester und zul. Weg von $(u \to u')$ nach $(w' \to w)$ **then**
20: **goto:** 6 ▸ vorzeitiger Abbruch, durch „schwächere“ Prüfung möglich
21: **end if**
22: **for all** Original-Pfeilpaare der Form $(u'' \to u)$ und $(w \to w'') \in E$ **do**
23: **if** Einziger kürzester zul. Weg von $(u'' \to u)$ nach $(w \to w'')$ beinhaltet p^* **then**
24: **if** $p_{v,v} = NULL$ **then** ▸ Keine Schlinge vom Shortcut-Pfeil zu überbrücken?
25: Erzeuge neuen Shortcut-Pfeil $(u \ldots w)$ mit
$e_1((u \ldots w)) = (u,v)$ und $e_2((u \ldots w)) = (v,w)$ sowie
$c_{\text{ch}}(u \ldots w) = c_{\text{ch}}(u,v) + c_{\text{ch}}(v,w)$ und füge ihn zu E hinzu
26: **else** ▸ Formal überbrückt jeder Shortcut-Pfeil exakt zwei andere Pfeile, daher...
27: Erzeuge sukzessive die notwendigen Shortcut-Schlingen über $p_{v,v}$,
erzeuge Shortcut-Pfeil $(u \ldots v)$ aus (u,v) und der Schlinge über $p_{v,v}$,
erzeuge Shortcut-Pfeil $(u \ldots w)$ aus $(u \ldots v)$ und (v,w) und
aktualisiere dabei jeweils e_1, e_2 und c_{ch} analog zu Zeile 25
28: **end if**
29: **end if**
30: **end for**
31: **end for**
32: Entferne alle zu v adjazenten (Shortcut-)Pfeile temporär aus E
33: Entferne v temporär aus V ▸ Ende der Kontraktion von v
34: **end while**
35: Füge alle temporär entfernten Knoten zu V und alle temporär entfernten (Shortcut-)Pfeile wieder zu E hinzu.

Ergebnis: $G_{\text{ch}} = (V, E, c_{\text{ch}}, e_1, e_2, TR)$ beschreibt eine vollständige Contraction Hierarchy mit Berücksichtigung von Abbiegebeschränkungen.

6.5 Implementierungsdetails der Verfahren

In diesem Abschnitt werden Besonderheiten bei der Implementierung der in diesem Kapitel vorgestellten Verfahren näher beschrieben. Wie auch in Abschnitt 5.5, S. 167 ff., geht es dabei weitgehend um Performance-Verbesserungen.

6.5.1 Implementierungsdetails der adaptiven Wegsuche

Die Implementierung der adaptiven Wegsuche weicht an einigen Stellen vom Pseudocode ab. In diesem Abschnitt werden die Differenzen herausgearbeitet und begründet.

Vereinfachte erste Iteration Bei der adaptiven Wegsuche sind stets ein Start- und ein Zielknoten gegeben. Von diesen ausgehend wird je Iteration der Vorwärts- bzw. Rückwärtssuche in alle aus- bzw. eingehenden Pfeile expandiert. Dabei werden nur solche Pfeile expandiert, die keine Abbiegebeschränkung verletzen[29]. Da bei einem knotenbasierten Verfahren – die adaptive Wegsuche beginnt als ein solches – mit der ersten Iteration auch der erste Pfeil eines Weges erreicht wird, kann an dieser Stelle unmöglich eine Abbiegebeschränkung verletzt werden. Die Prüfung muss also erst ab der zweiten Iteration der Vorwärts- und Rückwärtssuche erfolgen.

Index bei mehrfacher Aufnahme von Knoten Die Möglichkeit, einen Knoten mehrfach in die Horizont- und Lösungsmengen aufzunehmen, verlangt eine Adaption der Indizes für diese Mengen. Bislang kamen hierfür Hashtabellen zum Einsatz, deren Schlüssel Knoten-Ids sind, zu denen die – um die nötigen Attribute erweiterten[30] – Knoten gespeichert wurden. Doch in einer Hashtabelle kann zu jedem Schlüssel nur höchstens ein Eintrag vorliegen. Die vorliegende Implementierung überkommt diese Limitierung, indem die Werte Knotenobjekte sind, die neben den bislang vorgestellten Attributen `distanz`, `vorgänger`, `erreichtVia` und `verbotsfrei` ein zusätzliches Attribut `nächstesPendant` besitzen, dessen Datentyp seinerseits ein Knotenobjekt ist. Dies ermöglicht es, eine einfach-verkettete Liste aus Knotenobjekten zu erstellen.

Soll nun ein Knoten bei der adaptiven Wegsuche indiziert werden, wird zunächst geprüft, ob im Index bereits ein Objekt zu ebendiesem Knoten existiert. Ist dies nicht der Fall, kann der Knoten wie üblich indiziert werden. Wurde derselbe Knoten jedoch vorab schon einmal der Horizontmenge hinzugefügt, so existiert bereits ein Eintrag in der Index-Tabelle. Für eine Mehrfach-Aufnahme desselben Knotens gemäß der Fälle (3), (7) oder (8) in Abbildung 6.7, S. 213, in den Index wird das neue Knotenobjekt dann als `nächstesPendant` des bereits existierenden Eintrags gespeichert.

Zu jedem Hash-Schlüssel kann somit eine Knotenliste gespeichert werden, deren Knoten jeweils dieselbe Id besitzen. Die Unterscheidung zwischen Knoten mit gleicher Id erfolgt dann über ihre `erreichtVia`-Attribute. Diese sind ihrerseits innerhalb einer solchen Liste eindeutig, denn sie repräsentieren den pfeilbasierten Teil der adaptiven Wegsuche.

Die Verwaltung derartiger Knoten-Listen erfordert nur einen sehr kleinen und konstanten Mehraufwand: In der Hashtabelle wird dazu stets das im Rahmen einer Wegsuche zuerst gefundene Knotenobjekt gespeichert. Alle nachfolgenden *Pendants* dieses

[29] siehe Zeile 15 in Verfahren 18, S. 215,
[30] siehe Abschnitt 5.5.6 auf Seite 176, Stichwort „Temporäre, erweiterte Knotenobjekte“

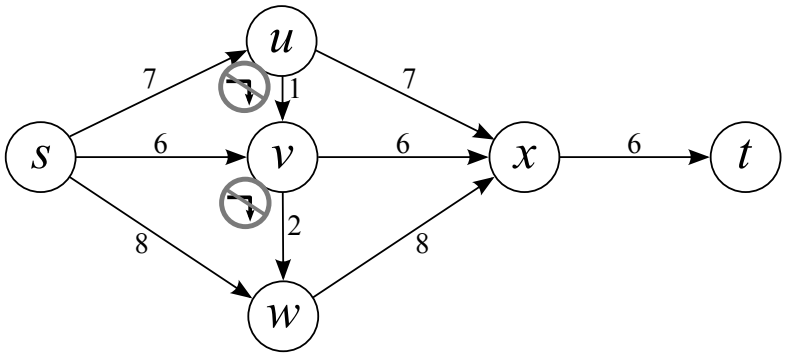
Knotens, die also über andere Pfeile erreicht wurden, werden stets an die *zweite* Position innerhalb der Knotenliste gespeichert. Dies hat den Vorteil, dass das zu einem Hash-Schlüssel gespeicherte Objekt im Verlauf der Suche niemals ausgetauscht, sondern allenfalls sein `nächstesPendant`-Attribut angepasst werden muss.

Zu besseren Verdeutlichung der Knotenlisten gibt Abbildung 6.14, S. 236, ein Beispiel. Im Graphen aus Abbildung 6.14a werde die Wegsuche von Knoten s nach t betrachtet. Nach der ersten Iteration, in der Knoten s in die Lösungsmenge aufgenommen wird, werden in der zweiten und dritten Iteration Knoten u und v ebenfalls in die Lösungsmenge aufgenommen, jedoch jeweils über einen *verbotsgefährdeten* Weg. Über beide Knoten wird Knoten x – ebenfalls verbotsgefährdet – erreicht und daher zweimal in die Horizontmenge aufgenommen. In Abbildungen 6.14b und 6.14c ist dies durch die Schreibweise x_u bzw. x_v ausgedrückt. Der Inhalt des Hash-Index nach dieser Iteration ist in Abbildung 6.14c dargestellt. Für den Hash-Schlüssel „x“ wurde eine Knotenliste bestehend aus den Knotenobjekten x_v und x_u wie oben beschrieben angelegt.

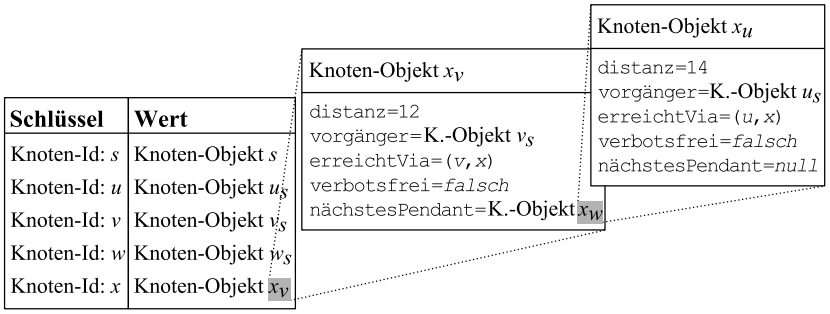
In der vierten Iteration gelangt Knoten w *verbotsfrei* in die Lösungsmenge und wiederum wird x erreicht und muss ein drittes Mal indiziert werden; nun in der Form x_w. Im Hash-Index wird beim Knotenobjekt x_v daraufhin lediglich das `nächstesPendant`-Attribut angepasst, so dass auf x_v nun x_w und dann x_u folgt.

Markieren statt Entfernen von Knoten Der Pseudocode zur adaptiven Wegsuche sieht vor, beim Eintritt eines *verbotsfrei* erreichten Knotens v in die Lösungsmenge alle weiteren möglichen Vorkommen von v aus der Horizontmenge zu entfernen (siehe Zeile 12 in Verfahren 18, S. 215). Die vorliegende Implementierung verzichtet in dieser Situation auf das Löschen der Knoten aus der Prioritätswarteschlange, in der die Horizontmenge vorgehalten wird. Stattdessen werden die entsprechenden Knoten über ein weiteres Knoten-Attribut als obsolet gekennzeichnet. Gelangen diese Knotenobjekte in der Folge bei einer Dijkstra-Suche an den Anfang der Prioritätswarteschlange (siehe Zeile 5 in Verfahren 18, S. 215), werden sie zwar aus der Horizontmenge entfernt, aber nicht mehr expandiert. Das Vorgehen ähnelt in dieser Hinsicht daher einer Knotenblockierung gemäß Abschnitt 6.3.2.2, S. 227.

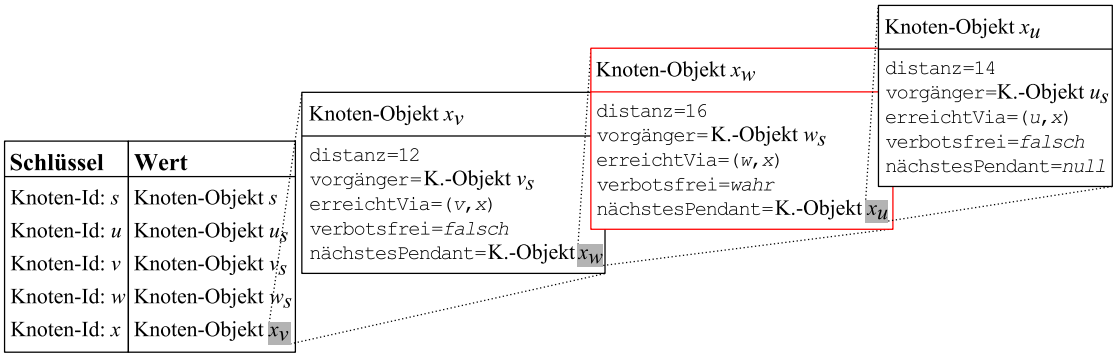
Auf diese Weise werden obsolete Knotenobjekte nicht in jedem Fall aus der Prioritätswarteschlange entfernt. Der Aufwand, sie zu suchen und im Anschluss die Heap-Eigenschaft der Warteschlange wiederherzustellen, entfällt daher mitunter, was sich positiv auf die Laufzeit auswirkt. Einfache Laufzeit-Tests bestätigten diese Einschätzung; der Unterschied im Vergleich zum direkten Löschen obsoleter Knotenobjekte ist jedoch gering.



(a) Beispielgraph für Mehrfachaufnahme eines Knotens



(b) Dargestellt ist die Hashtabelle nach der dritten Iteration bei der Wegsuche von Knoten s nach t. Knoten x wurde zum zweiten Mal aufgenommen.



(c) Dargestellt ist die Hashtabelle nach der vierten Iteration bei der Wegsuche von Knoten s nach t. Knoten x wurde zum dritten Mal aufgenommen und an die zweite Position (rot hervorgehoben) innerhalb der Knoten-Liste in der Hashtabelle eingefügt.

Abbildung 6.14: Hash-Index bei mehrfacher Knotenaufnahme in d. adaptiven Wegsuche

6.5.2 Möglichkeiten der Parallelisierung bei der Erzeugung einer Contraction Hierarchy

Da die Erzeugung einer Abbiegebeschränkungen berücksichtigenden Contraction Hierarchy pfeilbasiert erfolgen muss und somit die Geschwindigkeitsvorteile der adaptiven Wegsuche[31] nicht ausschöpfen kann, steigt der Aufwand im Vergleich zu einer CH *ohne Abbiegebeschränkungen* erheblich. Wie beschrieben, muss bei der Kontraktion von Knoten v nicht nur überprüft werden, ob v Teil des einzigen kürzesten Weges zwischen zweier seiner Nachbarknoten u und w ist. Stattdessen muss ein Shortcut-Pfeil $((u,v),(v,w))$ genau dann eingefügt werden, wenn der einzige kürzeste Weg zwischen zwei zu u und w adjazenten Original-Pfeilen $(u' \rightarrow u)$ und $(w \rightarrow w')$ über v führt.

Im Vergleich zu einer Abbiegebeschränkungen ignorierenden Implementierung wächst damit der Aufwand einer Knotenkontraktion quadratisch mit dem durchschnittlichen Knotengrad des zugrunde liegenden Graphen. Zusätzlich verstärkt sich die Auswirkung während der Hierarchie-Erzeugung selbst, da hier durch das Einfügen neuer Shortcut-Pfeile besonders bei der Kontraktion der letzten Knoten der durchschnittliche Knotengrad überproportional wächst. Der Vergleich von Abbildung 6.15 und Abbildung 5.15, S. 166, zeigt den deutlichen Unterschied insbesondere bei der Erzeugung einer Contraction Hierarchy für *kürzeste* Wege.

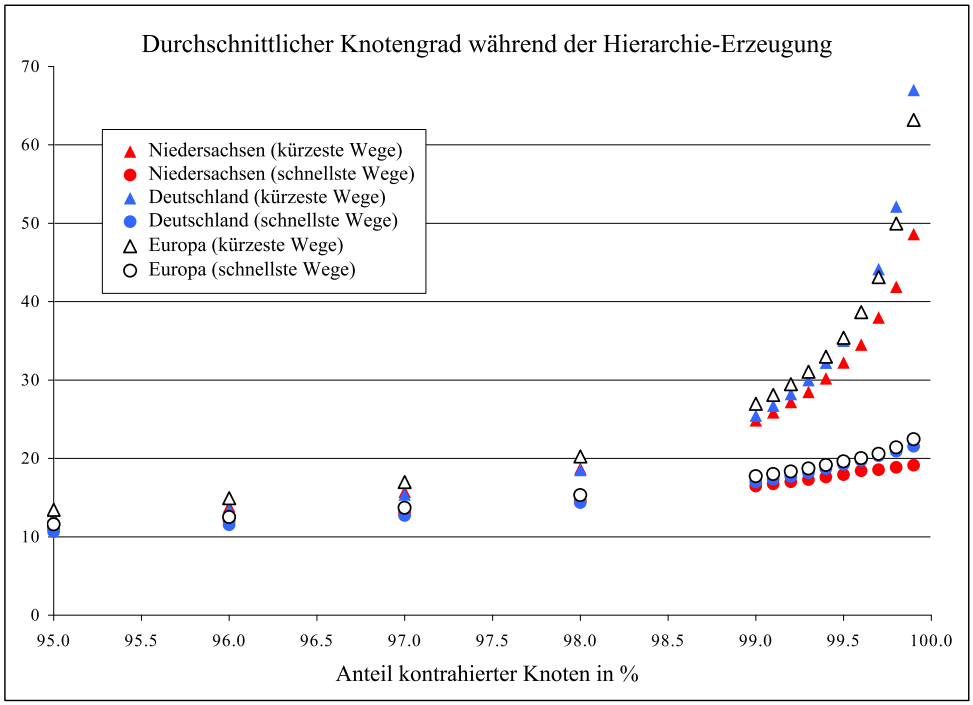


Abbildung 6.15: Durchschnittlicher Knotengrad während der Hierarchie-Erzeugung
Deutlich erkennbar ist der überproportionale Anstieg gegen Ende der Hierarchie-Erzeugung. Im Vergleich zu Abbildung 5.15, S. 166, ist der Anstieg deutlich steiler.
Zugrunde liegende Knotensortierung: $2R{+}8EQ{+}4OEQ{+}1Q{+}1RAND$

[31] siehe zum Beispiel Verfahren 18, S. 215, in Abschnitt 6.2.4

Die Erzeugung der Hierarchie zu parallelisieren ist daher sowohl aufgrund der höheren Komplexität einer einzelnen Knotenkontraktion als auch der insgesamt größeren Zahl von Pfeilen und damit des durchschnittlichen Knotengrads vorteilhafter als bei der Hierarchie-Erzeugung ohne Berücksichtigung von Abbiegebeschränkungen. Der Vergleich der in den Abbildungen 6.16 und 5.18 auf den Seiten 238 und 171 dargestellten Messungen unterstreicht diese Einschätzung.

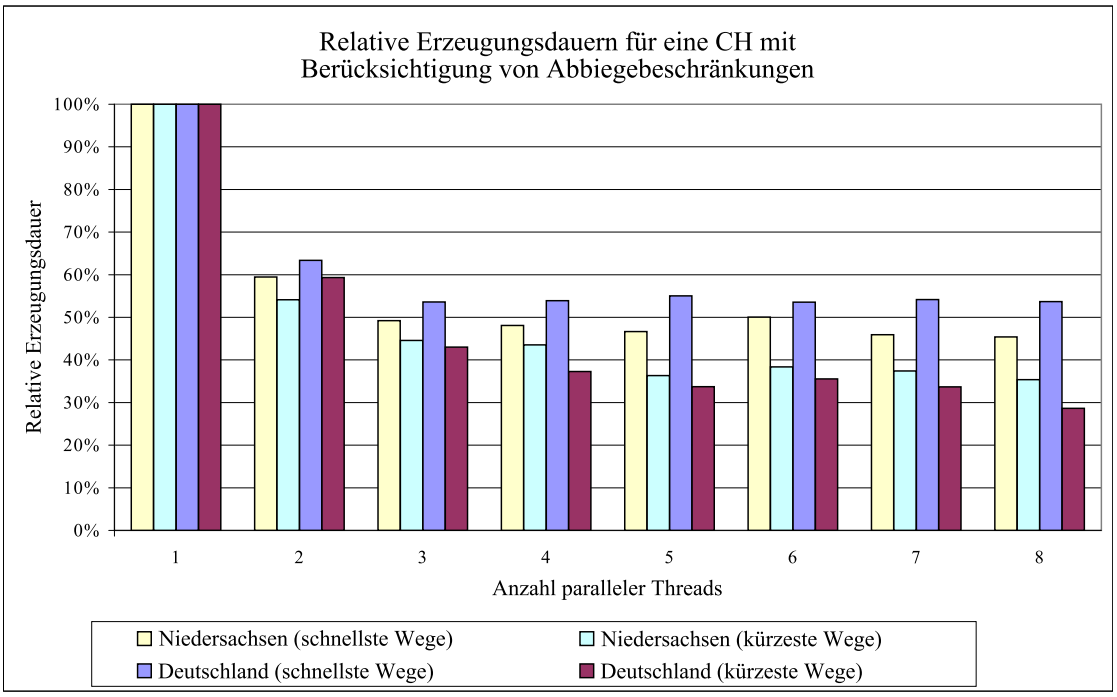


Abbildung 6.16: Parallele CH-Erzeugung mit Abbiegebeschränkungen
Die Parallelisierung skaliert im Vergleich zur CH-Erzeugung ohne Abbiegebeschränkungen (siehe Abbildung 5.18, S. 171) deutlich besser, da pro gleichzeitig bearbeiteter Teilaufgaben mehr Berechnungen anfallen. Zugrunde liegende Knotensortierung: $2R{+}8EQ{+}4OEQ{+}1Q{+}1RAND$

6.6 Benchmarks für Contraction Hierarchies mit Abbiegebeschränkungen

Die in diesem Abschnitt vorgestellten Benchmarks orientieren sich im Aufbau an denjenigen, die bereits in Abschnitt 5.6, S. 177 ff., präsentiert wurden. Allerdings wird an dieser Stelle der Fokus auf den Vergleich der *Verfahren* gelegt und weniger auf die konkrete Implementierung. So erfolgt die Sortierung der Horizontmengen stets auf Grundlage von Priority Queues[32]. Weiterhin verwenden alle hier vorgestellten Implementierungen *Hashtabellen* für die Pfeil- bzw. Knoten-Indizierung[33].

Wie auch bei den Benchmarks zu Contraction Hierarchies *ohne* Abbiegebeschränkungen in Abschnitt 5.6, S. 177 ff., werden viele Messungen mittels Boxplot-Diagrammen dargestellt. Der Übersichtlichkeit halber erfolgt die Darstellung hier erneut ohne Antennen. Die vollständigen Diagramme inklusive Antennen zeigt Anhang B, S. 319 ff.

Für diese Benchmarks wurde ausschließlich die OSM-Straßenkarte Europas verwendet.[34] Tabelle 6.4 listet die wesentlichen Eigenschaften der daraus generierten Contraction Hierarchies für schnellste und für kürzeste Wege. Die Schnellste-Wege-Metrik basierte auf der Fahrzeugkategorie „Transporter“.[35]

	Europa	
Wege:	schnellste	kürzeste
Anzahl Knoten	15 462 474	15 462 474
Anzahl Original-Pfeile	37 249 828	37 249 559
Anzahl Shortcut-Pfeile	58 073 740	68 306 383
(davon Shortcut-Schlingen)	4 916 546	5 309 349
Summe der Pfeile	95 323 568	105 555 942
Anzahl Abbiegegebote	37 845	
Anzahl Abbiegeverbote	17 167	

Tabelle 6.4: Kennzahlen der verwendeten Straßenkarten

Der Vergleich mit Tabelle 5.2, S. 179, zeigt den deutlichen erhöhten Speicherbedarf für die Karten mit Abbiegebeschränkungen. Für beide betrachteten Metriken sind mehr als doppelt so viele Shortcut-Pfeile notwendig als bei den Contraction Hierarchies ohne Abbiegebeschränkungen.

[32] siehe auch Abschnitt 5.6.3, S. 181 ff.
[33] siehe auch Abschnitt 5.6.4, S. 183 ff.
[34] Zu deren Herkunft siehe Kapitel 3, S. 61 ff., und insbesondere Tabelle 3.6, S. 85.
[35] für das Geschwindigkeitsprofil siehe Tabelle 3.5, S. 83

6.6.1 Benchmarks zur pfeilbasierten Wegsuche

Dieser Abschnitt stellt Messungen bezüglich der in Abschnitt 6.3.1, S. 219 ff., behandelten Varianten der Wegsuche in Contraction Hierarchies vor. In Analogie zur Wegsuche in Contraction Hierarchies *ohne* Berücksichtigung von Abbiegebeschränkungen[36] werden die Auswirkungen von

- Pfeilblockierung,
- zielgerichteter Suche sowie
- der Parallelisierung mit unterschiedlicher Schrittweite

auf die Performance der Wegsuche untersucht.

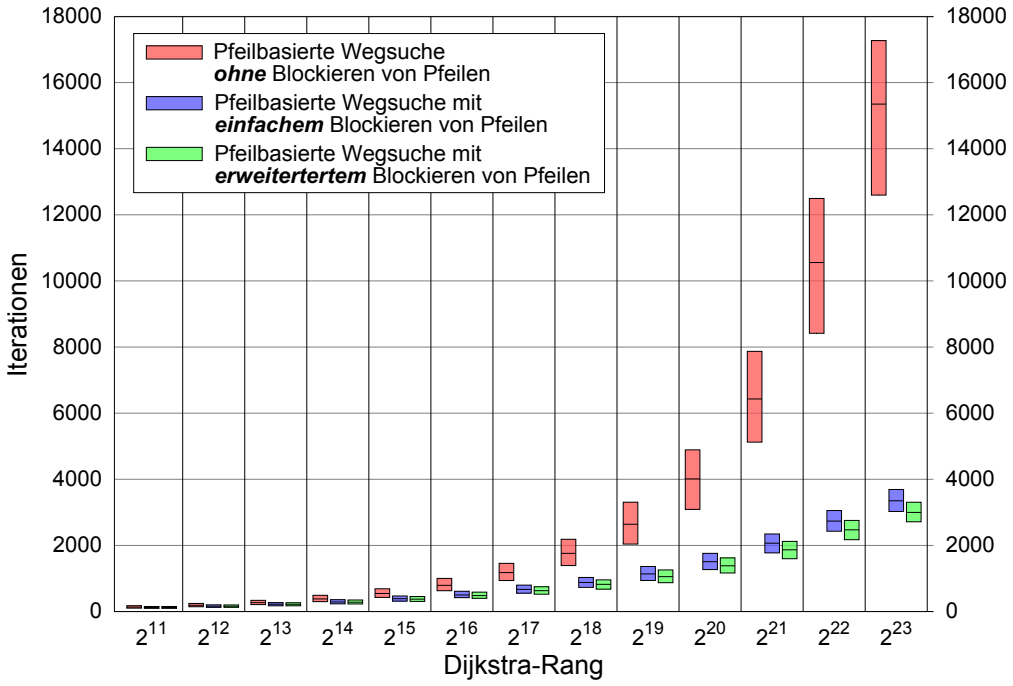
6.6.1.1 Benchmarks zur Blockierung von Pfeilen

Dieser Abschnitt umfasst die Messungen zu den in Abschnitt 6.3.1.2 beschriebenen Techniken der Pfeilblockierung. In Anlehnung an Abschnitt 5.6.5, bei dem die Knotenblockierung in Wegsuchen in Contraction Hierarchies *ohne* Berücksichtigung von Abbiegebeschränkungen untersucht wurde, werden bei der Pfeilblockierung für Wegsuchen *mit* Abbiegebeschränkungen drei Szenarien betrachtet:

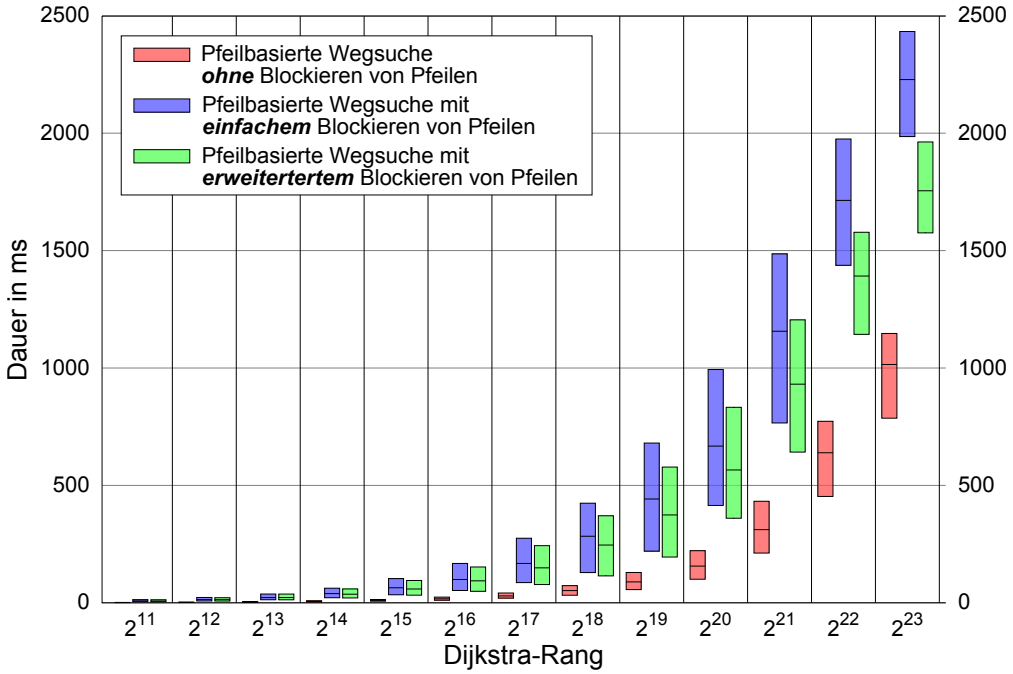
1. kein Blockieren von Pfeilen,
2. einfaches Blockieren von Pfeilen sowie
3. erweitertes Blockieren von Pfeilen.

Abbildungen 6.17 und 6.18 auf Seite 242 fassen die Ergebnisse der Wegsuchen für Dijkstra-Ränge im Zweierpotenzen-Bereich zusammen. Beim Vergleich der beiden Abbildungen ist die unterschiedliche Skalierung der Y-Achse zu beachten. Die zugehörigen Abbildungen mit Antennen und Ausreißer befinden sich im Anhang auf Seite 328 f.

[36] siehe Abschnitt 5.6, S. 177 ff.

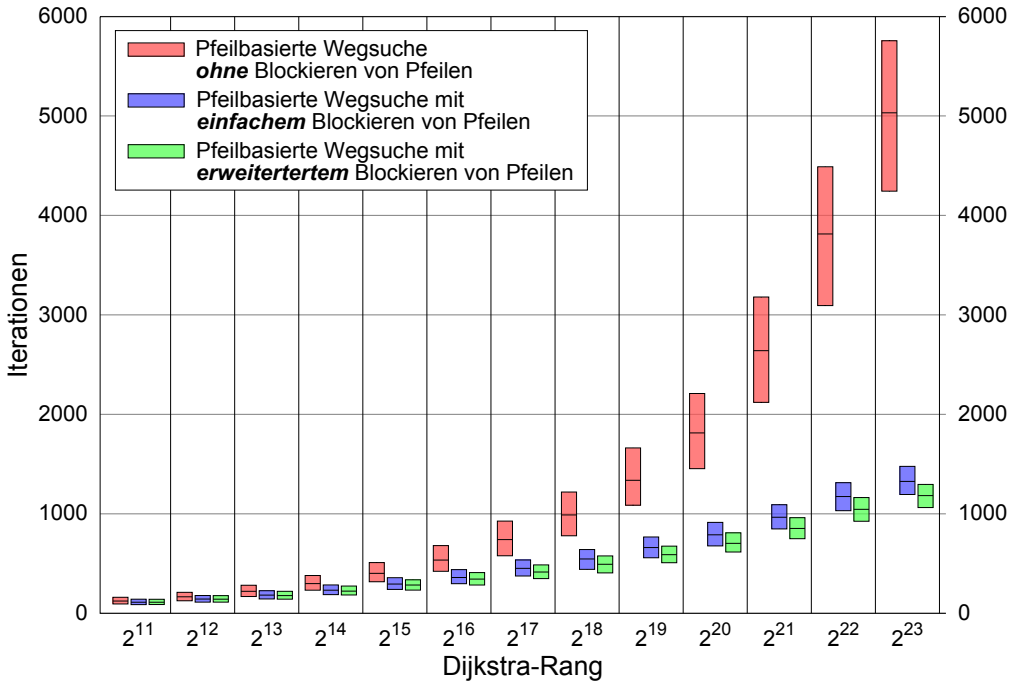


(a) Benötigte Iterationen

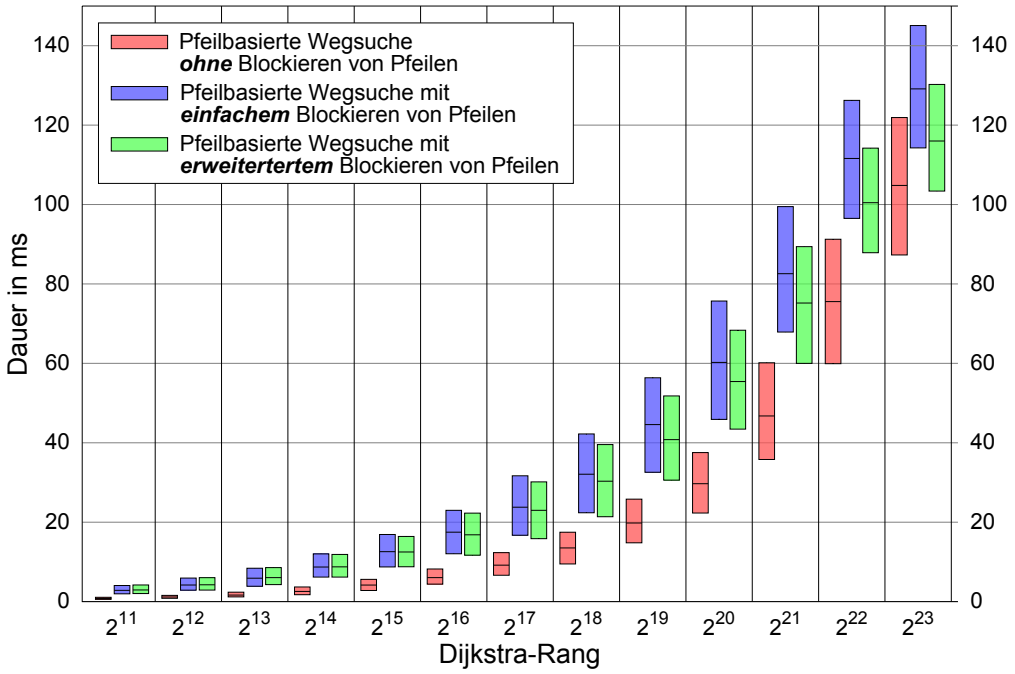


(b) Suchdauer

Abbildung 6.17: Benchmark für Pfeilblockierung bei Suchen nach *kürzesten* Wegen Zugrunde liegendes Verfahren ist die pfeilbasierte Wegsuche in Contraction Hierarchies.



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.18: Benchmark für Pfeilblockierung bei Suchen nach *schnellsten* Wegen Zugrunde liegendes Verfahren ist die pfeilbasierte Wegsuche in Contraction Hierarchies.

Interpretation der Messergebnisse Bezogen auf die notwendigen Iterationen können wie auch bei der abbiegebeschränkungsfreien Wegsuche sehr große Einsparungen durch die Blockierungen erzielt werden. Allerdings gehen diese zu Lasten der Performance. Der Mehraufwand für die Untersuchung, ob ein Pfeile blockiert werden kann, ist hier so hoch, dass die Such-Variante *ohne* Pfeilblockierung ihre Ergebnisse am schnellsten liefert.

Wiederum benötigt die Suche nach *kürzesten* Wegen erheblich mehr Iterationen und damit einhergehend auch Zeit als die Suche nach *schnellsten* Wegen. Daher sind die zu beobachtenden Unterschiede bei den Such-Varianten hier deutlicher.

Im direkten Vergleich mit der abbiegebeschränkungsfreien Wegsuche[37] benötigt die pfeilbasierte Wegsuche etwa zwei- bis dreimal so viele Iterationen. Bezogen auf die Suchdauer ist der Unterschied aufgrund der aufwendigeren Suche nach Brückenknoten erheblich stärker ausgeprägt.

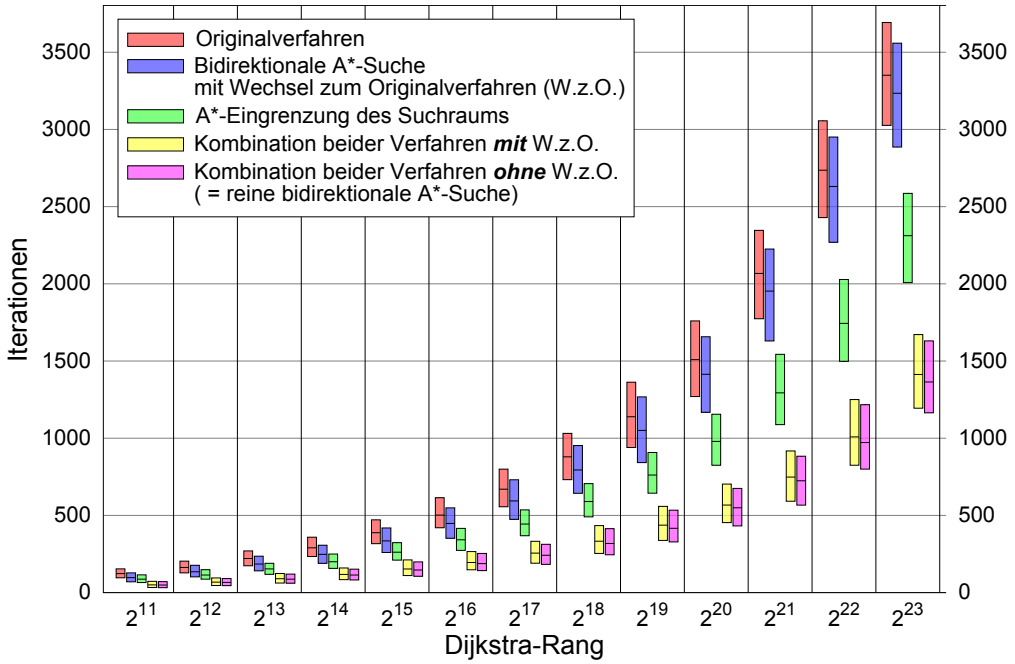
6.6.1.2 Benchmarks zur zielgerichteten Wegsuche

In diesem Abschnitt werden um eine Zielrichtung erweiterte Varianten der pfeilbasierten Wegsuche untersucht. Ihre Grundlagen sind in Abschnitt 6.3.1.3 erläutert. Die Messung liefert eine Gegenüberstellung des originalen, pfeilbasierten Verfahrens mit den folgenden vier Varianten:

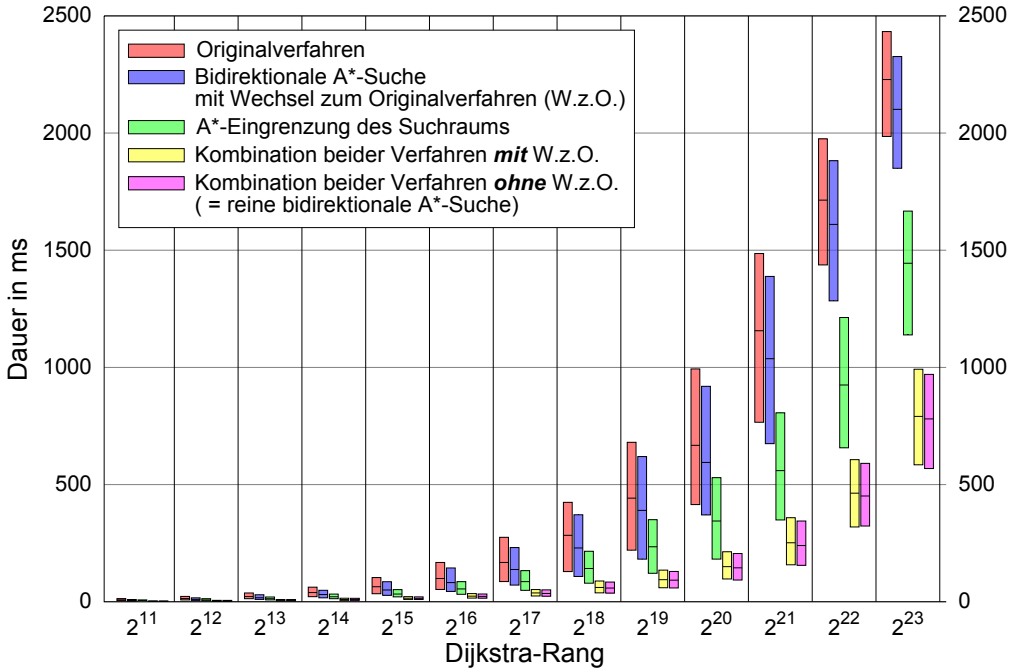
1. der A*-Eingrenzung des Suchraums[38],
2. der bidirektionalen A*-Suche mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens in Anlehnung an Nowak u. a. [2012][39],
3. der Kombination aus 1. und 2. mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens in Anlehnung an Nowak u. a. [2012] und
4. der Kombination aus 1. und 2. als reine bidirektionale A*-Suche.

Damit die Messungen besser mit denen der Verfahren der abbiegebeschränkungsfreien Suche aus Abschnitt 5.6.6, S. 188 ff., vergleichbar sind, arbeiten alle vier Verfahren mit *einfachem Blockieren von Pfeilen*[40]. Abbildungen 6.19 und 6.20 zeigen die Messergebnisse der zielgerichteten Wegsuche als Boxplot-Diagramme für die Suche nach kürzesten bzw. schnellsten Wegen. Die vollständigen Diagramme inklusive Antennen und Ausreißer zeigen die Abbildungen B.11 und B.12 im Anhang auf Seite 330 f.

[37] siehe Abbildungen 5.22 und 5.23 auf Seite 186 bzw. 187
[38] siehe Abschnitt 5.2.2.1, S. 150
[39] siehe Abschnitt 5.2.2.2, S. 152
[40] siehe auch Abschnitt 6.3.1.2, S. 221

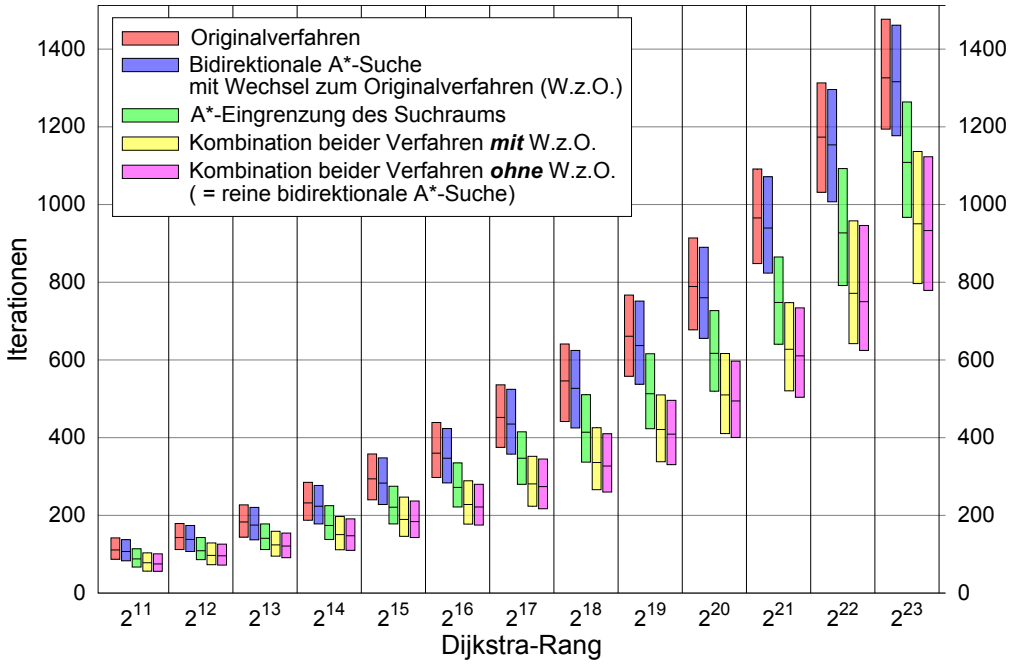


(a) Benötigte Iterationen

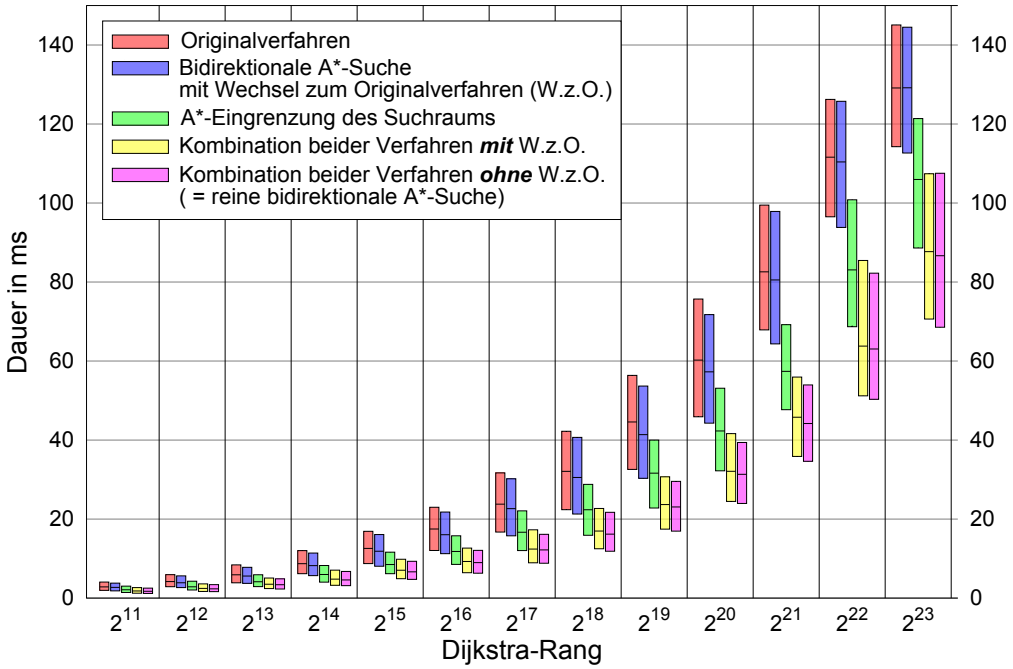


(b) Suchdauer

Abbildung 6.19: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Alle Verfahren wurden in der pfeilbasierten Variante durchgeführt.



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.20: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Alle Verfahren wurden in der pfeilbasierten Variante durchgeführt.

Interpretation der Messergebnisse Die Erweiterung der Wegsuche um eine Zielrichtungskomponente wirkt sich deutlich positiv auf die Anzahl der benötigten Iterationen als auch auf die Suchdauer aus. Die relativen Ersparnisse bei den Iterationen bewegen sich auf annähernd gleichem Niveau wie bei den Benchmarks zur Wegsuche ohne Abbiegebeschränkungen[41]. Zur Verdeutlichung stellt Abbildung 6.21 die Mediane der benötigten Iterationen pro Dijkstra-Rang bei Einsatz des Originalverfahrens den Medianen bei Einsatz der rein bidirektionalen A*-Wegsuche für die Suche nach schnellsten Wegen gegenüber.

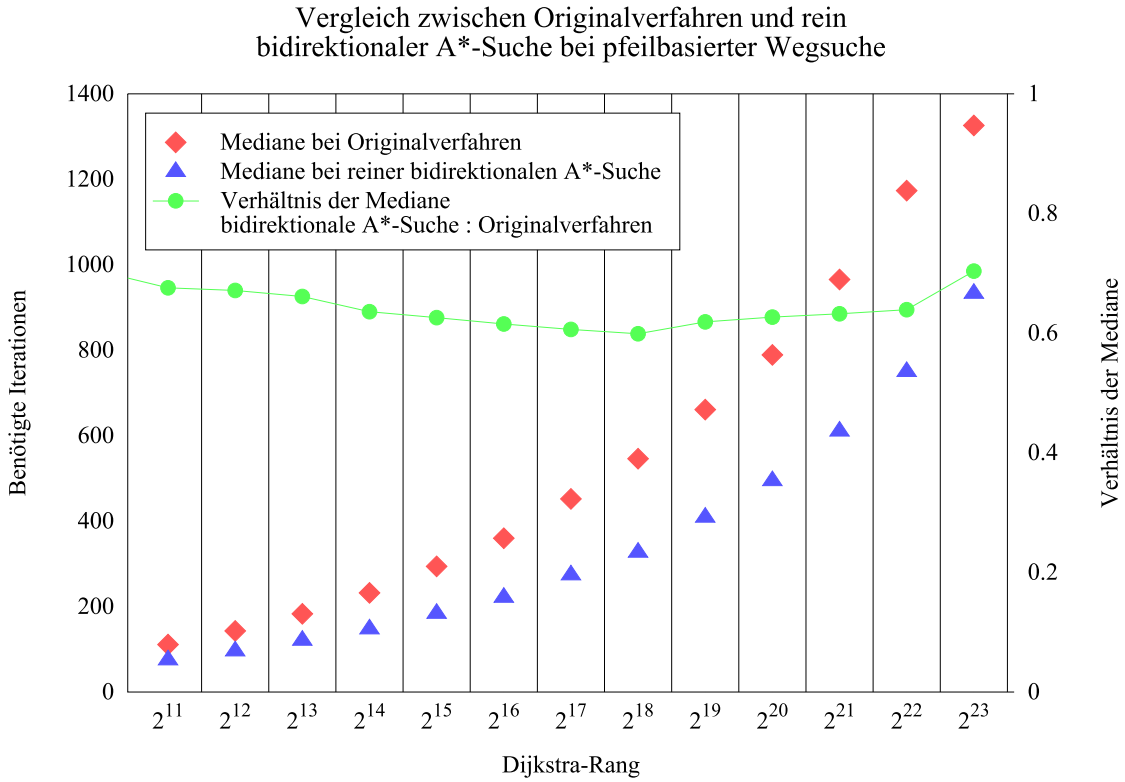


Abbildung 6.21: Relative Einsparungen bei zielgerichteter pfeilbasierter Suche nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Bezugsgröße für die Mediane ist die Anzahl der benötigten Iterationen.

Bezogen auf die Suchdauern lassen sich deutliche Verbesserungen beobachten. Dies liegt an der ohnehin längeren Dauer einer einzelnen Dijkstra-Iteration. War bei den Verfahren in Abschnitt 5.6.6 der Zusatzaufwand für die Restschätzungen für das A*-Verfahren bezogen auf die Dauer einer einzelnen Iteration verhältnismäßig groß, ist dies bei der pfeilbasierten Wegsuche nicht mehr der Fall.

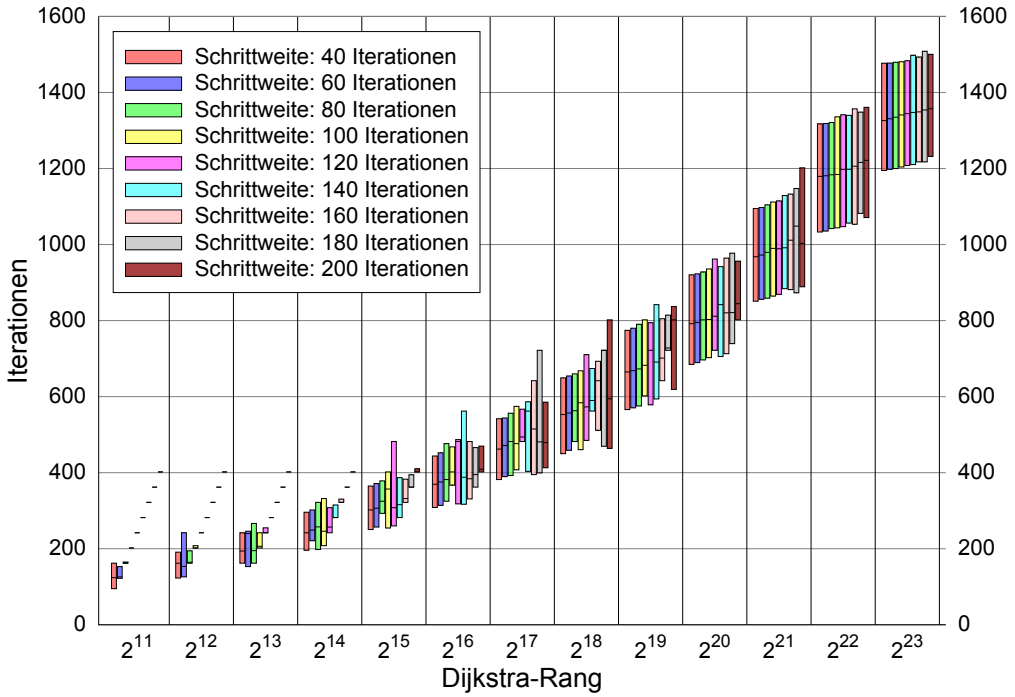
[41] siehe Abbildungen 5.24 und 5.25 auf Seite 189 bzw. 190

Entgegen den Messreihen zu Wegsuchen ohne Abbiegebeschränkungen existiert bei den Messungen zur pfeilbasierten Wegsuche ein Unterschied bei den notwendigen Iterationen zwischen der *reinen A*-Suche* und der Suchvariante, die zum *Standardverfahren mit A*-Eingrenzung des Suchraums* wechselt. Offenbar kommen in den Benchmarks diejenigen Fälle vor, bei denen die reine A*-Suche schneller weitere Brückenknoten identifiziert, wodurch im Folgenden tatsächlich noch Einträge der Horizontmengen verworfen werden können[42].

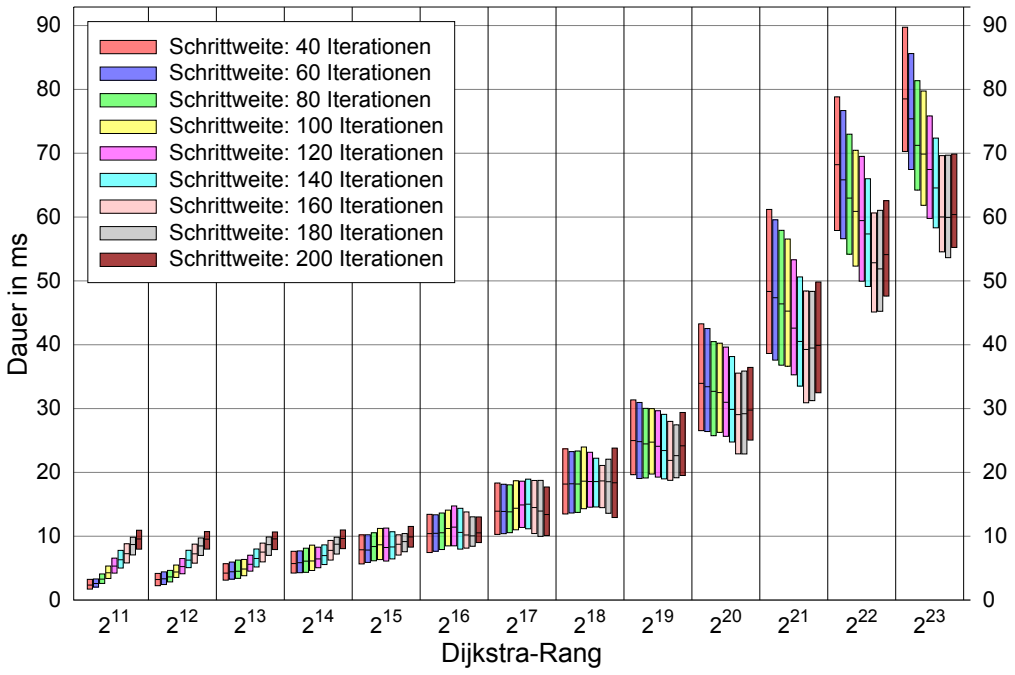
6.6.1.3 Benchmarks zur Schrittweite bei paralleler Wegsuche

Dieser Abschnitt liefert zur pfeilbasierten Wegsuche abschließend noch eine kurze Analyse der möglichen Beschleunigung durch Parallelisierung. Zwecks einer besseren Vergleichbarkeit der Ergebnisse mit den Implementierungen zur abbiegebeschränkungsfreien Wegsuche liegt den nachfolgenden Benchmarks eine parallele nicht-zielgerichtete Implementierung mit *einfacher* Pfeilblockierung gemäß Abschnitt 6.3.1.2 zugrunde. Die Messergebnisse sind in Abbildung 6.22, S. 248, visualisiert.

[42] siehe auch Abschnitt 5.6.6 für eine ausführlichere Formulierung der theoretischen Einsparungen der reinen A*-Wegsuche



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.22: Benchmark für parallele pfeilbasierte Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen

Interpretation der Messergebnisse Im Vergleich mit der nicht-parallelen Implementierung[43] kann die Suchgeschwindigkeit nahezu verdoppelt werden. Dies liegt im Wesentlichen am vergleichsweise hohen Aufwand pro einzelner Iteration. Das Ausmaß der Beschleunigung ist dabei natürlich abhängig vom Dijkstra-Rang des gesuchten Weges sowie der Schrittweite. Mit zunehmendem Dijkstra-Rang gilt dabei allgemein: je größer die Schrittweite, desto lohnenswerter ist die parallele Suche. Dies war vorab zu erwarten.

Besonders markant erscheinen dagegen die Messungen der benötigten Iterationen. Hier kommt es bei den ersten in Abbildung 6.22 dargestellten Dijkstra-Rängen zu sehr stark gleichförmigen Ergebnissen. Die Mehrzahl der 1 000 Wegsuchen für den Benchmark benötigen bei Schrittweiten zwischen 80 und 200 Iterationen jeweils *dieselbe* Anzahl Iterationen. Bei genauerer Betrachtung der Ergebnisse fällt auf, dass für eine Schrittweite von n dieser Wert stets bei $2n + 2$ liegt. Der Grund hierfür liegt in der Implementierung der parallelen Suche: Bei den Wegsuchen mit diesen kleineren Dijkstra-Rängen werden die Brückenknoten, die zum schnellsten Weg gehören, bereits nach dem *ersten* Durchgang der parallelen Vorwärts- und Rückwärtssuche gefunden. Dies erklärt $2n$ Iterationen. Im Anschluss wird ein weiterer Durchgang von n unabhängigen, parallelen Iterationen der Vorwärts- und Rückwärtssuche begonnen. Doch dabei wird sofort festgestellt, dass beide Suchrichtungen nur noch Knoten in ihren Horizontmengen besitzen, zu bzw. von denen die Fahrzeit länger dauert als die Fahrtzeit des aktuell schnellsten bekannten Weges. Daher brechen beide Suchrichtungen bei der ersten Iteration des zweiten Durchgangs ab. Dies erklärt die noch fehlenden 2 Iterationen.

6.6.2 Benchmarks zur adaptiven Wegsuche

Dieser Abschnitt ist der Auswertung der Benchmarks für die adaptive Wegsuche gewidmet. Er folgt dem Schema aus Abschnitt 6.6.1, der entsprechende Ergebnisse für die *pfeilbasierte* Wegsuche präsentierte. Insbesondere werden wiederum drei Arten der Suchbeschleunigungen näher untersucht; namentlich sind dies:

- die Knotenblockierung,
- die zielgerichteten Suche sowie
- die Parallelisierung mit unterschiedlicher Schrittweite.

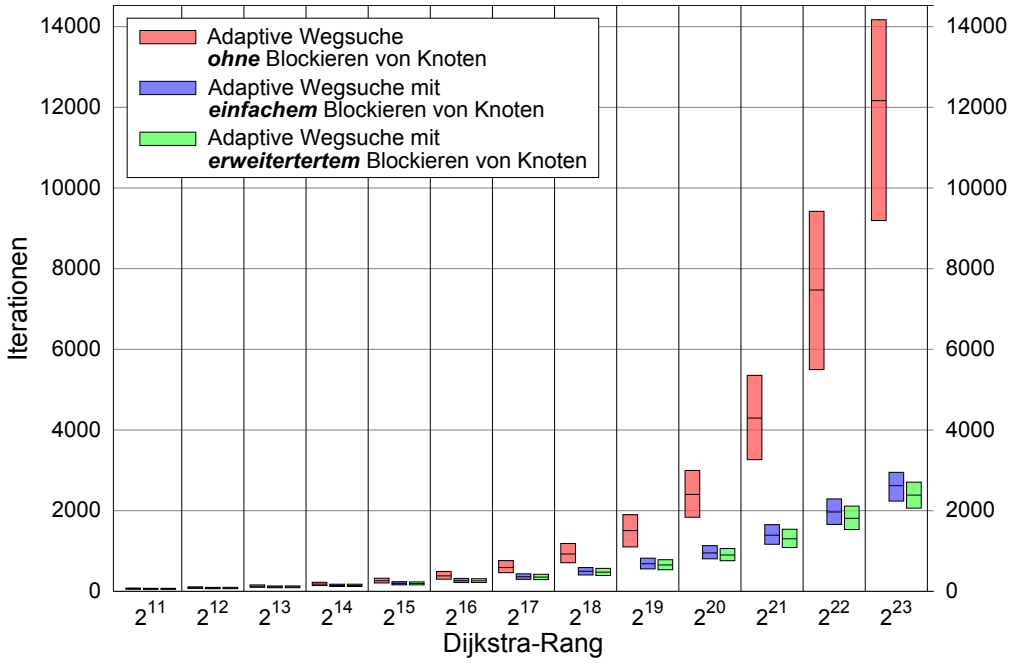
6.6.2.1 Benchmarks zur Blockierung von Knoten

An dieser Stelle werden die Ergebnisse der Benchmark-Durchläufe für die Verfahren aus Abschnitt 6.3.2.2, S. 227 ff., präsentiert. Wie auch in Abschnitten 5.6.5 und 6.6.1.1 wird zwischen drei Arten der Blockierung unterschieden:

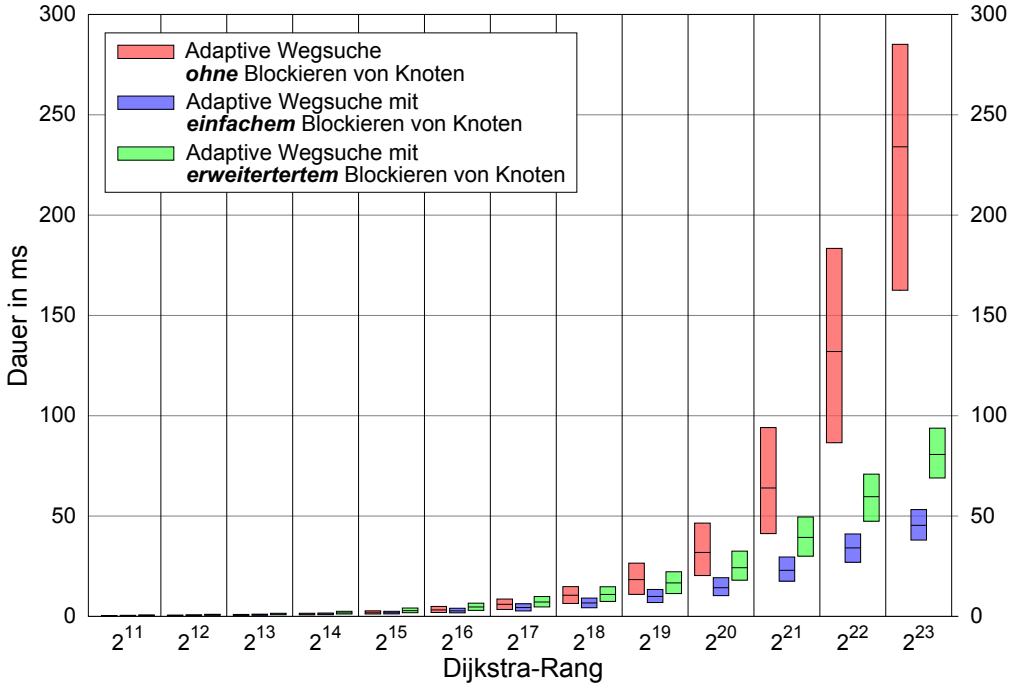
1. kein Blockieren von Knoten,
2. einfaches Blockieren von Knoten sowie
3. erweitertes Blockieren von Knoten.

Die Ergebnisse sind in Abbildung 6.23, S. 250, für die Suche nach kürzesten und Abbildung 6.24, S. 251, für die Suche nach schnellsten Wegen zusammengefasst.

[43] siehe Messergebnisse in Abbildung 6.20b, S. 245

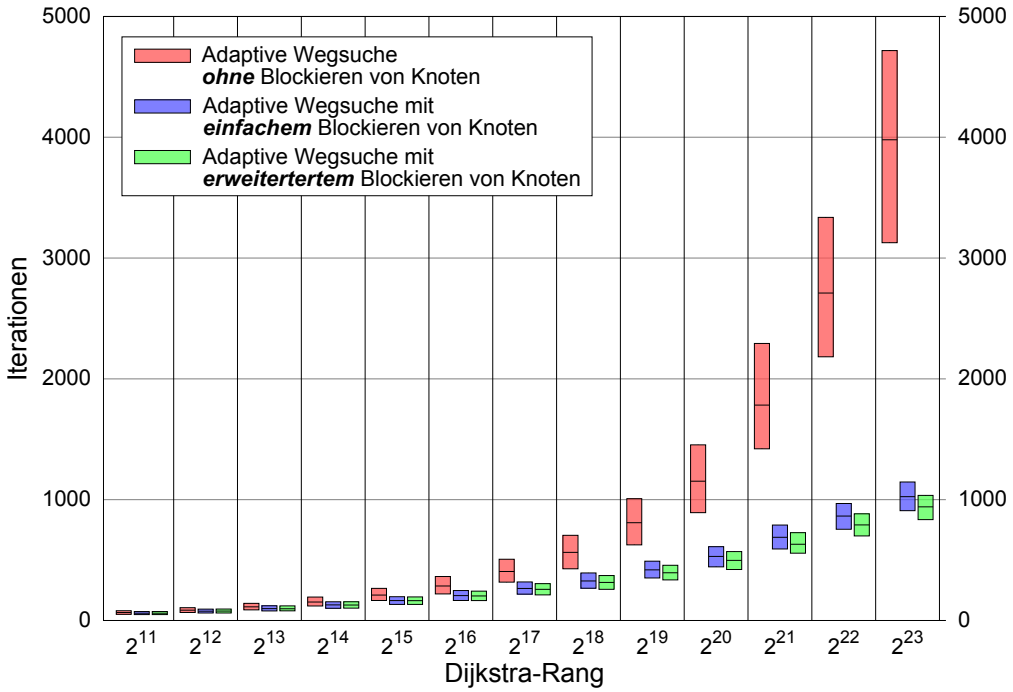


(a) Benötigte Iterationen

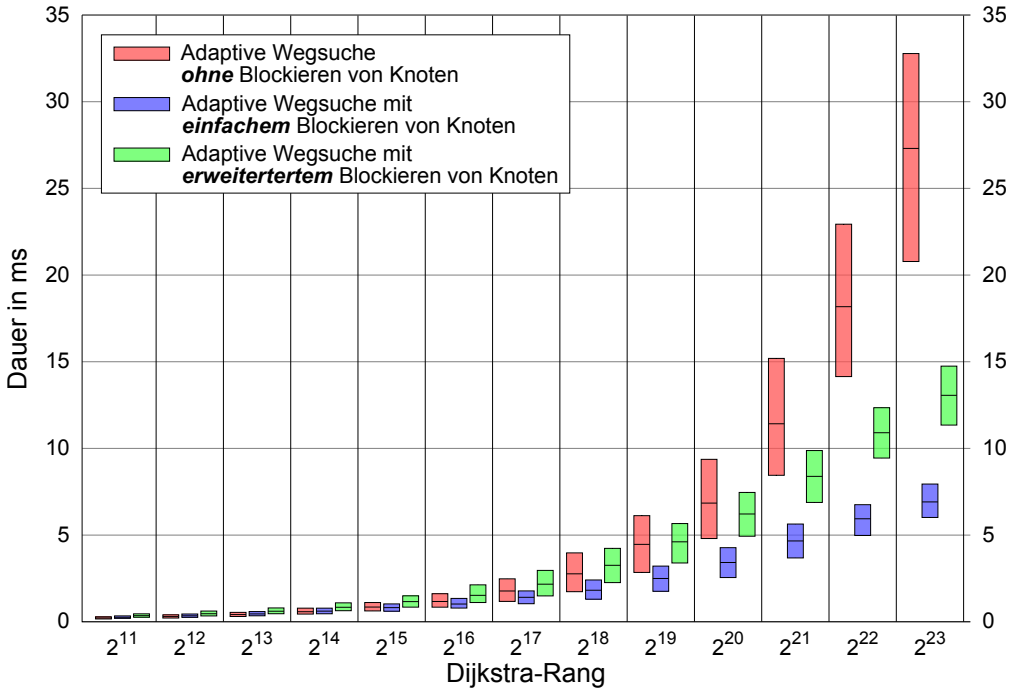


(b) Suchdauer

Abbildung 6.23: Benchmark für Knotenblockierung bei Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Zugrunde liegendes Verfahren ist die adaptive Wegsuche in Contraction Hierarchies.



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.24: Benchmark für Knotenblockierung bei Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Zugrunde liegendes Verfahren ist die adaptive Wegsuche in Contraction Hierarchies.

Interpretation der Messergebnisse Für die Interpretation ist der Vergleich der Ergebnisse mit den Benchmarkergebnissen zur *pfeilbasierten* Wegsuche aus Abschnitt 6.6.1.1 interessant. Die adaptive Wegsuche ist ihrem pfeilbasierten Pendant sowohl hinsichtlich der Anzahl der benötigten Iterationen als auch hinsichtlich der Suchdauer überlegen. Dies liegt an einer Mischung aus den vorteilhaften Eigenschaften der adaptiven Wegsuche[44] und der Häufigkeit der Abbiegebeschränkungen auf der den Benchmarks zugrunde liegenden Europa-Karte[45]. Der Unterschied ist bei der Suchdauer deutlich größer als bei der Anzahl der Dijkstra-Iterationen.

Bei genauerer Betrachtung zeigt sich allerdings, dass die relative Ersparnis der Iterationen mit zunehmendem Dijkstra-Rang abnimmt. Grund hierfür ist, dass bei der adaptiven Suche mit jeder Iteration die Wahrscheinlichkeit steigt, einen Pfeil über einen *verbotsgefährdeten* (Teil-)Weg zu erreichen[46].

Im Vergleich mit den Ergebnissen zur Wegsuche *ohne* Berücksichtigung von Abbiegebeschränkungen auf den Abbildungen 5.22, S. 186, und 5.23, S. 187, lässt sich als Unterschied kein konstanter Faktor erkennen. Bis zu einem Dijkstra-Rang von knapp 2^{20} liegen beide Implementierungen bei ähnlichen Werten für die benötigten Iterationen. Erst bei höheren Dijkstra-Rängen divergieren die Ergebnisse, da die adaptive Wegsuche mit immer größerer Wahrscheinlichkeit auf Abbiegebeschränkungen trifft, die sie zur Mehrfachaufnahme von Knoten und damit einhergehend zusätzlichen Iterationen zwingt.

Darüber hinaus verlangsamen die erforderlichen Prüfungen, die aufgrund der Berücksichtigung von Abbiegebeschränkungen entstehen, dagegen die adaptive Wegsuche im Vergleich zur beschränkungsfreien Suche deutlich. Das Ausmaß bewegt sich hier in Größenordnungen vom 5–8-fachen der Suchdauern. Diese Beobachtung gilt unabhängig vom Dijkstra-Rang.

[44] siehe hierzu insbesondere Abschnitt 6.2.4, S. 218
[45] für deren Details siehe Kapitel 3 und besonders Tabelle 3.6, S. 85
[46] für die genaue Begründung siehe Abschnitt 6.2.4, S. 211 ff.

6.6.2.2 Benchmarks zur zielgerichteten Wegsuche

In Abschnitt 6.3.2.3 wurde die Erweiterung der adaptiven Wegsuche um eine zielgerichtete Ausbreitung erläutert. Dieser Abschnitt präsentiert die zugehörigen Benchmark-Messungen und -Interpretationen. In Analogie zu den Abschnitten 5.6.6 und 6.6.1.2 werden wiederum vier Varianten der zielgerichteten Wegsuche evaluiert:

1. die A*-Eingrenzung des Suchraums[47],

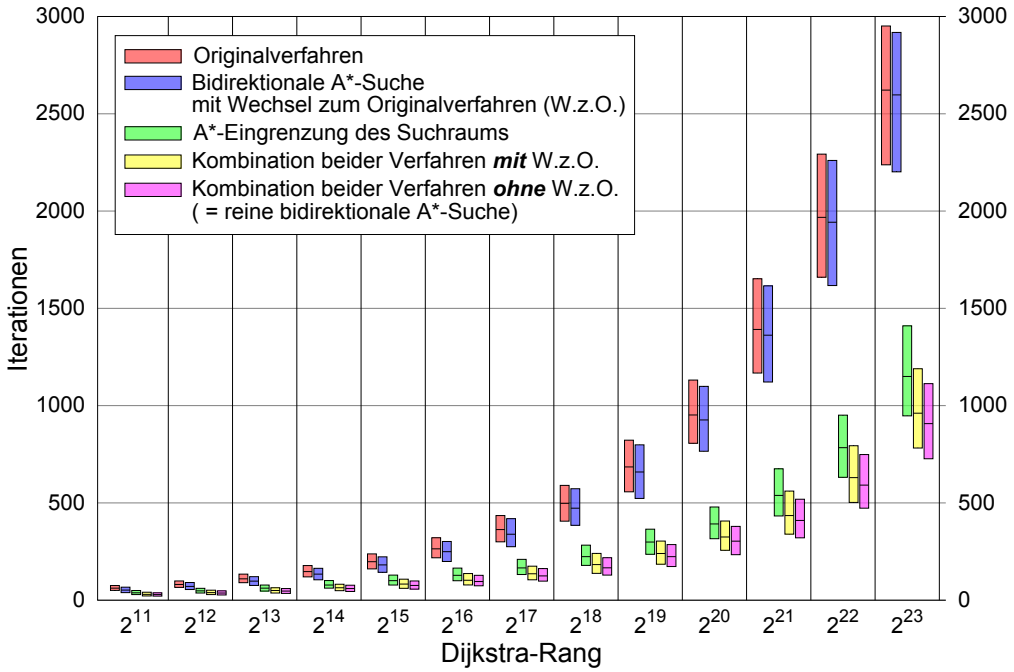
2. die bidirektionale A*-Suche mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens in Anlehnung an Nowak u. a. [2012][48],

3. die Kombination aus 1. und 2. mit Wechsel zum Originalverfahren beim Finden eines Brückenknotens in Anlehnung an Nowak u. a. [2012] und

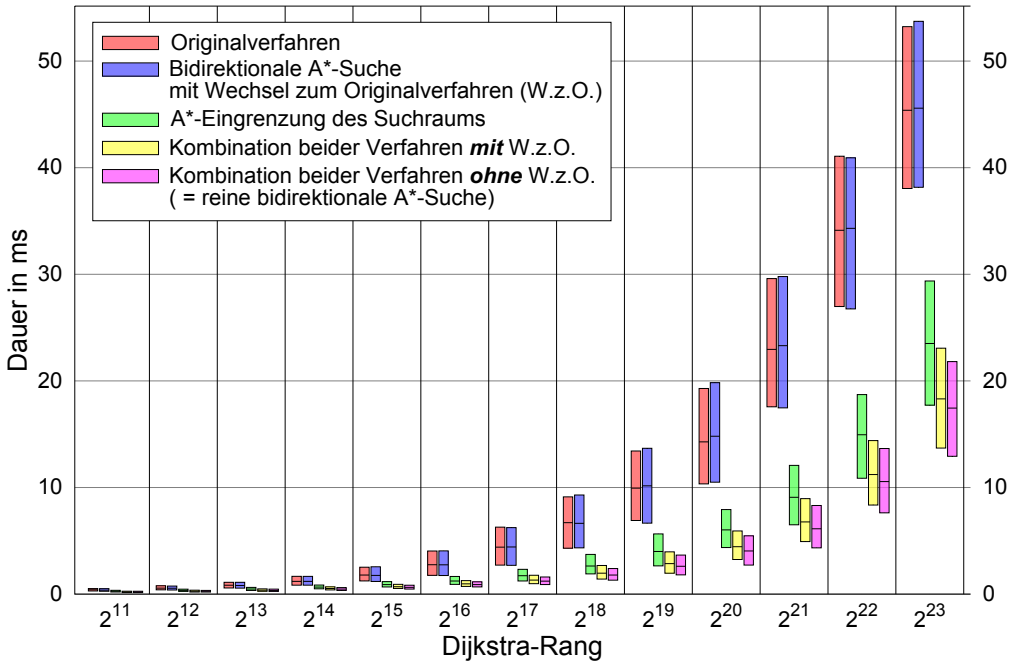
4. die Kombination aus 1. und 2. als reine bidirektionale A*-Suche.

Die gemessenen Implementierungen verwenden das *einfache Blockieren*[49] von Knoten, damit sich die Ergebnisse besser mit denen der pfeilbasierten und der abbiegebeschränkungsfreien Wegsuche vergleichen lassen. Abbildung 6.25, S. 254, und Abbildung 6.26, S. 255, zeigen die Messergebnisse der zielgerichteten Wegsuche als Boxplot-Diagramme für die Suche nach kürzesten bzw. schnellsten Wegen. Abbildungen B.16 und B.17 im Anhang auf den Seiten 335 und 336 zeigen die Diagramme inklusive Antennen und Ausreißer.

[47] siehe Abschnitt 5.2.2.1, S. 150
[48] siehe Abschnitt 5.2.2.2, S. 152
[49] siehe Abschnitt 6.3.2.2, S. 227

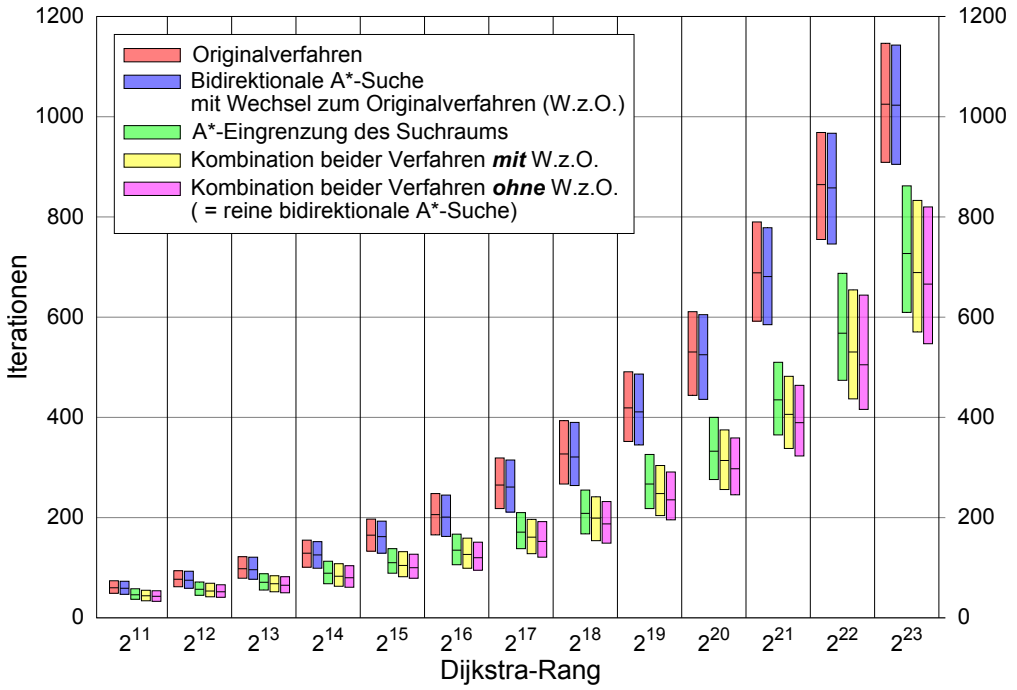


(a) Benötigte Iterationen

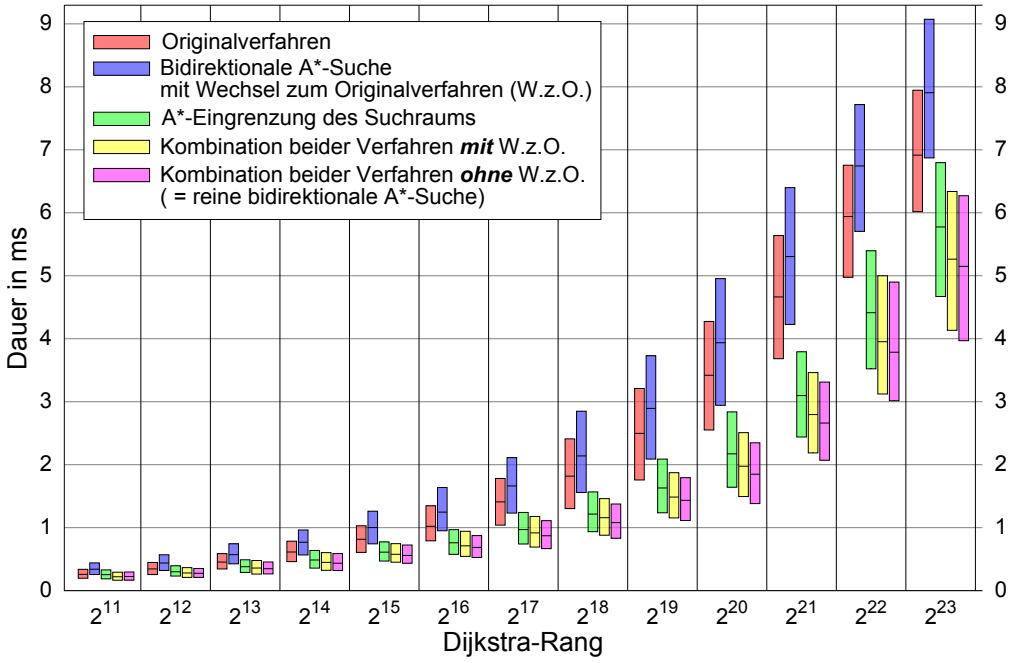


(b) Suchdauer

Abbildung 6.25: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Alle Verfahren wurden in der adaptiven Variante durchgeführt.



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.26: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Alle Verfahren wurden in der adaptiven Variante durchgeführt.

Interpretation der Messergebnisse Erneut zeigen sich die Vorteile einer Zielrichtungs-Komponente bei der Wegsuche empirisch. Größenordnungsmäßig lassen sich etwa 66 % bzw. 45 % der Dijkstra-Iterationen bei der Suche nach kürzesten bzw. schnellsten Wegen einsparen. In ähnlichen Dimensionen liegt die Zeitersparnis der Implementierungen.

Etwas überraschend ist dabei, dass bei der pfeilbasierten Wegsuche nach *kürzesten* Wegen zwar eine fast identische Verbesserung zu verzeichnen war. Doch bei den Ergebnissen zur Suche nach *schnellsten* Wegen ließ sich dort nur eine Verringerung der Iterationen von unter 40 % beobachten; die zielgerichtete adaptive Suche spart hier *über* 40 %. Zur Verdeutlichung dieses Umstands dient der Vergleich der Abbildung 6.21, S. 246, mit Abbildung 6.27.

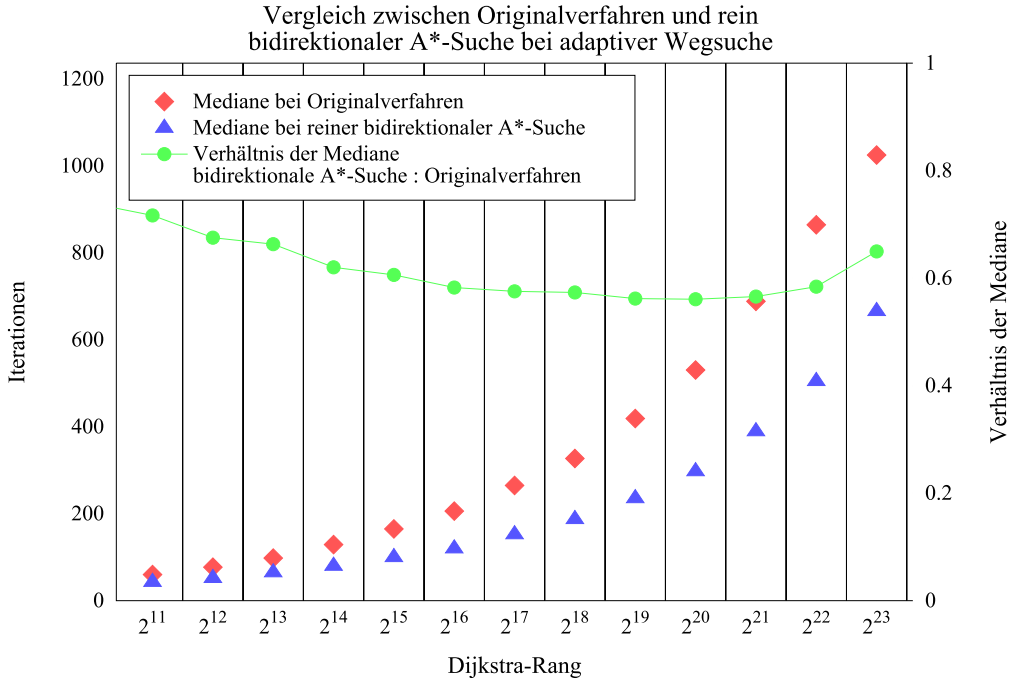


Abbildung 6.27: Relative Einsparungen bei zielgerichteter, adaptiver Suche nach schnellsten Wegen in CH mit Abbiegebeschränkungen
Bezugsgröße für die Mediane ist die Anzahl der benötigten Iterationen.

Der Unterschied ergibt sich offenbar aus den zugrunde liegenden Kartendaten: Schnellste Wege verlaufen häufiger auf Autobahnen oder ähnlichen Straßenkategorien. An diesen Straßenkategorien befinden sich jedoch verhältnismäßig weniger Abbiegebeschränkungen als an anderen Kategorien. Die adaptive Suche nimmt Knoten dieser Kategorien daher mit höherer Wahrscheinlichkeit über *verbotsfreie* Wege in die Horizont- und später in die Lösungsmenge auf. Dadurch werden diese Knoten im Anschluss nicht noch einmal der Horizontmenge hinzugefügt, selbst wenn die Suche sie erneut über einen anderen Pfeil erreicht. So kommt die adaptive Suche mit noch weniger Iterationen aus als die pfeilbasierte, die einen Knoten stets so oft erreichen und expandieren kann, wie dieser mit Pfeilen in Fahrtrichtung verbunden ist. In Abbildung 6.28, S. 257, ist dies beim direkten Vergleich der pfeilbasierten und der adaptiven Suche nach schnellsten Wegen, jeweils in der in der rein bidirektionalen A*-Variante, deutlich zu erkennen.

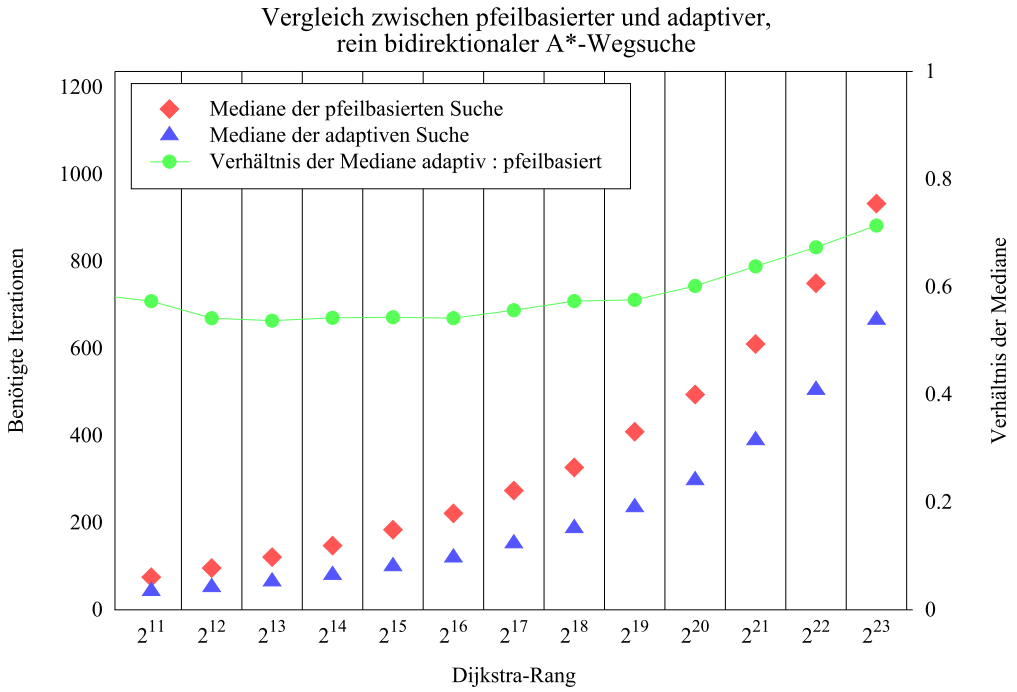


Abbildung 6.28: Direkter Vergleich der zielgerichteten pfeilbasierten mit der zielgerichteten adaptiven Wegsuche für schnellste Wege in Contraction Hierarchies mit Abbiegebeschränkungen
Bezugsgröße für die Mediane ist die Anzahl der benötigten Iterationen.

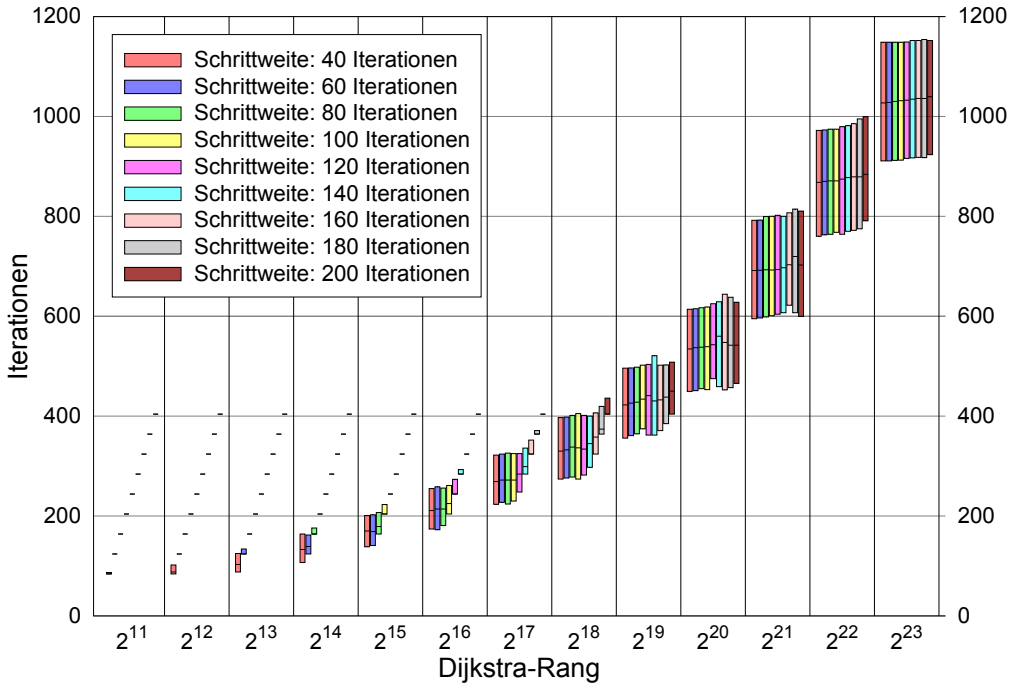
Beschleunigung

Im Vergleich mit dem Standard-Dijkstra-Verfahren zeigt sich dasselbe Prinzip wie auch bei der Wegsuche ohne Abbiegebeschränkungen[50]: Die Suchbeschleunigung mit dem CH-Verfahren im Vergleich zur Standard-Dijkstra-Suche wächst mit steigendem Dijkstra-Rang zwischen Start- und Zielknoten. Bei der Suche nach kürzesten Wegen werden Beschleunigungen um mehr als den Faktor 8 700 erreicht; bei der Suche nach schnellsten Wegen ist es mehr als Faktor 11 800.[51]

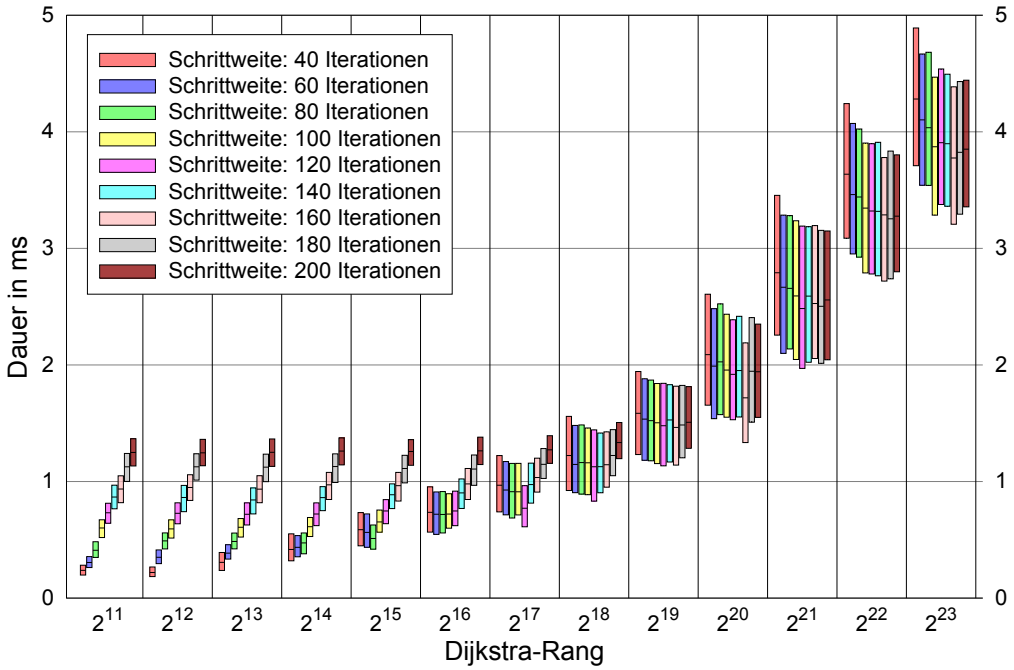
6.6.2.3 Benchmarks zur Schrittweite bei paralleler Wegsuche

Abbildung 6.29, S. 258, zeigt die Messergebnisse zu den parallelen Suchen nach schnellsten Wegen mit Schrittweiten zwischen 40 und 200 Iterationen, wenn die adaptive Wegsuche mit *einfacher* Knotenblockierung gemäß Abschnitt 6.3.2.2 zugrunde gelegt wird. Ergänzend sind in Abbildung 6.30, S. 259, die Werte zur *zielgerichteten*, parallelen, adaptiven Wegsuche mit ebenfalls einfacher Knotenblockierung nach Abschnitt 6.3.2.3 dargestellt. Letztere ist die schnellste, Abbiegebeschränkungen berücksichtigende Wegsuche, die in dieser Arbeit implementiert wurde.

[50] siehe Abschnitt 5.6.6, Seite 193, Stichwort: „Beschleunigung“
[51] siehe Abbildungen 6.25a und 6.26a jeweils bei Dijkstra-Rang 2^{23}

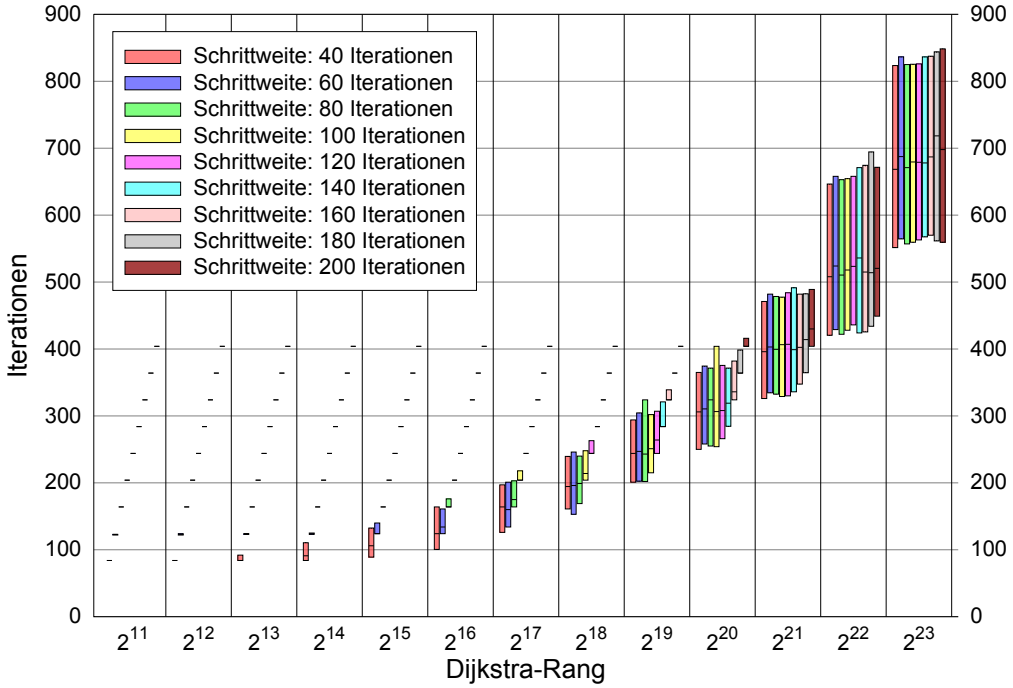


(a) Benötigte Iterationen

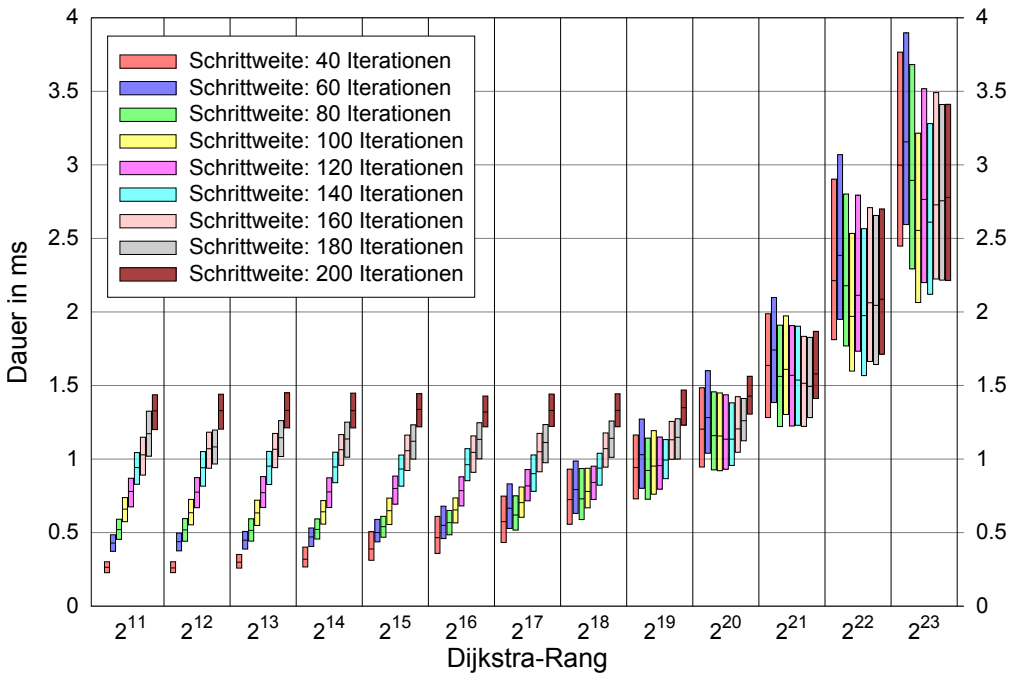


(b) Suchdauer

Abbildung 6.29: Benchmark für parallele, adaptive Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen (parallele Ausführung von Verfahren 22, S. 226)



(a) Benötigte Iterationen



(b) Suchdauer

Abbildung 6.30: Benchmark für parallele, adaptive A*-Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen (basierend auf Verfahren 22, S. 226)

Interpretation der Messergebnisse Die Ergebnisse sind im Wesentlichen mit den Ergebnissen zur parallelen, pfeilbasierten Wegsuche in Abschnitt 6.6.1.3 vergleichbar. Insbesondere gilt analog, dass

- bezüglich der Suchdauer große Einsparungen erzielt werden können; eine Verdopplung der Suchgeschwindigkeit im Vergleich zur Standardsuche bzw. zielgerichteten Wegsuche ist annähernd möglich,
- die Suchbeschleunigung stark abhängig ist vom Dijkstra-Rang der Wegsuche, so dass es auch zu einer Verlangsamung der Suche kommen kann, falls die Schrittweite zu groß gewählt wurde,
- bei Wegsuchen zu kleineren Dijkstra-Rängen zu große Schrittweiten n unnötige Iterationen nach sich ziehen, wodurch uniform $2n+2$ Iterationen benötigt werden.[52]

6.6.3 Zusammenfassung der Benchmarkergebnisse

Die wesentlichen Erkenntnisse, die durch die Benchmarks zur pfeilbasierten und zur adaptiven Wegsuche mit Abbiegebeschränkungen gewonnen werden konnten, sind:

- Durch die Blockierung von Pfeilen bzw. Knoten während der Wegsuche können viele Dijkstra-Iterationen eingespart werden. Sowohl in der pfeilbasierten als auch in der adaptiven Wegsuche ist diesbezüglich die *erweiterte* Blockierung am effektivsten. Demgegenüber steht aber die absolute Suchgeschwindigkeit. Weder die einfache noch die erweiterte Pfeilblockierung arbeiteten schneller als die untersuchte Implementierung der pfeilbasierten Suche *ohne* Blockierung. Bei der adaptiven Wegsuche konnten die schnellsten Ergebnisse dagegen mit *einfachem* Blockieren von Knoten erreicht werden. Das Ergebnis fällt hier analog zu den Messungen der Wegsuchen in Contraction Hierarchies *ohne* Abbiegebeschränkungen in Abschnitt 5.6.5, S. 185, aus.
- Die Zielrichtung kann sowohl bei der pfeilbasierten als auch bei der adaptiven Wegsuche überzeugen. Bei der Suche nach schnellsten Wegen mit der pfeilbasierten Wegsuche kann etwa ein Drittel der Iterationen eingespart werden (siehe Abbildung 6.21, S. 246). Bei der adaptiven Wegsuche sind es sogar etwa 40 % (siehe Abbildung 6.27, S. 256).
- Durch Parallelisierung der Vorwärts- und Rückwärtssuche lässt sich die Suchgeschwindigkeit nahezu verdoppeln. Die Erkenntnis gilt sowohl für die pfeilbasierte als auch für die adaptive Wegsuche gleichermaßen. Wie auch bei der Wegsuche ohne Abbiegebeschränkungen variiert das Ergebnis allerdings mit der verwendeten Schrittweite. Bei zu großen Schrittweiten sind daher auch langsamere Suchen möglich.
- Im direkten Vergleich der pfeilbasierten und der adaptiven Wegsuche ist letztere klar vorzuziehen. Die adaptive Wegsuche benötigt ihrerseits gut 40 % weniger Iterationen als die pfeilbasierte Wegsuche (siehe Abbildung 6.28, S. 257).

[52] Zur näheren Erläuterung siehe Abschnitt 6.6.1.3 auf Seite 249.

7 Entscheidungsunterstützung für das Konsolidierungsproblem

Inhalt

Dieses Kapitel beschäftigt sich mit einer besonderen Problemstellung bei Transportdienstleistern wie der in Abschnitt 2.1, S. 35 ff., vorgestellten IN tIME Express Logistik GmbH: dem *Konsolidierungsproblem*. Vereinfacht ausgedrückt geht es um die Frage, welche Kundenaufträge gemeinsam vom selben Fahrzeug parallel ausgeführt werden sollten. Durch eine solche Konsolidierung der Frachten wird versucht, die Ladekapazität der Fahrzeuge des Transportunternehmens besser auszulasten. Gleichzeitig erhöht eine gelungene Frachtkonsolidierung auch die Wirtschaftlichkeit des Unternehmens. Aufbauend

auf den Erkenntnissen der schnellen Wegsucheverfahren aus den beiden vorangegangenen Kapiteln wird ein Szenario für ein Entscheidungsunterstützungssystem entwickelt, das bei der operativen Tourenplanung hilft, Konsolidierungspotential aufzudecken.

Das Kapitel gliedert sich wie folgt: Im nachfolgenden Abschnitt wird eine formale Beschreibung des Tourenplanungsproblems entwickelt, wie es bei IN tIME anzutreffen ist. Ausgehend von einer vereinfachten Formulierung werden in den Abschnitten 7.1.2 und 7.1.3 zwei wesentliche Ergänzungen – namentlich *Umlademöglichkeiten* und *Teilladungen* – integriert.

Abschnitt 7.2 begründet schließlich die Ableitung des Konsolidierungsproblems als Entscheidungsunterstützungsproblem. Nach einer Diskussion der konkreten Modellausgestaltung und Implementierungsdetails in Abschnitt 7.3 wird der Praxisnutzen, der sich aus der Bearbeitung des Konsolidierungsproblems ergibt, anhand einer ausführlichen empirischen Untersuchung in Abschnitt 7.4 erörtert. In diese Untersuchung fließen alle in den vorangegangenen Kapiteln dieser Arbeit gewonnenen Erkenntnisse über Wegsucheverfahren auf digitalen Straßenkarten mit ein.

7.1 Allgemeine Problembeschreibung und -definition

Problemstellungen aus dem Gebiet der Tourenplanung sind in der Literatur weit verbreitet und genießen bis heute rege Aufmerksamkeit in der Forschung. Der nachfolgende Abschnitt liefert hierzu eine knappe Übersicht der weitesten verbreiteten Ausprägungen der Problemstellung. Eine abschließende Auflistung kann jedoch im Rahmen dieser Arbeit nicht stattfinden. Dafür wird auf weiterführende Literatur verwiesen[1].

Dieser Abschnitt liefert eine detaillierte Beschreibung der Problemstellung bei der IN tIME Express Logistik GmbH, die mithilfe geeigneter Verfahren im weiteren Verlauf des Kapitels bearbeitet wird. Basierend auf einer einfachen Betrachtungsweise in Abschnitt 7.1.1 wird in den Abschnitten 7.1.2 und 7.1.3 dazu ein ausführliches mathematisches Modell entwickelt.

7.1.1 Einfache Modellierung des Tourenplanungsproblems bei der IN tIME Express Logistik GmbH

Für den Einstieg wird zunächst nur eine vereinfachte Modellierung des Tourenplanungsproblems bei IN tIME vorgestellt. Hierfür werden aus der Literatur bekannte Problemformulierungen zugrunde gelegt und erweitert.

Der nachfolgende Abschnitt 7.1.1.1 gibt einen kurzen Überblick über verschiedene Varianten der Tourenplanung. Die Beschreibungen verzichten zugunsten des leichteren Zugangs auf streng formalisierte Definitionen. Eine präzise mathematische Modellierung liefert Abschnitt 7.1.1.2.

[1] beispielsweise Domschke [1997], Parragh u. a. [2008], Savelsbergh u. Sol [1995] oder Vahrenkamp [2003], die jeweils weiterführende Quellen nennen

7.1.1.1 Natürlich-sprachliche Formulierung der Problemstellung

Die Grundlage bildet das so genannte *Standardproblem der Tourenplanung*. In dessen Formulierung werden neben wichtigen Begriffen wie *Route*, *Tour* und *Tourenplan* auch die Zielfunktion sowie übergreifende Nebenbedingungen eingeführt.

Das Standardproblem der Tourenplanung[2]
Gegeben ist für dieses deterministische Problem eine Menge von Kunden mit spezifischen Güter-Bedarfen. Ferner steht eine homogene Fahrzeugflotte mit beliebig vielen Fahrzeugen mit einer festgelegten Ladekapazität an einem Depot zur Verfügung.[3] Ausgehend von diesem Depot sollen sämtliche Kunden beliefert werden.[4] Ebenfalls bekannt sind sämtliche symmetrischen Entfernungen zwischen den Kunden untereinander sowie zwischen dem Depot und den Kunden. Ziel ist die vollständige Bedienung der Kunden durch Fahrzeuge, wobei die insgesamt zurückgelegte Wegstrecke minimiert wird. Die Lösung muss drei Nebenbedingen berücksichtigen:

1. Jeder Kunde wird durch genau ein Fahrzeug bedient.
2. Die Fahrzeugkapazität wird von jedem Fahrzeug eingehalten.
3. Alle Fahrzeuge kehren wieder zum Depot zurück.

VRP

Sowohl in der englischsprachigen als auch in der deutschen Literatur hat sich für das Standardproblem der Tourenplanung der Begriff des *Vehicle Routing Problem* (VRP) etabliert (vgl. u.a. [Solomon, 1987], [Savelsbergh u. Sol, 1995], [Vahrenkamp, 2003] oder [Domschke, 2007]). Häufig ist es das Ziel, anstelle der Wegstrecke, die Gesamt-Fahrtdauer oder die Fahrtkosten zu minimieren. Vereinfachend werden diese oft als proportional zu der zurückgelegten Wegstrecke angesehen. Somit kann ein abstrakter Kostenbegriff verwendet werden, der sich in einem zu minimierenden Zielfunktionswert ausdrückt.[5]

Tour, Route, Tourenplan

Eine Lösung dieser Problemstellung umfasst sowohl die Zuordnung von Kunden zu Fahrzeugen als auch die Reihenfolge der Bedienung der Kunden. Im Kontext des VRP wird die Menge aller Kunden, die vom selben Fahrzeug auf einer beim Depot beginnenden und endenden Fahrt bedient werden, *Tour* genannt. Mit dem Begriff *Route* ist die Reihenfolge der Kunden-Bedienung innerhalb einer Tour gemeint. Dabei werden in der Regel zwischen je zwei Kunden optimale Wege verwendet.[6] Ein *Tourenplan* ist dann eine Menge von Touren und deren Routen, welche die obige Problemstellung unter Berücksichtigung aller drei oben genannten Nebenbedingungen löst. (vgl. [Domschke, 1997, S. 206])

[2] vgl. [Vahrenkamp, 2003, S. 240 f.]
[3] Für die mathematische Formulierung ist bezüglich der Art der Güter und der Fahrzeugkapazität wichtig, dass beide mit derselben Einheit ausgedrückt werden können, z. B. in Tonnen oder Liter.
[4] Das Problem lässt sich alternativ auch so formulieren, dass bei den Kunden jeweils eine bestimmte Menge des Guts *abgeholt* und zum Depot transportiert werden muss. Die Perspektive ist für die Lösung des Problems unerheblich.
[5] Ein ähnlich weit gefasster Kostenbegriff wird in Kapitel 4 auf Seite 30 verwendet.
[6] Aufgrund der umgangssprachlichen Nähe der Begriffe *Route* und *Weg* sei hier auf den semantischen Unterschied in diesem Kapitel explizit hingewiesen.

Für das Standardproblem der Tourenplanung existieren viele Erweiterungen, die zusätzliche, praxisnahe Nebenbedingungen abbilden. Die wichtigsten von ihnen werden nachfolgend vorgestellt. Sie umfassen

- den Einbezug von Zeitfenstern,
- Aufträge, zu denen eine Abholung und eine Auslieferung gehören, und
- Fahrzeugflotten mit unterschiedlichen Fahrzeugen

TW

Zu den häufigsten Erweiterungen des Standardproblems der Tourenplanung gehört der Einbezug von Zeitfenstern. Sie stellen zusätzliche Nebenbedingungen dar und begründen sich beispielsweise durch Öffnungszeiten bei den Sendern oder Empfängern der Güter. In der Literatur werden Problemstellungen mit Zeitfenstern meist durch den Zusatz der beiden Buchstaben „TW“ (für „time windows“) angedeutet, so dass beispielsweise VRPTW das *Vehicle Routing Problem with Time Windows* meint (vgl. [Parragh u. a., 2008, S. 85]).

In der Modellierung wird zwischen *harten* und *weichen* Zeitfenstern unterschieden. Erstere stellen eine Nebenbedingung dar, die nicht verletzt werden darf. Bei letzteren sind dagegen Verletzung der Zeitfenster erlaubt, gehen in diesem Fall aber mit Strafkosten in die Zielfunktion ein (vgl. [Vahrenkamp, 2003, S. 259]). Sind bei weichen Zeitfenstern die Strafkosten hinreichend hoch, lassen sich somit auch de facto harte Zeitfenster modellieren.

VRPPD

Beim VRP besteht jeder Auftrag lediglich aus einem (Kunden-)Ort, an dem ein bestimmter Güterbedarf zu decken ist. Eine weiter gefasste Problemstellung entsteht, wenn Aufträge sowohl Abholungen als auch Auslieferungen umfassen dürfen. In der englischsprachigen Literatur wird dies als *Vehicle Routing Problem with Pickup and Delivery* (VRPPD) bezeichnet; ein deutschsprachiges Äquivalent wäre etwa *Tourenplanung mit Abholung und Auslieferung* (vgl. [Parragh u. a., 2008, S. 81]). Das VRPPD beinhaltet das VRP als Spezialfall, bei dem sämtliche Abholungsorte auf denselben Ort zusammenfallen, nämlich dem Depot.

PDVRP

Bisweilen wird unterschieden, ob mit den Gütern einer Abholung *jede* Auslieferung bedient werden kann, oder ob die Zuordnung eindeutig festgelegt ist. Parragh u. a. [2008, S. 82] beschreiben den ersten Fall mit *Pickup and Delivery Vehicle Routing Problem* (PDVRP), welches sich um ein Transportproblem mit nur einer einzigen Güterart handelt. Es wird häufig im Kontext der so genannten Retrologistik beim innerbetrieblichen Transport standardisierter Güter mit hohem Umschlag wie Container oder Europoolpaletten[7] angetroffen: Manche Standorte, d. h. Kunden, benötigen für den Versand von Waren zusätzliche Leer-Container, und andere Standorte haben von diesen einen Überhang, so dass bei ihnen Container abgeholt werden müssen.

PDP

Der zweite Fall trifft auf Aufträge mit inhomogenen Gütern zu, wie sie bei Transportunternehmen wie IN tIME vorkommen, und wird häufig als „reines“ *Pickup and Delivery Problem* (PDP) bezeichnet. Im Deutschen könnte dieser Fall *Tourenplanung mit fixer Abholung und Auslieferung* genannt werden.

[7] umgangssprachlich auch „Europaletten“ genannt

Eine Variante des PDP ist das *Dial A Ride Problem* (DARP), bei dem anstelle von Gütern Personen befördert werden. Häufig wird beim DARP durch zusätzliche Nebenbedingungen eine Art „Passagier-Komfort“ gefordert, der beispielsweise verhindern soll, dass einzelne Fahrgäste übermäßig große Umwege erdulden müssen. (vgl. [Parragh u. a., 2008, S. 82]) DARP

In der Realität sind oft Problemstellungen anzutreffen, bei denen für den Transport eine Flotte aus verschiedenartigen Fahrzeugen zur Verfügung steht. Die Kennzeichnung der Fahrzeuge innerhalb der mathematischen Modellierung erfolgt hier anhand einer fahrzeugbezogenen Indexierung. Gelegentlich wird in der Literatur bei der Problembeschreibung „MF“ für *mixed fleet* als Präfix verwendet (vgl. bspw. [Wassan u. Osman, 2002]). MF

Eine Besonderheit bei IN tIME ist die Vergütung der Subunternehmer. Denn diese erhalten einen unterschiedlichen Tarif pro gefahrene Kilometer in Abhängigkeit davon, ob es sich um eine Leerfahrt, also eine Fahrt ohne Fracht für einen Auftrag an Bord, oder um eine Lastfahrt handelt. Da der Kostensatz für Leerfahrten geringer ist, lohnt es sich für Subunternehmer häufig, nach Beendigung eines Auftrags auf einen Folgeauftrag in der Nähe des aktuellen Standorts zu warten, anstatt mit leerem Fahrzeug die Heimreise anzutreten.

7.1.1.2 Mathematische Formulierung der Problemstellung

Im Folgenden wird eine allgemeine mathematische Problemformulierung gegeben, die mit Ausnahme der weichen Zeitfenster alle oben beschriebenen Sonderfälle beinhaltet. Hinsichtlich der Struktur handelt es sich um eine Anlehnung[8] an und Erweiterung[9] der Darstellungen bei Savelsbergh u. Sol [1995] und Parragh u. a. [2008]. Die Formulierung verwendet harte Zeitfenster, um in jedem Fall eine pünktliche Bedienung der Kunden des Transportunternehmens sicherzustellen.

Anmerkung: Die Formulierung stellt natürlich nur eine von vielen denkbaren Schreibweisen dar. Sie dient der Definition des Problems, zielt dabei jedoch nicht auf eine besonders effiziente Lösung ab. Wenn es der Lesbarkeit dient, enthält sie mitunter auch Redundanzen wie beispielsweise überflüssige Pfeile in der Graph-Repräsentation des Problems.

Gegeben ist eine Auftragsmenge A und die zur Verfügung stehende Fahrzeugflotte F. Zu jedem Auftrag $a \in A$ gehört eine Frachtmenge q_a, die von einem Beladungsort o_a^+ zu einem Entladungsort o_a^- transportiert werden soll. Jedes Fahrzeug $k \in F$ besitzt eine spezifische Kapazität C^k.

[8] Im Gegensatz zu Savelsbergh u. Sol [1995] wird hier nicht erlaubt, dass zu einem Auftrag mehrere Be- und/oder Entladungsorte gehören können, da dies in den hier untersuchten Vergangenheitsdaten von IN tIME nur in den seltensten Fällen vorkommt.

[9] Weder bei Savelsbergh u. Sol [1995] noch bei Parragh u. a. [2008] werden Umladungen, Teilladungen oder auftragsbezogene Dauern von Umladungen berücksichtigt.

Die Lösung wird mithilfe eines speziell konstruierten, gerichteten Graphen $G = (V, E)$[10] angegeben, dessen Aufbau im Folgenden erläutert wird. Die Knoten V des Graphen setzen sich zusammen aus:

- den individuellen aktuellen Positionen k^+ der Fahrzeuge $\forall k \in F$,
- den individuellen Zielorten k^- der Fahrzeuge $\forall k \in F$,
- allen Beladungsorten o_a^+ der Aufträge $\forall a \in A$ sowie
- allen Entladungsorten o_a^- der Aufträge $\forall a \in A$.

Zur vereinfachten Referenzierung seien

- $K^+ = \{k^+ \mid k \in F\}$ die Menge aller aktuellen Fahrzeugpositionen,
- $K^- = \{k^- \mid k \in F\}$ die Menge aller Zielorte der Fahrzeuge,
- $K = K^+ \cup K^-$ die Menge aller fahrzeugbezogenen Orte,
- $O^+ = \{o_a^+ \mid a \in A\}$ die Menge aller Beladungsorte,
- $O^- = \{o_a^- \mid a \in A\}$ die Menge aller Entladungsorte mit $O^+ \cap O^- = \emptyset$ und
- $O = O^+ \cup O^-$ die Menge aller auftragsbezogenen Orte.

Die Pfeile E des Graphen bestehen aus

- Pfeilen von k^+ zu k^- mit $k \in F$,
- Pfeilen von allen Elementen aus K^+ zu allen Elementen aus O^+,
- Pfeilen zwischen je zwei Elementen aus O und
- Pfeilen von allen Elementen aus O^- zu allen Elementen aus K^-.

Anschaulich bedeutet dies, dass der Graph einen Kern besitzt, der aus der paarweise verknüpften Menge aller auftragsbezogenen Orte besteht. Zu diesem Kern führen Pfeile von den aktuellen Fahrzeugpositionen und aus ihm heraus führen Pfeile zu den Zielorten der Fahrzeuge. Außerdem existieren Pfeile, welche die aktuellen Fahrzeugpositionen direkt mit den Zielorten der Fahrzeuge verbinden, ohne dass ein Knoten aus dem Kern des Graphen dabei erreicht wird. Zusammengefasst ergibt sich damit der Graph $G = (V, E)$ mit $V = K \cup O$ und $E = \{(k^+, k^-) \mid k \in F\} \cup (K^+ \times O^+) \cup (O \times O) \cup (O^- \times K^-)$. Abbildung 7.1, S. 267, zeigt schematisch einen Beispielgraphen für die Fahrzeugflotte $F = \{k_1, k_2\}$ sowie die Auftragsmenge $A = \{1, 2\}$. Die Verbindungen zwischen den Orten aus O sind dort der Übersicht halber als Linien dargestellt und symbolisieren jeweils zwei entgegengesetzte Pfeile.

[10] zur Definition von Graphen: siehe auch Abschnitt 1.3 auf Seite 29

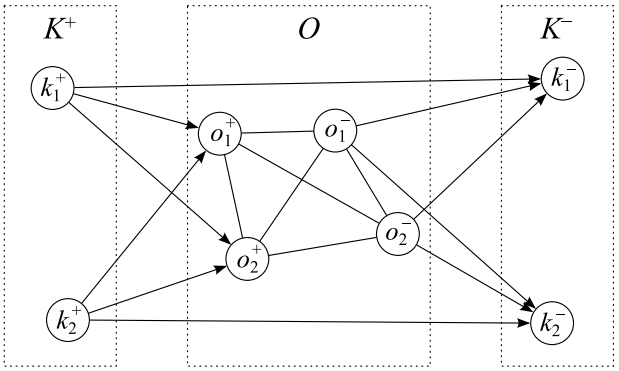


Abbildung 7.1: Schematischer Graph für das PDP bei IN tIME
Dargestellt ist ein Szenario mit zwei Fahrzeugen und zwei Aufträgen. Linien ohne Pfeilspitzen sind als zwei entgegengesetzte Pfeile zu interpretieren.

Im Beispielgraphen sind die Mengen der fahrzeug- und der auftragsbezogenen Orte disjunkt ($K \cap O = \emptyset$). Dies ist jedoch keine zwingende Anforderung an die Problemstellung. Es kann durchaus vorkommen, dass sich ein Fahrzeug z. B. aktuell an einem Kundenort befindet, von dem aus ein neuer Auftrag abgeholt werden soll. Im schematischen Graph existiert dann beim betreffenden Knoten eine Schlinge mit Null-Bewertung.

Die Beladungsorte O^+ und die Entladungsorte O^- sind dagegen stets disjunkt. Wenn bei einem Kunden sowohl Güter abzuholen als auch anzuliefern sind, existieren im zugehörigen Graphen zwei unterschiedliche Knoten für Be- und Entladung am selben Punkt. Dieselbe Aussage gilt für Kunden, zu denen mehrere Aufträge gehören: Auch für solche Kunden existieren mehrere Knoten am selben Punkt.

Inhaltlich entsprechen die Pfeile aus G den schnellsten Wegen zwischen zwei Orten. Es sei daran erinnert, dass dieser Graph mitnichten eine digitale Straßenkarte repräsentiert. Die genannten Wege sind *auf der Straße* zwar fahrzeugabhängig und für ihre konkrete Ermittlung eignen sich daher die Verfahren aus den Kapiteln 4, 5 und 6. Doch für die Formulierung einer Lösung des Optimierungsproblems in diesem Kapitel wird durch die Nutzung eines Pfeils aus G die *Route* und nicht der *Weg* definiert[11]. Unterschiedliche Wege zwischen denselben Orten wirken sich in G durch unterschiedliche zeitliche und monetäre Bewertungen aus. Beide Aspekte werden nachfolgend eingeführt.

Zeitliche Aspekte

Für die zeitliche Komponente der Problemstellung sind Zeitfenster, die Dauern für Be- und Entladungen sowie die Fahrtzeiten zu berücksichtigen. Für die Beladungen sind die Zeitfenster $[e_a^+; l_a^+]$ vorgegeben und für die Entladungen die Zeitfenster $[e_a^-; l_a^-]$. Die Beladung der zum Auftrag $a \in A$ gehörigen Fracht benötigt fahrzeugunabhängig t_a^+ Zeiteinheiten und die Entladung t_a^- Zeiteinheiten. Darüber hinaus sind für heterogene Fahrzeugflotten die Fahrzeiten fahrzeugabhängig: t_{uv}^k $(k \in F,\ (u,v) \in E)$ ist die Zeit, die Fahrzeug k für die Fahrt entlang des Pfeils (u,v) benötigt. t_a^+, t_a^- und t_{uv}^k sind nicht-negativ.

[11] zum Unterschied zwischen den Begriffen *Route* und *Weg*: siehe Seite 263

Kostenaspekte

Das Modell umfasst für die Fahrt eines Fahrzeugs $k \in F$ entlang eines Pfeils $(u, v) \in E$ fahrzeugabhängige Fahrtkosten c^k_{uv} bzw. $\hat{c}^k_{uv}$, die entstehen, wenn das entsprechende Fahrzeug entlang (u, v) fährt und dabei mindestens eine Sendung transportiert bzw. ohne Sendung fährt. Auf diese Weise lassen sich unterschiedlich hohe Kosten für Leerfahrten modellieren, wobei zumeist gilt: $c^k_{uv} > \hat{c}^k_{uv}$.

Die Entscheidungsvariablen in dieser Problemstellung umfassen

1. die Routen der Fahrzeuge,
2. die Transportrouten der zu den Aufträgen gehörenden Sendungen und
3. die Zeitpunkte des Beginns der Bedienung, d. h. der Be- oder Entladung, für alle Fahrzeuge bei den Kunden.

Fahrzeugrouten Die Darstellung der Fahrzeugrouten erfolgt anhand von Binärvariablen der Form x^k_{uv} ($(u, v) \in E, k \in F$), die genau dann den Wert 1 annehmen, wenn Fahrzeug k von Ort u nach Ort v fährt.

Sendungsbezogene Transportrouten Ebenfalls mit binären Variablen werden die Routen der Transporte definiert: y^{ka}_{uv} ($a \in A$,$k \in F$,$(u, v) \in E$) ist genau dann 1, wenn Fahrzeug k die zu Auftrag a gehörende Fracht über den Pfeil (u, v) transportiert.

Zeitpunkte der Kundenbedienungen Als dritte Entscheidungsvariable kennzeichnet B^k_u ($u \in V, k \in F$) den Zeitpunkt der Ankunft bei Ort u durch das Fahrzeug k. Beschreibt u einen Kunden-Ort, ist B^k_u damit der Beginn der Bedienung dieses Kunden durch k. Durch die B^k_u lassen sich auch Lösungen mit identischen Routen, aber unterschiedlicher zeitlicher Abfolge differenzieren.

Die Routen- und Transport-Variablen x^k_{uv} und y^{ka}_{uv} für alle $k \in F, (u, v) \in E$ und $a \in A$ können jeweils als Matrix X und Y geschrieben werden; für die Bedien-Zeitpunkte B^k_u genügt ein Vektor B. Die zu minimierenden Zielfunktion Z kumuliert alle Kosten für Leerfahrten und Lastfahrten:

$$Z(X, Y, B) = \sum_{(u,v)\in E} \sum_{k\in F} \max \left\{ x^k_{uv} \hat{c}^k_{uv},\ y^{ka}_{uv} c^k_{uv} \mid a \in A \right\}. \tag{7.1}$$

Nachfolgend sind die Nebenbedingungen aufgelistet. Ihre Bedeutungen werden direkt im Anschluss erläutert.

$$\sum_{\substack{k\in F,\\ (o^+_a,v)\in E}} y^{ka}_{o^+_a v} = \sum_{\substack{k\in F,\\ (v,o^-_a)\in E}} y^{ka}_{v o^-_a} = 1 \qquad \forall a \in A \tag{7.2a}$$

$$\sum_{(k^+,v)\in E} x^k_{k^+ v} = \sum_{(v,k^-)\in E} x^k_{v k^-} = 1 \qquad \forall k \in F \tag{7.2b}$$

$$\sum_{(u,v)\in E} x^k_{uv} - \sum_{(v,w)\in E} x^k_{vw} = 0 \qquad \forall v \in O,\ k \in F \tag{7.2c}$$

$$y_{uv}^{ka} = 1 \Rightarrow x_{uv}^{k} = 1 \qquad \forall (u,v) \in E,\ a \in A,\ k \in F \quad (7.2d)$$

$$y_{uv}^{ka} = 1 \Rightarrow \sum_{(v,w)\in E} y_{vw}^{ka} = 1 \quad \forall a \in A, k \in F, (u,v) \in E \,|\, v \neq o_a^- \quad (7.2e)$$

$$\sum_{a\in A,} y_{uv}^{ka} q_a \leq C^k \qquad \forall (u,v) \in E,\ k \in F \quad (7.2f)$$

$$B_{k^+}^{k} = 0 \qquad \forall k \in F \quad (7.2g)$$

$$x_{uv}^{k} = 1 \Rightarrow B_u^k + t_{uv}^k \leq B_v^k \qquad \forall (u,v) \in E,\ k \in F \quad (7.2h)$$

$$B_v^k + \max\left\{0, \sum_{(u,v)\in E} y_{uv}^{ka} - \sum_{(v,w)\in E} y_{vw}^{ka}\right\} t_a^-$$

$$+ \max\left\{0, \sum_{(v,w)\in E} y_{vw}^{ka} - \sum_{(u,v)\in E} y_{uv}^{ka}\right\} t_a^+ + t_{vw}^k \leq B_w^k \qquad \forall v \in O,\ a \in A,\ k \in F \quad (7.2i)$$

$$x_{o_a^+ v}^{k} = 1 \Rightarrow e_a^+ \leq B_{o_a^+}^k \leq l_a^+ \qquad \forall a \in A,\ (o_a^+, v) \in E,\ k \in F \quad (7.2j)$$

$$x_{o_a^- v}^{k} = 1 \Rightarrow e_a^- \leq B_{o_a^-}^k \leq l_a^- \qquad \forall a \in A,\ (o_a^-, v) \in E,\ k \in F \quad (7.2k)$$

$$x_{uv}^{k} \in \{0,1\} \qquad \forall (u,v) \in E,\ k \in F \quad (7.2l)$$

$$y_{uv}^{ka} \in \{0,1\} \quad \forall (u,v) \in E,\ a \in A,\ k \in F \quad (7.2m)$$

$$B_u^k \geq 0 \qquad \forall u \in V,\ k \in F \quad (7.2n)$$

Nebenbedingung (7.2a) stellt sicher, dass jeder Auftrag abgeholt und ausgeliefert wird. (7.2b) gewährleistet, dass die Routen aller Fahrzeuge von ihren aktuellen Positionen beginnen und an ihren Zielorten enden. (7.2c) erzeugt zusammenhängende Routen und (7.2d) und (7.2e) sorgen dafür, dass auch die Fracht der Aufträge nur über zusammenhängende Routen transportiert wird. Mithilfe von (7.2f) werden die Fahrzeugkapazitäten eingehalten. Die Nebenbedingungen (7.2g)–(7.2k) sorgen für den korrekten zeitlichen Ablauf und (7.2l)–(7.2n) definieren die Wertebereiche der Entscheidungsvariablen.

Zum Verständnis von Nebenbedingung (7.2i) ist zu berücksichtigen, dass jedes Element der auftragsbezogenen Orte O zu genau einem Auftrag gehört, d. h., an jedem Ort aus O findet entweder eine Beladung oder eine Entladung genau eines Auftrages statt. Sollten sowohl Beladung als auch Entladung zweier Aufträge am selben geographischen Ort stattfinden, enthält der zugehörige Graph zwei Knoten am selben Punkt. Die in (7.2i) eingehenden Maxima beschreiben die *Differenz der Fahrzeugbeladung* bezüglich des Auftrags a am Knoten $v \in O$. Nimmt das erste Maximum den Wert 1 an, findet eine Entladung des Auftrags a am Knoten v statt, nimmt das zweite Maximum den Wert 1 an, handelt es sich um eine Beladung. Aufgrund der Konstruktion des Graphen ist es ausgeschlossen, dass beide Maxima den Wert 1 annehmen, und für alle Aufträge, die nicht zu Knoten v gehören, nehmen beide Maxima den Wert null an.

Einige Nebenbedingungen müssen nicht explizit angegeben werden, da sie in einer die Zielfunktion minimierenden Lösung implizit erfüllt sind. Hierzu gehören unter anderem Forderungen wie die nachfolgenden:

$$\sum_{(u,k^+)\in E} x^k_{uk^+} = \sum_{(k^-,v)\in E} x^k_{k^-v} = 0 \qquad \forall k \in F \tag{7.3a}$$

$$x^k_{uv} = 0 \Rightarrow y^{ka}_{uv} = 0 \qquad \forall (u,v) \in E,\ a \in A,\ k \in F \tag{7.3b}$$

$$B^k_{o_a^+} \leq B^k_{o_a^-} \qquad \forall a \in A,\ k \in F \tag{7.3c}$$

Die unnötige Nebenbedingung (7.3a) formuliert, dass sämtliche aktuellen Fahrzeugstandorte nie zusätzlich erreicht werden und die Zielorte der Fahrzeuge nicht wieder verlassen werden. Da durch diese zusätzlichen Strecken aber ohnehin Mehrkosten entstünden, können sie nicht Teil einer Lösung sein, welche die Zielfunktion minimiert. Ebenfalls aufgrund der Zielfunktion redundant ist Nebenbedingung (7.3b), die unmögliche zusätzliche Transporte ausschließt. (7.3c) kann aufgrund von (7.2g) in Kombination mit (7.2i), (7.2a) und (7.2d) entfallen, die bereits die zeitliche Komponente hinreichend berücksichtigen.

Die hier vorgestellte Formulierung beinhaltet sämtliche vorab vorgestellten Problemstellungen als Spezialfälle. Tabelle 7.1 zeigt dies beispielhaft für das PDP, das VRPPD und das VRP. Da das VRP zu den NP-schweren Problemen gehört, trifft dies also auch auf seine Verallgemeinerung in Form des hier vorgestellten Problems zu (vgl. [Vahrenkamp, 2003, S. 241]).

Spezialfall	Voraussetzungen
PDP	$F = \{k\}^\dagger$ $k^+ = k^- =$ Depot $\hat{c}_{uv} = c_{uv}\ \forall (u,v) \in E$ $e^+_a = e^-_a = 0\ \forall a \in A$ $l^+_a = l^-_a = \infty\ \forall a \in A$ $t^+_a = t^-_a = 0\ \forall a \in A$
VRPPD	wie PDP und zusätzlich Depot $\in \{o^+_a, o^-_a\}\ \forall a \in A$
VRP	wie VRPPD und zusätzlich $o^+_a =$ Depot $\forall a \in A$

† Daher entfällt der Fahrzeugindex für alle weiteren Variablen und Konstanten in dieser Tabelle.

Tabelle 7.1: Spezialfälle in der einfachen Modellierung

7.1.2 Erweiterung des Modells um Umlademöglichkeiten

Ein wesentlich größerer Lösungsraum entsteht, wenn erlaubt wird, dass die zu den Aufträgen gehörenden Sendungen umgeladen werden dürfen. In der Praxis ist dieses Szenario sehr häufig anzutreffen, insbesondere bei großen Dienstleistern wie Post- oder Paketdiensten. Diese organisieren die Touren nach der so genannten *Hub-and-Spoke-Architektur*. Hierbei bildet ein Unternehmensort die zentrale *Nabe* (engl. *hub*) und ist über Transportverbindungen mit vielen anderen Logistikzentren des Unternehmens über *Speichen* (engl. *spoke*) genannte Wege verbunden (vgl. [Pfohl, 2004, S. 128 f.]). So kann jedes weitere Logistikzentrum durch Hinzufügen einer einzigen Speiche zur Nabe an das Transportnetzwerk angeschlossen werden und es entsteht eine näherungsweise sternförmige Struktur um die zentrale Nabe.

Sofern es die Kunden-Zeitfenster zulassen, können durch Umladungen auch bei Transportunternehmen wie IN tIME Einsparungen erzielt werden – beispielsweise durch Konsolidierungen. Ladevorgänge benötigen jedoch aufgrund der Sendungsstruktur in vielen Fällen Hilfsmittel wie Hubwagen oder Gabelstapler und können deshalb nicht überall stattfinden. Dies trifft beispielsweise auf Güter zu, die auf Europoolpaletten transportiert werden. Bei IN tIME sind daher grundsätzlich nur die Niederlassungen für Umladungen vorgesehen.

Deadlocks

Falls Lösungen zeitlich unzureichend synchronisiert sind, besteht bei diesem Tourenplanungsszenario die Gefahr von *Deadlocks*: Durch wechselseitige Abhängigkeiten warten unter Umständen zwei Fahrzeuge endlos auf die Anlieferung einer Sendung des jeweils anderen Fahrzeugs an Umladeknoten (vgl. [Oertel, 2000, S. 21 f.]). Abbildung 7.2 zeigt den Aufbau einer Deadlock-Situation schematisch.

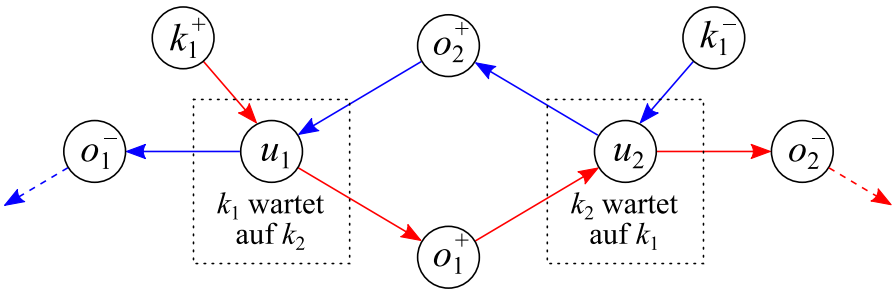


Abbildung 7.2: Deadlock bei Tourenplanung mit Umladungen
Fahrzeug k_1 wartet bei Umladeort u_1 auf die Sendung zu Auftrag o_2 durch Fahrzeug k_2. Gleichzeitig wartet Fahrzeug k_2 bei Umladeort u_2 auf die Sendung zu Auftrag o_1 durch Fahrzeug k_1.
Rot dargestellt: Route von Fahrzeug k_1
Blau dargestellt: Route von Fahrzeug k_2
Quelle: in Anlehnung an Oertel [2000, S. 21]

Deadlocks können verhindert werden, indem die zeitliche Reihenfolge des vollständigen Tourenplans berücksichtigt wird (vgl. [Oertel, 2000, S. 21 f.]). Die hier vorgestellte Problemformulierung beinhaltet schon die zeitliche Komponente aufgrund der Einhaltung der Kunden-Zeitfenster. Lösungen, die Deadlocks enthalten, verletzen damit bereits Nebenbedingungen und müssen deshalb nicht durch zusätzliche Nebenbedingungen ausgeschlossen werden.

Der Einbezug von Umladeorten muss natürlich auch in der mathematischen Formulierung berücksichtigt werden. Zusätzlich zur bisherigen Problemspezifikation stehe daher eine Menge U von optionalen Umladeorten zur Verfügung mit $U \cap O = \emptyset$. Wenn die Sendung eines Auftrags umgeladen wird, entstehen eine Zwischenbeladung, die für den Auftrag $a \in A$ fahrzeugunabhängig mit t_a^{z+} Zeiteinheiten veranschlagt wird, und eine Zwischenentladung, die mit t_a^{z-} Zeiteinheiten in die Rechnung einfließt.

Der gerichtete Graph $G = (V, E)$, in dem die Lösung abgebildet wird, muss um die Umladeorte als Knoten erweitert werden. Außerdem erhält er zusätzliche Pfeile $\{(u, v) \mid (u, v) \in (U \times O) \cup (O \times U) \cup (K^+ \times U) \cup (U \times K^-)\}$, um die Umladeorte an den restlichen Graphen anzubinden. Liegt einer der Umladeorte an einem auftragsbezogenen Ort, entstehen im zugehörigen Graphen zwei Knoten am selben Punkt, nämlich ein Knoten für die Be- oder Entladung und ein Knoten für die Umladung. Abbildung 7.3 zeigt schematisch die Struktur des veränderten Graphen.

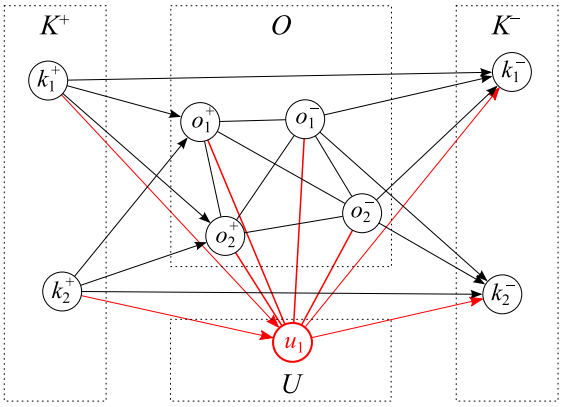


Abbildung 7.3: Schematischer Graph für das PDP bei IN tIME mit Umladeorten
Dargestellt ist ein Szenario mit zwei Fahrzeugen, zwei Aufträgen sowie einem möglichen Umladeort. Rot hervorgehoben sind die neu hinzugekommenen Elemente im Vergleich zu Abbildung 7.1, S. 267.

Die Zielfunktion (7.1), S. 268, bleibt unverändert. Um sicherzustellen, dass die Routen weiterhin zusammenhängend sind, muss Nebenbedingung (7.2c) ersetzt werden, so dass (7.2c′) entsteht:

$$\sum_{(u,v)\in E} x_{uv}^k - \sum_{(v,w)\in E} x_{vw}^k = 0 \qquad \forall v \in O \cup U,\ k \in F \quad (7.2c')$$

Die Forderung nach lückenlosem Transport in (7.2e) muss angepasst werden, da auch mehrere Umladungen am selben Umladeort möglich sind. Sie lautet nun:

$$y_{uv}^{ka} = 1 \Rightarrow \sum_{(v,w)\in E} y_{vw}^{ka} = 1 \; \forall a \in A, k \in F, (u, v) \in E \mid v \notin U \cup \{o_a^-\} \quad (7.2e')$$

Damit die Umladungen auch zeitlich berücksichtigt werden, ist zusätzlich zu Nebenbedingung (7.2i) die neue Nebenbedingung (7.2o) einzuhalten, die sich auf die Umladeorte bezieht. Da an einem Umladeort *mehrere* Aufträge umgeladen werden können, muss hier die Summe über alle Aufträge gebildet werden:

$$B_v^k + \sum_{a \in A} \max\left\{0, \sum_{(u,v)\in E} y_{uv}^{ka} - \sum_{(v,w)\in E} y_{vw}^{ka}\right\} t_a^{z-} + \sum_{a \in A} \max\left\{0, \sum_{(v,w)\in E} y_{vw}^{ka} - \sum_{(u,v)\in E} y_{uv}^{ka}\right\} t_a^{z+} + t_{vw}^k \leq B_w^k \qquad \forall v \in U,\ k \in F \quad (7.2o)$$

Um die Transport-Flusserhaltung darüber hinaus auch für die Transporte über Umladeorte zu gewährleisten, muss eine zweite zusätzliche Nebenbedingung eingehalten werden:

$$\sum_{\substack{k\in F \\ (u,v)\in E}} y_{uv}^{ka} - \sum_{\substack{k\in F \\ (v,w)\in E}} y_{vw}^{ka} = 0 \qquad \forall v \in U,\ a \in A \quad (7.2p)$$

7.1.3 Erweiterung des Modells um Teilladungen

Je nach Problemstellung ist es manchmal erlaubt, die Fracht eines Auftrags gegebenenfalls auf mehrere Fahrzeuge zu verteilen. Dies ermöglicht eine bessere Fahrzeugauslastung und damit Kosteneinsparungen auf Seiten des Transportdienstleisters. In der Literatur wird ein Szenario dieser Art als VRP mit *Split Load* oder auch *Split Delivery* bezeichnet. Entsprechend existieren sowohl *SL* als auch *SD* als kennzeichnende Abkürzung bei der Problembeschreibung.[12][13] Durch die Aufteilung einer Sendung in theoretisch beliebige Teilmengen vergrößert sich der Lösungsraum im Vergleich zur bisherigen Modellierung erheblich. SL

In der mathematischen Formulierung benötigt die vollständige Beladung bzw. Entladung der zum Auftrag $a \in A$ gehörigen Fracht fahrzeugunabhängig jeweils t_a^+ bzw. t_a^- Zeiteinheiten. Eine Zwischenbeladung bzw. -entladung dieser Fracht wird mit t_a^{z+} bzw. t_a^{z-} Zeiteinheiten berücksichtigt. Diese Zeiten werden nunmehr als linear skalierbar und proportional zur entsprechenden Gütermenge angenommen, d. h., wird von einem Fahrzeug nur die Hälfte einer Sendung abgeholt, wird hierfür auch nur die Hälfte der Beladungszeit veranschlagt.[14]

Modellierung der Teilladungen

Die Modellierung eines Transports von Teilladungen erfolgt anhand der Variable y. Dazu wird ihr Wertebereich von binär auf das vollständige Intervall $[0; 1]$ erweitert, wobei dieses als Prozentsatz zu interpretieren ist: $y_{uv}^{ka} = 0,75$ gibt beispielsweise an, dass das

[12] *SL* verwenden beispielsweise Nowak u. a. [2008], während Archetti u. a. [2006] *SD* bevorzugen.

[13] Nowak u. a. [2008, S. 34] heben hervor, dass nur Aufträge als geteilt gelten sollten, die ansonsten auch von einem einzelnen Fahrzeug bedient werden könnten. Ein Auftrag a gilt nach dieser Argumentation nicht als geteilt, wenn kein Fahrzeug der zur Verfügung stehende Flotte eine ausreichend große Kapazität besitzt, um a allein auszuführen.

[14] Dieses Vorgehen führt im Extremfall zu unrealistischen Zeitdauern. Denn unabhängig vom Anteil der umzuladenden Fracht entstehen Rüstzeiten wie z. B. für das Ausfüllen von Frachtpapieren oder das Heranholen eines Hubwagens. Für das hier vorgestellte Modell ist die vereinfachende Betrachtung jedoch ausreichend.

Fahrzeug k drei Viertel der zu Auftrag a gehörigen Sendungen von Ort u nach Ort v transportiert.

Die Zielfunktion unterscheidet mithilfe der Signum-Funktion die Kosten für Leer- und Lastfahrten. Sie lautet:

$$Z(X,Y,B) = \sum_{(u,v)\in E} \sum_{k\in F} \max\left\{x_{uv}^k \hat{c}_{uv}^k,\ \operatorname{sgn}(y_{uv}^{ka}) c_{uv}^k \mid a \in A\right\} \tag{7.1''}$$

Durch den veränderten Wertebereich der Transport-Variablen y müssen die meisten Nebenbedingung angepasst werden, die sich auf den Transport der Sendungen beziehen. Die veränderten Fassungen sind nachfolgend durch ein $''$ gekennzeichnet. Eine Ausnahme bilden die Nebenbedingungen (7.2i) und (7.2o), die in ihrer aktuellen Form übernommen werden können.

$$y_{uv}^{ka} > 0 \Rightarrow x_{uv}^k = 1 \qquad \forall (u,v) \in E,\ a \in A,\ k \in F \tag{7.2d''}$$

$$y_{uv}^{ka} > 0 \Rightarrow \sum_{(v,w)\in E} y_{vw}^{ka} = y_{uv}^{ka} \qquad \forall a \in A,\ k \in F,\ (u,v) \in E \mid v \notin U \cup \{o_a^-\} \tag{7.2e''}$$

$$y_{uv}^{ka} \in [0;1] \qquad \forall u,v \in V,\ a \in A,\ k \in F \tag{7.2m''}$$

Wenn – wie bei IN tIME üblich – gefordert wird, dass die Sendungen der Kunden jeweils mit nur einem Fahrzeug abgeholt und ausgeliefert werden, lässt sich dies mittels weiterer Nebenbedingungen erzwingen.

$$x_{uo_a^+}^k = 1 \Rightarrow y_{uo_a^+}^{ka} = 1 \qquad \forall (u, o_a^+) \in E,\ a \in A,\ k \in F$$

$$x_{o_a^- v}^k = 1 \Rightarrow y_{o_a^- v}^{ka} = 1 \qquad \forall (o_a^-, v) \in E,\ a \in A,\ k \in F$$

7.1.4 Grenzen des Modells

Da ein Modell in der Regel nur einen Teil der Realität abbildet, ergeben sich naturgemäß Grenzen in der Darstellung. Die drei wichtigsten werden im Folgenden erläutert.

Handhabung von Ladevorgängen Zwar berücksichtigt das Modell Zeitdauern für die Be- und Entladung sowie für alle Umladungen jedes Auftrags, doch in der Realität können diese starken Schwankungen unterliegen. Insbesondere spielt in der Praxis zusätzlich noch der Verlader selbst eine wichtige Rolle. So kann es beispielsweise in großen Automobilwerken phasenweise zu regelrechten „Lieferstaus“ kommen, wenn zu viele Be- und Entladungen gleichzeitig abgefertigt werden sollen. Falls ein Fahrzeug noch Fracht für weitere Kunden geladen hat, verzögert sich dann entsprechend deren Auslieferungszeit.

Weiterhin sind in der vorliegenden Modellierung mehrere gleichzeitige parallele Be- und Entladungen an den einzelnen Standorten möglich. Realistisch betrachtet, ist deren gleichzeitige Bearbeitung jedoch sowohl durch die Verfügbarkeit von Arbeitskräften, den Platz an den Ladestellen als auch die Anzahl der Hilfsmittel wie etwa Gabelstapler begrenzt.

Kein Einbezug von Fahrermodellen Die vorliegende Problemformulierung berücksichtigt darüber hinaus die Einhaltung der Sozialvorschriften gemäß VO (EG) 561/2006 nicht. Sie wurden an dieser Stelle bewusst ausgelassen, um auf die „klassische“ Tourenplanung zu fokussieren. Für eine vollständige formale Beschreibung der VO (EG) 561/2006, die über die Darstellung in Abschnitt 2.3 hinausgeht, sei daher auf weiterführende Literatur verwiesen.[15]

Deterministisch-statische Modellierung Alle Einflussgrößen werden als bekannt und unveränderlich vorausgesetzt, obwohl beide Aspekte in der Praxis so nur seltenen Fällen anzutreffen sind. Einerseits lässt sich das Eintreffen neuer Aufträge im Allgemeinen nicht vorhersagen. Andererseits können dynamisch auftretende Störungen wie Verkehrsstaus oder Fahrzeugausfälle zu veränderten Rahmenbedingungen führen.

7.2 Ableitung des Konsolidierungsproblems

Tourenplanungsprobleme, wie das im vorangegangenen Abschnitt vorgestellte allgemeine PDP, lassen sich bislang nur in begrenztem Maße vollständig lösen. Dies liegt im Wesentlichen an ihrer Zugehörigkeit zur Klasse der NP-schweren Probleme, welche die Rechenzeit größenordnungsmäßig exponentiell mit der Problemgröße anwachsen lässt (vgl. [Vahrenkamp, 2003, S. 241]). Selbst neuere Verfahren benötigen daher zur optimalen Lösung verhältnismäßig kleiner Probleminstanzen bereits mehrere Minuten bis sogar Stunden.[16]

So verwundert es nicht, dass keine der bei Parragh u. a. [2008, S. 107] geführten Benchmark-Instanzen für verschiedene Variationen des VRPPD mehr als 1 000 Aufträge vorsehen. Alle bis auf zwei der elf aufgeführten Benchmarks arbeiten dabei mit weniger als 300 Kundenaufträgen – bei den für diese Arbeit vorliegenden Vergangenheitsdaten sind zumeist Hunderte und bis zu etwa 1 500 Aufträge parallel in Bearbeitung.

In den hier genannten Quellen sowie den meisten in der Literatur veröffentlichten Benchmarks werden zudem die Distanzen der Routen einer vollständigen Distanzmatrix entnommen und nicht etwa – beispielsweise mit den in den Kapiteln 4–6 vorgestellten Verfahren – *on demand* berechnet.

[15] Beispielsweise formuliert Goel [2010] die wesentlichen Aspekte der VO (EG) 561/2006 und führt mehrere Verfahren zum Finden eines zulässigen Tourenplans an. Kopfer u. Meyer [2009] liefern ein vollständiges (Optimierung-)Modell für ein Rundreiseproblem unter Berücksichtigung der VO (EG) 561/2006. Beide Quellen betrachten jedoch nur Problemstellungen ohne den in Abschnitt 2.3.2 auf Seite 53 f. beschriebenen *Mehrfahrerbetrieb.*

[16] Beispielsweise untersuchen Pisinger u. Ropke [2007] unter anderem künstliche VRPTW-Instanzen mit 100 Kunden, für deren optimale Lösung sie mit bekannten als auch neuen Verfahren mehrere Minuten Rechenzeit benötigen.

Wassan u. Osman [2002, S. 776] vergleichen heuristische Lösungsverfahren zu VRP-Instanzen mit heterogener Flotte und fahrzeugabhängigen Fahrtkosten. Die größten beiden dort berichteten Probleminstanzen umfassen 100 Kunden und werden in 53 bzw. 100 Minuten Rechenzeit gelöst.

Auch Oertel [2000, S.72 ff.] untersucht PDPTW-Problemstellungen aus der Praxis mit jeweils „etwa 70 Aufträgen“ (ebd.). Die von ihm vorgestellten Heuristiken benötigen hierfür zwischen fünf und zwanzig Minuten Rechenzeit (vgl. [Oertel, 2000, S. 75]).

7.2.1 Grenzen der automatischen Disposition

Nicht zuletzt ist auch aus pragmatischen Überlegungen ein Tourenplan, der eine Kostenfunktion wie (7.1″) gänzlich minimiert, für die Praxis kaum relevant. Dies hat neben dem sehr großen Rechenaufwand noch drei weitere wesentliche Gründe:

1. Ein Tourenplan, der solch eine Kosten-Zielfunktion minimiert, beinhaltet erwartungsgemäß nur wenig zeitliche Puffer. Dies macht ihn anfällig für in der Realität häufig auftretende dynamische Störungen wie Fahrzeugausfälle oder Verkehrsstaus.

2. Für Transportunternehmen, bei denen Aufträge nur wenig Vorlaufzeit besitzen, ist es nicht sinnvoll, bei jedem Eintreffen eines neuen Auftrags den aktuellen Tourenplan von neuem zu berechnen. Selbst wenn dies innerhalb akzeptabler Zeit möglich wäre, enthielte ein neu berechneter Tourenplan in vielen Fällen veränderte Routen – auch für Fahrzeuge, die den neuen Auftrag nicht bedienen. Dies würde erheblich mehr Kommunikation mit den Fahrern erfordern und dadurch ihre Belastung aufgrund sich häufig ändernder Routenführung erhöhen.

3. Kein aktuell beherrschbares, mathematisches Modell kann alle Entscheidungskriterien beinhalten, die zu einer realen Tourenplanung führen. Bezogen auf quantifizierbare Daten bezeichnet Drexl [2012, S. 60 f.] dieses Phänomen als *Forschungslücke*[17]. Viele dieser Kriterien sind darüber hinaus nur schwer in Zahlen auszudrücken; manche lassen sich sogar kaum erfassen. Hierzu gehören unter anderem die folgenden Aspekte:

   Gleichauslastung der Fahrer durch die Disposition Wenn Transportdienstleister mit Subunternehmern zusammenarbeiten, müssen sie darauf achten, diese in etwa gleichmäßig auszulasten. Ansonsten drohen seltener beauftragte Subunternehmer aufgrund von finanziellen Einbußen oder Unzufriedenheit zu Mitbewerbern abzuwandern.

   Eignung der Fahrzeuge Die Erfassung der Fahrzeugdaten erfolgt in der Regel nur mit einem abstrahierten Ladevolumen. Zu jedem Fahrzeug werden dabei die Kantenlängen eines Quaders angegeben, der den nutzbaren Laderaum repräsentiert. Diese Angaben sind verständlicherweise nicht exakt. Dem Autor sind beispielsweise Fälle bekannt, in denen sich Disponenten gegen die Zuweisung eines bestimmten Fahrzeugs für einen Auftrag entschieden, da sie aus Erfahrung wussten, dass dessen Radkästen den unteren Laderaum zu stark verjüngten, um die Sendung sicher zu transportieren.

[17] im englischen Original: *research gap*

Subjektive Aspekte bei der Disposition Letztlich wirken auch ganz persönliche Gründe auf eine Tourenplanung ein. Dies können schlechte Erfahrungen mit einem Fahrer bei einem bestimmten Kunden sein. Aber auch sonstige Gründe beeinflussen mitunter die Routen, wenn etwa einem Fahrer ermöglicht werden soll, private Termine einzuhalten.

Zusammengenommen führt dies dazu, dass auf absehbare Zeit kein Software-System in der Lage sein wird, eine vollständig automatisierte Disposition durchzuführen. Vielmehr konzentriert sich daher der Software-Einsatz auf Szenarien der Entscheidungsunterstützung. (vgl. [Drexl, 2012, S. 61])

Unternehmen, wie die in dieser Arbeit untersuchte IN tIME Express Logistik GmbH, versuchen aus den oben genannten Gründen beim Eintreffen eines neuen Auftrags, diesen in den aktuellen Tourenplan zu *integrieren*, anstatt einen vollständig neuen Tourenplan zu generieren. Da hierbei möglichst geringe zusätzliche Kosten entstehen sollen, werden Lösungen bevorzugt, die ein Zusammenlegen des neuen Auftrags mit einem bereits geplanten Auftrag ermöglichen. Gleichzeitig sollte das Einfügen des neuen Auftrags nicht zu viele Änderungen der bereits in Durchführung befindlichen Fahrten verursachen. Diese Überlegungen führen zu einer Problemstellung, die hier *Konsolidierungsproblem* genannt werden soll und bereits zu Beginn dieses Kapitels umrissen wurde.

7.2.2 Definition des Konsolidierungsproblems

Gegeben ist ein vollständiger Tourenplan, bestehend aus aktuell durchgeführten sowie für die Zukunft geplanten Touren. Für einen neuen Auftrag $\hat{a}$ sind ein oder mehrere andere noch nicht vollständig bearbeitete Aufträge gesucht, die sich für eine Konsolidierung entlang der zu $\hat{a}$ gehörigen Strecke oder Teilstrecken davon eignen. Hierbei sind Umladungen und auch Transporte von Teilladungen der zu $\hat{a}$ gehörigen Sendungen erlaubt, sofern sie weder die Einhaltung der Zeitfenster von $\hat{a}$ noch die der anderen Aufträge gefährden.

Für die Lösung des Konsolidierungsproblems eignet sich besonders der Einsatz eines Entscheidungsunterstützungssystems[18]: Zwar können viele Arbeitsschritte und Berechnungen – insbesondere die Ermittlung schnellster Wege – automatisiert erfolgen. Doch dies gilt nicht für die gesamte Dispositionsentscheidung, da sie wie oben erläutert zu komplex ist. Die Anwendungsfälle eines EUS für die Lösung des Konsolidierungsproblems beinhalten

- die Berechnung sämtlicher theoretisch möglichen Konsolidierungen,
- die Bewertung und Sortierung der möglichen Konsolidierungen,
- gegebenenfalls das Herausfiltern ungünstiger Konsolidierungen, bevor ein Disponent eine Entscheidung trifft, sowie stets
- eine geeignete Darstellung der ermittelten Konsolidierungsoptionen für den Disponenten.

[18] siehe hierzu auch Seite 24 ff.

Aus praktischen Gründen sollte das System die Verfügbarkeit von *konkreten* Fahrzeugen für den Transport je nach Anwendungsfall hintanstellen. Da bei dem hier betrachteten Unternehmen regelmäßig Fahrzeuge als verfügbar bzw. nicht mehr verfügbar gemeldet werden, reicht es für die Entscheidungsunterstützung im Allgemeinen aus, mit abstrakten *Fahrzeugkategorien*[19] zu rechnen. Erst wenn der Disponent eine Entscheidung treffen möchte, können in einem zweiten Schritt konkrete verfügbare Fahrzeuge für den Transport ermittelt werden.

7.3 Modellierung des Konsolidierungsproblems in der Implementierung

Vor der empirischen Ermittlung des Einsparungspotentials durch Konsolidierungen im nachfolgenden Abschnitt liefert dieser Abschnitt zunächst eine Erläuterung, mit welchen Werten die Rahmenbedingungen des Konsolidierungsproblems initialisiert werden. Zusätzlich wird die effiziente Modellierung der Aufträge und ihrer Teilungen durch Vorabberechnung grundsätzlich möglicher Umladungen näher beleuchtet.

7.3.1 Modellierung zeitbezogener Vorgänge

Sowohl die Fahrtdauern als auch die Dauern für Beladungen und Entladungen der Güter müssen für eine empirische Untersuchung mit sinnvollen Werten belegt werden. Zwar liegen im Rahmen der Vergangenheitsdaten eine Reihe von Fahrtzeiten und Dauern für Ladevorgänge vor. Doch diese spiegeln natürlich nur die Ergebnisse der tatsächlich durchgeführten Tourenpläne wider.

Bei dieser Untersuchung werden dagegen sehr viele retrospektiv plausible alternative Tourenpläne generiert. Um sie mit denen aus der Vergangenheit vergleichen zu können, müssen also auch die Fahrtzeiten anderer Routen und die Zeiten für (Um-)Ladungen an anderen Orten angegeben werden.

Die Abschätzung der Fahrtdauern wurde bereits in Abschnitt 5.5.4, S. 172 ff., hergeleitet und wird auch in dieser Untersuchung verwendet. Sie basiert auf der OSM-Straßenkarte Europas, deren Extraktion in Kapitel 3 beschriebenen wurde. Darauf aufbauend wurde eine Contraction Hierarchy mit Abbiegebeschränkungen für schnellste Wege berechnet, bei der die Fahrzeugkategorie „Transporter“ zugrunde liegt. Es handelt sich dabei um dieselbe Contraction Hierarchy mit der Knotensortierung $2R+8EQ+4OEQ+1Q+1RAND$, die auch in Kapitel 6, S. 199 ff., für die Benchmarks[20] verwendet wurde. Die schnellsten Wege anderer Fahrzeugkategorien werden daher zwar heuristisch ermittelt, weichen aber nur sehr selten von den tatsächlichen schnellsten Wegen ab.[21] Die Abschätzungen der Fahrtzeiten entlang jedes Weges sind jedoch für alle Fahrzeugkategorien von gleicher Qualität.

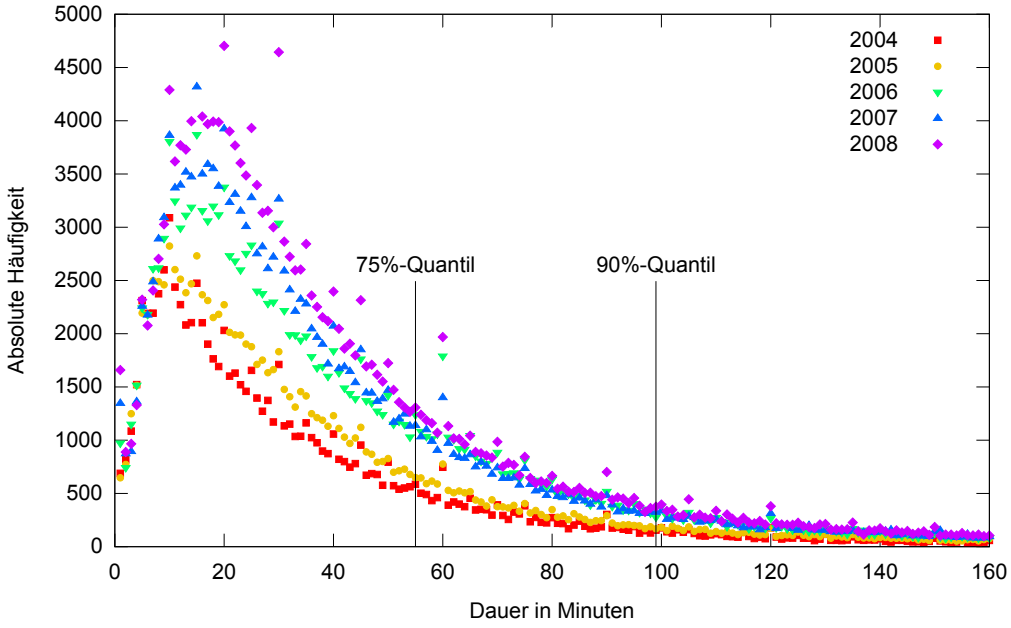
[19] Fahrzeuge der gleichen Kategorie besitzen ein ähnliches Ladevolumen und Geschwindigkeitsprofil.
[20] für die Benchmarks: siehe Abschnitt 6.6, S. 239 ff.
[21] Eine Abweichung kann entstehen, wenn in der „Transporter-Contraction-Hierarchy“ Shortcut-Pfeile fehlen, die für die Berechnung eines schnellsten Weges für eine andere Fahrzeugkategorie notwendig sind.

Die Abschätzungen der Dauern für Ladevorgänge sowie die Auswirkungen der Vorschriften zu den Lenk- und Ruhezeiten sind noch zu treffen. Sie werden nachfolgend hergeleitet.

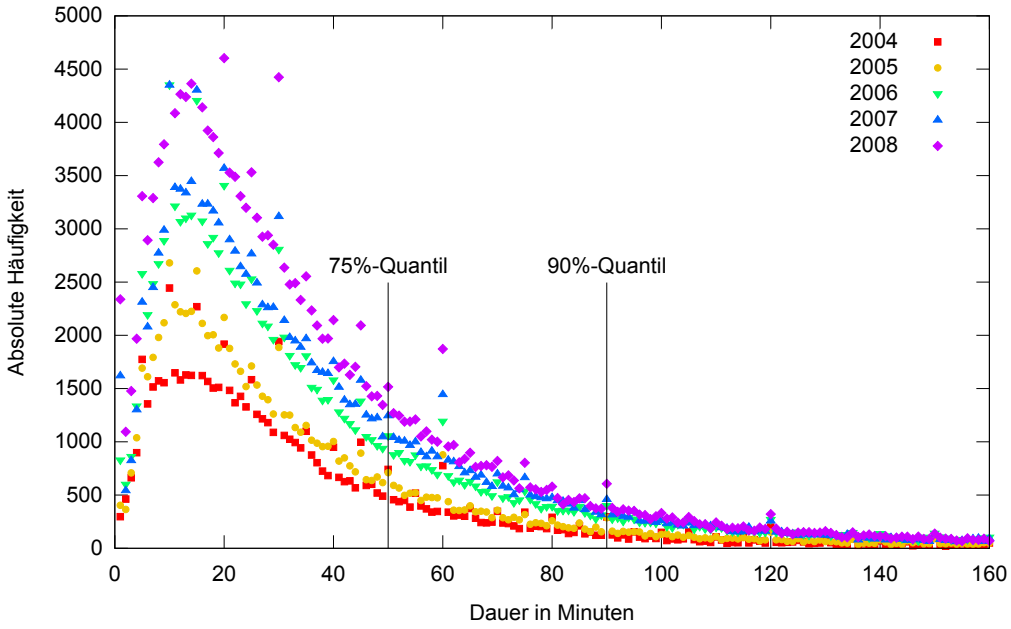
Dauern von Ladevorgängen

Die Implementierung der Benchmarks unterscheidet vier Arten von Ladevorgängen: Startbeladungen, Zwischenentladungen, Zwischenbeladungen und Endentladungen, entsprechend der Parameter t_a^+, t_a^{z-}, t_a^{z+} und t_a^- aus dem mathematischen Modell. Die Zwischenladevorgänge geschehen dabei nur an ausgewählten Umladeorten, den 21 Niederlassungen des Transportunternehmens. Für die Berechnung der Benchmarks werden unabhängig vom Auftrag jeweils konstante Werte für die Dauern dieser Ladevorgangsarten angenommen. Zwar könnten diese Dauern anhand der Vergangenheitsdaten sogar pro Auftrag rekonstruiert werden, doch dieses Wissen lag den Disponenten in der Vergangenheit ebenfalls nicht vor. Somit arbeiten die Benchmarks mit einem vergleichbaren „Kenntnisstand“.

Die Verteilung der Dauern für Ladevorgänge gemäß den vorliegenden Vergangenheitsdaten zeigen Abbildungen 7.4 und 7.5. Zwar sind für alle Messreihen deutliche Modalwerte in der Häufigkeitsverteilung zu erkennen, doch es sind jeweils nur etwa ein Viertel aller Ladevorgänge, die in weniger Zeit als diese Maxima absolviert wurden.

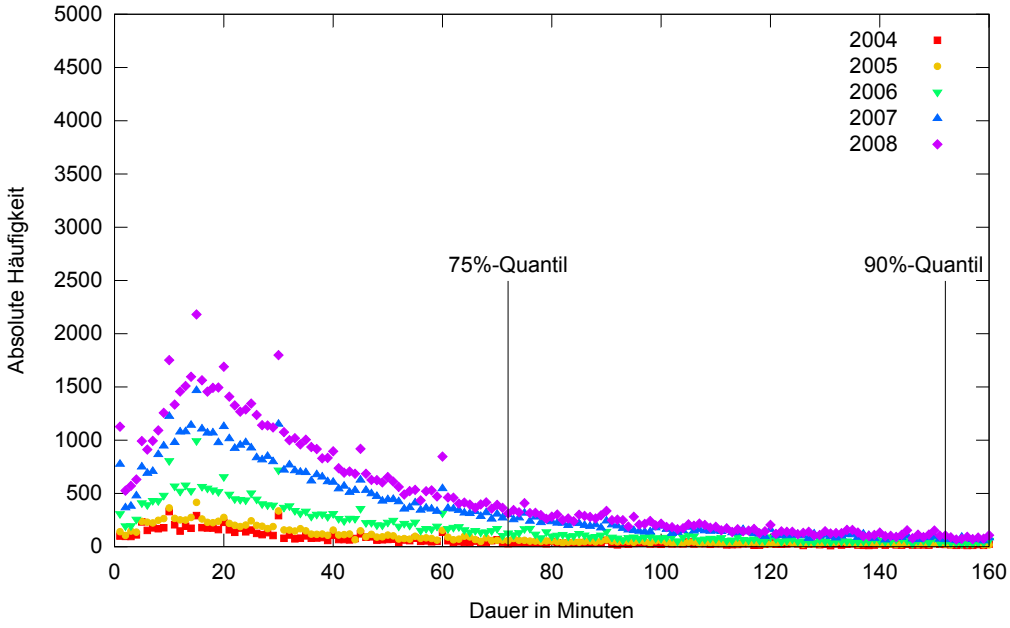


(a) Startbeladungen

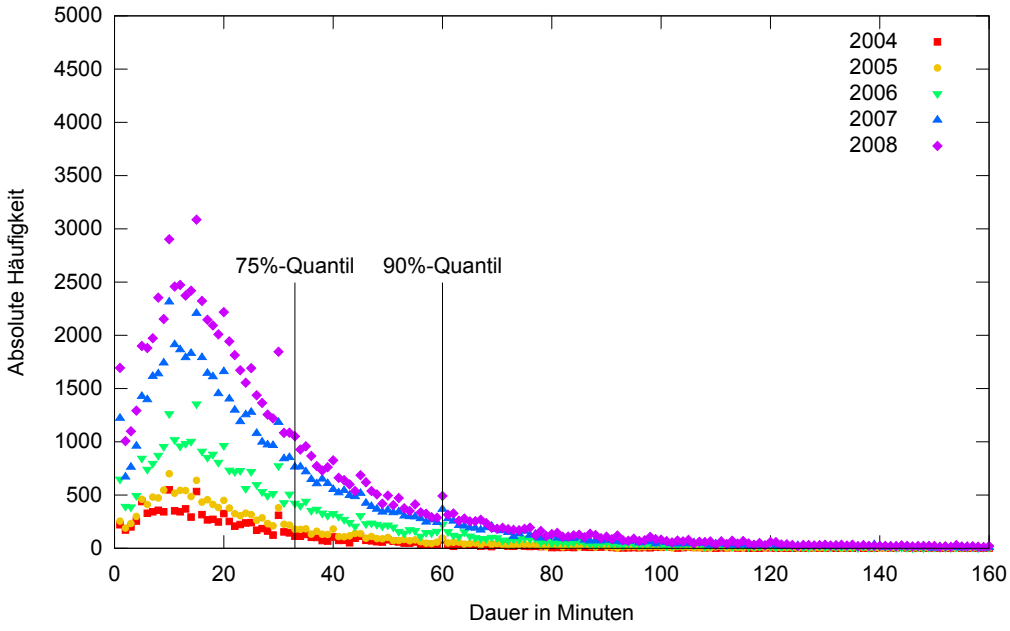


(b) Endentladungen

Abbildung 7.4: Dauern der Be- und Entladungsvorgänge bei IN tIME (1)
Die Quantile beziehen sich auf das Jahr 2008.
Dargestellt ist ein Ausschnitt aller Werte ohne Ausreißer.



(a) Zwischenbeladungen



(b) Zwischenentladungen

Abbildung 7.5: Dauern der Be- und Entladungsvorgänge bei IN tIME (2)
Die Quantile beziehen sich auf das Jahr 2008.
Dargestellt ist ein Ausschnitt aller Werte ohne Ausreißer.

Damit die Benchmarks nicht von zu optimistischen Dauern ausgehen, wurden für das Jahr 2008 und jede Art von Ladevorgang deshalb jeweils das 75%- als auch das 90%-Quantil berechnet. Tabelle 7.2 zeigt die Ergebnisse dieser Untersuchung.

Art des Ladevorgangs	75%-Quantil	90%-Quantil
Startbeladung	55 min	99 min
Endentladung	50 min	90 min
Zwischenbeladung	72 min	152 min
Zwischenentladung	33 min	60 min

Tabelle 7.2: Dauern der Arten von Ladevorgängen im Jahr 2008

Zwei Benchmarkvarianten: B75 und B90

Die Ergebnisse der Benchmarks werden stets sowohl mit den optimistischeren Werten, also den Dauern der 75%-Quantile, als auch mit den pessimistischeren Werten des 90%-Quantils angegeben Die erste Variante wird fortan mit *B75* und die zweite mit *B90* referenziert. Dies erlaubt eine belastbarere Einschätzung des Konsolidierungspotentials.

Einbezug der Lenk- und Ruhezeiten

In Anbetracht der Komplexität der Lenk- und Ruhezeiten für angestellte Fahrer[22] sowie der Ausrichtung des hier vorgestellten EUS auf eine fahrzeugunabhängige Sichtweise erfolgt die Berechnung für die Abschätzung der Fahrtzeiten im Rahmen der Messungen in diesem Kapitel heuristisch: Als Minimal-Anforderung gilt, dass ein Fahrer spätestens nach 4,5 Stunden Fahrt eine mindestens 45-minütige Pause einzulegen hat[23]. Somit kann das Fahrzeug nicht sofort nach den 270 Minuten (4,5 Stunden), sondern erst nach 315 Minuten weiterfahren. Dies ergibt einen Korrekturfaktor von $\frac{315}{270} = \frac{7}{6}$. Die sich aus der Kombination von den der Wegstrecke zugrunde liegenden Kantenkategorien mit dem eingesetzten Fahrzeug ergebenden Fahrtdauern werden mit diesem Korrekturfaktor multipliziert, so dass ein durchschnittlich realitätsnäherer Wert entsteht.

7.3.2 Lokalisierung von Kundenorten und Ortungen

Innerhalb der für diese Arbeit zur Verfügung stehenden Vergangenheitsdaten liegen für alle Transporte und Ortungen die Angaben nicht in Form von Adressen, sondern Längen- und Breitengraden vor. Um Routen zwischen ihnen auf einer Contraction Hierarchy ermitteln zu können, müssen diesen Geokoordinaten daher Knoten der CH zugewiesen werden. Wie es sich zeigte, reicht es nicht aus, hierfür einfach denjenigen Knoten der digitalen Straßenkarte zu verwenden, der die kürzeste Luftdistanz zu den entsprechenden Koordinaten aufweist.

[22] siehe auch Abschnitt 2.3, S. 49 ff.
[23] siehe hierzu Seite 50, Stichwort „Arbeitszeiten“

Zwar wurde bei der Extraktion der hier zugrunde liegenden OSM-Daten versucht, eine möglichst „navigierbare“ Karte zu erzeugen[24], doch nicht alle Sonderfälle konnten dabei effizient berücksichtigt werden. Stattdessen kann es vorkommen, dass in der Karte kleine Bereiche existieren, aus denen kein Weg hinausführt. Abbildung 7.6 zeigt schematisch, wie solche Bereiche aufgebaut sind. Anschaulich formuliert könnte hier von einem *Senke-Bereich* der Karte gesprochen werden, da in diesen ausschließlich Wege hinein führen. Spiegelbildlich können auch *Quellen-Bereiche* auf der Karte existieren.

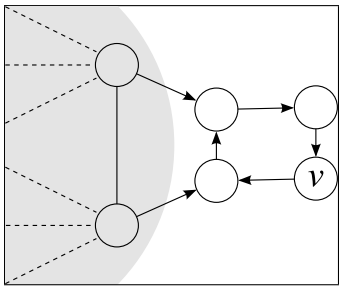


Abbildung 7.6: Beispiel für einen Senke-Bereich der Karte
Grau gekennzeichnet ist der Bereich, in dem sich die meisten Knoten des Graphen befinden. Knoten v ist zwar von diesem Bereich ausgehend erreichbar, doch es existiert kein bei v beginnender Weg in den Bereich *hinein*.

Es befinden sich weniger als 0,1 % aller Knoten aus Karten, die gemäß Kapitel 3 erstellt wurden, in einem Quellen- oder Senken-Bereich. Für die Zuweisung von Geokoordinaten zu einem Knoten der Straßenkarte wird daher derjenige Knoten verwendet, der die geringste Luftdistanz zu den gegebenen Koordinaten besitzt und gleichzeitig weder in einem Quellen- noch in einem Senken-Bereich der Karte liegt. Statistische Informationen der auf diese Weise zugewiesenen Knoten und den ihnen zugrunde liegenden Geokoordinaten liefert Tabelle 7.3 in Kombination mit Abbildung 7.7, S. 284.

Kennzahl	Wert
Kleinste Distanz	0 m
Median der Distanzen	49 m
Größte Distanz[†]	3 743 943 m

[†] Dieser Wert beruht allerdings auf fehlerhaften Kunden-Koordinaten in den Vergangenheitsdaten.

Tabelle 7.3: Distanz-Kennzahlen bei Knotenzuweisungen von Geokoordinaten
Die angegebenen Werte sind gerundet auf drei Nachkommastellen und beziehen sich auf alle 91 232 Kundenorte aus den Vergangenheitsdaten. Abbildung 7.7, S. 284, zeigt einen Ausschnitt der Verteilung der Werte.

[24] siehe Abschnitt 3.3, S. 66 ff.

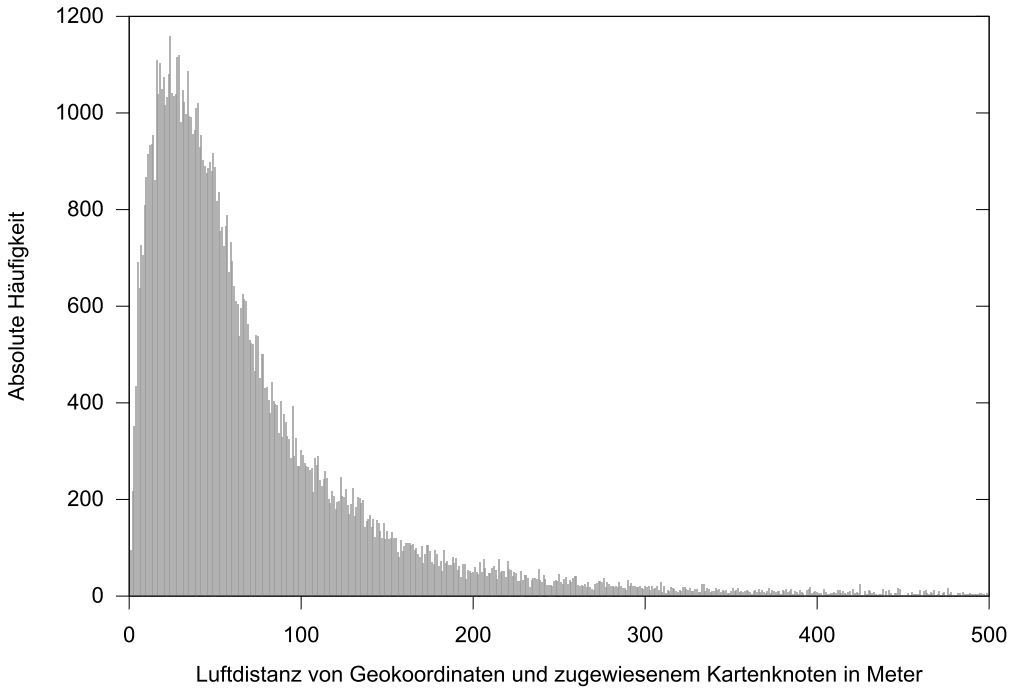


Abbildung 7.7: Distanzen bei Knotenzuweisungen von Kundenkoordinaten
Dargestellt ist ein Ausschnitt sämtlicher Werte. Die größte – auf fehlerhaften Kundenkoordinaten beruhende – Distanz liegt bei 3 743 943 Metern und ist damit ein deutlicher Ausreißer.

7.3.3 Kostenermittlung innerhalb der Benchmarks

Untersuchungsgegenstand der Benchmarks ist das Kosteneinsparungspotential durch die Konsolidierung der Sendungen einzelner Aufträge. Die hier betrachteten Kosten entstehen durch den Transport der Sendungen durch Fahrzeuge und deren jeweilige Tarife[25]. Die Benchmarks arbeiten zwar nicht mit einzelnen Fahrzeugen, unterscheiden aber zwischen *Fahrzeugkategorien*. Tabelle 7.4 gibt einen Überblick über die zugrunde gelegten Tarife und Kapazitäten der einzelnen Fahrzeugkategorien. Die verwendeten Tarife entsprechen den im Jahr 2008 marktüblichen Preisen.

Fahrzeugkategorie	Tarif	Nutzlast	Palettenstellplätze	Anteil 2008
Pkw	0,30 €/km	200 kg	0	15 %
(Klein-)Transporter	0,36 €/km	1200 kg	4	58 %
Transporter mit Anh.	0,41 €/km	3200 kg	7	7 %
Lkw (< 7 Tonnen)	0,53 €/km	2600 kg	15	15 %

Tabelle 7.4: Fahrzeugkategorien und ihre Tarife und Kapazitäten in den Benchmarks

[25] siehe auch S. 268, Stichwort „Kostenaspekte“

Der Grund für die Nutzung genau dieser Kategorien ist, dass sie für den innerhalb der Benchmarks betrachteten Zeitraum die mit großem Abstand wichtigsten Kategorien darstellen: Sie wurden bei über 95 % aller im Jahr 2008 erzeugten Frachtbriefe hinterlegt. Andere Kategorien, die zwar in den Vergangenheitsdaten vorkommen, aber in den Benchmarks nicht zur Konsolidierung herangezogen werden, sind *Sattelzug* oder *Flugzeug*.[26]

7.3.4 Effizientes Caching per Einzelauftrag

Die Modellierung der Aufträge durch eine spezielle Datenstruktur erleichtert die Berechnungen und ermöglicht dadurch zusätzlich schnelle Reaktionszeiten auf dynamische Veränderungen im Rahmen eines Echtzeit-EUS. Verwendet wird in dieser Arbeit der von Nowak u. a. [2009] vorgestellte *Einzelauftrag*, dessen Grundstruktur Abbildung 7.8 darstellt. Dieser ist besonders für den Einsatz in Tourenplanungsproblemen geeignet, bei denen eine endliche Anzahl von Umladeorten zur Verfügung steht.

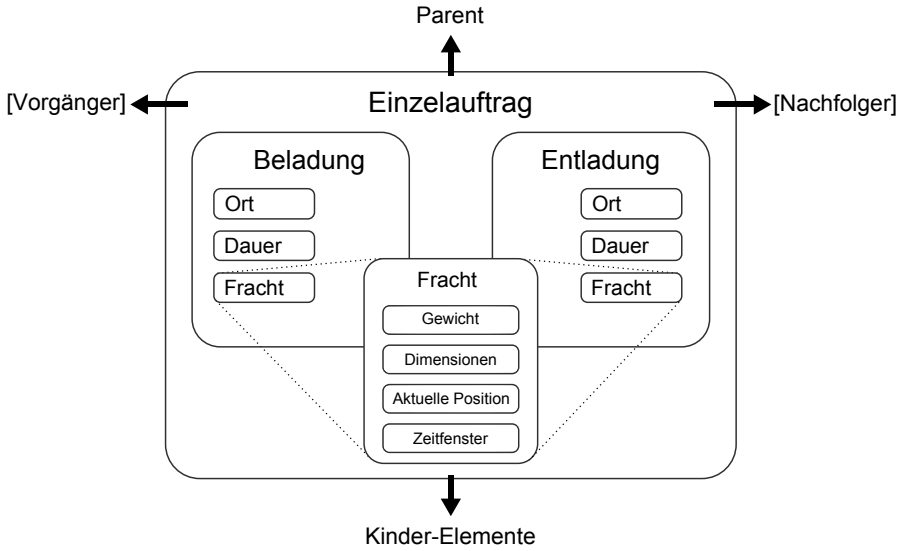


Abbildung 7.8: Der Einzelauftrag als Datenstruktur
Hervorgehoben ist die sowohl bei der Be- als auch Entladung zugrunde liegende Fracht, die daher intern nur einmal zu speichern ist.
Quelle: Nowak u. a. [2009, S. 3625]

Ein Einzelauftrag kann an einem Umladeort *geteilt* werden, wodurch eine Umladung dargestellt wird. Hieraus entstehen zwei neue Einzelaufträge, die als *Kinder* des ursprünglichen Einzelauftrags gespeichert werden. Abbildung 7.9, S. 286, zeigt das Prinzip schematisch.

[26] Eine detailliertere Diskussion zur Auswahl konsolidierungsfähiger Aufträge innerhalb der Benchmarks findet sich in Abschnitt 7.4.3 auf Seite 297, Stichwort „Verzicht auf Konsolidierungen“

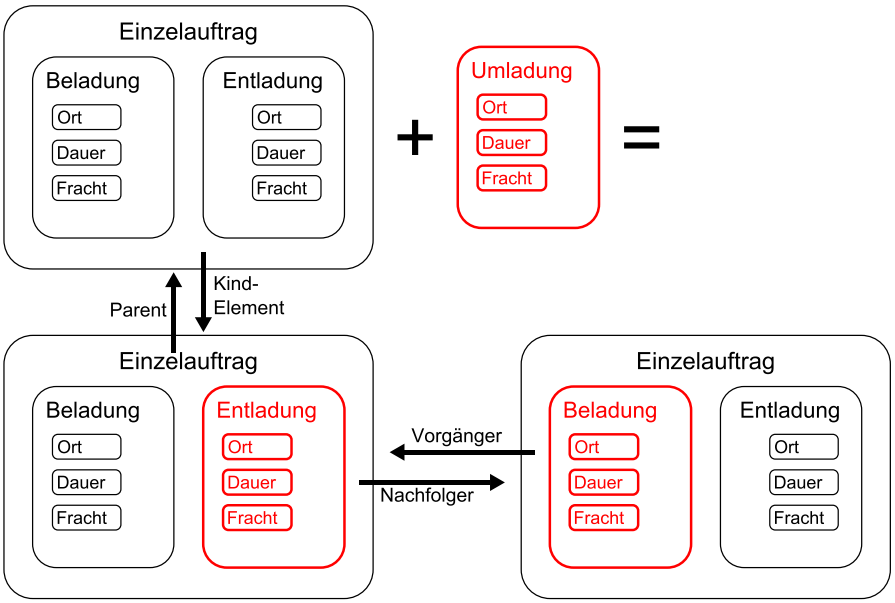


Abbildung 7.9: Teilung eines Einzelauftrags

Sind mehrere Umladungen möglich, kann rekursiv geteilt werden, so dass ein Teilungsbaum entsteht, wie ihn Abbildung 7.10, S. 287, zeigt. Hier wurde ein Einzelauftrag bis zu dreimal umgeladen. Alle zu einer Umladung gehörigen Orte sind zur besseren Unterscheidung mit jeweils derselben Farbe markiert. Teilungen, welche das im Ursprungsauftrag gegebene Zeitfenster aufgrund zu langer Fahrt- und Ladezeiten gefährden, werden im Teilungsbaum verworfen und nicht gespeichert. Im Folgenden werden die Wurzel-Elemente dieser Bäume auch *Top-Level-Einzelaufträge* genannt. Sie entsprechen den ungeteilten Original-Aufträgen, beinhalten also keine Umladungen.

Top-Level-Einzelaufträge

Alle theoretisch durchführbaren Teilungen eines Auftrags können so vorberechnet und speichereffizient in der Baumstruktur abgelegt werden. Nachfolgende Konsolidierungsberechnungen können dann auf diese Informationen zurückgreifen.

Zusätzlich eignet sich die Baumstruktur zur Beschleunigung der Berechnung der Konsolidierungen: Bei der Suche nach Konsolidierungsoptionen für einen Auftrag a sollte grundsätzlich zuerst der Top-Level-Einzelauftrag eines anderen Auftrags b geprüft werden. Nur falls eine Konsolidierung von a und b möglich ist, ist die nachfolgende Untersuchung der Kinder-Elemente von b als potentielle Konsolidierungspartner zu a überhaupt sinnvoll. Ist dagegen beispielsweise die gemeinsame Ladung von a und b zu groß für alle zur Verfügung stehenden Fahrzeugkategorien, kommt auch eine Konsolidierung von a mit den Kinder-Elementen von b nicht in Frage.

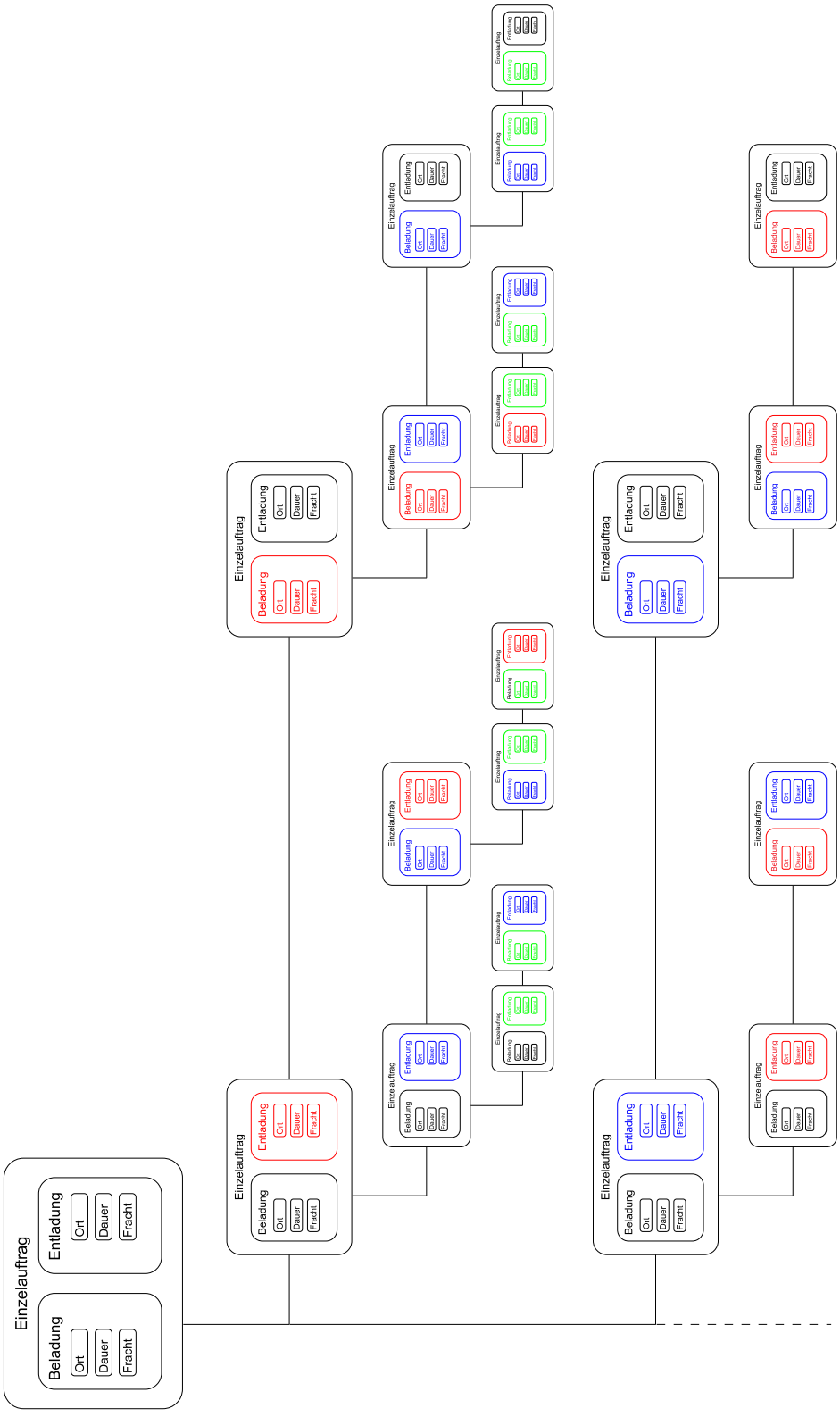


Abbildung 7.10: Teilungsbaum für einen Einzelauftrag

7.4 Benchmarks für das Konsolidierungsproblem

Den Benchmarks, anhand derer die Überlegungen aus diesem Kapitel untersucht werden, liegen die jüngsten zur Verfügung stehenden Vergangenheitsdaten zugrunde. Dies gestattet die beste Übertragbarkeit der Ergebnisse auf die Gegenwart, da das Transportunternehmen im vorliegenden Zeitraum von 2004–2008 ein merkliches Auftragswachstum verzeichnete (siehe Abbildung 7.11).

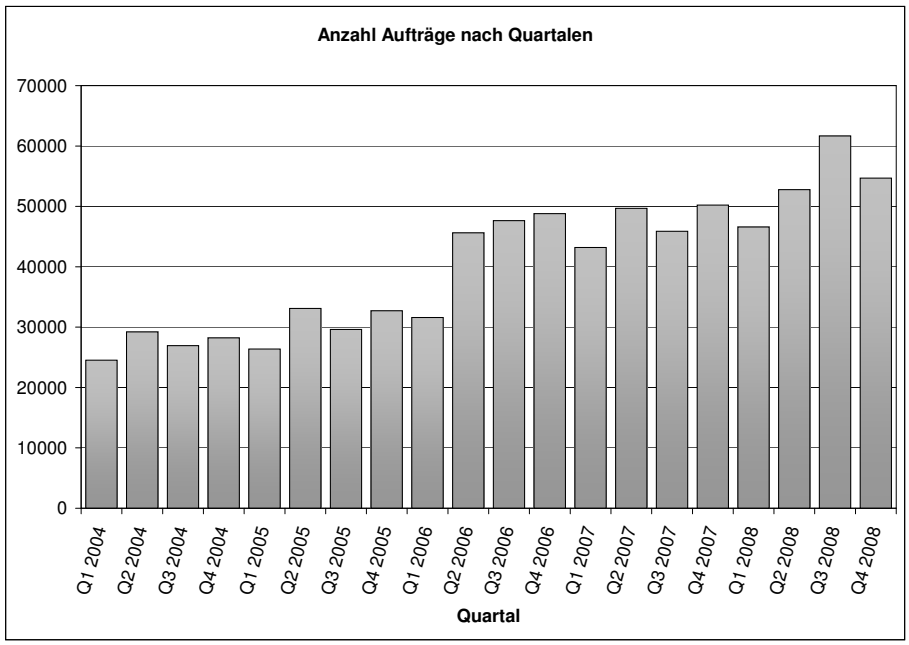


Abbildung 7.11: Aufträge in den Vergangenheitsdaten nach Quartal

Beginnend mit dem 1.1.2008 um 04:00 Uhr und endend mit dem 10.12.2008 um 8:00 Uhr wurden für jeden Tag im Abstand von vier Stunden Momentaufnahmen der aktuellen Dispositionssituation angefertigt.[27] Die Umstellungen zur Sommer- bzw. Winterzeit sind in der Weise ignoriert, dass ausschließlich die Uhrzeiten 00:00, 04:00, 08:00, 12:00, 16:00 und 20:00 vorkommen. D. h., aufgrund der Umstellung zur Sommer- bzw. Winterzeit existiert in den Benchmarks je ein Fall, bei dem zwei Momentaufnahmen drei bzw. fünf Stunden auseinander liegen.

Jede der so extrahierten 2 066 Situationen beinhaltet die Informationen zu allen zum jeweiligen Zeitpunkt bereits bekannten, jedoch noch nicht vollständig bearbeiteten Aufträgen. Insbesondere bestehen diese Auftragsinformationen aus

- der zum Zeitpunkt der Betrachtung aktuellen Positionen der Frachten,
- der zu den Aufträgen gehörenden Sendungsstruktur[28]
- der vom Kunden angegebenen Fahrzeugkategorie gemäß Tabelle 7.4, S. 284.

[27] Nach dem 10.12.2008 um 8:00 Uhr konnten in den Vergangenheitsdaten nicht mehr alle Aufträge bis zu ihrer Auslieferung nachvollzogen werden, da sie teils ins Jahr 2009 fielen, zu dem für diese Arbeit keine Daten vorliegen.

[28] sofern extrahierbar, siehe zur Datenqualität auch Abschnitt 2.4.1, S. 57 ff.

Im betrachteten Zeitraum sind insgesamt 208 286 Aufträge vollständig bearbeitet worden, d. h., ihre zugehörige Startbeladung, als auch die zugehörige Endentladung fiel in den betrachteten Zeitraum. Die zeitliche Verteilung aller im Jahr 2008 erfassten Aufträge lässt sich anhand Abbildung 7.11, S. 288, quartalsweise abschätzen.[29]

7.4.1 Inhaltliche Abgrenzung der Benchmarks

Nachfolgend sind wichtige Abgrenzungen der Benchmarks aufgelistet. Sie dienen vor allem der methodischen Transparenz und dem besseren Verständnis. Vor ihrem Hintergrund sind die Benchmark-Ergebnisse in den Abschnitten 7.4.4–7.4.6 zu verstehen.

- Die Benchmarks sind keine Simulation. Sie liefern Abschätzungen über das im Vergleich zu den real durchgeführten Transporten zusätzliche Einsparungspotential durch Konsolidierung der Aufträge in den Vergangenheitsdaten. Dabei treffen sie jedoch keine Dispositionsentscheidungen. Insbesondere können aufgrund der Datenlage nur schwer Aussagen über die konkrete Verfügbarkeit von Fahrzeugen zu einem bestimmten Zeitpunkt in der Vergangenheit getroffen werden[30].
- Die Benchmarks berücksichtigen keine Preise für Fähren oder Maut. Sie verwenden zwar Wasserwege und mautpflichtige Straßen, veranschlagen hierfür jedoch keine Extrakosten. Ausgleichend wurde dies bei den Bewertungen der Original-Strecken ebenfalls nicht getan.
- Die Benchmarks berücksichtigen keine unvorhergesehenen Ereignisse wie Staus, Fahrzeugausfälle oder nachträgliche Änderungen an den Auftragsdetails.
- Aufgrund der Lücken in der Beschreibung der zugehörigen Frachten kann häufig nicht genau entschieden werden, ob eine Fahrzeugkategorie ausreichend Kapazität für die Sendungen zweier zu konsolidierender Aufträge besitzt. Allerdings ist zu fast jedem Auftrag in den Vergangenheitsdaten hinterlegt, welche Fahrzeugkategorie gemäß Tabelle 7.4, S. 284, vom Kunden gewünscht wurde.[31] Daher wird innerhalb der Benchmarks bei einer Konsolidierung sichergestellt, dass eine Fahrzeugkategorie gewählt wird, die *mindestens* der von den Kunden gewünschten entspricht. Die Reihenfolge der in Tabelle 7.4, S. 284, genannten Kategorien ist hier in aufsteigender Größe zu verstehen.
- Die Lücken in den Frachtbeschreibungen sind auch der Grund dafür, dass in den nachfolgenden Benchmarks die Möglichkeit von *Teilladungen*, wie sie in Abschnitt 7.1.3, S. 273 ff., formuliert wurden, nicht näher untersucht wird. Die Qualität der Angaben zur Menge der zu einem Auftrag gehörigen Frachtstücke sowie zu ihrer genauen Beschaffenheit reicht nicht aus, um hier mithilfe eines Benchmarks belastbare Ergebnisse zu ermitteln.
- In den Original-Strecken existieren mitunter Wege, die Zyklen enthalten, d. h. denselben Ort zweimal anfahren, ohne dass zwischendurch eine dokumentierte Be-

[29] Obwohl der Zeitraum, der von den *Benchmarks* abgedeckt wird, das Jahr 2008 nicht vollständig umfasst, liegen dennoch die Auftrags-*Eingänge* für das gesamte Jahr vor.

[30] siehe hierzu auch S. 277, Stichwort „Subjektive Aspekte bei der Disposition“

[31] Lediglich bei 105 Aufträgen, entsprechend 0,05 % der betrachteten 208 286 Aufträge, fehlte diese Information.

oder Entladung vorgenommen wurde. Diese Strecken lassen sich durch oben genannte unvorhergesehene Ereignisse erklären. Ein Szenario, das zu Zyklen führte, entstand beispielsweise, wenn eine Sendung versehentlich unvollständig abgeholt worden war und dieser Umstand erst später auffiel. Der Fahrer kehrte zurück zum Absender und fuhr somit den Zyklus. In den hier verwendeten Benchmarks kommen Zyklen bei 2 258 von 208 286 Aufträgen, entsprechend 1,08 % der Fälle vor.

Solche Zyklen würden bei den Benchmarks zu vorteilhaften Ergebnissen zugunsten der Konsolidierungen führen. Um diesem Effekt entgegenzuwirken werden sämtliche Zyklen beinhaltenden Routen bei den Original-Strecken ignoriert, so dass hierfür keine „Umweg-Kosten“ in der Kostenrechnung entstehen.

7.4.2 Vorab-Vergleich der gefahrenen Routen mit Direktfahrten

Direktfahrten als Vergleich

Um die Ergebnisse der Benchmarks in einem geeigneten Kontext untersuchen zu können, wurden für sämtliche der 208 286 oben genannten Aufträge zunächst die Kosten ausgerechnet, die bei einer *Direktfahrt* entstanden wären. Hierzu wurde diejenige Fahrzeugkategorie zugrunde gelegt, die im Frachtbrief vermerkt war. Sämtliche Entfernungsmessungen erfolgten anhand der in Kapitel 3 beschriebenen Europa-Karte.

In den Vergangenheitsdaten sind darüber hinaus einzelne Fahrzeugpositionen der real gefahrenen Routen gespeichert. In der Annahme, dass zwischen diesen Positionen stets die für das zugehörige Fahrzeug schnellsten Wege gewählt wurde, lassen sich stückweise die wahrscheinlichsten und somit „pseudo-realen“ Wege rekonstruieren. Für manche Routen gelang dies zwar aufgrund fehlender bzw. falscher Daten nicht. Hierzu gehören auch Routen, zu deren Start- oder Zielort sich kein Knoten auf der Karte innerhalb eines Radius von weniger als zehn Kilometern ermitteln ließ. Dieser Fall kam jedoch lediglich bei zehn Aufträgen vor.

Verteilung der Kosten bei Konsolidierungen

Darauf aufbauend können „Pseudo-Realkosten“ für sämtliche Aufträge berechnet werden: Für jede Teil-Strecke ergeben sich Kosten aus dem Tarif der verwendeten Fahrzeugkategorie. Wurden mehrere Frachten gleichzeitig transportiert – wurde also konsolidiert –, dann geschah die Kostenberechnung dieser Teilstrecke anteilig für alle beteiligten Aufträge. Auf diese Weise können die „Pseudo-Realkosten“ (PRK) eines Auftrags geringer sein als die Kosten der zugehörigen theoretischen Direktfahrt.

Einen vergleichenden Überblick dieser Vorab-Messung liefert Tabelle 7.5, S. 291. Sie fokussiert im unteren Abschnitt auf Aufträge, für die sowohl die Wegberechnung der „pseudo-realen Route“ als auch der direkten Route erfolgreich verlief.

Hervorzuheben ist hier der Anteil von Fahrten, die größere Kosten verursachten, als es für die Direktfahrt ermittelt wurde. Dies kann aus mehreren Gründen geschehen. Einer davon ist mangelnde Verfügbarkeit von Fahrzeugen. Denn wenn nur Fahrzeuge zur Verfügung stehen, die einen teureren Tarif verwenden als es für den Transport eines Auftrags nötig wäre, entstehen die beobachteten Mehrkosten. Ein Beispiel ist der Transport einer Sendung, die für einen Pkw geeignet wäre, durch einen Lkw, da kein Pkw in der Nähe des Abholungsortes bereit stand.

	Anzahl	Anteil[I]	Strecke[II]	Anteil	Kosten[III]	Anteil
Aufträge insgesamt	208 286	100 %				
DF ermittelbar	208 276	~100 %				
„PRK“ ermittelbar[IV]	205 016	98 %	83,9	100 %	27,6	100 %
günstiger als DF ausgeführt	75 976	37 %[V]	37,6	45 %	9,6	35 %
teurer als DF ausgeführt	56 005	27 %[V]	28,2	34 %	10,8	39 %

[I] an 2008 durchgeführten Aufträgen (gerundet)
[II] in Millionen Kilometer
[III] in Millionen Euro
[IV] Nicht ermittelbar waren die Kosten für Aufträge, deren „pseudo-realen“ Wege sich nicht rekonstruieren ließen.
[V] bezogen auf die 205 016 Aufträge, zu denen sowohl Direktrouten als auch „pseudo-reale“ Kosten ermittelt werden konnte

Tabelle 7.5: Vergleich „pseudo-realer“ Routen und theoretischer Direktfahrten in 2008
„DF“ steht für *Direktfahrt*
„PRK“ steht für *„Pseudo-Realkosten“*

Ein weiterer Grund für Mehrkosten liegt bei der Disposition. Mitunter wurden bei IN tIME Frachten vorsorglich zu Filialen transportiert, weil eine Konsolidierung mit anderen Sendungen möglich erschien. Fand die Konsolidierung dann jedoch nicht statt, erzeugte der Gesamttransport dieser Frachten einen Umweg und damit Mehrkosten.

Diejenigen Aufträge, die gemäß der Vergangenheitsdaten kostengünstiger transportiert wurden als eine Direktfahrt, machen anteilig 37 % der Aufträge, jedoch 45 % der Weglänge aus: Sie sind damit also überdurchschnittlich lang. Dies legt nahe, dass tendenziell Langstrecken bei der bisherigen Disposition für Konsolidierungen in Betracht gezogen wurden. Gründe hierfür sind vermutlich das größere Einsparungspotential als bei Kurzstrecken sowie entsprechend großzügiger geplante Zeitfenster.

Doch auch diejenigen Aufträge, die zu höheren Kosten als eine theoretische Direktfahrt ausgeführt wurden, sind überdurchschnittlich lang. Hier liegt im Analogieschluss noch weiteres Potential für die Konsolidierung.

7.4.3 Das rekursive Savingsverfahren

In diesem Abschnitt wird ein neues Verfahren präsentiert, welches das Konsolidierungsproblem heuristisch löst. Die Vorgehensweise ist an das von Clarke u. Wright [1964] beschriebene Savingsverfahren angelehnt: Für sämtliche Paare (a_i, a_j) aus offenen Aufträgen A mit $a_i, a_j \in A$ wird das Kosteneinsparungspotential, d. h. das *Saving*, durch Konsolidierung berechnet. Ist – beispielsweise aufgrund von Zeitfenster-Restriktionen – keine Konsolidierung möglich, wird die Paarung verworfen. Anschließend werden die übrig gebliebenen Paarungen absteigend nach ihren Savingswerten sortiert und in dieser Reihenfolge als Konsolidierung eingeplant. Nach einer geplanten Konsolidierung zweier Aufträge werden alle weiteren Savingswerte der Liste verworfen, die sich auf einen dieser beiden Aufträge beziehen. So wird kein Auftrag mehrfach eingeplant. Durch rekursive

Aufrufe des Verfahrens können weitere Aufträge der Planung hinzugefügt werden. Für die Berechnung kommt die in Abschnitt 7.3.4 vorgestellte Datenstruktur *Einzelauftrag* zum Einsatz.

Nachfolgend wird beschrieben, wie durch den rekursiven Aufruf des Verfahrens ein Planungsverbund aus wechselseitig abhängigen Aufträgen entstehen kann. Nach einem Beispiel für ein mögliches Ergebnis des Verfahrens folgt die Erläuterung des Pseudocodes und der Pseudocode selbst.

„Lose Enden:“ Geschwister-Einzelaufträge

Die Konsolidierung eines Top-Level-Einzelauftrags a mit einem aus einer Teilung hervorgegangenen Einzelauftrag b führt unweigerlich zu einem „losen Ende“: Der zu b gehörige Vorgänger- bzw. Nachfolger-Einzelauftrag – im Pseudocode ist dieser als `geschwister`-Attribut eines Einzelauftrags modelliert – muss in diesem Fall noch eingeplant werden. Dies ist insbesondere bei der Berechnung der Savings relevant. Für die Bearbeitung des übrigen Geschwister-Einzelauftrags *b.geschwister* stehen zwei Alternativen zur Verfügung:

1. Er wird ohne weitere Konsolidierungen direkt gefahren.
2. Er wird seinerseits möglichst kostengünstig konsolidiert.

Der erste Fall schmälert die Kosteneinsparung der ursprünglichen Konsolidierung. Bei der Berechnung der Savingswerte gehen die Kosten für die Direktfahrt daher negativ mit ein[32].

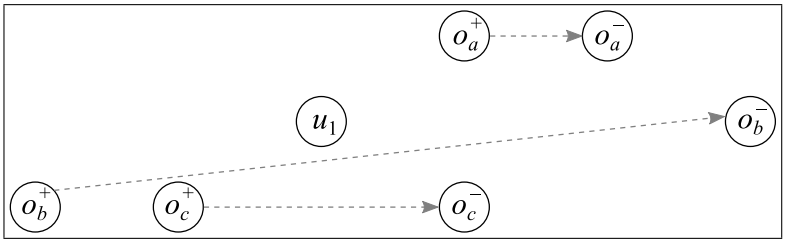
Im zweiten Fall können dagegen sogar noch weitere Kosteneinsparungen erzielt werden, indem rekursiv nach einer Konsolidierungsmöglichkeit für *b.geschwister* gesucht wird. Dabei muss jedoch unbedingt beachtet werden, dass die Zeitfenster von *b.geschwister* aufgrund der ursprünglich geplanten Konsolidierung von a und b möglicherweise eingeschränkt wurden.

Konsolidierungsketten

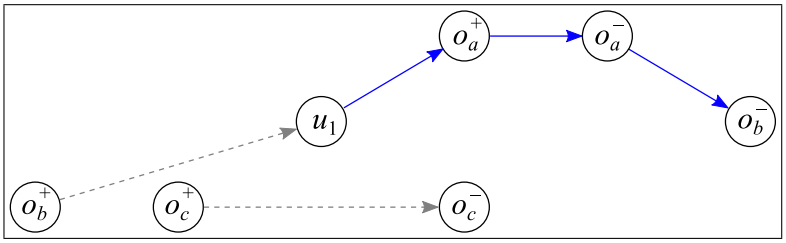
Offensichtlich können sich rekursiv regelrechte „Konsolidierungsketten“ bilden. Die Kettenglieder bestehen aus Konsolidierungen zweier (Teil-)Aufträge und maximal einer Direktfahrt eines Teil-Auftrags. Abbildung 7.12, S. 293, zeigt ein schematisches Beispiel für die Entstehung einer Konsolidierungskette. Gegeben sind hier die drei Aufträge a, b und c sowie ein Umladeort u_1, an dem alle Aufträge potentiell umgeladen werden dürfen. Gesucht ist ein kostengünstigster Tourenplan.

Das rekursive Savingsverfahren beginnt damit, nach einer Konsolidierungsmöglichkeit für a zu suchen. Eine Konsolidierung mit dem Nachfolger-Einzelauftrag des bei u_1 geteilten Einzelauftrags b liefert eine Ersparnis gegenüber einer Direktfahrt von a (Abbildung 7.12b, S. 293). Als „loses Ende“ verbleibt bei dieser Planung der Einzelauftrag von o_b^+ nach u_1. Für diesen ergibt die rekursive Suche nach weiteren Konsolidierungspartnern einen Treffer für den ersten Teil von Auftrag c, wenn dieser ebenfalls bei u_1 umgeladen wird (Abbildung 7.12c, S. 293). Im Beispiel wird das daraus entstehende „lose Ende“ von u_1 nach o_c^- als Direktfahrt eingeplant (Abbildung 7.12d, S. 293).

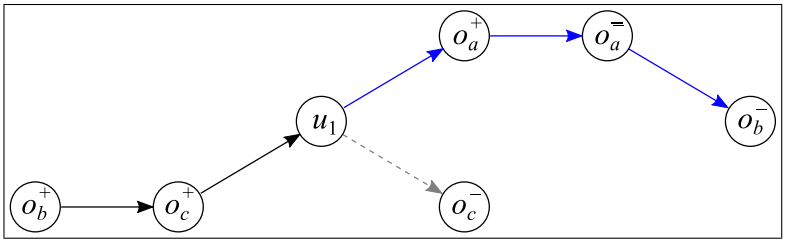
[32] siehe auch Zeile 2 in Verfahren 25, S. 296



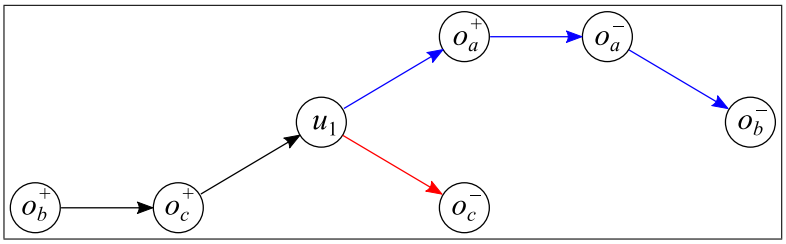
(a) Ausgangslage: Planungsbeginn mit Auftrag a.



(b) Nach erster Konsolidierung: Bei Auftrag b entsteht ein „loses Ende“ für den aus der Teilung bei u_1 hervorgegangenen Einzelauftrag von o_b^+ nach u_1.



(c) Nach zweiter und damit rekursiver Konsolidierung: Bei Auftrag c entsteht ein „loses Ende“ für den aus der Teilung bei u_1 hervorgegangenen Einzelauftrag von u_1 nach o_c^-.



(d) Fertige Konsolidierungskette: Der Einzelauftrag von u_1 nach o_c^- ist als Direktfahrt eingeplant. Dieses Ergebnis ist erreichbar, wenn in Verfahren 24 , S. 295, Parameter `RMAX` ≥ 2 gewählt ist.

Abbildung 7.12: Beispiel für den Aufbau einer Konsolidierungskette
Durchgezogene Pfeile einer Farbe symbolisieren eingeplante Routen.
Grau gestrichelte Pfeile symbolisieren noch zu bearbeitende (Kinder-)Einzelaufträge.
Zeitfenster sind nicht mit angegeben.

Typen der Konsolidierung Bei der Konsolidierung zweier Aufträge können zwei Typen auftreten, die sich hinsichtlich der Reihenfolge der Auftragsbearbeitung unterscheiden. Seien a und b zwei Aufträge. Gemäß der in Abschnitt 7.1 eingeführten Notation sind damit o_a^+ und o_b^+ bzw. o_a^- und o_b^- die zugehörigen Beladungs- bzw. Entladungsorte[33]. O. B. d. A. werde stets a vor b abgeholt. Eine beide Aufträge bearbeitende Route kann somit entweder die Form $\langle \ldots, o_a^+, \ldots, o_b^+, \ldots o_b^-, \ldots, o_a^-, \ldots \rangle$ oder die Form $\langle \ldots, o_a^+, \ldots, o_b^+, \ldots o_a^-, \ldots, o_b^-, \ldots \rangle$ besitzen. Die erste Form wird im Folgenden mit Typ *ABBA*, die zweite mit Typ *ABAB* referenziert.

Der Pseudocode für das rekursive Savingsverfahren besteht aus einem Haupt- und einem Unterprogramm und wird nachfolgend näher erläutert. Das Hauptprogramm (Verfahren 24, S. 295) iteriert über alle Top-Level-Einzelaufträge und sucht per Aufruf des Unterprogramms (Verfahren 25, S. 296) nach dem günstigsten Teil-Tourenplan für jeden dieser Top-Level-Einzelaufträge. Ein Teil-Tourenplan besteht entweder aus einer Direktfahrt oder aus einer Konsolidierungskette.

Verfahren 25 nimmt einen zu konsolidierenden Einzelauftrag b als Parameter entgegen. Es besteht aus drei Teilen:

Teil 1 Zunächst werden die Kosten einer Direktfahrt für b ermittelt und als beste bekannte Lösung tp_{opt} notiert.

Teil 2 Danach wird untersucht, ob sich durch eine Konsolidierung von b mit anderen Top-Level-Einzelaufträgen eine Ersparnis gegenüber tp_{opt} erreichen lässt. Gegebenenfalls wird tp_{opt} überschrieben.

Teil 3 Schließlich wird geprüft, ob sich b mit Kinder-Einzelaufträgen konsolidieren lässt, die durch eine Teilung eines Top-Level-Einzelauftrags an einem der Umladeorte entstanden. Nur in diesem Teil sind „lose Enden“ zu berücksichtigen. Sie werden ihrerseits als Parameter beim rekursiven Aufruf an Verfahren 25 übergeben und so entweder als Direktfahrt oder innerhalb einer Konsolidierungskette in den Tourenplan eingeplant.

Die Kosten eines Teil-Tourenplans *tp* werden im Pseudocode als Attribut `kosten` notiert. Die Einsparungen, die sich durch die Verwendung von *tp* ergeben, sind im Attribut `saving` enthalten. Die Behandlung eines „losen Endes“ in Teil 3 ergibt einen weiteren Teil-Tourenplan. Dieser wird dem erzeugenden Teil-Tourenplan *tp* in Form des Attributs `konsequenz` zugeordnet. Daher lassen sich dessen Anteil an der Gesamtersparnis in Verfahren 25, Zeile 21, mittels des Ausdrucks *tp.konsequenz.saving* referenzieren.

Die Berechnung des Savingswertes eines Teil-Tourenplans geschieht in mehreren Schritten und bedarf einer Erklärung. Im Unterprogramm gehen die Kosten für eine Direktfahrt eines Einzelauftrags *negativ* in den Savingswert ein (Verfahren 25, Zeile 2). Dies ist für die korrekte Berechnung der Savingswerte derjenigen Einzelaufträge notwendig, deren Einplanung *rekursiv* über ein „loses Ende“ erfolgt. Damit der Savingswert für einen nicht-konsolidierten Top-Level-Einzelauftrag b jedoch exakt bei null liegt, schlägt das Hauptprogramm in jedem Fall die Kosten für die Direktfahrt von b wieder auf (Verfahren 24, Zeile 4).

[33] siehe Seite 266

In Verfahren 25 wird in den Zeilen 7 und 14 ein konsolidierender Teil-Tourenplan erzeugt. Dieser meint stets die kostengünstigere Konsolidierung aus *ABBA* und *ABAB*. Darin eingeplant ist außerdem die kostengünstigste Fahrzeugkategorie mit ausreichender Kapazität. Beide Überlegungen fehlen im Pseudocode zugunsten der Lesbarkeit.

Der Parameter $\texttt{RMAX} \in \mathbb{N}$ in Verfahren 25 gibt die maximale Rekursionstiefe bei der Bildung von Konsolidierungsketten an – dies entspricht der maximalen Kettenlänge. Besitzt er den Wert null, findet überhaupt keine Konsolidierung statt; ist er eins, wird ein wie oben beschriebenes, durch eine Konsolidierung entstehendes „loses Ende“ stets per Direktfahrt eingeplant. Offensichtlich erweitert sich der Lösungsraum $\mathcal{L}_{\texttt{RMAX}}$ mit zunehmender Größe von RMAX, so dass für $\texttt{RMAX} < \texttt{RMAX}'$ stets $\mathcal{L}_{\texttt{RMAX}} \subset \mathcal{L}_{\texttt{RMAX}'}$ folgt.

Greedy-Algorithmus

Gleichzeitig bedeutet ein höherer Wert von RMAX nicht zwangsläufig ein kostengünstigeres Ergebnis bei Verfahren 24. Die Heuristik ist ein *Greedy-Algorithmus* und liefert damit in den meisten Fällen nur *lokale* Minima der Zielfunktion. Auf die Problemstellung der Konsolidierung bezogen bedeutet dies konkret, dass die Einplanung der Teilpläne in absteigender Reihenfolge ihrer Savingswerte unter gewissen Umständen ungeschickter als eine alternative Sortierung ist. Dies ist etwa dann der Fall, wenn eine Konsolidierung zweier Aufträge a und b zwar für sich betrachtet die größte Ersparnis ermöglicht, aber durch die Entscheidung, diese Konsolidierung durchzuführen, zwei andere Konsolidierungen von a und b mit je einem *anderen* Auftrag zunichte gemacht werden, die in Summe ein größeres Einsparpotential besäßen.

Verfahren 24 Rekursives Savingsverfahren

Gegeben: Eine Menge offener Top-Level-Einzelaufträge A, die vorab höchstens einmal geteilt wurden, maximale Rekursionstiefe *RMAX*

1: *savingsListe* $\coloneqq \emptyset$ ▸ Initialisierung
2: **for all** $b \in A$ **do**
3: Berechne Teil-Plan *tp* für *b* mit Verfahren 25, S. 296, mit *rek* $\coloneqq 0$, leerer Tabuliste und *RMAX*
4: *tp.saving* $\coloneqq$ *tp.saving* + Direktfahrt(*b*).*kosten* ▸ ergibt 0, falls *b* nicht konsolidiert werden konnte
5: **if** *tp.saving* > 0 **then**
6: Nimm *tp* in *savingsListe* auf
7: **end if**
8: **end for**
9: *teilPlanListe* $\coloneqq \emptyset$ ▸ beinhaltet das Ergebnis
10: Sortiere *savingsListe* absteigend nach Savingswerten
11: **for all** *tp* $\in$ *savingsListe* **do**
12: Nimm *tp* in *teilPlanListe* auf
13: Entferne sämtliche $tp' \in$ *savingsListe* aus *savingsListe*, die Überschneidungen mit den in *tp* enthaltenen (Kinder-)Einzelaufträgen besitzen
14: **end for**
15: **return** *teilPlanListe*

Ergebnis: Ein suboptimaler Tourenplan, der sämtliche Aufträge in A in plausible Routen inklusive der bei der Berechnung zugrunde gelegten Fahrzeugkategorie einteilt.

Verfahren 25 *RMAX* Konsequenzen berücksichtigende Konsolidierung

Gegeben: Eine Menge offener Top-Level-Einzelaufträge A, ein zu konsolidierender Einzelauftrag $b \in A$, Tabuliste an Einzelaufträgen *tabu*, aktuelle Rekursionstiefe *rek*, maximale Rekursionstiefe *RMAX*

Teil 1

$tp_{opt} \coloneqq$ Direktfahrt(b) ▸ d. h. keine Konsolidierung
$tp_{opt}.saving \coloneqq -tp_{opt}.kosten$ ▸ Direktfahrt bewirkt keine Ersparnis, sondern Kosten!
if $rek \geq RMAX$ **then**
  return tp_{opt} ▸ Rekursionsabbruch
end if

Teil 2

for all $a \in A \setminus (tabu \cup b)$ **do**
  Erzeuge Teil-Tourenplan tp, der a und b konsolidiert
  $tp.saving \coloneqq$ Direktfahrt$(a).kosten - tp.kosten$
  if tp ist zulässig **und** $tp.saving > tp_{opt}.saving$ **then**
    $tp_{opt} \coloneqq tp$
  end if

Teil 3

  $kinder_a \coloneqq$ durch Teilung von a an Umladeorten plausible Kinder-Einzelaufträge
  for all $a' \in kinder_a \setminus tabu$ **do**
    Erzeuge Teil-Tourenplan tp', der a' und b konsolidiert
    ▸ $tp'.saving$ berücksichtigt hier noch nicht das „lose Ende“
    $tp'.saving \coloneqq$ Direktfahrt$(a'.parent).kosten - tp'.kosten$
    if $tp'.saving < 0$ **then**
      goto 13 ▸ starke Beschleunigung des Verfahrens
    end if
    $tabuRek \coloneqq tabu \cup$ (Teil-)Aufträge in tp' sowie deren Eltern
    $tp'.konsequenz \coloneqq$ rekursiver Aufruf des Verfahrens mit $a'.geschwister$, $tabuRek$, $rek + 1$ und $RMAX$
    $tp'.saving \coloneqq tp'.saving + tp'.konsequenz.saving$ ▸ $tp'.saving$ nun inkl. „losem Ende“
    if tp' ist zulässig **und** $tp'.saving > tp_{opt}.saving$ **then**
      $tp_{opt} \coloneqq tp'$
    end if
  end for
end for
return tp_{opt}

Ergebnis: Ein plausibler, suboptimaler Teil-Tourenplan, der b nach Möglichkeit mit einem anderen (Teil-)Auftrag konsolidiert und rekursiv bis zu *RMAX* Konsequenzen durch weitere Konsolidierungen abdeckt; eine Direktfahrt von b, falls kein solcher Teil-Tourenplan gefunden werden konnte.

Aufwandsabschätzung

Ebenfalls von `RMAX` abhängig ist der Aufwand des Verfahrens: Bei jeder Prüfung, ob ein Einzelauftrag $a \in A$ mit einem anderen konsolidiert werden kann, kommen grund-

sätzlich alle Einzelaufträge sowie deren Kinder-Einzelaufträge, die nicht zum selben Original-Auftrag wie a gehören, in Frage. Für n Top-Level-Einzelaufträge und insgesamt m Kinder-Einzelaufträge besitzt jeder Aufruf von Verfahren 25, S. 296, damit einen Aufwand in der Größenordnung $\mathcal{O}(m)$. Da Verfahren 25 innerhalb von Verfahren 24, S. 295, einmal pro Top-Level-Einzelauftrag aufgerufen wird, ergibt sich für Verfahren 24 insgesamt eine Aufwandsabschätzung von $\mathcal{O}(n \times m^{\texttt{RMAX}})$.

Heuristische Beschleunigung

Vor diesem Hintergrund versteht sich der in Verfahren 25, Zeile 16, eingefügte Schleifenabbruch: Wenn der ermittelte Savingswert bereits *ohne* Einbezug der Kosten für das „lose Ende“ keine Ersparnis bewirkt, ist es unwahrscheinlich, dass weitere Glieder an der Konsolidierungskette diesen Umstand ändern können. Die Untersuchung wird zugunsten einer schnelleren Ausführung des Verfahrens an dieser Stelle abgebrochen.

Die Messreihen in den nachfolgenden Abschnitten prüfen für die 2 066 Momentaufnahmen des Jahres 2008[34], ob im direkten Vergleich mit der tatsächlichen Disposition noch zusätzliches Konsolidierungspotential vorhanden war. Der nachfolgende Abschnitt 7.4.4 beschreibt die Messergebnisse für die Konsolidierung der Aufträge *ohne Umladungen*; es werden also ausschließlich Top-Level-Einzelaufträge betrachtet. In Abschnitt 7.4.5 wird anschließend untersucht, inwieweit das Erlauben von Teilungen der Einzelaufträge zusätzliche Kostenersparnisse ermöglicht. Dazu wird für jeden Top-Level-Einzelauftrag eine Teilung an allen 21 Niederlassungen von IN tIME in Betracht gezogen. Eine mehrfache Teilung wird für diese Messungen dagegen ausgeschlossen, da dies auch bei IN tIME nur in besonderen Ausnahmefällen geschah. Betrachtet werden die Ergebnisse für $\texttt{RMAX} = 1$ und $\texttt{RMAX} = 2$.

Zuordnung der Zeitpunkte

Bei der Interpretation der Ergebnisse ist zu beachten, dass die Einsparungen, d. h. die Summe der Savingswerte, für eine einzelne Momentaufnahme auf der theoretischen Konsolidierung derjenigen Aufträge beruht, die gemäß der Vergangenheitsdaten zum Zeitpunkt der Momentaufnahme *bekannt* waren. Dies schließt explizit auch Aufträge mit ein, deren Bearbeitungszeitfenster aus Perspektive der jeweiligen Momentaufnahme *in der Zukunft* liegt. Die Zuordnung von Zeitpunkten und Einsparungen bezieht sich somit auf den Zeitpunkt der *Einplanung* und nicht auf den Zeitpunkt der – theoretischen – *Realisierung*. Dies ist insofern konsequent, als dass das in diesem Kapitel vorgestellte Verfahren zur Aufdeckung von Konsolidierungspotential für den Einsatz innerhalb eines Entscheidungsunterstützungssystems ausgerichtet ist. Auch dabei liegt der Fokus auf der geschickten *Planung*.

Weitere Beschränkungen

Verzicht auf Konsolidierung Damit die Ergebnisse der Benchmarks besser auf die Realität übertragbar sind, werden zusätzliche Beschränkungen vorgenommen. Dadurch kommen zwei Aufträge *nicht* für eine gemeinsame Konsolidierung in Frage, wenn

- einer der Aufträge gemäß der Vergangenheitsdaten zum Zeitpunkt der Momentaufnahme bereits in Bearbeitung war,
- einer der Aufträge gemäß der Vergangenheitsdaten ohnehin Teil einer Konsolidierung war; d. h., die Benchmarks entdecken damit *zusätzliches* Konsolidierungspotential,

[34] siehe Seite 288

- beide Aufträge denselben Start- oder Zielort besitzen,[35]
- die konsolidierte Route aufgrund der Zeitfenster eine Wartezeit von mehr als 45 Minuten einkalkulieren muss[36],
- die Planung der Konsolidierung mit Blick auf die Kunden-Zeitfenster einen Zeitpuffer von weniger als zehn Minuten zulässt,
- bei einem der Aufträge keine vom Kunden gewünschte Fahrzeugkategorie in den Vergangenheitsdaten hinterlegt ist,
- bei einem der Aufträge eine vom Kunden gewünschte Fahrzeugkategorie hinterlegt ist, die nicht aus Tabelle 7.4, S. 284, stammt,
- die konsolidierte Fracht die Kapazität jedes der in Tabelle 7.4, S. 284, aufgelisteten Fahrzeuge bezüglich der Nutzlast oder der Palettenstellplätze übersteigt oder
- ein Fehler bei der Routenberechnung auftritt, wobei folgende Fälle als Fehler interpretiert werden:
  - Einem beteiligten Ort aus den Vergangenheitsdaten kann kein Knoten der OSM-Karte mit weniger als zehn Kilometer Luftdistanz zugewiesen werden. Dieser Fehler tritt bei den Aufträgen des Jahres 2008 insgesamt zehnmal auf (siehe Tabelle 7.5, S. 291).
  - In den Vergangenheitsdaten kann kein zusammenhängender Originaltransport einer beteiligten Sendung nachvollzogen werden. Dies geschieht, wenn in den Daten Informationen zu Start- oder Zwischenbeladungen oder zu End- oder Zwischenentladungen fehlen. Für den betrachteten Zeitraum trifft dies auf 3 260 Aufträge zu (siehe Tabelle 7.5, S. 291).

7.4.4 Benchmarks ohne Umladungen

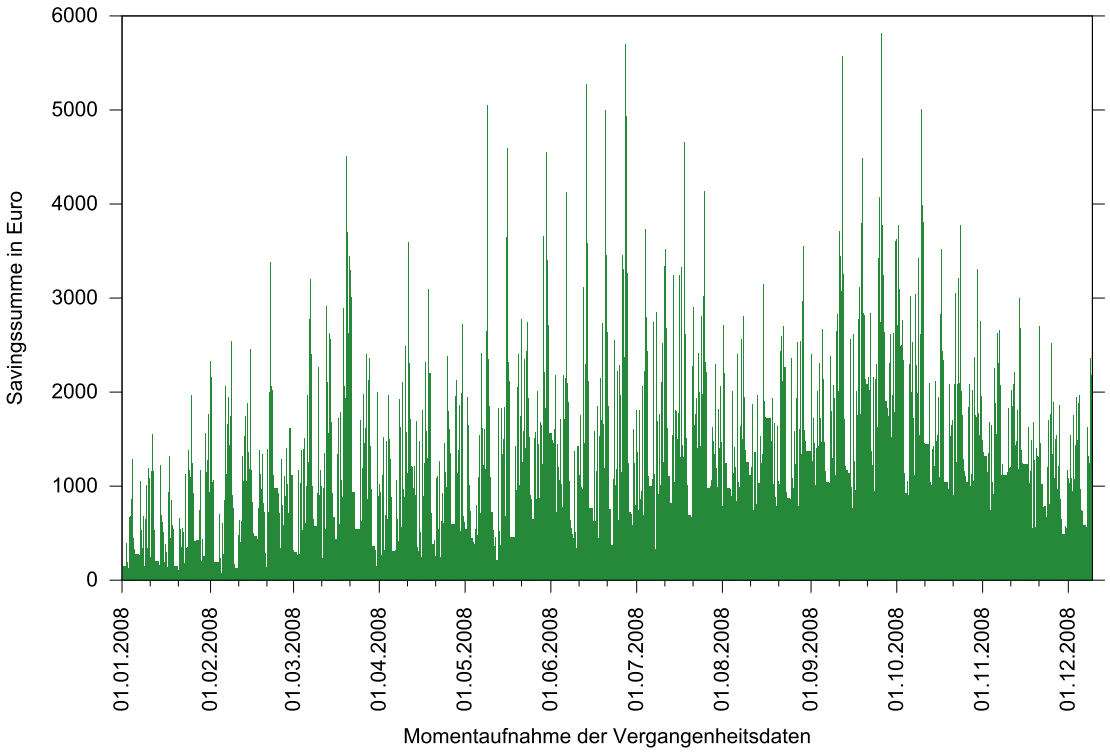
In diesem Abschnitt wird das Einsparungspotential durch Konsolidierung untersucht, wenn keine Umladungen erlaubt sind. Dies entspricht der Durchführung der Benchmarks mit Verfahren 24, S. 295, und Parameter `RMAX` = 1. Eine tiefere Rekursion ist ohnehin nicht möglich, da ohne Teilungen Teil 3 in Verfahren 25, S. 296, keine Auswirkungen hat.

Abbildung 7.13, S. 299, zeigt sämtliche Savingssummen sortiert nach dem Zeitpunkt der zugehörigen Momentaufnahmen. Dazu ergänzend sind in Tabelle 7.6, S. 300, die wesentlichen Ergebnisse für die beiden Benchmarkvarianten B75 und B90[37] und die 2 066 Momentaufnahmen des Jahres 2008 zusammengefasst. Zusätzlich angegeben sind Durchschnittswerte der Summen und Anzahl der Savings nach Wochentagen und nach Uhrzeiten.

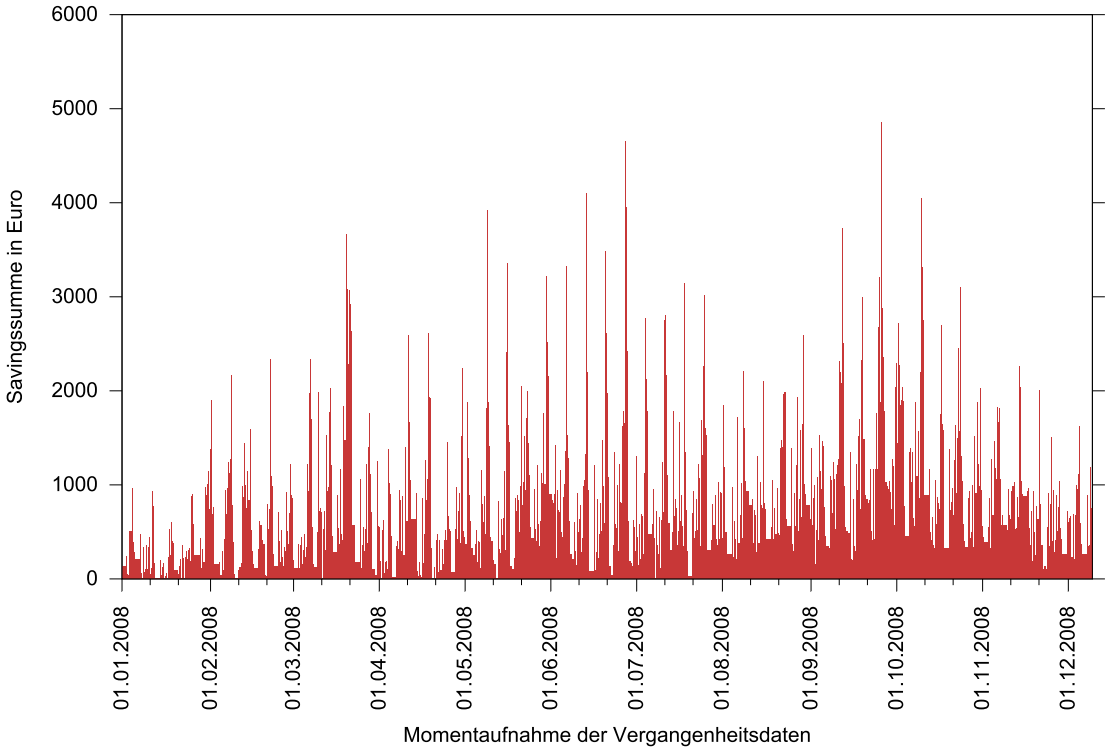
[35] Der Autor geht davon aus, dass in der Realität eine Konsolidierungsmöglichkeit nicht übersehen worden wäre, wenn zwei Aufträge zeitlich überlappend vom oder zum selben Kunden führten. Dass jedoch gemäß der Vergangenheitsdaten beide Aufträge nicht konsolidiert wurden, spricht in diesen Fällen vielmehr dafür, dass die zugehörigen Frachten – trotz möglicherweise andersartiger Kennzeichnung in den Vergangenheitsdaten – nicht konsolidierbar waren.

[36] Dieser Wert wurde aufgrund der Praxiserfahrungen des Autors gewählt.

[37] siehe Seite 282, Stichwort „Dauern von Ladevorgängen“



(a) Ergebnisse für B75



(b) Ergebnisse für B90

Abbildung 7.13: Ergebnisse der Benchmarks ohne Umladungen

	B75		B90	
	Summe	Savings	Summe	Savings
Minimum	35	1	0	0
Maximum	5 820	59	4 851	37
Durchschnitt	1 264	19	678	8
Median	1 086	18	492	7
Durchschnitt montags	1 034	17	466	6
Durchschnitt dienstags	1 018	17	447	7
Durchschnitt mittwochs	1 137	18	578	7
Durchschnitt donnerstags	1 440	21	758	9
Durchschnitt freitags	1 994	24	1 340	12
Durchschnitt samstags	1 422	20	877	9
Durchschnitt sonntags	814	15	337	5
Durchschnitt 0 Uhr	1 067	17	560	7
Durchschnitt 4 Uhr	1 032	16	543	6
Durchschnitt 8 Uhr	971	15	526	6
Durchschnitt 12 Uhr	1 489	22	786	9
Durchschnitt 16 Uhr	1 744	24	974	11
Durchschnitt 20 Uhr	1 284	19	679	8

Tabelle 7.6: Ergebnisse der Benchmarks ohne Umladungen
(alle Summen in €; alle Werte gerundet)

Deutlich zeigt sich das größte Potential an Freitagen: Hier waren gemäß der Vergangenheitsdaten durchschnittlich die meisten Aufträge im System. Häufig hatten diese zudem einen längeren Planungshorizont, etwa ins Wochenende hinein oder sogar bis zum nächsten Wochenbeginn. Entsprechend groß waren ihre Zeitfenster.

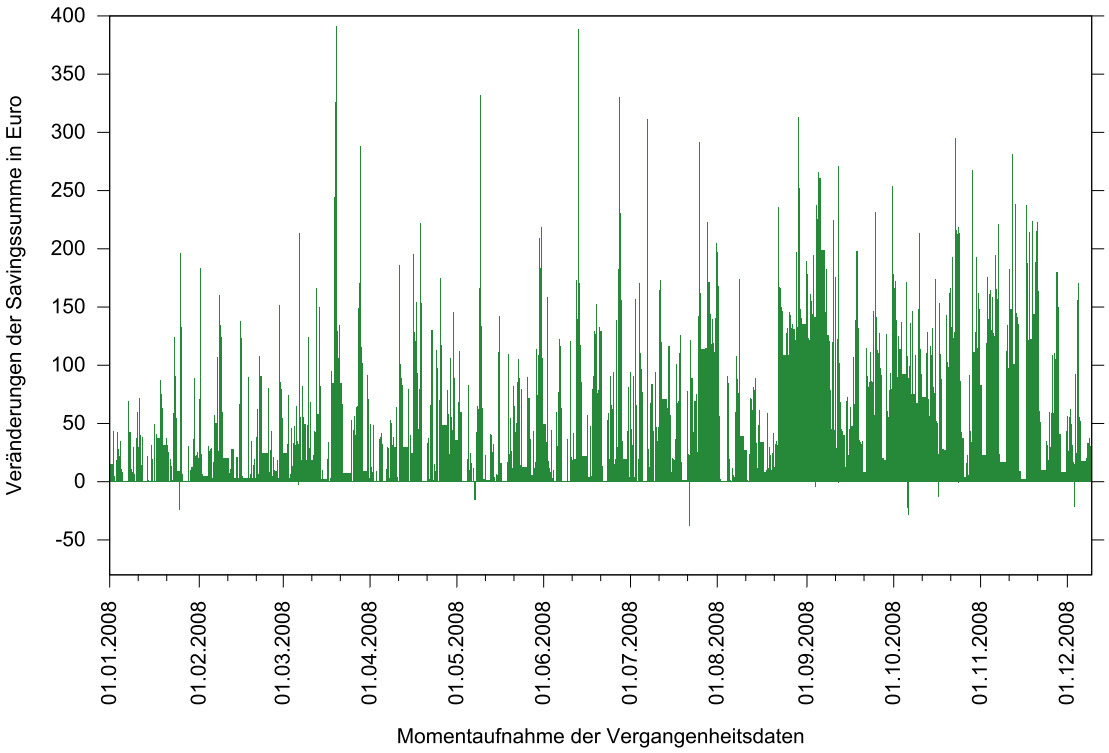
Ebenfalls auffällig ist die Verteilung bei den Uhrzeiten der Momentaufnahmen. In beiden Benchmarkvarianten konnten durchschnittlich die größten Einsparungen am frühen Nachmittag erzielt werden. Danach erfolgte durchschnittlich eine Abnahme der Einsparpotentiale bis inklusive der 8-Uhr-Momentaufnahmen. Zu den 12-Uhr-Benchmarks erfolgte der deutlichste Anstieg. Dies hat vermutlich mit der Struktur der Auftraggeber bei IN tIME zu tun: Die meisten Kunden erkennen Bedarf an Sonderfahrten prinzipbedingt erst im Verlauf des Tages, nachdem das Tagesgeschäft eine Weile angelaufen ist.

Für die optimistischeren Annahmen bezüglich der Dauern von Ladevorgängen bei B75 bleibt festzuhalten, dass für sämtliche Momentaufnahmen – zumindest theoretisch – eine Kosteneinsparung per Konsolidierung ermittelbar war. Bei B90 konnte dagegen für 33 Momentaufnahmen, entsprechend etwa 1,6 %, keine Verbesserung gefunden werden.

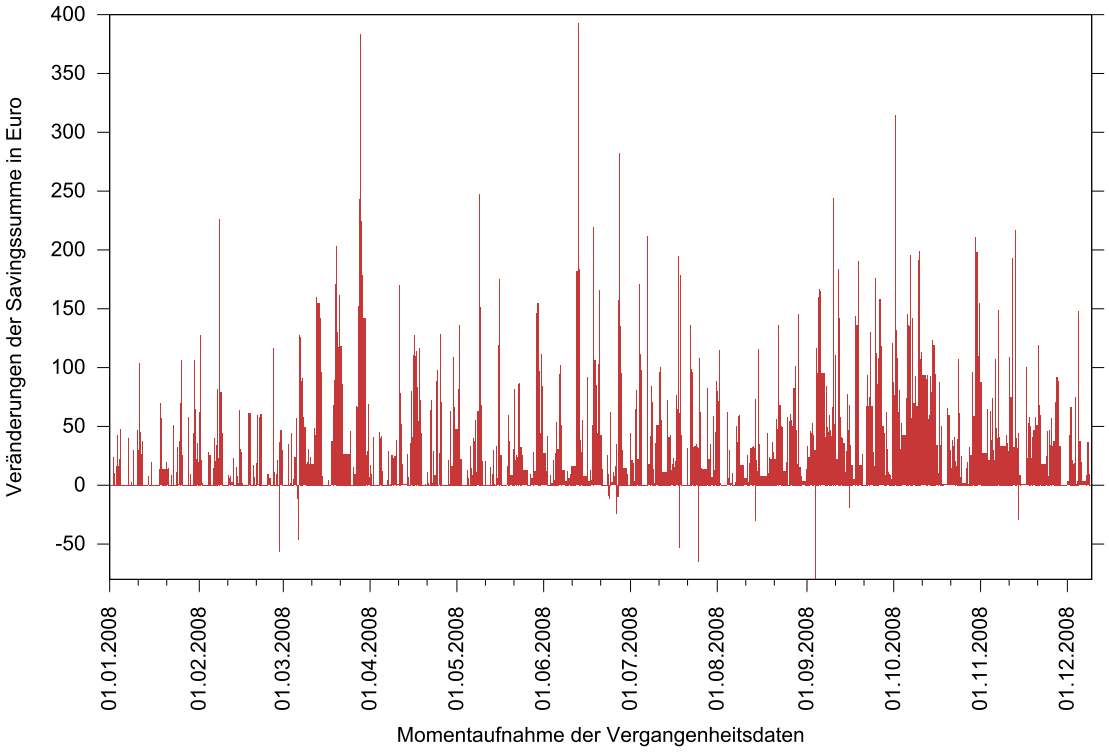
7.4.5 Benchmarks mit Umladungen

In diesem Abschnitt werden die Benchmarkergebnisse für die 2 066 Momentaufnahmen präsentiert, wenn jeder Top-Level-Einzelauftrag an jedem zur Verfügung stehenden Umladeort maximal einmal geteilt wird. Zunächst werden die Ergebnisse für Verfahren 24, S. 295, mit Parameter `RMAX` = 1 ausgeführt; anschließend für `RMAX` = 2. Bei letzterem ergibt sich pro Konsolidierung maximal eine weitere Konsolidierung eines „losen Endes“ sowie eine Direktfahrt. Damit ist ein (Teil-)Tourenplan wie in Abbildung 7.12, S. 293, möglich.

Ergebnisse für `RMAX` = 1 Abbildung 7.14, S. 302, zeigt die Auswertung der Benchmarks für erlaubte Umladungen und Parameter `RMAX` = 1. Für einen geeigneteren Vergleich mit den Ergebnissen des vorangegangenen Abschnitts (siehe Abbildung 7.13, S. 299, Konsolidierungen *ohne* Umladungen) sind anstelle der absoluten Werte die *Veränderungen* der Savingswerte dargestellt.



(a) Ergebnisse für B75



(b) Ergebnisse für B90

Abbildung 7.14: Ergebnisse der Benchmarks mit Umladungen und `RMAX` = 1

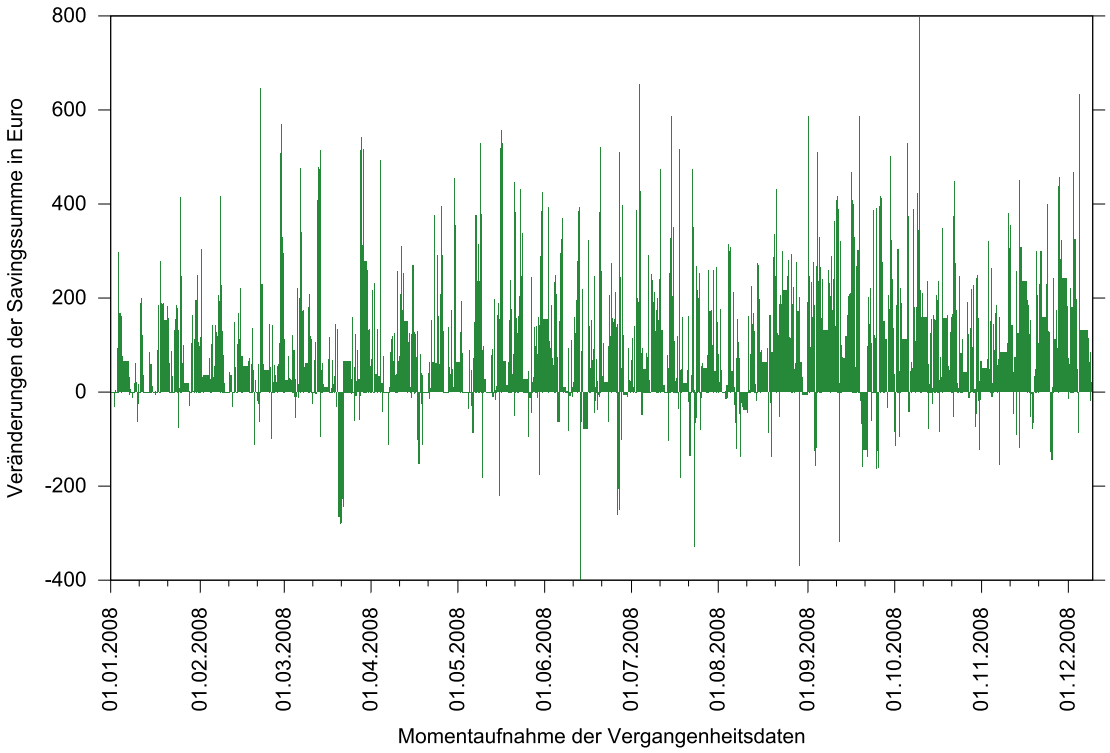
Für B75 kann durch den Einbezug von Umladungen fast durchgängig weiteres Einsparungspotential aufgedeckt werden. Lediglich bei 14 Momentaufnahmen wurden ungünstigere Tourenpläne berechnet; 1 852 Tourenpläne waren günstiger und für die restlichen 200 ergab sich auf der Kostenseite kein Unterschied. Durchschnittlich erzeugten die Tourenpläne um 57€ geringere Kosten als bei den Benchmarks ohne Umladungen.

Die Konfiguration von B90 erlaubt grundsätzlich geringere Einsparungen. Dies macht sich auch bei erlaubten Umladungen bemerkbar: Die durchschnittliche zusätzliche Einsparung lag bei 34€. Für 1 615 Momentaufnahmen des Benchmarks konnten günstigere Lösungen gefunden werden, 433 entfielen mit gleichen Kosten und 18 Tourenpläne erzeugten höhere Kosten als bei den Benchmarks ohne Umladungen.

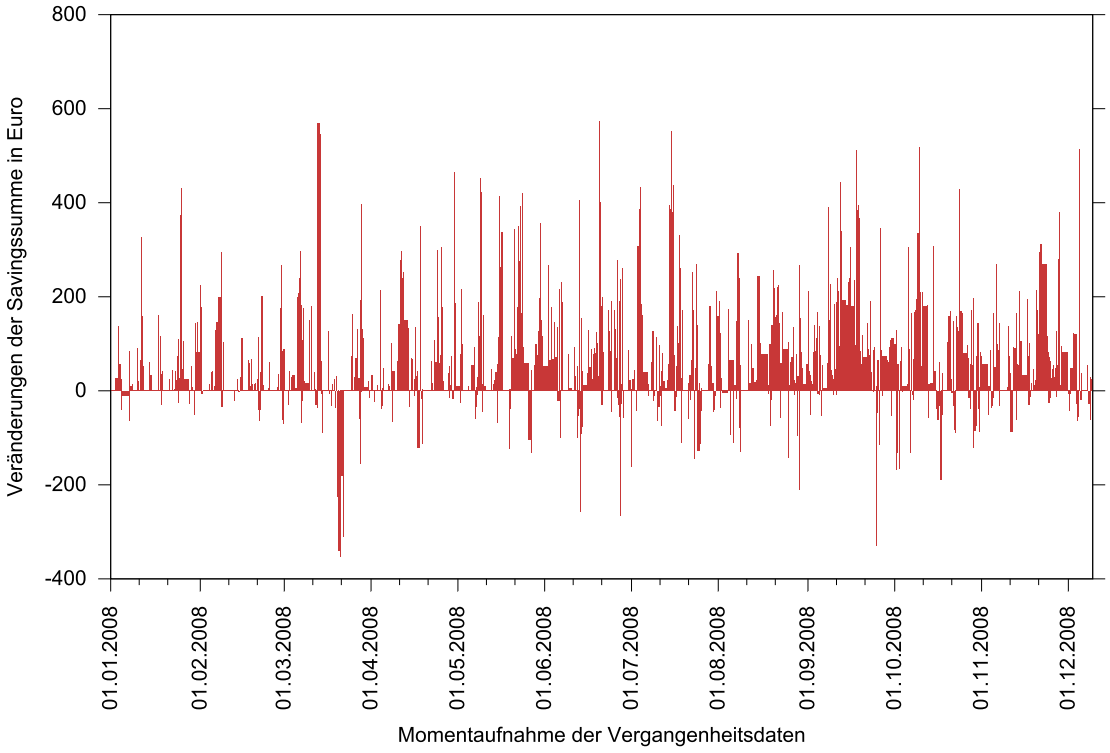
Ergebnisse für `RMAX` = 2 In Abbildung 7.15, S. 304, sind die Ergebnisse von Verfahren 24, S. 295, mit Parameter `RMAX` = 2 dargestellt. Wiederum sind die Veränderungen im Vergleich zu den Benchmarks *ohne* Umladungen angegeben, denn ein Vergleich mit den Ergebnissen zu `RMAX` = 1 unterläge zu vielen Schwankungen, um leicht interpretierbar zu sein.

Auffällig sind die stärkeren Schwankungen bei den Kosten als bei der Messung zu `RMAX` = 1. Zwar können bei B75 im Mittel gegenüber den Messungen *ohne* Umladungen zusätzliche 94€ eingespart werden. Doch bei 277 Tourenplänen entstehen höhere Kosten. Offenbar greift an dieser Stelle – zumindest teilweise – die Überlegung bezüglich des Greedy-Charakters von Verfahren 24 auf Seite 295[38]. In 162 Fällen blieben die Kosten unverändert und zu 1 627 Momentaufnahmen konnten günstigere Lösungen gefunden werden.

[38] Stichwort: „Greedy-Algorithmus“



(a) Ergebnisse für B75



(b) Ergebnisse für B90

Abbildung 7.15: Ergebnisse der Benchmarks mit Umladungen und `RMAX` = 2

Für B90 ergibt sich beim Schritt von `RMAX` = 1 auf `RMAX` = 2 ein ähnliches Bild: Der Benchmark liefert hier 305 ungünstigere, 471 kostenneutrale und 1 290 kostengünstigere Tourenpläne. Im Durchschnitt bedeutet dies eine zusätzliche Kosteneinsparung von 53€. Eine zusammenfassende Übersicht der hier genannten Ergebnisse liefert Tabelle 7.7.

		B75		B90	
		Wert	Anteil	Wert	Anteil
	Durchschnitt Savingssumme	1 321€		712€	
	Median Savingssumme	1 141€		526€	
	Maximum Δ	391€		393€	
	Minimum Δ	−38€		−80€	
`RMAX` = 1	Durchschnitt Δ	57€		34€	
	Median Δ	34€		18€	
	Häufigkeit $\Delta > 0$	1 852	89,6 %	1 615	78,2 %
	Häufigkeit $\Delta = 0$	200	9,7 %	433	21,0 %
	Häufigkeit $\Delta < 0$	14	0,7 %	18	0,8 %
	Durchschnitt Savingssumme	1 416€		766€	
	Median Savingssumme	1 232€		591€	
	Maximum Δ	797€		573€	
	Minimum Δ	−399€		−352€	
`RMAX` = 2	Durchschnitt Δ	94€		53€	
	Median Δ	65€		21€	
	Häufigkeit $\Delta > 0$	1 627	78,8 %	1 290	62,4 %
	Häufigkeit $\Delta = 0$	162	7,8 %	471	22,8 %
	Häufigkeit $\Delta < 0$	277	13,4 %	305	14,8 %

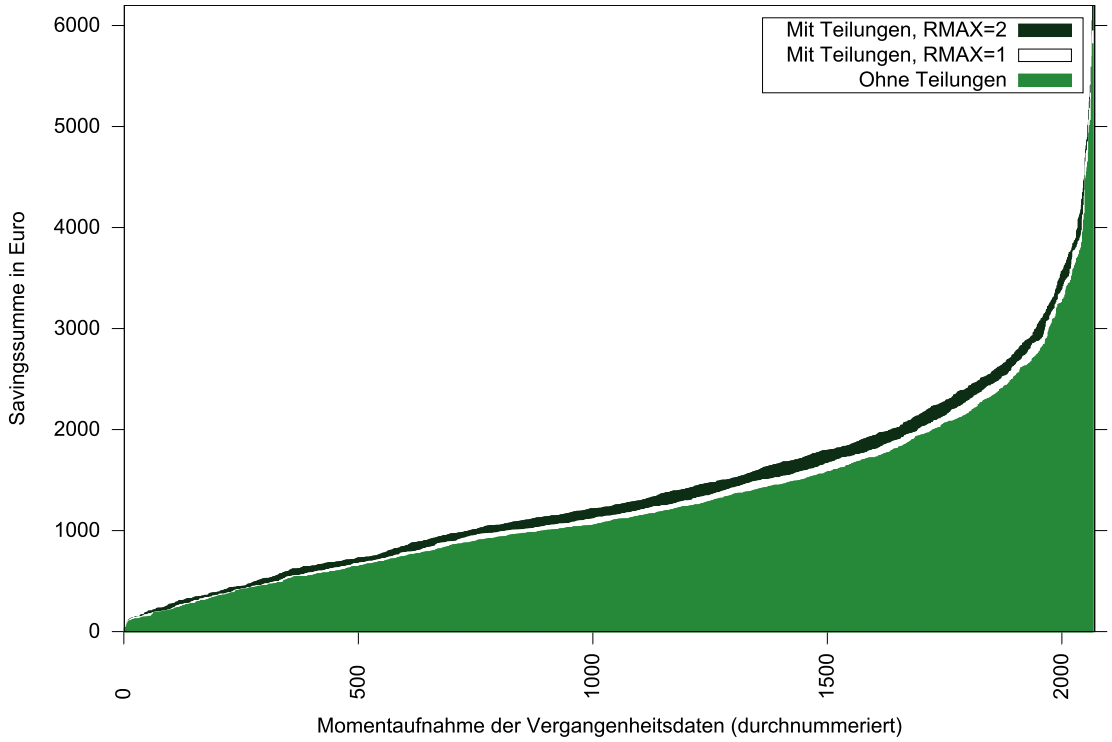
Tabelle 7.7: Ergebnisse der Benchmarks mit Umladungen
Δ-Werte verstehen sich als Differenz der Savingssummen zu denen der Benchmarks *ohne* Umladungen (siehe Tabelle 7.6, S. 300)
(alle Werte gerundet)

Beim Vergleich mit dem Ergebnis des Benchmarks *ohne* Umladungen[39] lässt sich feststellen:

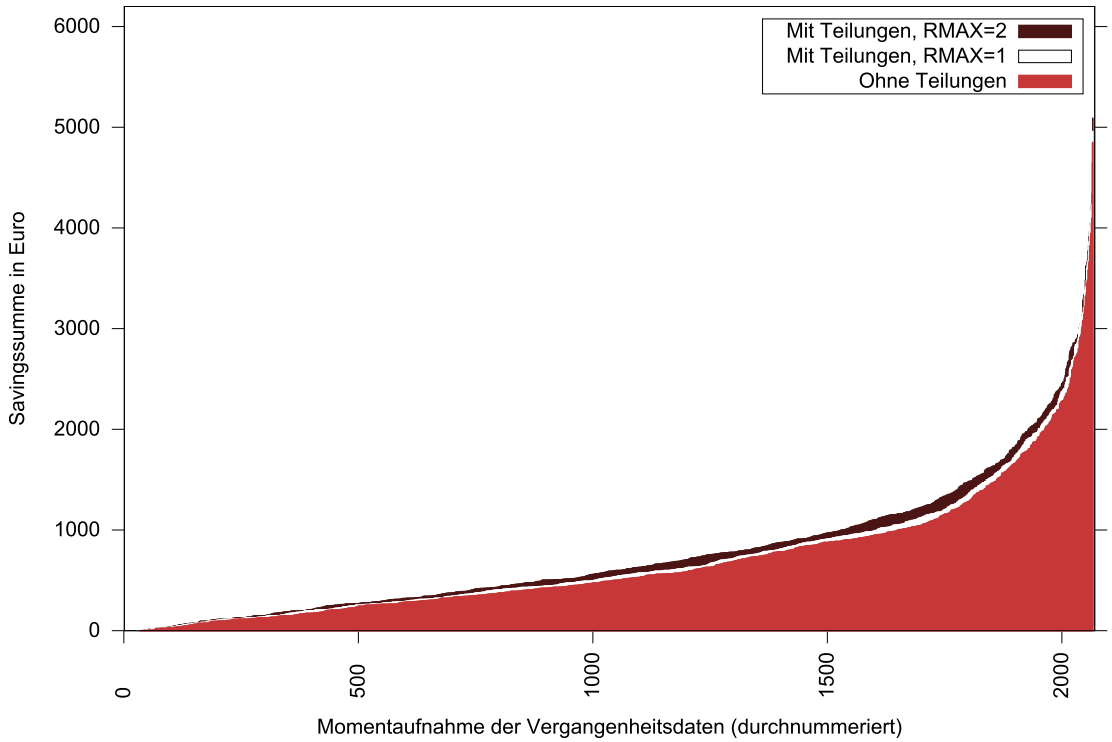
- Für `RMAX` = 1 erhöhen sich die durchschnittlichen Savingssummen erhöhen sich um etwa 5 %.
- Für `RMAX` = 2 ist sogar eine Steigerung um etwa 12 % zu messen.

Ergänzend dazu zeigt Abbildung 7.16, S. 306, die Ergebnisse von B75 und B90 sortiert nach Savingssummen. Der grundsätzlich positive Effekt von `RMAX` = 2 gegenüber von `RMAX` = 1 lässt sich gut erkennen, doch geht dieser einher mit einer höheren Laufzeit[40].

[39] siehe Tabelle 7.6, S. 300
[40] siehe Seite 296, Stichwort „Aufwandsabschätzung“



(a) Ergebnisse für B75



(b) Ergebnisse für B90

Abbildung 7.16: Ergebnisse der Benchmarks sortiert nach Savingssummen

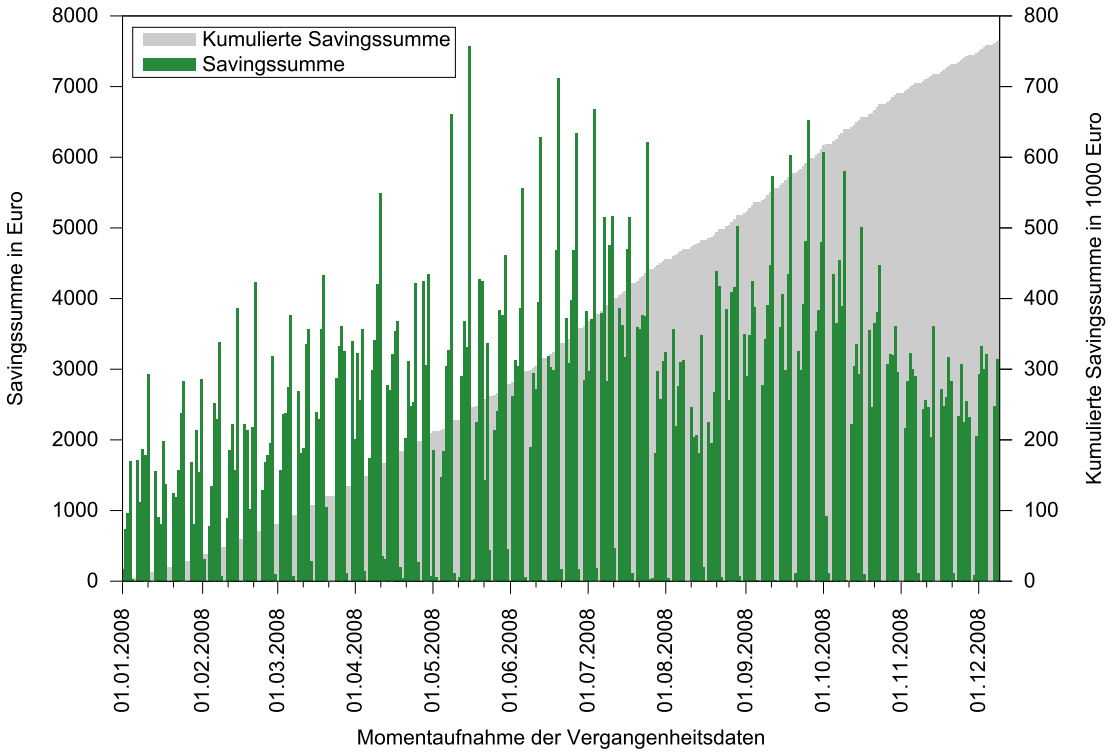
Der Verlauf der Kurven für B75 und B90 ist grundsätzlich ähnlich und zeigt einen markanten Anstieg bei den Momentaufnahmen mit den größten Savingssummen. Dies spiegelt sich ebenfalls beim Vergleich der *durchschnittlichen* Savingssumme und deren *Median* in Tabelle 7.7, S. 305, wider. Im Hinblick auf den ansonsten annähernd linearen Verlauf könnten diese Ausreißer ein Hinweis auf fehlerhafte Daten sein. Es ist denkbar, dass hier beispielsweise für gleich mehrere konsolidierte Aufträge irrtümlich besonders große Zeitfenster oder zu wenig Fracht angegeben waren, so dass sich große Kosteneinsparungen berechnen ließen.

Die in Tabelle 7.7, S. 305, angegebenen Einsparungen sind im richtigen Kontext zu interpretieren: Dass die durchschnittliche Savingssumme der B75-Benchmarks mit Parameter `RMAX` $= 2$ dort mit 1 416€ angegeben ist und die Momentaufnahmen jeweils vier Stunden voneinander entfernt liegen, bedeutet keineswegs, dass durchschnittlich täglich $6 \times 1\,416€ = 8\,496€$ einzusparen sind. Denn es kommt bei der Untersuchung durchaus vor, dass in zwei aufeinander folgenden Momentaufnahmen *dieselben* Aufträge zur Konsolidierung herangezogen werden, sei es innerhalb derselben oder auch in abweichenden Kombinationen. Die angegeben 1 416€ sind daher besser als untere Schranke für das *tägliche* Einsparungspotential im Rahmen des hier verwendeten Modells zu interpretieren. Die Schranke ist noch verbesserbar, wie im nachfolgenden Abschnitt gezeigt wird.

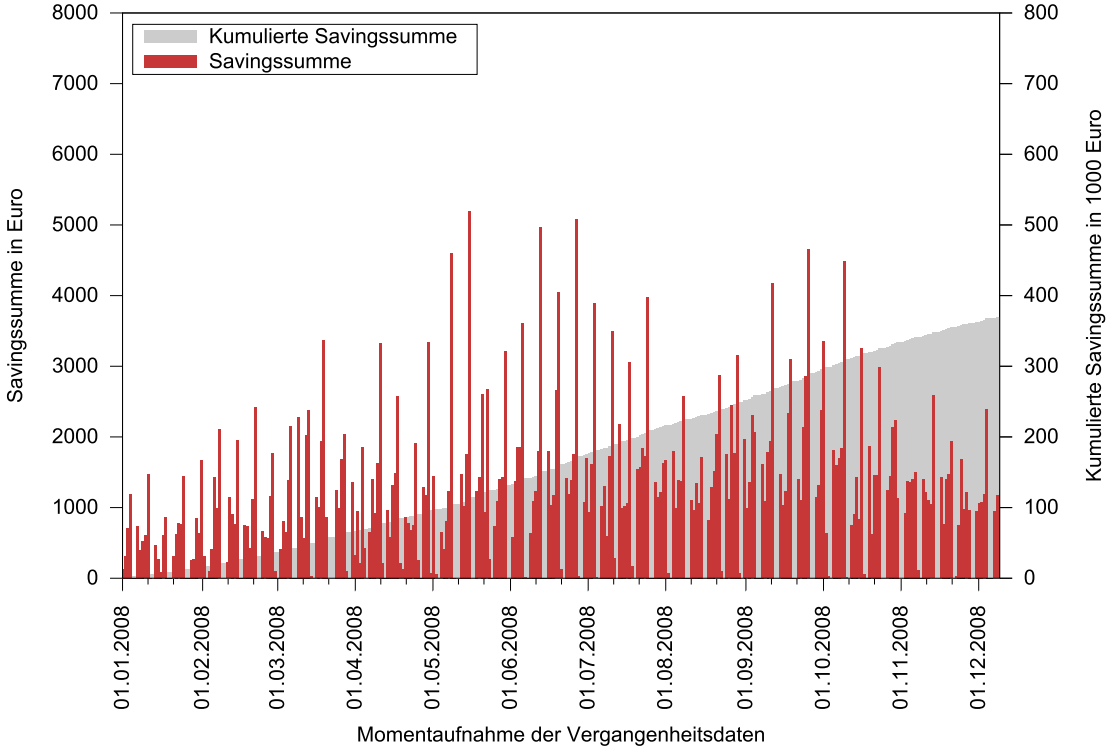
7.4.6 Benchmarks mit Umladungen und Tabulisten

Um zu verhindern, dass in den Benchmarks dieselben Aufträge mehrfach für eine Konsolidierung verwendet werden, eignet sich der Einsatz einer Tabuliste. Hier werden alle Aufträge gespeichert, die Teil einer geplanten Konsolidierung für eine Momentaufnahme sind. Für alle nachfolgenden Momentaufnahmen stehen diese Aufträge dann nicht mehr zur Verfügung. Auf diese Weise lassen sich die erreichten Savingssummen frei von Dopplungen tageweise zusammenfassen.

Abbildung 7.17, S. 308, zeigt die Ergebnisse der Messungen mit Tabulisten. Dargestellt sind die 345 tageweise erreichbaren Savingssummen für `RMAX` $= 2$ sowie die Kumulation dieser Einsparungen über den gesamten Benchmarkzeitraum jeweils für B75 und B90.



(a) Ergebnisse für B75



(b) Ergebnisse für B90

Abbildung 7.17: Ergebnisse der Benchmarks mit Umladungen, `RMAX` = 2 und Tabuliste

Erneut fällt die Auswirkung der zugrunde gelegten Dauern für Be- und Entladevorgänge deutlich auf. Während der B90-Benchmark als untere Schranke für die insgesamt einzusparenden Kosten einen Wert von 369 739€ liefert, kommt die B75-Variante mit 764 919€ auf mehr als doppelt so hohe Einsparungen[41]. Dies ist insofern nicht verwunderlich, als dass ähnliche relative Unterschiede auch in den Messungen der vorangegangenen Abschnitte auftraten (siehe Tabelle 7.6, S. 300, sowie Tabelle 7.7, S. 305).

Ebenfalls auffällig ist in dieser Messung die starke Ungleichverteilung über die Wochentage. Während das Konsolidierungspotential innerhalb der Benchmarks mit Tabulisten gemessen am Median innerhalb der Woche ansteigt, ist – im Vergleich mit den Vergangenheitsdaten – für Wochenenden beinahe keine zusätzliche Ersparnis zu erzielen. Abbildung 7.18 zeigt die Auswertung der nach Wochentagen zusammengefassten Ergebnisse der beiden Benchmarkvarianten als Boxplot-Diagramm. Box und Antennen[42] umfassen hier 90 % aller Werte, die übrigen 10 % sind als Ausreißer dargestellt.

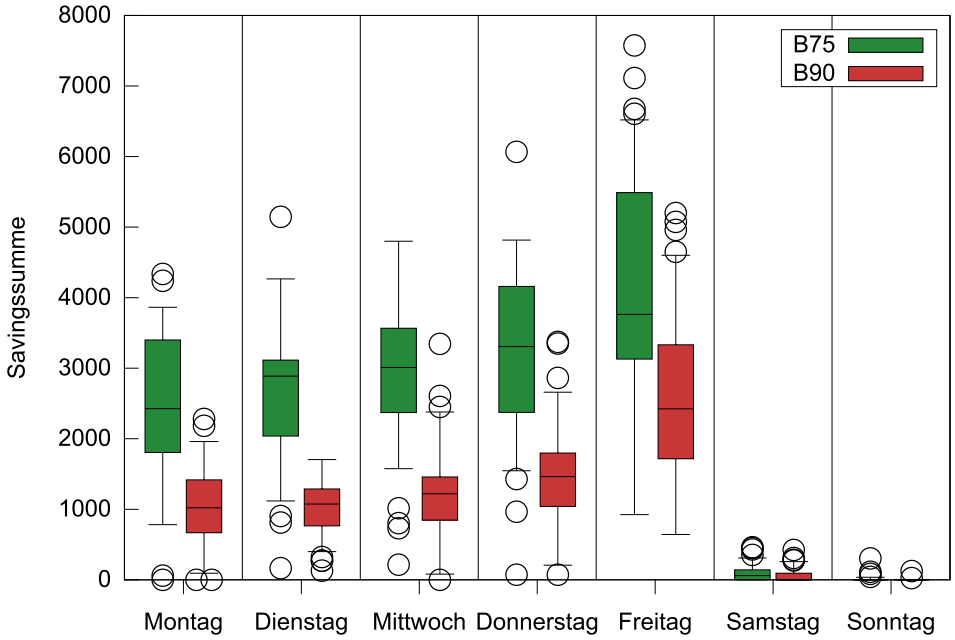


Abbildung 7.18: Ergebnisse der Benchmarks mit Tabuliste nach Wochentagen

Es besteht ein Zusammenhang zwischen der Tatsache, dass die größten Einsparungen durch Konsolidierungen an Freitagen erreicht wurden und sich gleichzeitig für Wochenenden keine nennenswerten Konsolidierungen ermitteln ließen. Denn innerhalb der Vergangenheitsdaten waren die am Wochenende zu bearbeitenden Aufträge zumeist bereits am Freitag oder sogar Donnerstag bekannt. Die Einplanung dieser Aufträge gelang somit „schon“ bei früheren Momentaufnahmen, zu denen die Einsparungen dann auch gezählt wurden[43].

[41] beide Euro-Werte gerundet
[42] zur Definition von Boxplot-Diagrammen: siehe Abschnitt 5.6.1, S. 178
[43] siehe auch Seite 297, Stichwort „Zuordnung der Zeitpunkte“

Abschließende Bemerkungen zu den Benchmarks Das hier vorgestellte, rekursive Savingsverfahren zur Lösung des Konsolidierungsproblems ließ sich erfolgreich auf die empirischen Vergangenheitsdaten der IN tIME Express Logistik GmbH anwenden. Gemäß Tabelle 7.5, S. 291, entstanden im betrachteten Benchmark-Zeitraum tatsächliche Fahrtkosten in Höhe von 27,6 Mio. Euro. Die Untersuchung legt nahe, dass mithilfe von Verfahren 24, S. 295, – etwa im Rahmen eines EUS – parallel zur manuellen Disposition zusätzliches Konsolidierungspotential in Höhe von mindestens 764 919€ hätte aufgedeckt werden können. Dies entspricht einer Kostenreduzierung um 2,8 % bezogen auf den zugrunde gelegten Benchmarkzeitraum des Jahres 2008.

Dieser Wert lässt sich als Indikator dafür interpretieren, dass die Disponenten bei IN tIME im betrachteten Zeitraum bereits gute Ergebnisse erzielten. Das bei dem Unternehmen in der Disposition eingesetzte EUS, welches im Jahr 2008 auf Basis von Luftdistanzen zwischen Fahrzeugen und auftragsbezogener Orte Vorschläge generierte, war in dieser Form vermutlich eine wesentliche Hilfe. Dennoch besteht offenbar noch Optimierungspotential.

Das Ausmaß dieses Optimierungspotentials übersteigt dabei erwartungsgemäß die oben genannten 2,8 %. Denn in diesen Wert fließen an mehreren Stellen *vorsichtige Annahmen* ein, von denen die wichtigsten auf Seite 297 beschrieben wurden[44]. Dass in den hier durchgeführten Messungen stets nur solche Aufträge konsolidiert wurden, die gemäß der Vergangenheitsdaten *nicht* konsolidiert wurden, stellt eine der größten Einschränkungen der Benchmarks dar. Es ist sehr wahrscheinlich, dass sich – retrospektiv – weitere Einsparungen generieren ließen, wenn die Benchmarks auch diese Aufträge umfassten.

Zudem wurde bei der zeitlichen Berücksichtigung der Ladevorgänge an Umladeorten im Rahmen der Benchmarks stets die Dauern der Zwischenbe- und -entladungen *aller* beteiligten Frachten einkalkuliert. Auch dies entspricht einer besonders vorsichtigen Annahme, denn in vielen Fällen kann eine Ent- und anschließende Beladung an einem Umladeort entfallen, wenn eines derjenigen Fahrzeuge für die Weiterfahrt verwendet wird, die eine an der Konsolidierung beteiligte Fracht zuvor angeliefert haben.

In gegenläufige Richtung sind einige der in Abschnitt 7.4.1 gelisteten Aspekte zu vermerken. Gerade die in diesem Modell vernachlässigten dynamischen Einflüsse wie Staus müssen für eine Übertragung in die Praxis einkalkuliert und gegebenenfalls vom Disponenten berücksichtigt werden.

Unklar ist, wie sich eine höhere Datenqualität insbesondere bezüglich der Frachtinformationen auf das berechnete Konsolidierungspotential auswirken würde. Denn einerseits wurde in den Benchmarks versucht, hier ebenfalls Vorsicht walten zu lassen[45]. Andererseits ist davon auszugehen, dass eine bessere Kenntnis der Frachten eine Konsolidierung teilweise auch verhindern würde.

[44] Stichwort „Verzicht auf Konsolidierung“
[45] siehe Abschnitt 7.4.1, S. 289 f.

8 Zusammenfassung und Ausblick

Die operative Disposition bei europaweit tätigen Expressversand-Dienstleistern wie der IN tIME Express Logistik GmbH ist eine komplexe und kombinatorisch aufwendige Problemstellung mit mehreren und teils konkurrierenden Zielstellungen. Während auf der einen Seite die Transportkosten zu minimieren sind, gilt es auf der anderen Seite, die Transportaufträge fristgemäß durchzuführen und gleichzeitig dafür zu sorgen, dass angestellte Fahrer ihre Lenk- und Ruhezeiten einhalten können.

Die Problemstellung ist aus drei Gründen für den Einsatz eines Entscheidungsunterstützungssystems sehr gut geeignet:

1. Die kombinatorische Vielfalt der möglichen Tourenpläne erschwert es menschlichen Disponenten die Übersicht über alle alternativen Entscheidungen zu bewahren und eine günstige auszuwählen.
2. Die Verfügbarkeit von digitalen Straßenkarten und leistungsstarken Wegsucheverfahren als wesentlicher Bestandteil der Tourenplanung begünstigt eine rechnergestützte Lösungsgenerierung.
3. Eine *vollautomatische* Disposition ist nach heutigem Stand noch ausgeschlossen. Weder existieren beherrschbare, mathematische Modelle, welche alle relevanten Entscheidungskriterien umfassen[1], noch werden alle diese Entscheidungskriterien in hinreichend strukturierter Form erfasst. Als Beispiele sind hier kurzfristige Verkehrsstörungen, die genaue Beschaffenheit der Frachten sowie die Form des Laderaums der eingesetzten Transportfahrzeuge zu nennen.

Die Untersuchungen der vorliegenden Arbeit umfassen schwerpunktmäßig zwei Elemente der Methoden- und Modellverwaltung in einem speziellen EUS, welches dabei helfen soll, das Konsolidierungspotential bei der operativen Disposition aufzudecken. Dies sind

- schnelle Wegsucheverfahren für digitale Straßenkarten in den Kapiteln 4–6 sowie
- eine effiziente Heuristik für die Tourenplanung mit Einbezug möglicher Frachtkonsolidierungen in Kapitel 7.

Die Datengrundlagen für sämtliche empirischen Messungen in dieser Arbeit werden in Kapitel 2 vorgestellt und hinsichtlich ihrer Qualität diskutiert. Hierzu gehören die Vergangenheitsdaten von IN tIME, die digitalen Straßenkarten des OpenStreetMap-Projekts sowie die einschlägigen Gesetze und Verordnungen für angestellte Fahrer im Bereich der straßengebundenen Transporte.

[1] siehe hierzu auch Abschnitt 7.2.1, S. 276

Während die Vergangenheitsdaten eine detaillierte Rekonstruktion der tatsächlich durchgeführten Transporte und eine recht genaue Abschätzung der zugehörigen Routen erlauben, existieren diverse Lücken bzw. Fehler in den Beschreibungen der Frachten. Dies ist auch auf die Verwendung von Freitexten in der Dispositionssoftware zurückzuführen.

Für die OSM-Straßenkarten gilt, dass auch sie Qualitätsschwankungen unterliegen. Aktuell sind dicht besiedelte Gebiete dabei tendenziell besser kartographiert als dünn besiedelte. In Bezug auf Routenplanung muss bei der Verwendung der Karten darauf geachtet werden, ein *zusammenhängendes* Straßennetz zu extrahieren.

Kapitel 3 beschäftigt sich im Anschluss mit den Einzelschritten beim Datenimport einer OSM-Straßenkarte. Es ist an der konkreten Implementierung orientiert und begründet mehrere Modellierungsentscheidungen wie die Entfernung von Nebenclustern und die Kantenobjekt-Verbindung mit praktischen Überlegungen. Die maßgeblichen Ergebnisse dieses Kapitels sind in Tabelle 3.6, S. 85, zusammengefasst und dienen der besseren Nachvollziehbarkeit des Datenimports.

Der erste Schwerpunkt der Arbeit beginnt in Kapitel 4 mit der Vorstellung ausgewählter Wegsucheverfahren aus der Literatur. Diese werden bezüglich der Suchbeschleunigung und – sofern nötig – des zusätzlichen Speicherbedarfs, dem so genannten *Overhead* mit dem unidirektionalen Dijkstra-Verfahren (Verfahren 5, S. 91) verglichen. Grundsätzlich gilt, dass die Suchbeschleunigung mit einem erhöhten Overhead einhergeht. Das Hub-Based-Labeling-Verfahren ermöglicht aktuell die schnellste Wegsuche, benötigt entsprechend aber auch den größten Overhead. Eine abschließende Übersicht der untersuchten Verfahren und der Vergleichskriterien befindet sich in Tabelle 4.2, S. 128.

Das Contraction-Hierarchies-Verfahren bietet ein besonders günstiges Verhältnis aus Suchbeschleunigung und Overhead und wird gesondert und ausführlich in Kapitel 5 behandelt. Vorgestellt werden zwei neue Erweiterungen um eine Zielrichtung der Wegsuche – die A*-Eingrenzung des Suchraums (Abschnitt 5.2.2.1) und die bidirektionale A*-Suche (Abschnitt 5.2.2.2) – sowie eine neue Erweiterung zur Parallelisierung der Wegsuche (Abschnitt 5.2.3). Die Erweiterungen sind miteinander kombinierbar und erweisen sich in den vorgestellten Benchmarks als leistungsstarke Ergänzungen zur Standardsuche in Contraction Hierarchies (Abschnitt 5.6.6):

- Mit der kombinierten Zielrichtung wird die Anzahl der notwendigen Dijkstra-Iterationen gegenüber der Standardsuche in Contraction Hierarchies deutlich verringert. Bei der Suche nach kürzesten Wegen ergibt sich eine Verbesserung um einen Faktor von mehr als 2. Für schnellste Wege entsteht ein Suchbeschleunigungsfaktor von etwa 1,5 (siehe Abbildung 5.26, S. 192).
- Bezogen auf die Suchdauer wurden im Rahmen dieser Arbeit die mit Abstand schnellsten gemessenen Wegsuchen durch der Kombination aller drei Erweiterungen erreicht (Abschnitt 5.6.7).

Im weiteren Verlauf des Kapitels werden Überlegungen für eine effiziente Implementierung von Contraction Hierarchies vorgestellt (Abschnitt 5.5). Es wird gezeigt, dass die Besonderheit von OSM-Straßenkarten zu berücksichtigen ist, für die gilt, dass die Knoten an Autobahnen tendenziell kleinere Ids besitzen (Abschnitt 5.5.2) und wie sich die

Hierarchie-Erzeugung parallelisieren lässt (Abschnitt 5.5.3). Für die Sortierung der Knoten der Horizontmengen während der Wegsuche werden vier Datenstrukturen in Betracht gezogen (Abschnitt 5.5.5), von denen die Prioritätswarteschlange in den Benchmarks auf einer Straßenkarte Europas die besten Ergebnisse liefert (Abschnitt 5.6.3). In weiteren Benchmarks wird empirisch gezeigt, dass sich durch eine aufwendigere Knotenblockierung zwar einige Iterationen bei der Wegsuche einsparen lassen, dies jedoch deutlich zu Lasten der Suchgeschwindigkeit geschieht (Abschnitt 5.6.5).

Kapitel 6 erweitert den ersten Schwerpunkt der Arbeit um die Berücksichtigung von Abbiegebeschränkungen. Dazu werden die drei in der Literatur am häufigsten beschriebenen Ansätze – die Graph-Transformation per Knotensplitting (Abschnitt 6.2.1), die pfeilbasierte Wegsuche (Abschnitt 6.2.2) und die knotenbasierte Suche in Pseudographen (Abschnitt 6.2.3) – sowie ein neues Verfahren, die adaptive Wegsuche (Abschnitt 6.2.4), vorgestellt und hinsichtlich ihrer Eignung für den Einsatz in Contraction Hierarchies diskutiert.[2] Die adaptive Wegsuche besitzt gegenüber den anderen Abbiegebeschränkungen berücksichtigenden Wegsucheverfahren zwei bedeutende Vorzüge:

1. Der Ursprungsgraph bleibt unverändert. Anders als bei der Graph-Transformation per Knotensplitting und der knotenbasierten Suche in Pseudographen ist weder ein Präprozess erforderlich, in dem neue Knoten und Pfeile berechnet werden, noch sind für eine Wegsuche zwischen zwei Knoten vorab künstliche, temporäre Knoten und Pfeile in den Graphen einzufügen.
2. Beginnt die Wegsuche in einem Gebiet ohne Abbiegebeschränkungen, verhält sie sich wie das knotenbasierte Dijkstra-Verfahren und nimmt die Knoten nur ein einziges Mal in die Horizont- und Lösungsmenge auf. Kein anderes der untersuchten Verfahren besitzt diese positive Eigenschaft. Die Adaptivität bewirkt, dass das Verfahren erst bei Erreichen einer Abbiegebeschränkung pfeilbasiert arbeitet und dadurch erlaubt, dass Knoten mehrfach erreicht werden. Sogar ein Wechsel zurück zur knotenbasierten Suche ist im Verlauf des Verfahrens möglich.

Sowohl die pfeilbasierte als auch die adaptive Wegsuche sind gut auf Contraction Hierarchies übertragbar. Ihre notwendigen Anpassungen werden in diesem Zusammenhang ausführlicher untersucht (Abschnitte 6.3.1 bzw. 6.3.2). Dabei ist mit der leichteren Identifikation von Brückenknoten während der Wegsuche ein zusätzlicher Vorzug der adaptiven Suche im Vergleich zur pfeilbasierten Suche hervorzuheben.

Die *Erzeugung* einer Contraction Hierarchy für Wegsuchen mit Abbiegebeschränkungen muss jedoch pfeilbasiert erfolgen (Abschnitt 6.4). Sie profitiert stärker von einer Parallelisierung als die Hierarchie-Erzeugung *ohne* Berücksichtigung von Abbiegebeschränkungen (Abschnitt 6.5.2).

Für die Implementierung der adaptiven Wegsuche wird eine Datenstruktur präsentiert, mit deren Hilfe sich die Mehrfachaufnahme von Knoten in die Horizont- und Lösungsmengen effizient realisieren lässt (Abschnitt 6.5.1). Die oben genannten Erweiterungen der Wegsuche um eine Zielrichtung und eine Parallelisierung sind gut mit der pfeilbasierten und mit der adaptiven Wegsuche in Contraction Hierarchies kombinierbar, was sich auch in Benchmarks zeigt (Abschnitte 6.6.1 bzw. 6.6.2).

[2] Die Grundidee der adaptiven Wegsuche wird zwar auch von Schmid [2001, S. 34 ff.] umrissen, ist dort jedoch in der Beschreibung lückenhaft.

Im direkten Vergleich zwischen der pfeilbasierten und der adaptiven Wegsuche ist die adaptive Wegsuche jedoch deutlich zu bevorzugen. Für schnellste Wege benötigt sie etwa 40 % weniger Iterationen (siehe Abbildung 6.28, S. 257).

Nicht alle Aspekte der Contraction Hierarchies konnten im Rahmen dieser Arbeit ausführlich betrachtet werden. Lohnenswert erscheint insbesondere eine vertiefende Untersuchung der nachfolgenden Punkte:

- *Die Knotensortierung während der Hierarchie-Erzeugung*
  Die Liste der in Abschnitt 5.3 aufgeführten Sortierkriterien ist selbstverständlich nicht vollständig. Zusätzliche bzw. alternative Kriterien sind ohne weiteres denkbar.[3] Außerdem muss die Sortierung nicht notwendigerweise nach einer einzigen, konstanten Linearkombination der Sortierkriterien erfolgen. Die Eigenschaften des Graphen während der Hierarchie-Erzeugung (Abschnitt 5.4) legen nahe, dass es sinnvoll sein könnte, die Knotensortierung zu wechseln, wenn der größte Anteil der Knoten des Ursprungsgraphen bereits kontrahiert ist. Auch hierarchische Sortierungen, die nur bei Gleichheit *eines* Kriteriums den Wert eines *anderen* Kriteriums heranziehen, sind vorstellbar.
- *Modellierung dynamischer Störungen*
  Eine interessante und praxisnahe Aufgabe entsteht mit der Fragestellung, wie dynamische Störungen bei der Wegsuche, wie Verkehrsstaus oder Baustellen, im Graphen dargestellt werden können. Anders als beim Dijkstra- oder A*-Verfahren ist es in diesen Fällen nicht einfach möglich, die Bewertung der betroffenen Pfeile im Graphen ad hoc zu verändern, da sich der Einfluss der Störung auch auf Shortcut-Pfeile erstrecken kann.
- *Verbindung mehrerer Contraction Hierarchies*
  Graphen ohne Shortcut-Pfeile und Knotenränge können sehr leicht miteinander verbunden werden, so dass sich im daraus entstehenden, vereinten Graphen Wege suchen lassen. Dieses Szenario entsteht zum Beispiel, wenn die Transportplanung eines Unternehmens nicht mehr nur national, sondern auch international stattfinden soll. Bei Contraction Hierarchies stellt sich die Frage, ob und wie es möglich ist, Hierarchien zu vereinen, ohne eine vollständige Neuberechnung durchführen zu müssen. Auch im Hinblick auf eine schnelle Hierarchie-Erzeugung könnten derartige Untersuchungen interessante Möglichkeiten für Parallelisierungen eröffnen.

Im zweiten Schwerpunkt der Arbeit wird in Kapitel 7 eine Heuristik zur Lösung eines speziellen Pickup-And-Delivery-Tourenplanungsproblems mit Zeitfenstern entwickelt, bei dem durch Umlademöglichkeiten Frachten unterschiedlicher Aufträge gemeinsam transportiert, d. h. konsolidiert, werden können. Zunächst wird eine natürlich-sprachliche Beschreibung des Standardproblems der Tourenplanung und einiger seiner Variationen gegeben (Abschnitt 7.1.1.1), auf welche die Entwicklung der mathematisch-formalen Problembeschreibung folgt (Abschnitt 7.1.1.2). Das mathematische Modell wird danach um die Möglichkeiten von Umladungen (Abschnitt 7.1.2) sowie von Teilladungen (Abschnitt 7.1.3) erweitert. Abschließend werden die wichtigsten Grenzen dieser Modellierung genannt (Abschnitt 7.1.4).

[3] Weitere Kriterien finden sich auch bei Geisberger [2008, S. 14 ff.] und Geisberger u. a. [2008, S. 322 ff.].

Ausgehend von der Erkenntnis, dass eine vollautomatische Disposition mit aktuellen Modellen und Methoden noch nicht realisierbar ist, wird ein Szenario für den Einsatz eines Entscheidungsunterstützungssystems im Rahmen der operativen Disposition entwickelt (Abschnitt 7.2). Im Hinblick auf eine mögliche Implementierung sind dabei mehreren Modellparametern konkrete Werte zuzuweisen. Hierzu gehören insbesondere Fahrtdauern, Dauern von Ladevorgängen, die Verortung von Kundenstandorten auf einer digitalen Straßenkarte sowie die berücksichtigten Fahrtkosten. Anhand der Vergangenheitsdaten lassen sich zu diesen Parametern jeweils sinnvolle Ausprägungen ermitteln (Abschnitt 7.3). Für die Dauern der Ladevorgänge werden mit dem Set *B75* und dem Set *B90* zwei unterschiedlich „schnelle“ Varianten der Ausprägungen extrahiert. Dies erlaubt eine Tourenplanung unter eher „optimistischen“ und unter eher „pessimistischen“ Annahmen. Ergänzend wird mit dem so genannten *Einzelauftrag* eine spezielle Datenstruktur vorgestellt, die eine Repräsentation aller potentiellen Umladungen für einen Kundenauftrag ermöglicht (Abschnitt 7.3.4).

Die Untersuchungen des Konsolidierungspotentials bei IN tIME basieren auf einem Benchmark, dem 2 066 Momentaufnahmen von Dispositions-Situationen mit offenen und noch nicht vollständig bearbeiteten Kundenaufträgen aus dem Jahr 2008 zugrunde liegen. Um eine bessere Einschätzung der realen Dispositionen zu erhalten, wird zunächst ermittelt, inwieweit in der Vergangenheit bereits Aufträge konsolidiert wurden. Der Vergleich zwischen den real entstandenen Transportkosten und den theoretischen Kosten für Direktfahrten aller Aufträge zeigt, dass bereits 35 % der Aufträge konsolidiert wurden (Abschnitt 7.4.2)[4].

Um *zusätzliches* Konsolidierungspotential aufzudecken, wird im Anschluss ein neues Verfahren präsentiert, das *rekursive Savingsverfahren*. Dieser Heuristik liegt die Überlegung zugrunde, dass durch Konsolidieren von Aufträgen mit Umladungen ein Planungsverbund entstehen kann, der viele Aufträge umfasst und hier als *Konsolidierungskette* bezeichnet wird (Abschnitt 7.4.3).[5] Das Verfahren besitzt als Stellparameter die Anzahl der Kettenglieder, d. h. der in wechselseitiger Abhängigkeit eingeplanten Aufträge, die in der Planung erlaubt werden sollen.

Angewendet auf den Benchmark in den Varianten B75 und B90 liefert das Verfahren Tourenpläne für die betrachteten Momentaufnahmen, welche besonders bei der Disposition für das Wochenende deutliches Einsparungspotential offenbaren. Dies gilt sowohl für den Fall, dass Umladungen ausgeschlossen sind (Abschnitt 7.4.4), als auch für den Fall, in dem Umladungen erlaubt sind (Abschnitt 7.4.5). Im Falle erlaubter Umladungen vergrößert sich das ermittelte Einsparungspotential um etwa 5 %, wenn Konsolidierungsketten maximal zwei Aufträge umfassen und um etwa 12 %, wenn drei Aufträge im Verbund konsolidiert werden dürfen.

Zuletzt wird das rekursive Savingsverfahren um Tabulisten ergänzt, mit denen verhindert wird, dass derselbe Auftrag in unterschiedlichen Momentaufnahmen des Benchmarks mehrfach betrachtet wird (Abschnitt 7.4.6). Die Benchmarkergebnisse sind dadurch noch besser auf die Realität übertragbar. Hervorzuheben ist, dass sie aufgrund vieler bewusst vorsichtiger Modellannahmen eine *untere Schranke* für das tatsächliche Einsparungspotential darstellen.

[4] siehe auch Tabelle 7.5, S. 291

[5] Für ein anschauliches Beispiel siehe Abbildung 7.12, S. 293.

Insgesamt wird als Einsparungspotential für den betrachteten Zeitraum im Jahr 2008 in der Variante B90 ein Wert von mindestens 369 739€, entsprechend einer Kostenreduktion von 1,3 % ermittelt. Die „optimistischere“ Variante B75 ergibt ein Potential von mindestens 764 919€, entsprechend einer Kostenreduktion von 2,8 %, und damit einen mehr als doppelt so hohen Wert.

Das vorgestellte Verfahren zum Aufdecken von Konsolidierungspotential bei der operativen Disposition ist ausgerichtet auf einen Einsatz innerhalb eines dialogorientierten Entscheidungsunterstützungssystems. Die Leistungsfähigkeit des Verfahrens konnte anhand von Benchmarks nachgewiesen werden. Für den Einsatz des Systems mit Disponenten als Benutzer ist in Hinblick auf Software-Ergonomie noch eine geeignete Darstellung der generierten Konsolidierungsketten zu entwickeln. Dieser Aspekt umfasst idealerweise nicht nur die Visualisierung der zeitlichen und räumlichen Dimensionen der beteiligten Transporte, sondern auch die Hervorhebung der Unterschiede zwischen verschiedenen Konsolidierungsvorschlägen des Systems.

Zukünftige Forschung sollte ebenfalls in die Erweiterung der vorhandenen Modelle für die Tourenplanung fließen, um die Lücken zur Praxis weiter zu schließen. Hier ist besonders die effiziente Modellierung der Lenk- und Ruhezeiten für die an den Transporten beteiligten Fahrer zu betonen.[6] Bedarf besteht ebenfalls an verfeinerten Modellen für die zu transportierenden Frachten, um beispielsweise auch die Beladung der Fahrzeuge optimieren zu können.

[6] Erste Ansätze hierzu finden sich bei Kopfer u. Meyer [2009]. Siehe auch Seite 275 in dieser Arbeit.

Anhang A

Dateiformat für Contraction Hierarchies

Nachfolgend wird das binäre Dateiformat beschrieben, in dem die dieser Arbeit zugrunde liegenden Contraction Hierarchies gespeichert sind. Als *Block* gekennzeichnete und per Leerzeilen voneinander getrennte Abschnitte können wiederholt hintereinander auftreten.

Zeilen, denen eine Raute (#) vorangestellt ist, sind Kommentare. Zwischen spitzen Klammern (<>) stehen die Beschreibungen und, durch einen Doppelpunkt davon abgetrennt, die Datentypen der einzulesenden Werte.

```
1  # Metadaten der Hierarchie
2  <Versionsnummer des Dateiformats:INTEGER>
3  <Name der Hierarchie:STRING>
4  <Gewichtungsfaktor für "‘Anzahl kontrahierter Nachbarknoten"’:INTEGER>
5  <Gewichtungsfaktor für "‘Pfeildifferenz"’:INTEGER>
6  <Gewichtungsfaktor für "‘Anzahl notwendiger Shortcut−Pfeile"’:INTEGER>
7  <Gewichtungsfaktor für "‘Kontraktionskosten"’:INTEGER>
8  <Gewichtungsfaktor für "‘Wegsuchekosten"’:INTEGER>
9  <Gewichtungsfaktor für "‘Pfeilquotient"’:INTEGER>
10 <Gewichtungsfaktor für "‘Originaler Pfeilquotient"’:INTEGER>
11 <Gewichtungsfaktor für "‘Zufälliger Bias"’:INTEGER>
12 <Gewichtungsfaktor für "‘Abbiegebeschränkungs−Flag"’:INTEGER>
13 <Kennzeichnung der zugrunde liegenden Metrik>
14
15
16 # Block: Knoten
17 <Anzahl der Knoten in der Karte:INTEGER>
18 # Pro Knoten der Karte folgt ein nachfolgend formatierter Einzelknoten−Block
19 # Bei Kreuzungsknoten mit Abbiegebeschr. wird die negative Knoten−Id gespeichert
20 <Knoten−Id:LONG>
21 <Knoten−Rang:INTEGER>
22 <Breitengrad:DOUBLE>
23 <Längengrad:DOUBLE>
24
25
26
27 # Block: Original−Kantenobjekte
28 <Anzahl der Original−Kantenobjekte in der Karte:INTEGER>
29 # Pro Original−Kantenobjekt der Karte folgt ein nachfolgend formatierter Einzel−Original−
       Kantenobjekt−Block
30 <Kantenobjekt−Id:INTEGER>
31 <Knoten−Id des niederrangigen Knotens:LONG>
32 <Knoten−Id des höherrangigen Knotens:LONG>
33 <Kantenlänge in Kilometer:DOUBLE>
34 <Flag für Fahrerlaubnis in Richtung höherrangiger Knoten:BOOLEAN>
```

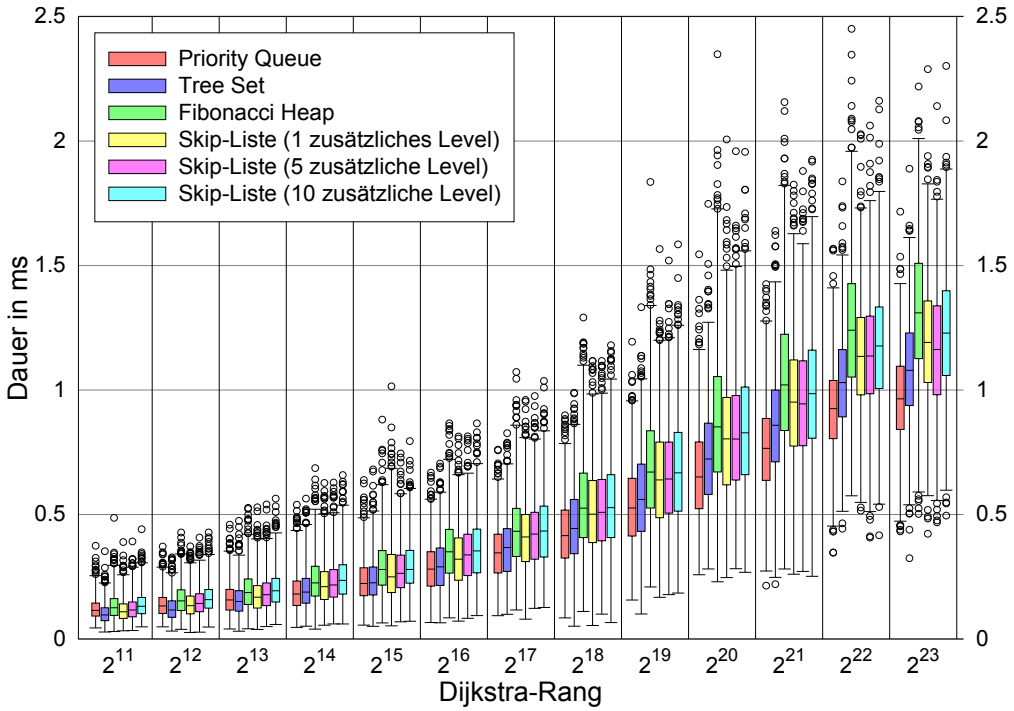
```
35 <Flag für Fahrerlaubnis in Richtung niederrangiger Knoten:BOOLEAN>
36 <Ordinal des Landes der Kantenkategorie:INTEGER>
37 <Ordinal der Straßenkategorie der Kantenkategorie:INTEGER>
38 <Geschwindigkeitsbegrenzung in km/h:SHORT>
39
40
41
42 # Block: Shortcut-Kantenobjekte
43 <Anzahl der Shortcut-Kantenobjekte in der Karte:INTEGER>
44 # Pro Shortcut-Kantenobjekt der Karte folgt ein nachfolgend formatierter Einzel-Shortcut-
       Kantenobjekt-Block
45 # Es wird zwischen Shortcut-Blöcken und Shortcut-Schlingen-Blöcken unterschieden
46 # Sie sind beim Parsen anhand des Vorzeichens der Kantenobjekt-Id unterscheidbar
47
48 # Shortcut-Block
49 <Kantenobjekt-Id:INTEGER>
50 <Knoten-Id des niederrangigen Knotens:LONG>
51 <Knoten-Id des höherrangigen Knotens:LONG>
52 <Flag für Fahrerlaubnis in Richtung höherrangiger Knoten:BOOLEAN>
53 <Flag für Fahrerlaubnis in Richtung niederrangiger Knoten:BOOLEAN>
54 <ID des ersten überbrückten Kantenobjekts:INTEGER>
55 <ID des zweiten überbrückten Kantenobjekts:INTEGER>
56
57 # Shortcut-Schlingen-Block
58 <negative Kantenobjekt-Id:LONG>
59 <Knoten-Id des beidseitigen Endknotens:INTEGER>
60 <Knoten-Id des überbrückten Knotens:INTEGER>
61 <ID des ersten überbrückten Kantenobjekts:INTEGER>
62 <ID des zweiten überbrückten Kantenobjekts:INTEGER>
63
64
65 # Block: Abbiegebeschränkungen
66 <Anzahl der Kreuzungsknoten mit Abbiegebeschränkungen in der Karte:INTEGER>
67 # Pro Kreuzungsknoten folgt ein nachfolgend formatierter Einzel-Kreuzungsknoten-Block
68 <Knoten-Id des Kreuzungsknotens:LONG>
69 <Anzahl der Abbiegegebote des Kreuzungsknotens:INTEGER>
70 # Pro Abbiegegebot folgt ein nachfolgend formatierter Block
71
72 # Abbiegegebots-Block-Anfang
73 <Kantenobjekt-Id des Original-Kantenobjekts von dem aus das Gebot besteht:INTEGER>
74 <Knoten-Id des zweiten Knotens dieses Original-Kantenobjekts:LONG>
75 <Kantenobjekt-Id des Original-Kantenobjekts zu dem das Gebot besteht:INTEGER>
76 <Knoten-Id des zweiten Knotens dieses Original-Kantenobjekts:LONG>
77 # Abbiegegebots-Block-Ende
78
79 <Anzahl der Abbiegeverbote des Kreuzungsknotens:INTEGER>
80 # Pro Abbiegeverbot folgt ein Block, der analog zum Abbiegegebots-Block formatiert ist
```

Listing A.1: Dateiformat für Contraction Hierarchies

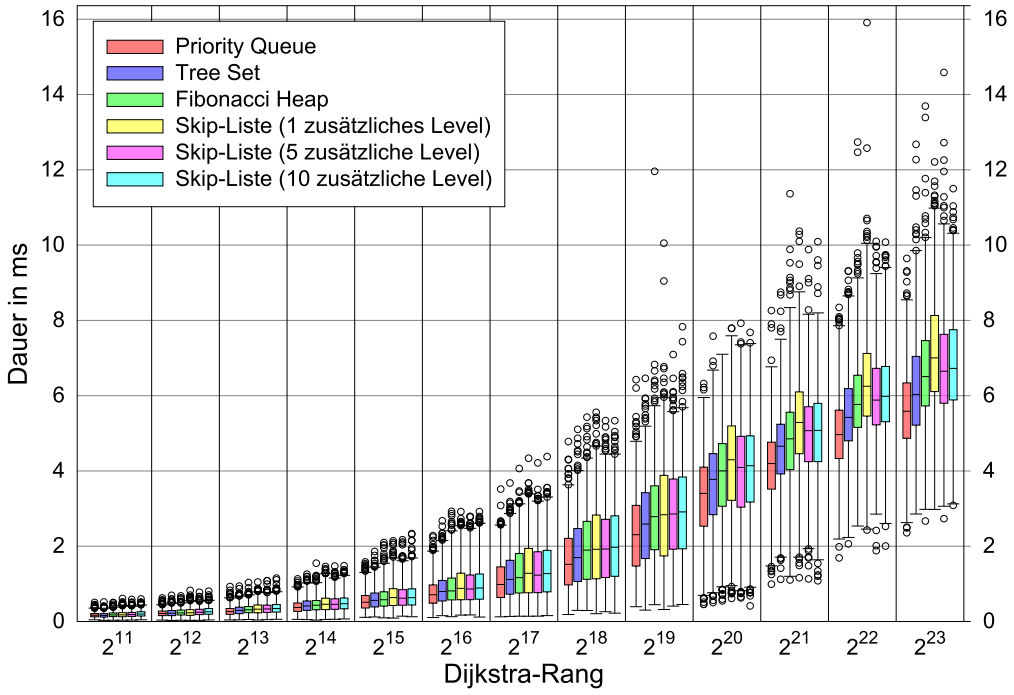
Anhang B

Vollständige Boxplot-Diagramme

Die Abbildungen dieses Anhangs sind die vollständigen Boxplot-Diagramme zu den Abschnitten 5.6 und 6.6. Die Antennen der Boxplots umfassen 99 % aller Messwerte um den Median. Das übrige Prozent ist als Ausreißer dargestellt.

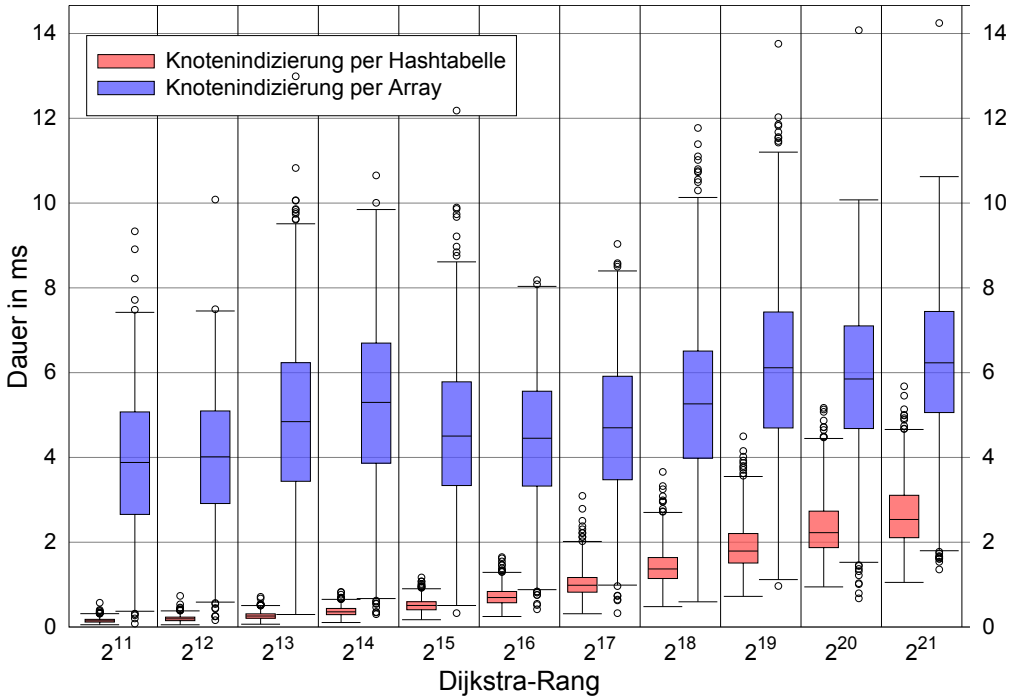


(a) Suche nach schnellsten Wegen

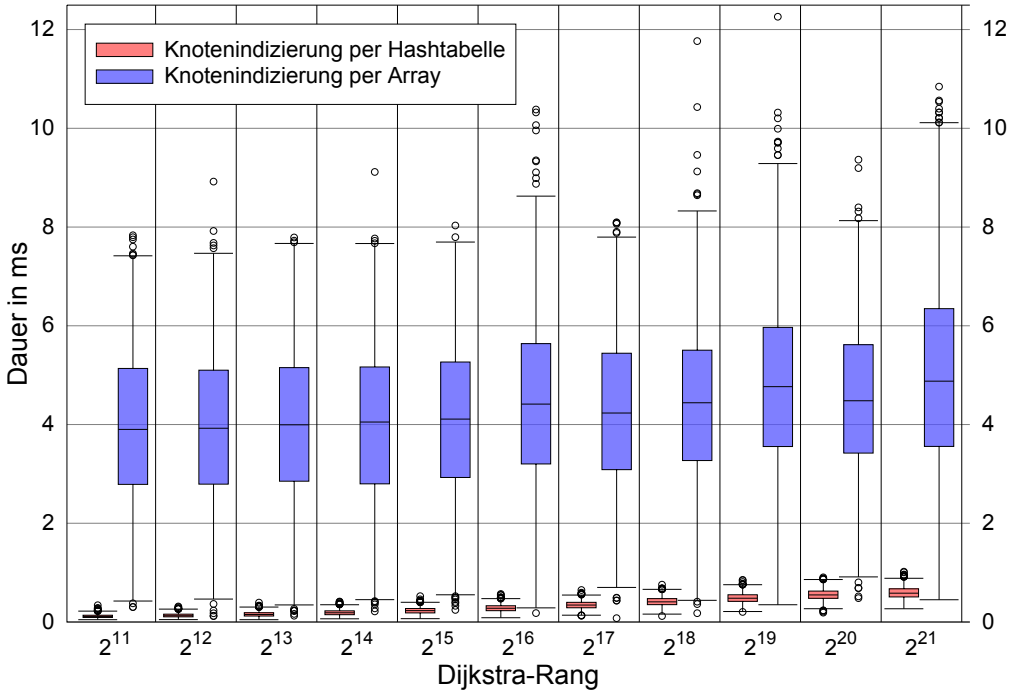


(b) Suche nach kürzesten Wegen

Abbildung B.1: Vergleich verschiedener Datenstrukturen bei der Knotensortierung
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.27, S. 194.

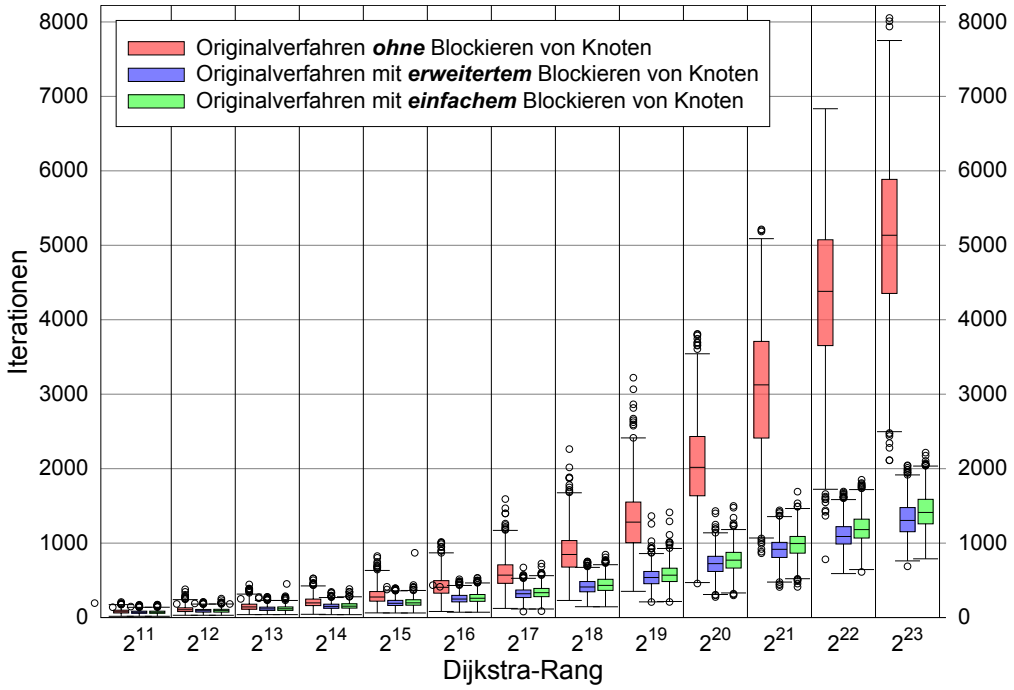


(a) Suche nach *kürzesten* Wegen

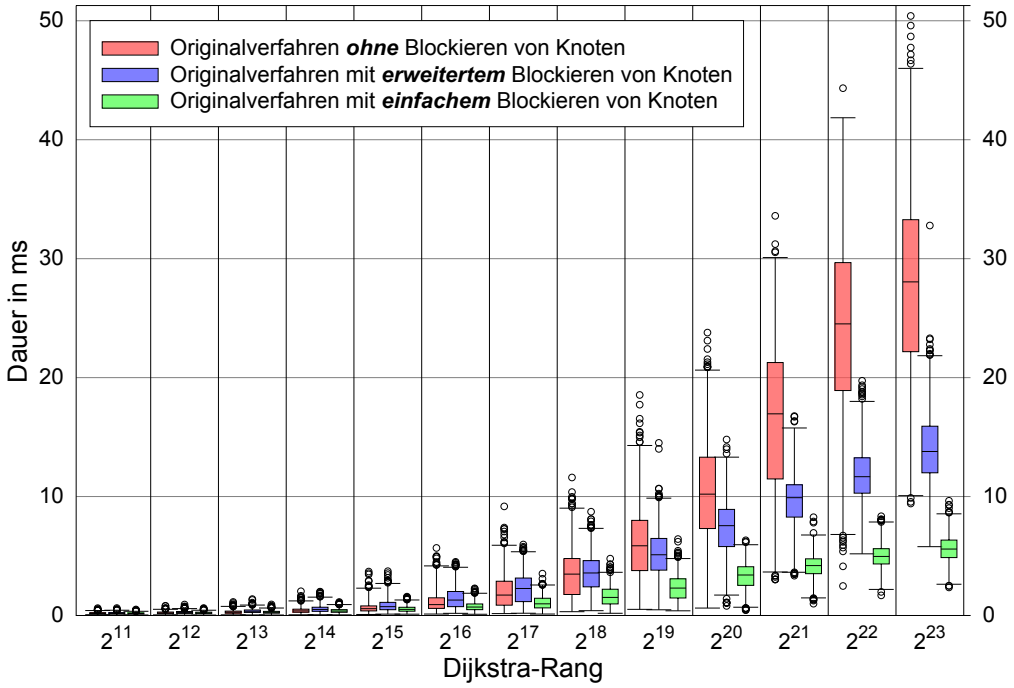


(b) Suche nach *schnellsten* Wegen

Abbildung B.2: Benchmark für die Indizierung der Knoten
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.27, S. 194.

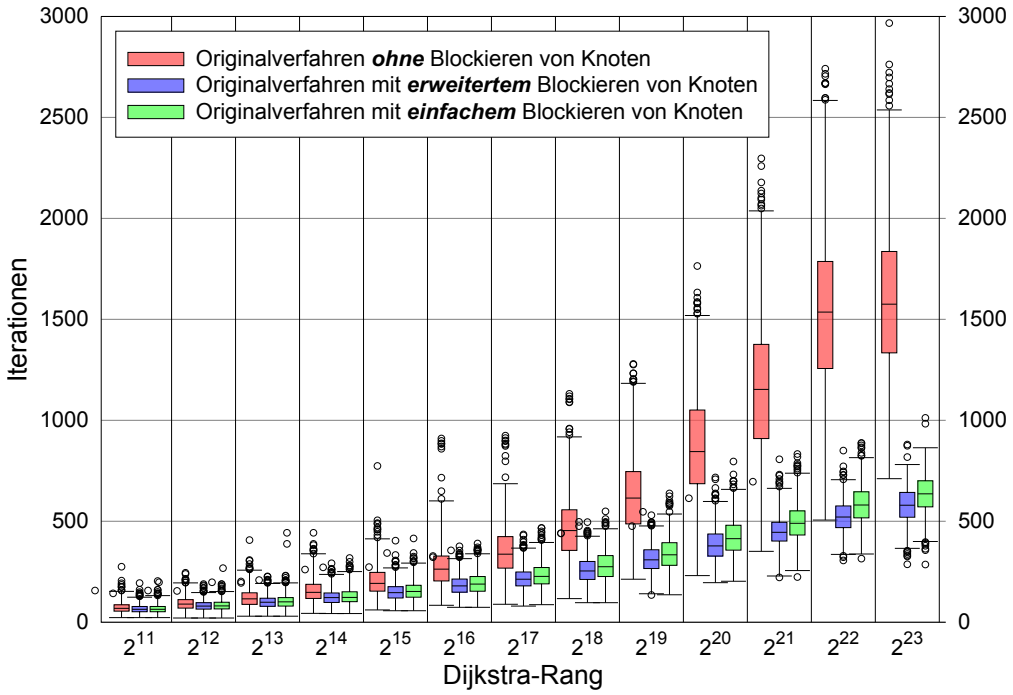


(a) Benötigte Iterationen

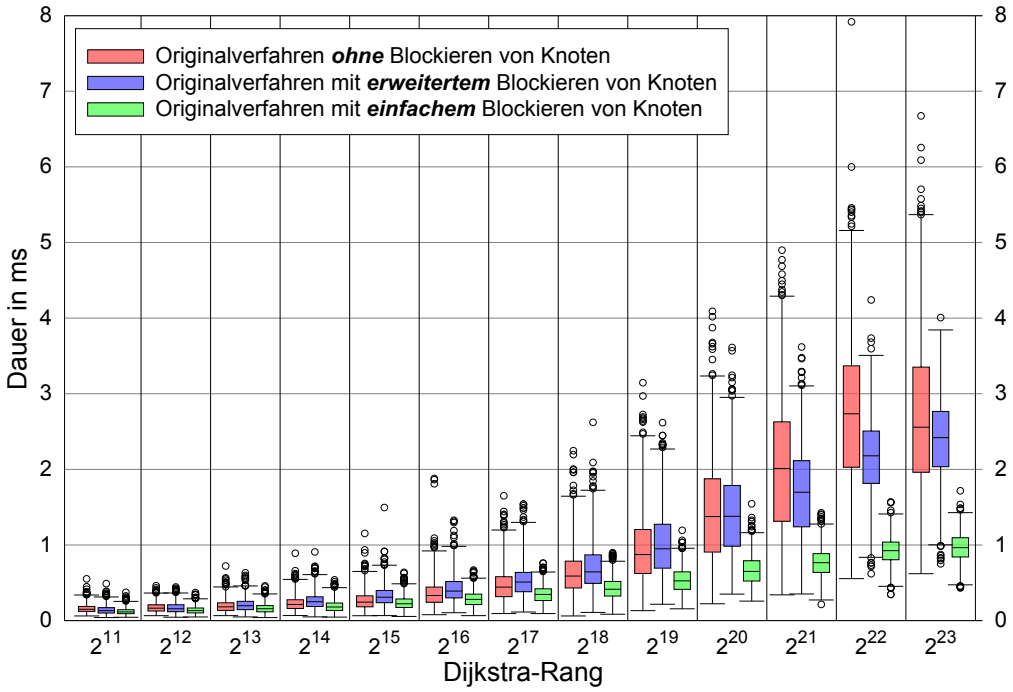


(b) Suchdauer

Abbildung B.3: Benchmark für Knotenblockierung bei Suchen nach *kürzesten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.22, S. 186.

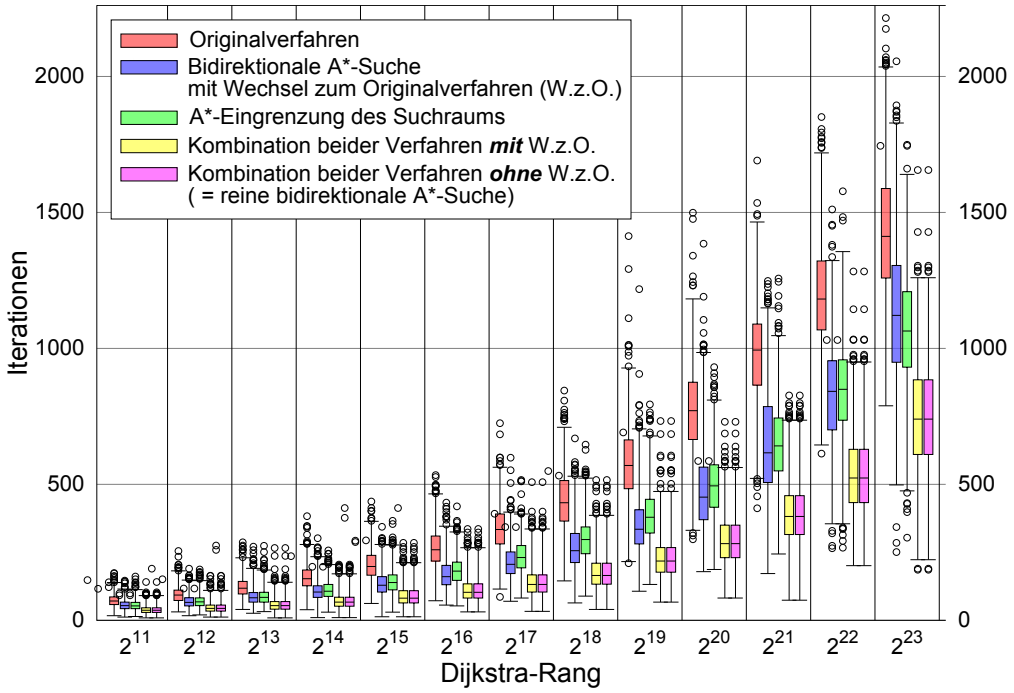


(a) Benötigte Iterationen

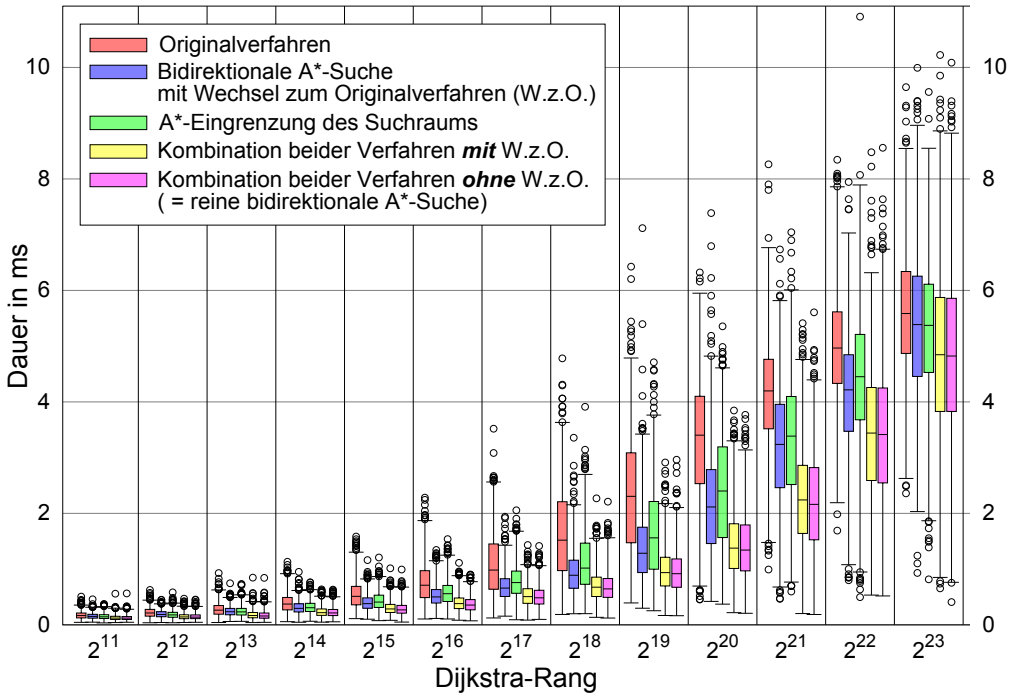


(b) Suchdauer

Abbildung B.4: Benchmark für Knotenblockierung bei Suchen nach *schnellsten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.23, S. 187.

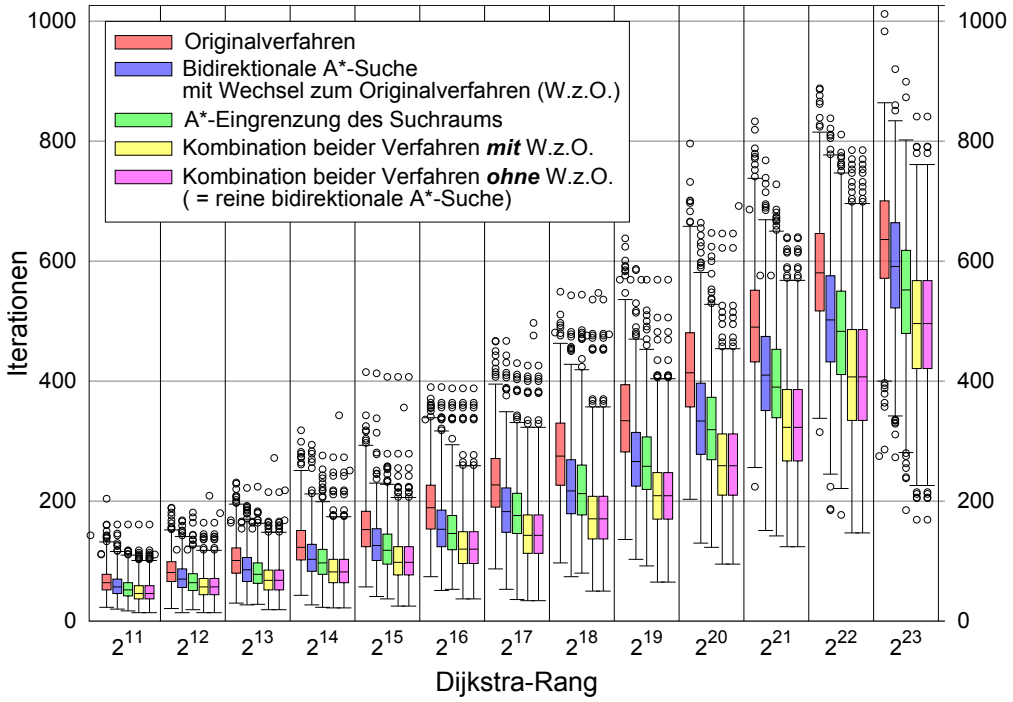


(a) Benötigte Iterationen

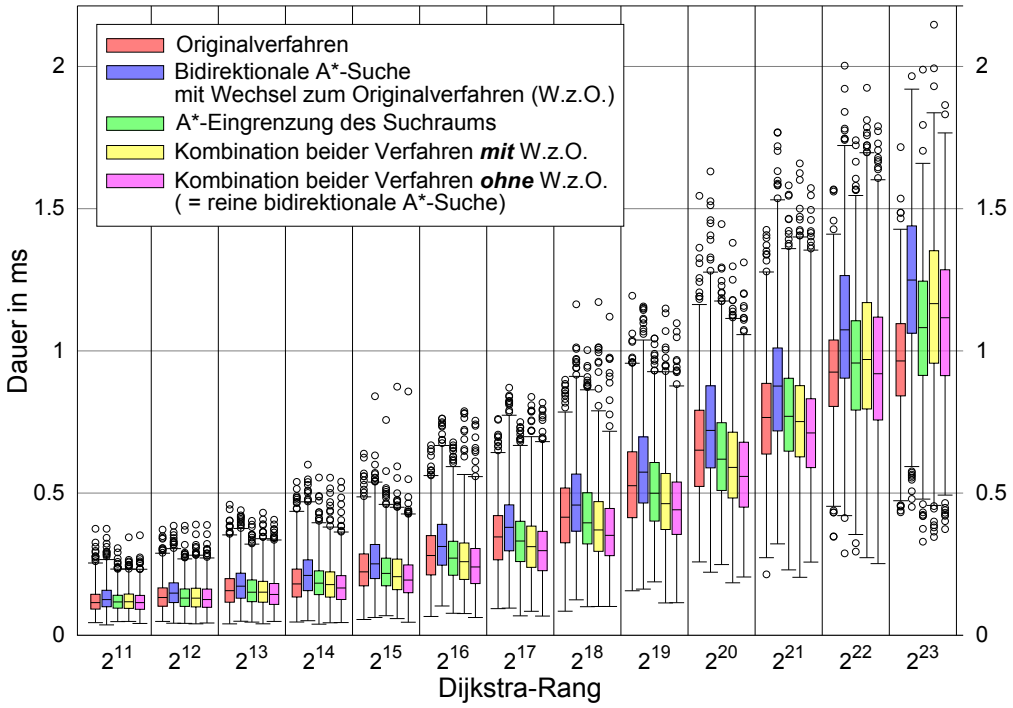


(b) Suchdauer

Abbildung B.5: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.24, S. 189.

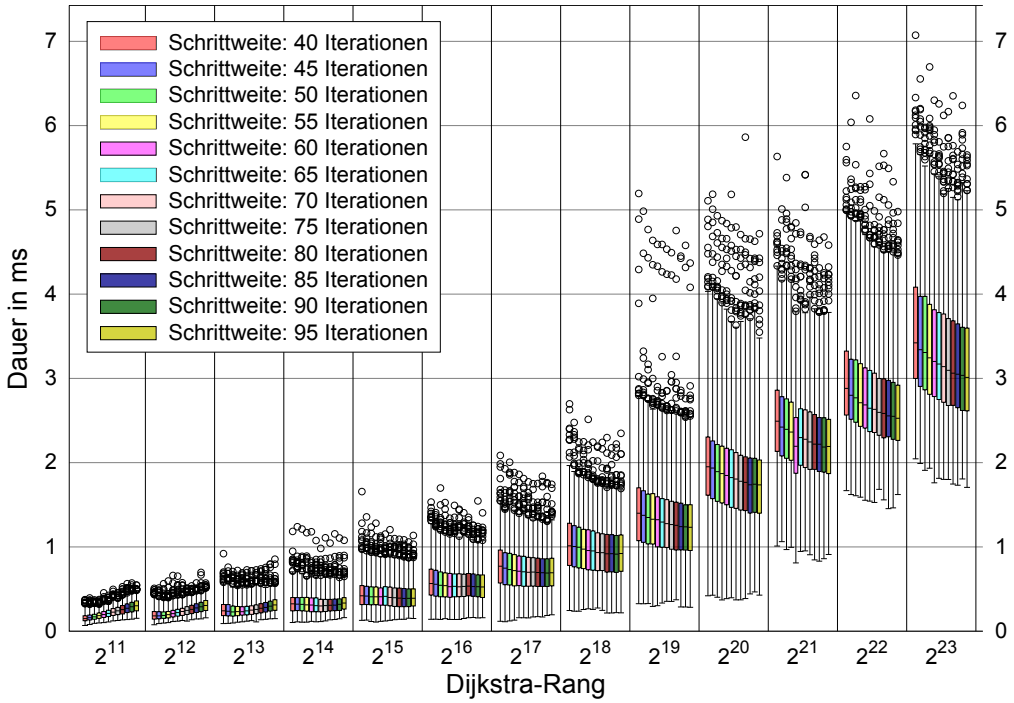


(a) Benötigte Iterationen

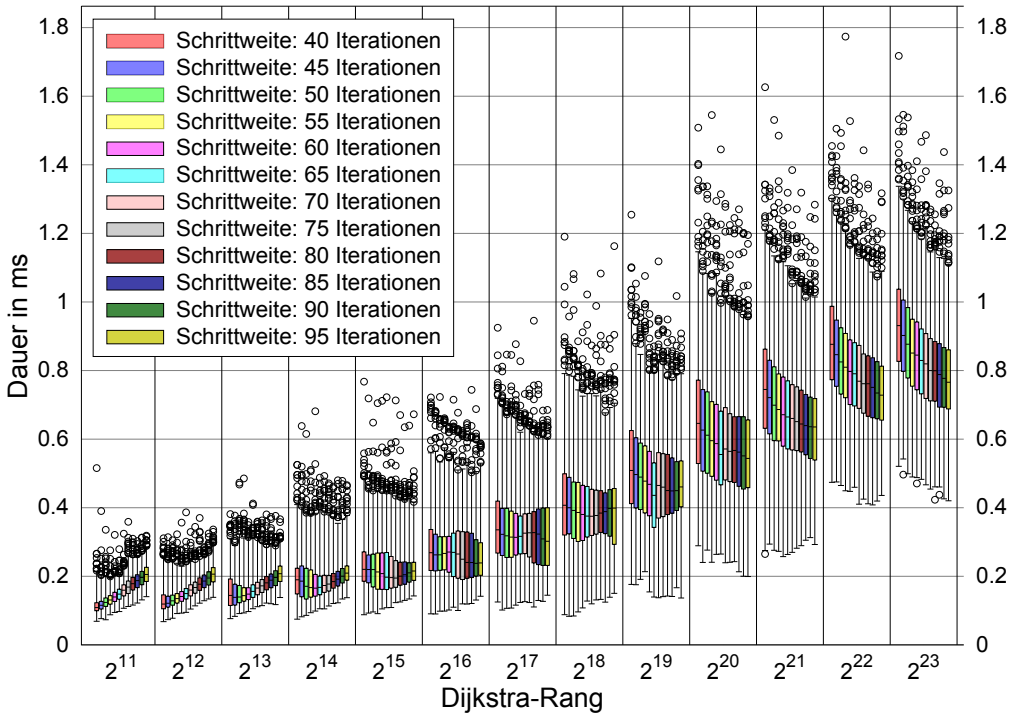


(b) Suchdauer

Abbildung B.6: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.25, S. 190.

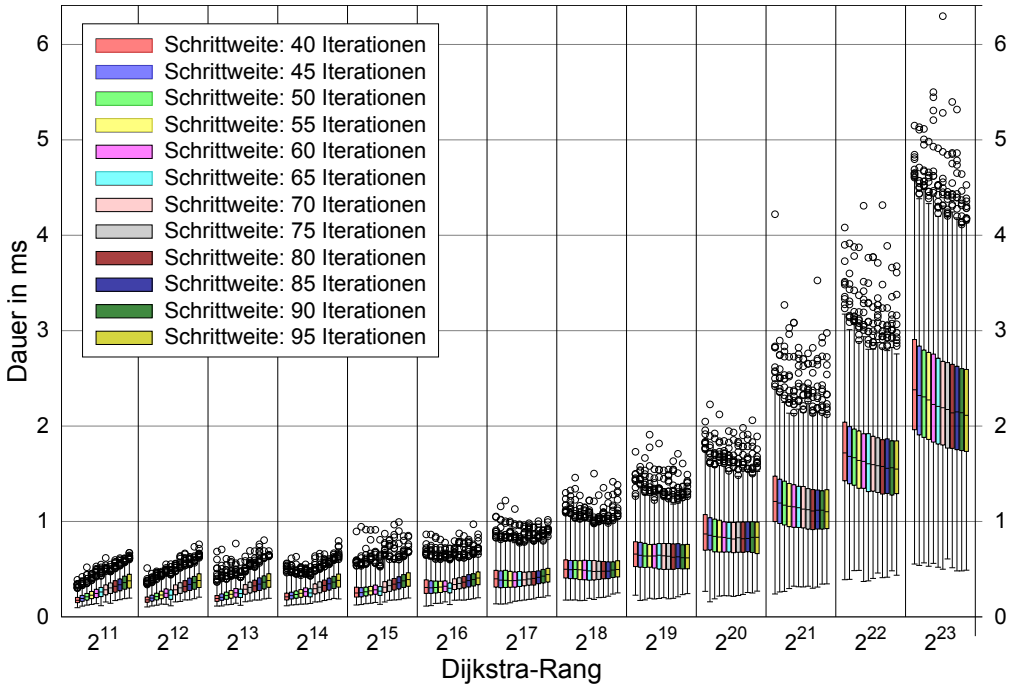


(a) Suche nach *kürzesten* Wegen

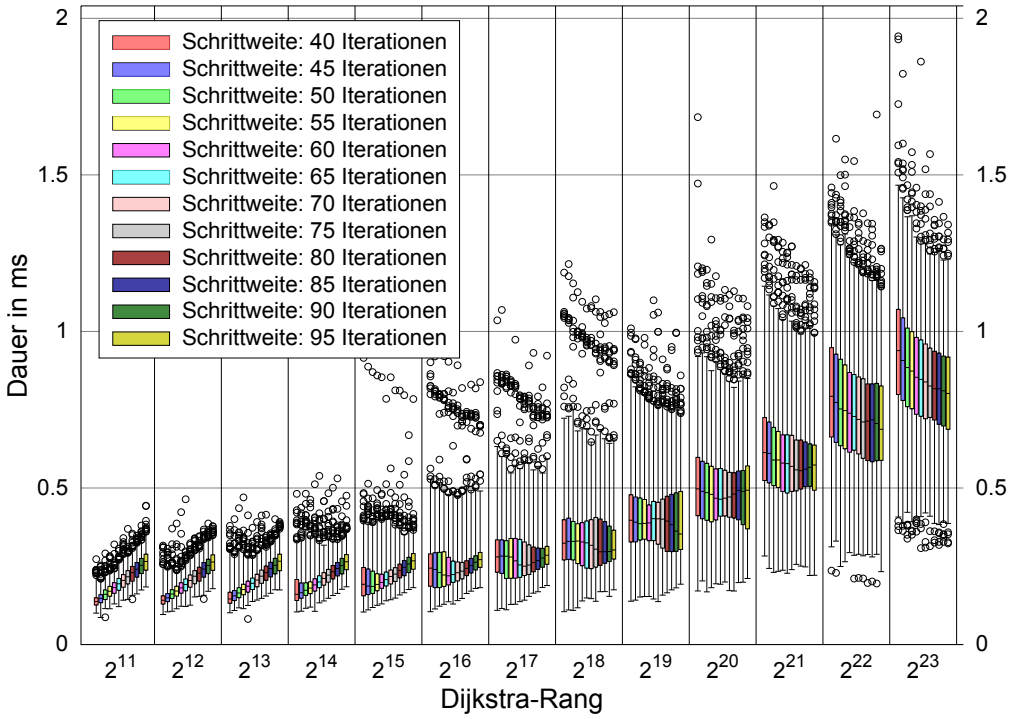


(b) Suche nach *schnellsten* Wegen

Abbildung B.7: Benchmark für parallele Standard-Wegsuchen in Contraction Hierarchies (Verfahren 11, S. 138)
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.27, S. 194.

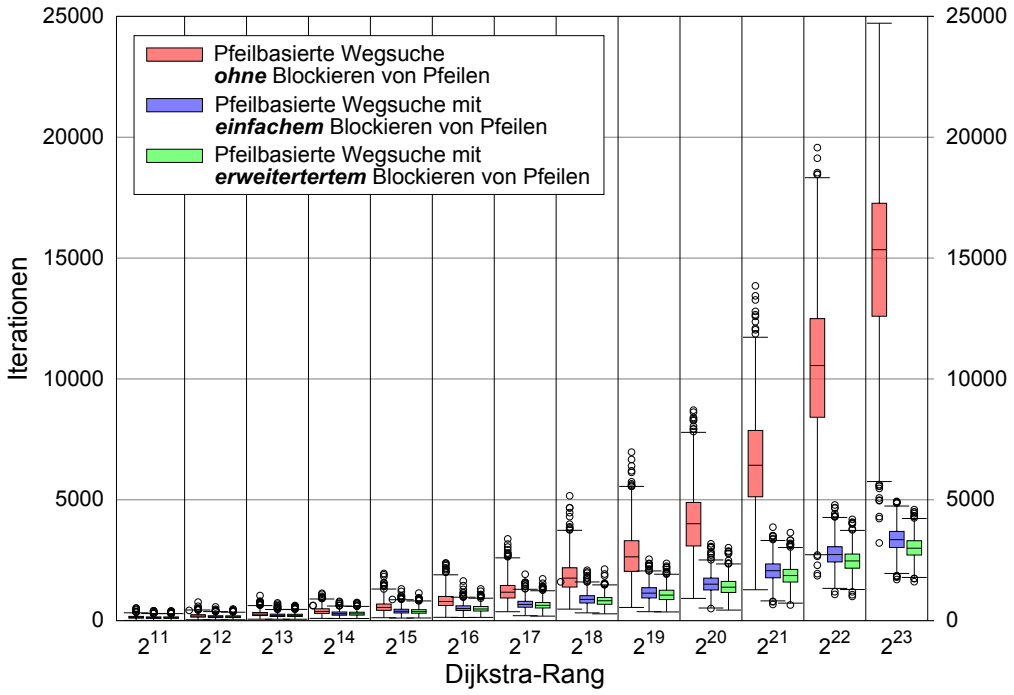


(a) Suche nach *kürzesten* Wegen

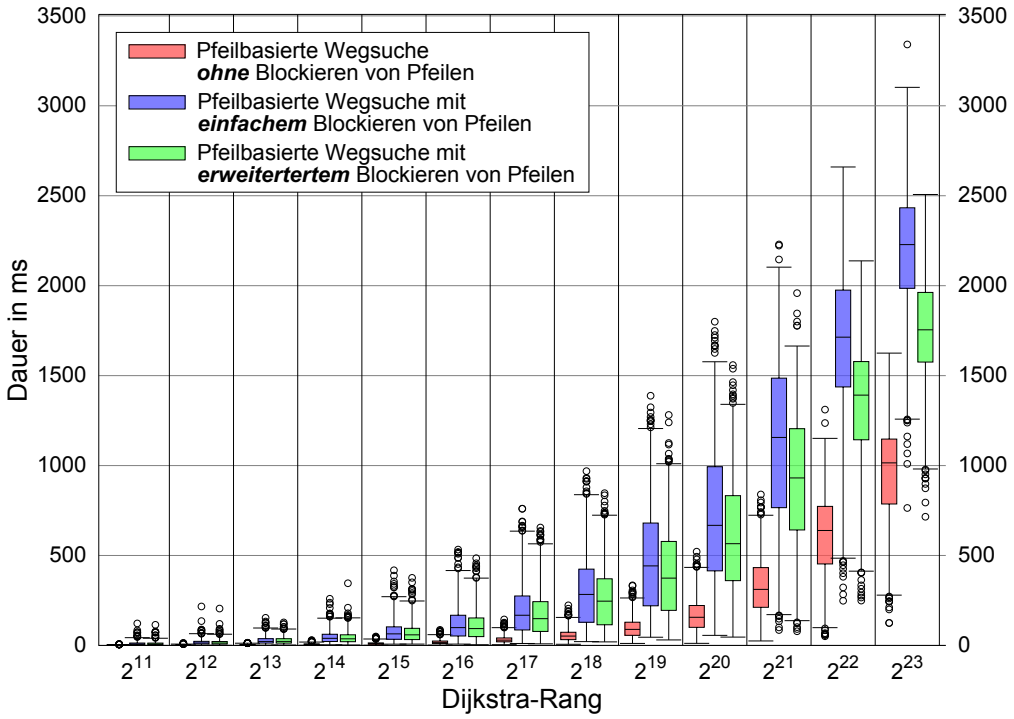


(b) Suche nach *schnellsten* Wegen

Abbildung B.8: Benchmark für parallele, zielgerichtete Wegsuchen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 5.28, S. 195.

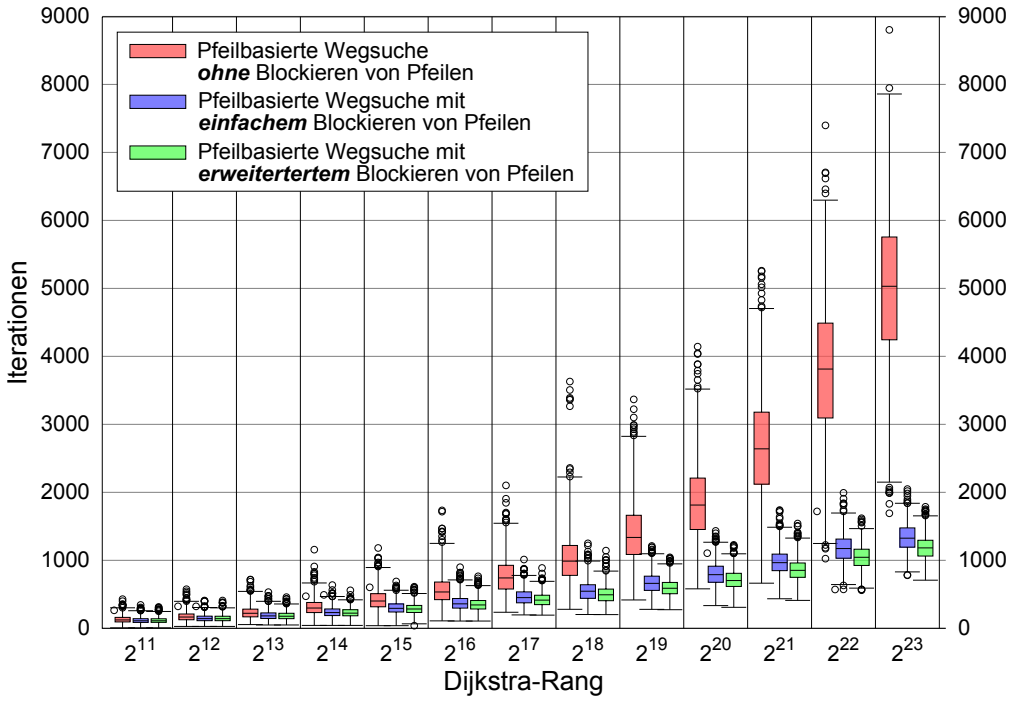


(a) Benötigte Iterationen

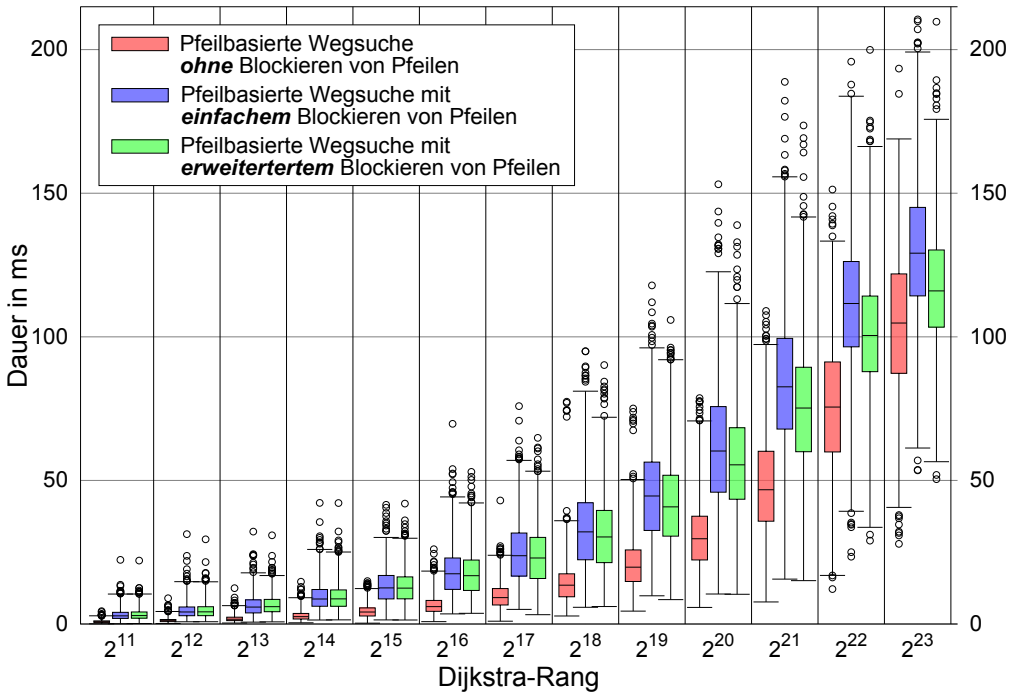


(b) Suchdauer

Abbildung B.9: Benchmark für Pfeilblockierung bei Suchen nach *kürzesten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.17, S. 241.

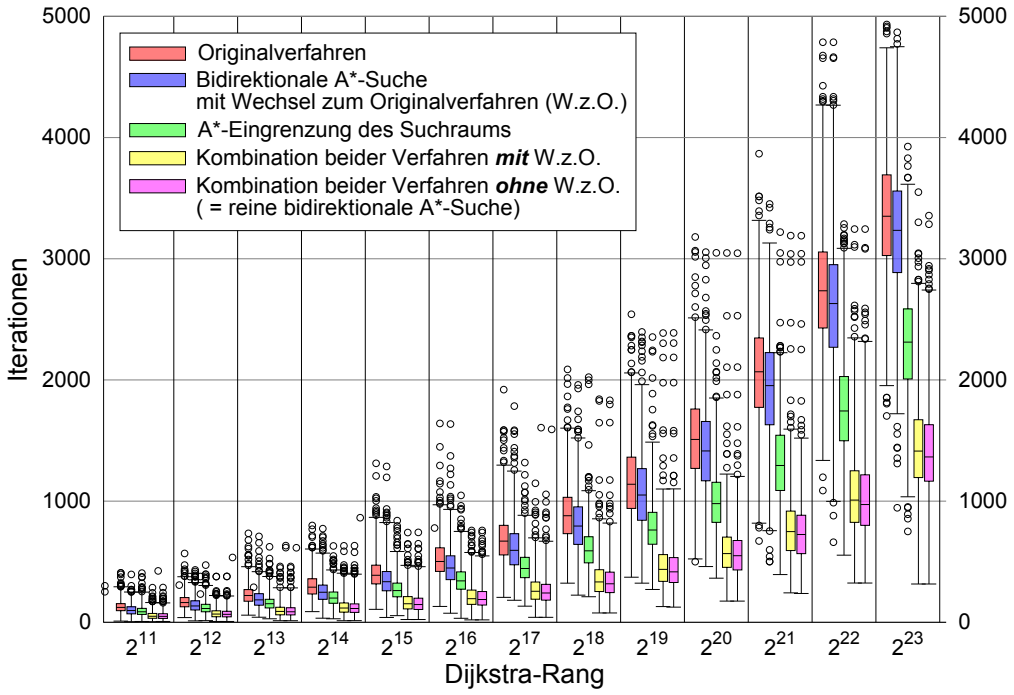


(a) Benötigte Iterationen

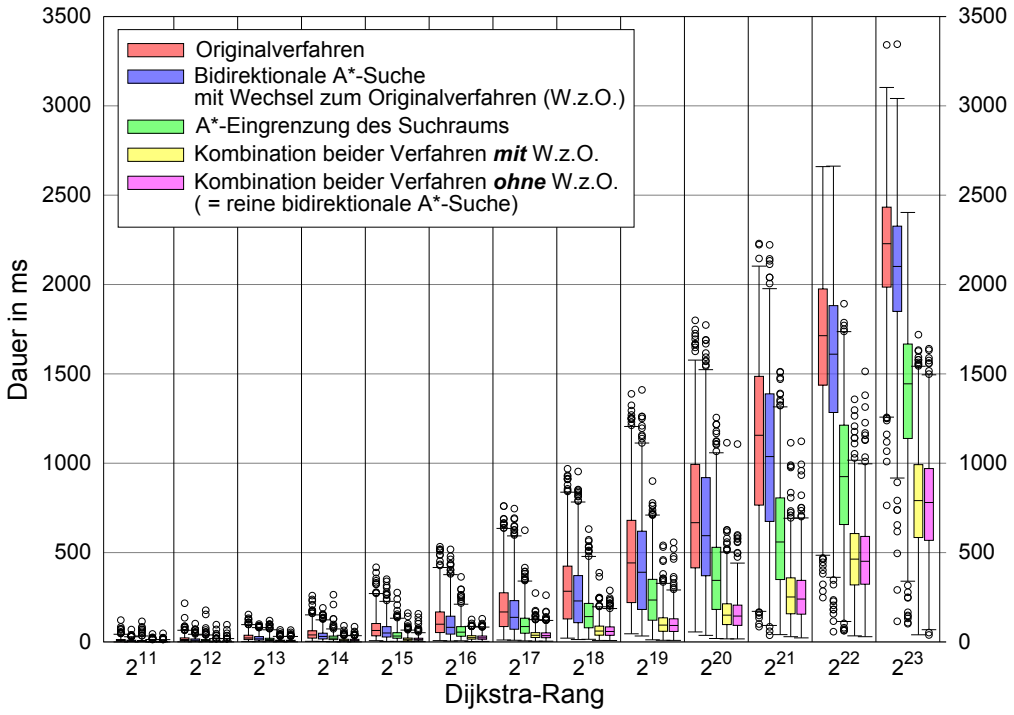


(b) Suchdauer

Abbildung B.10: Benchmark für Pfeilblockierung bei Suchen nach *schnellsten* Wegen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.18, S. 242.

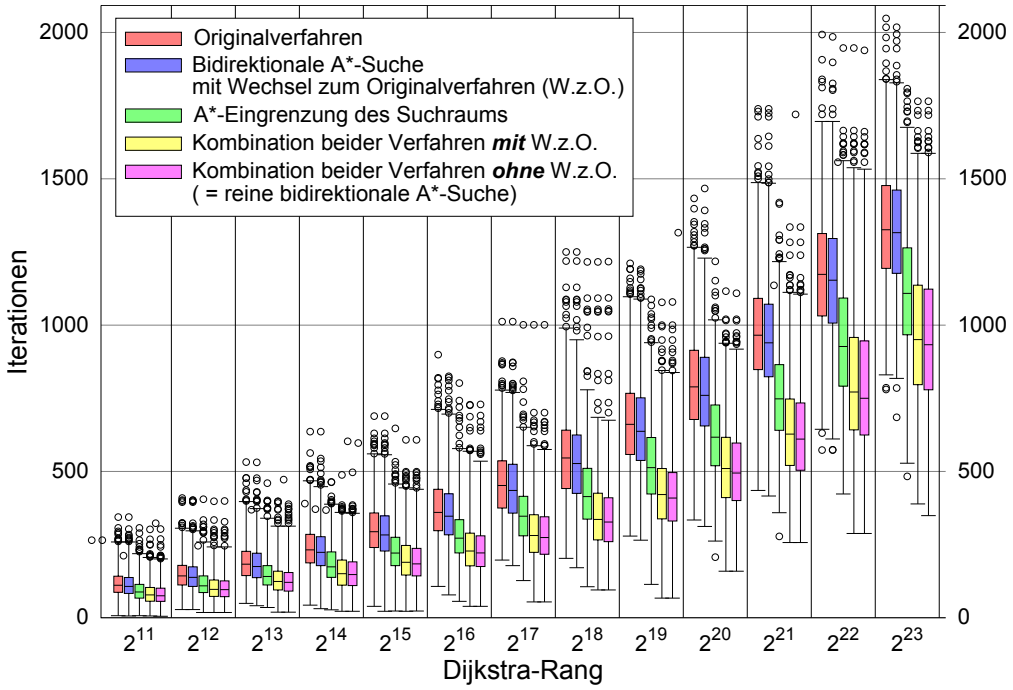


(a) Benötigte Iterationen

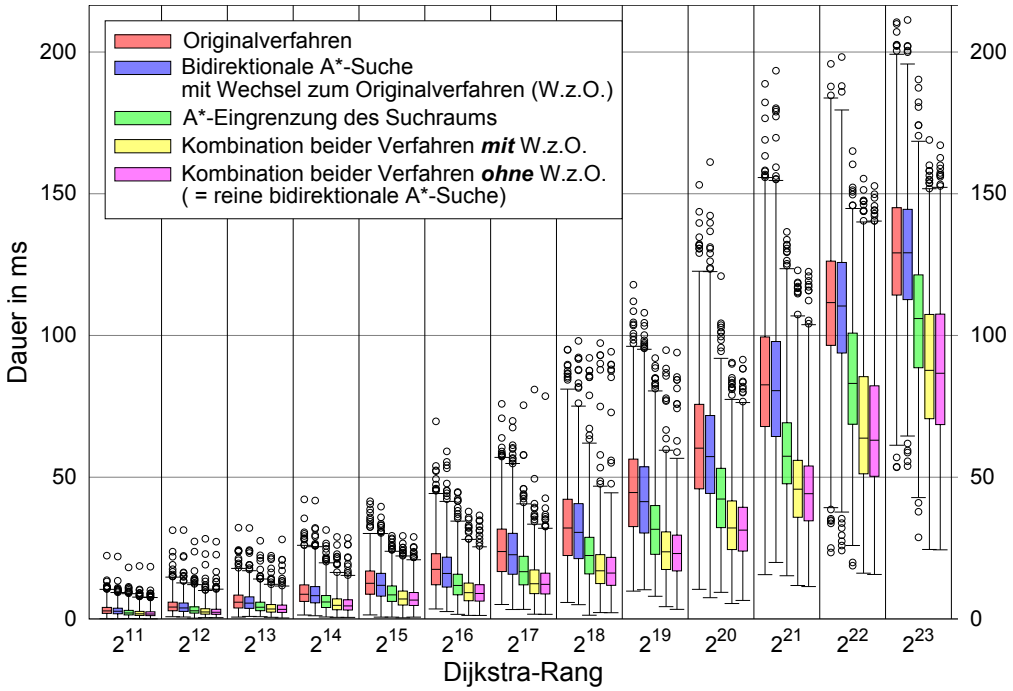


(b) Suchdauer

Abbildung B.11: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.19, S. 244.

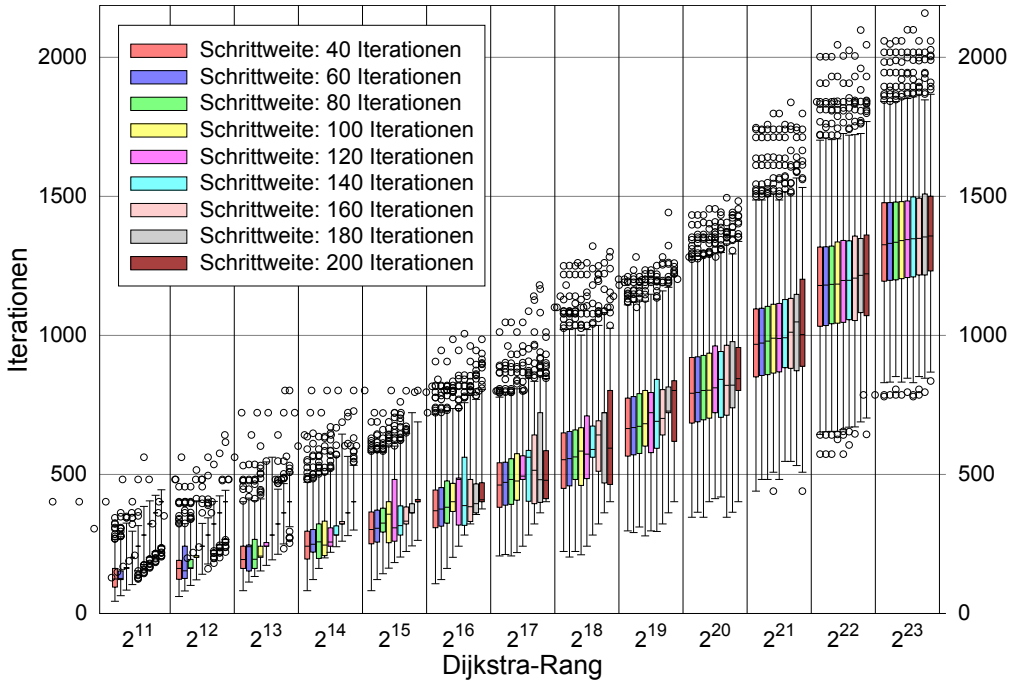


(a) Benötigte Iterationen

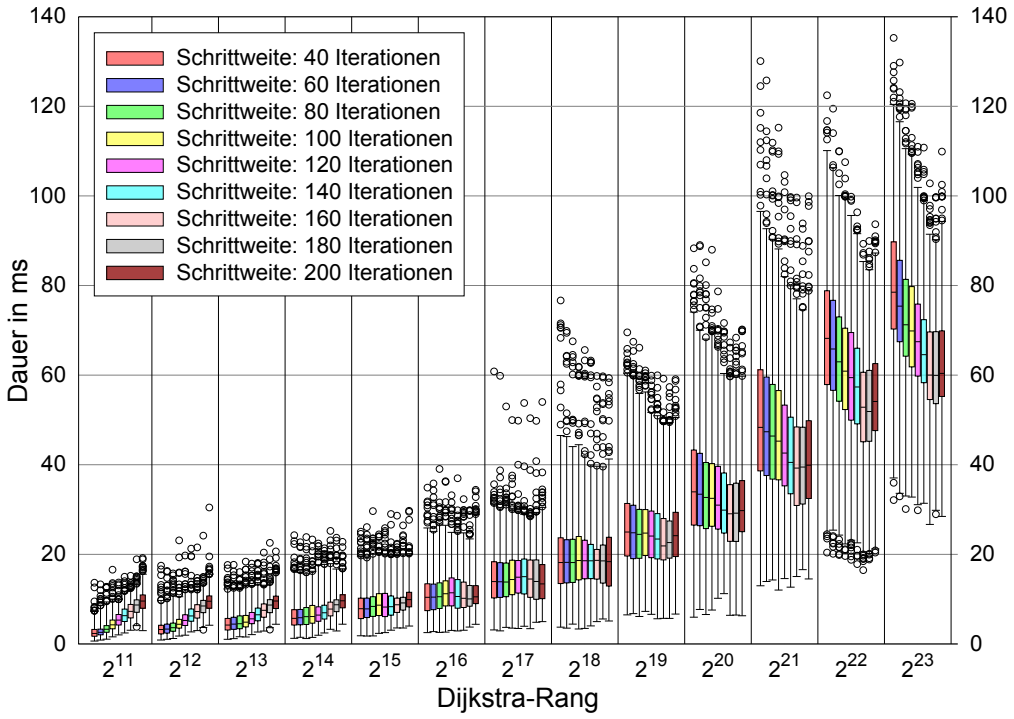


(b) Suchdauer

Abbildung B.12: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.20, S. 245.

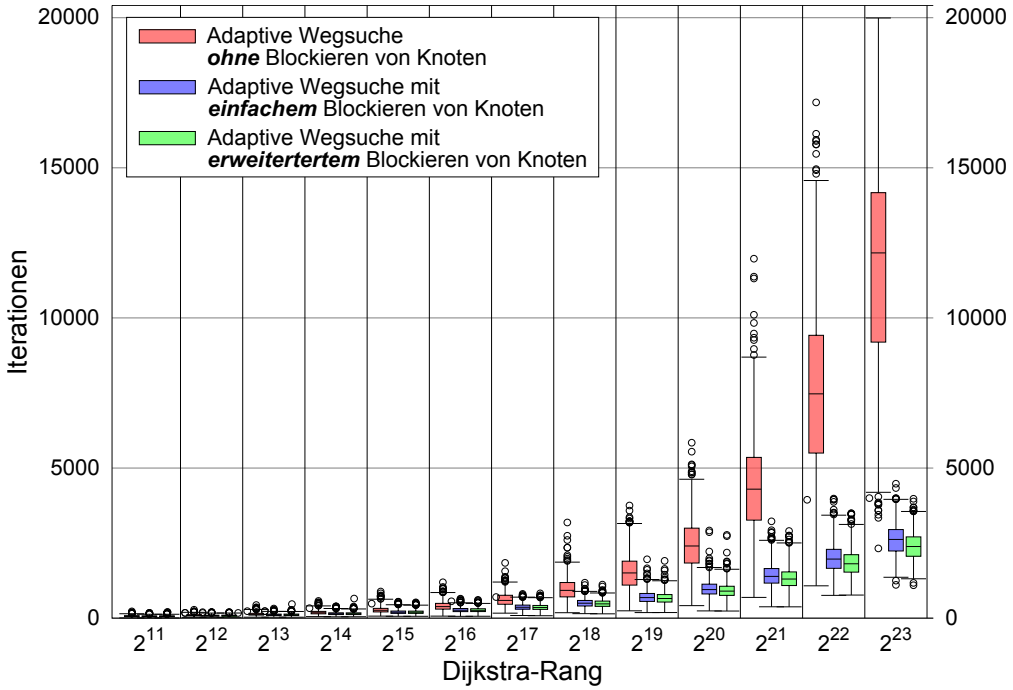


(a) Benötigte Iterationen

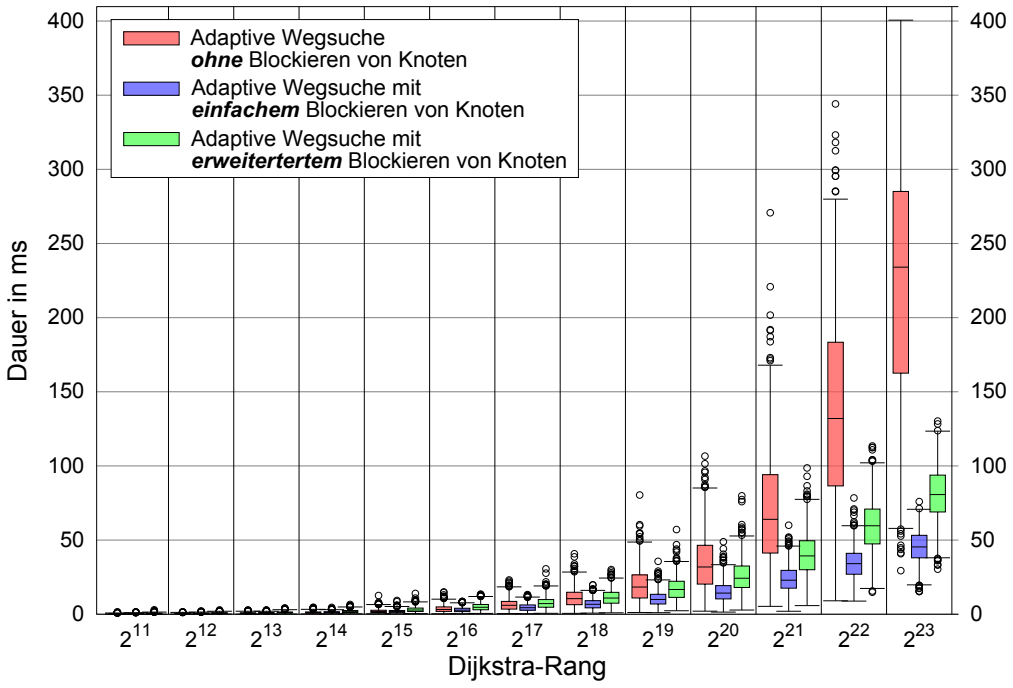


(b) Suchdauer

Abbildung B.13: Benchmark für parallele pfeilbasierte Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.22, S. 248.

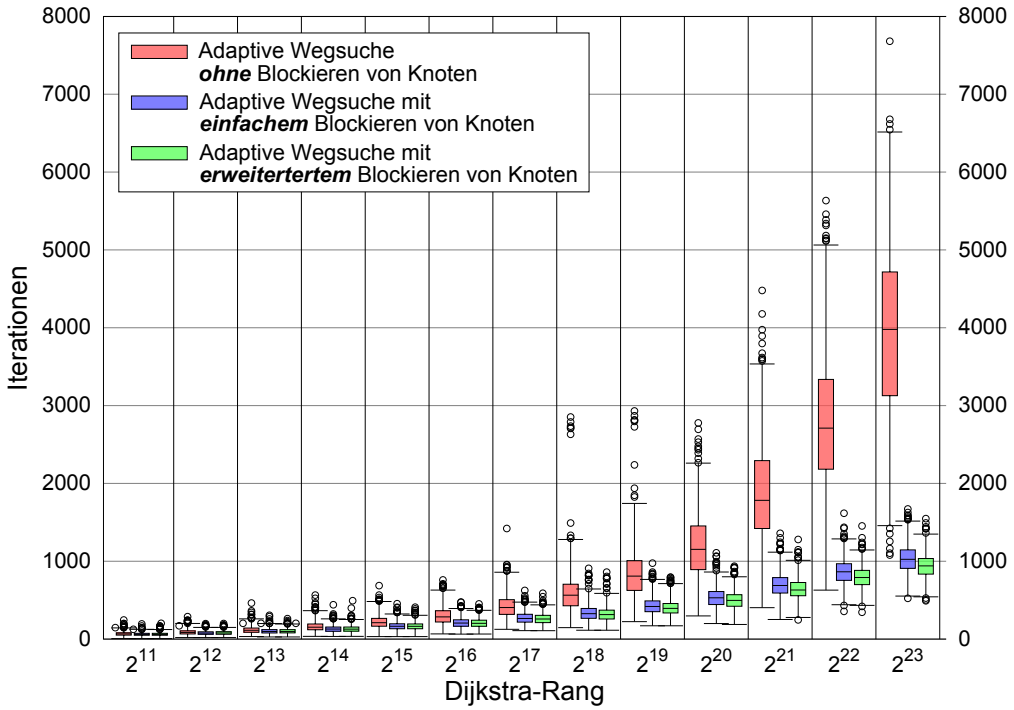


(a) Benötigte Iterationen

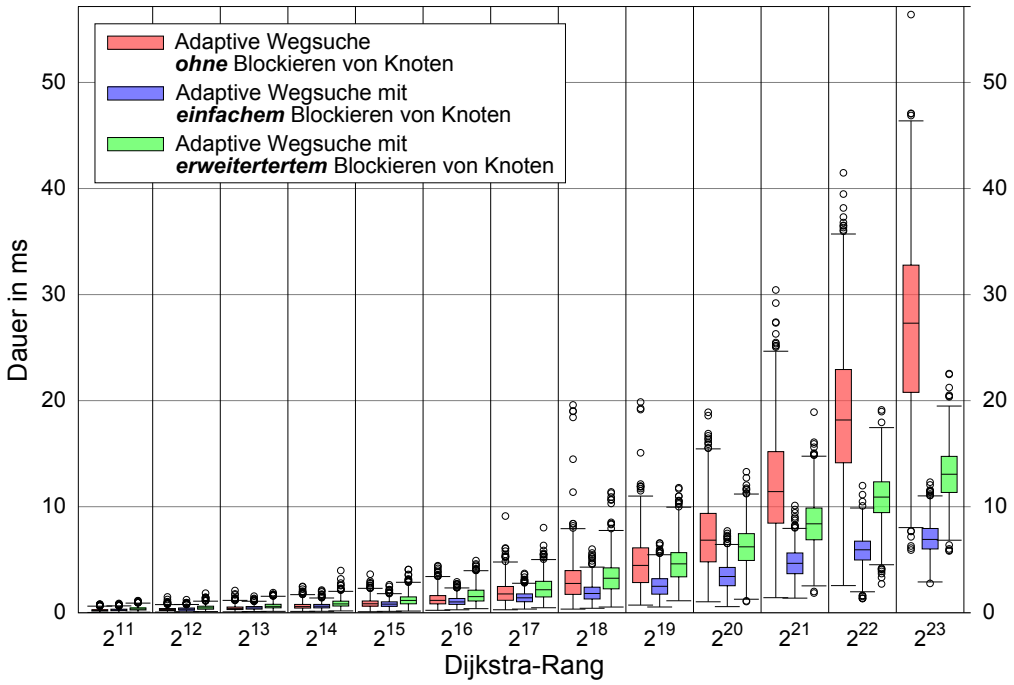


(b) Suchdauer

Abbildung B.14: Benchmark für Knotenblockierung bei Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.23, S. 250.

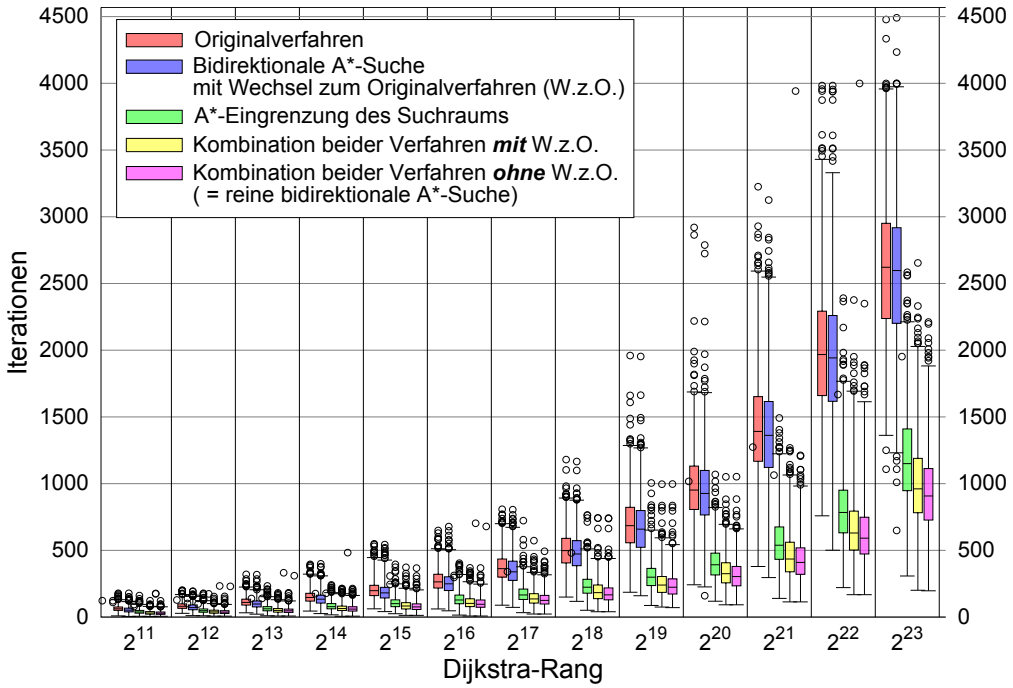


(a) Benötigte Iterationen

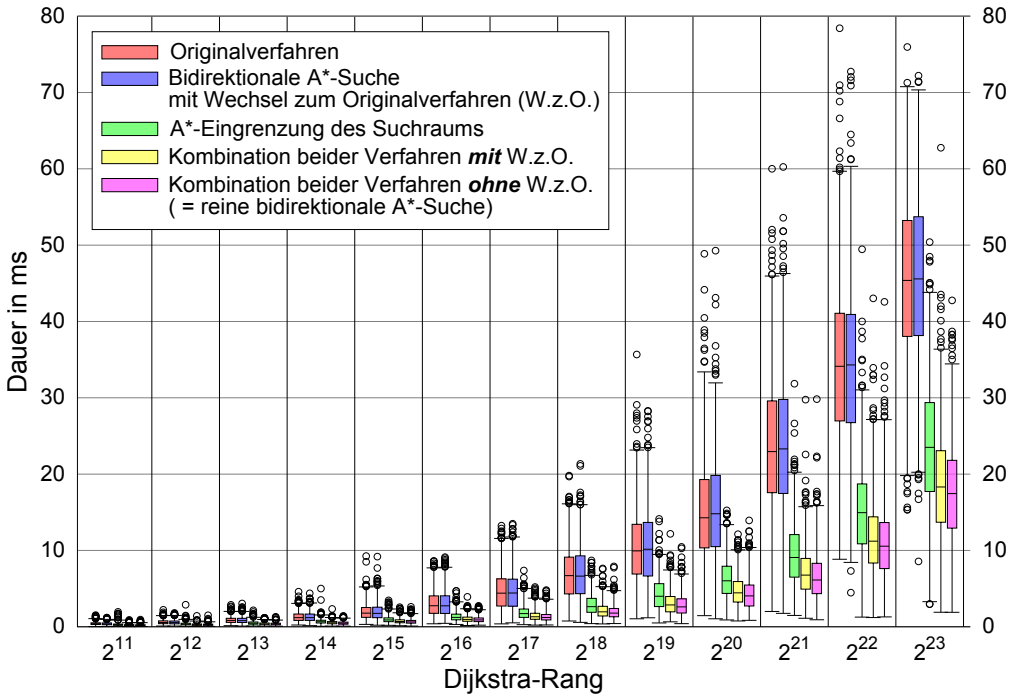


(b) Suchdauer

Abbildung B.15: Benchmark für Knotenblockierung bei Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.24, S. 251.

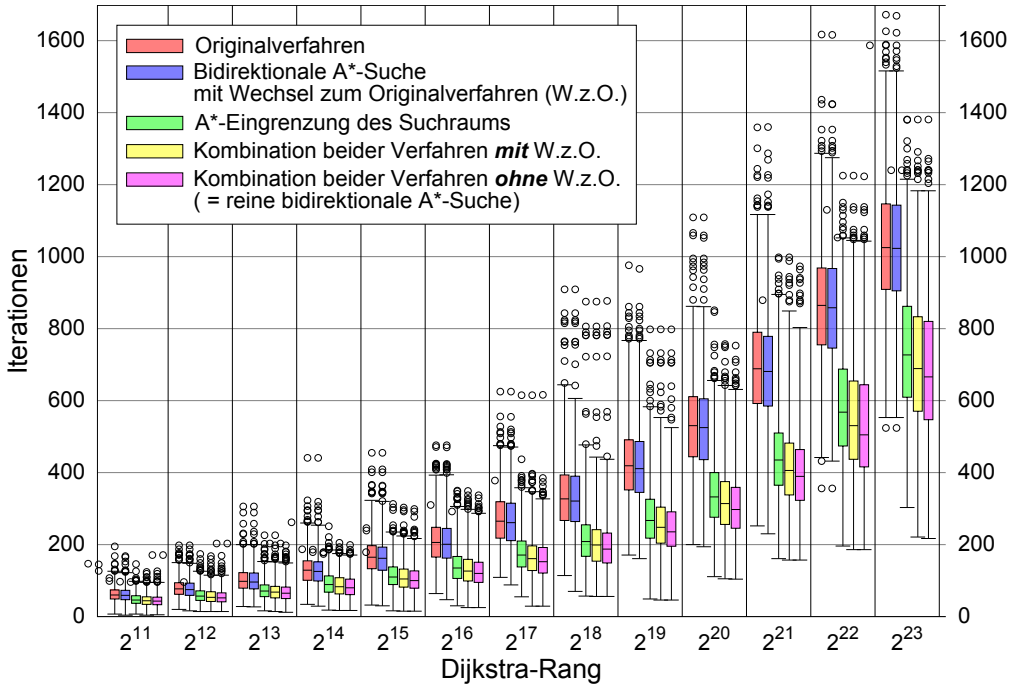


(a) Benötigte Iterationen

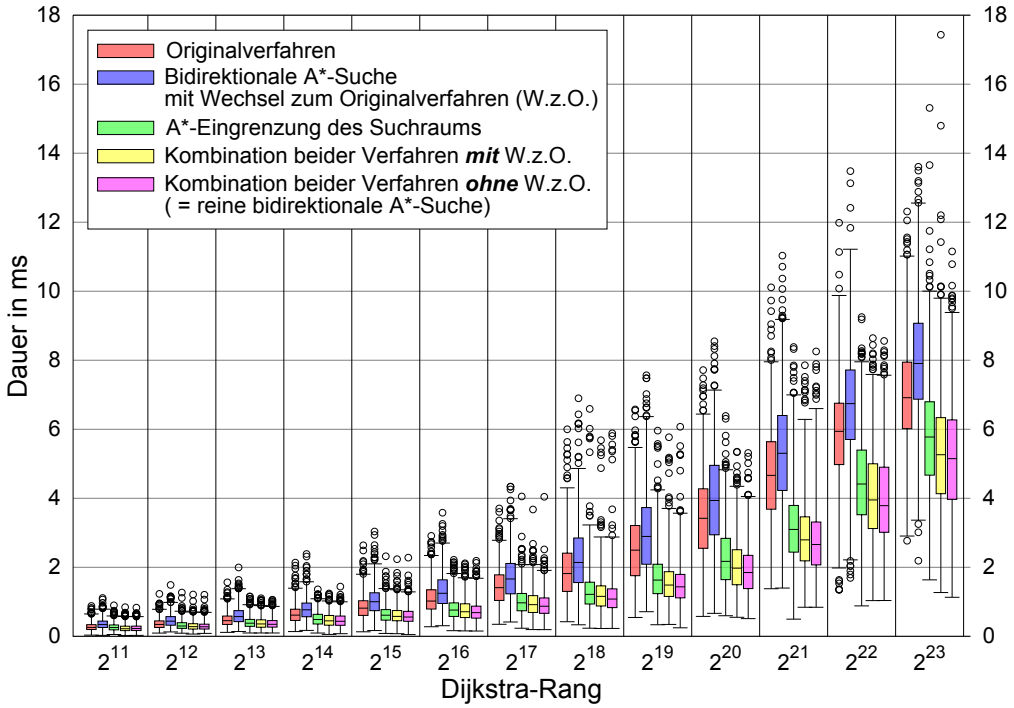


(b) Suchdauer

Abbildung B.16: Benchmark für zielgerichtete Suchen nach *kürzesten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.25, S. 254.

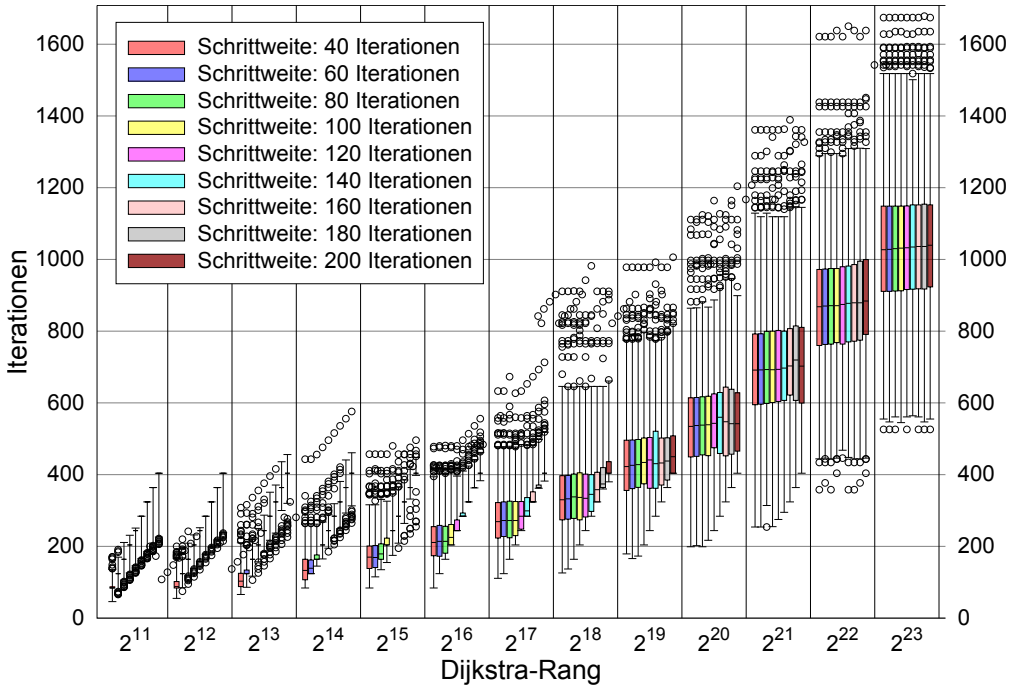


(a) Benötigte Iterationen

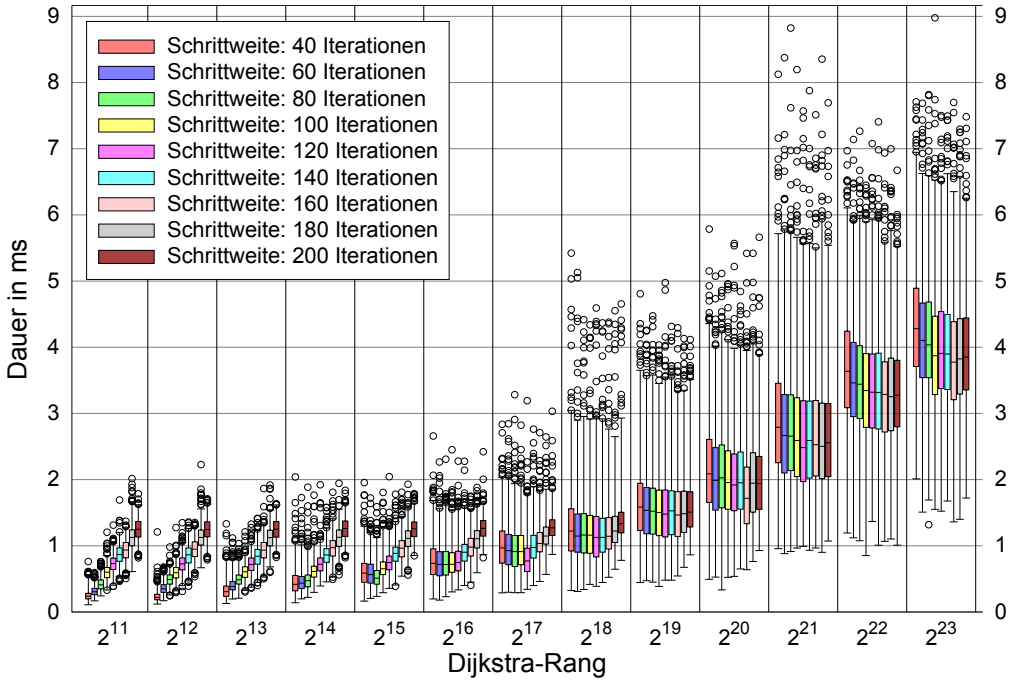


(b) Suchdauer

Abbildung B.17: Benchmark für zielgerichtete Suchen nach *schnellsten* Wegen in Contraction Hierarchies mit Abbiegebeschränkungen
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.26, S. 255.

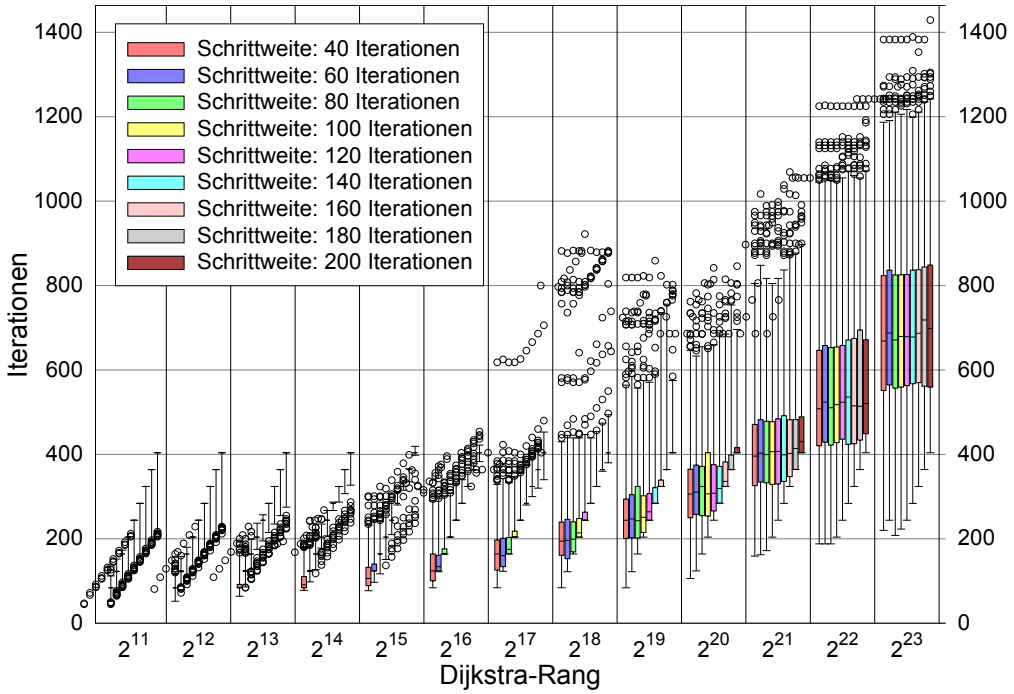


(a) Benötigte Iterationen

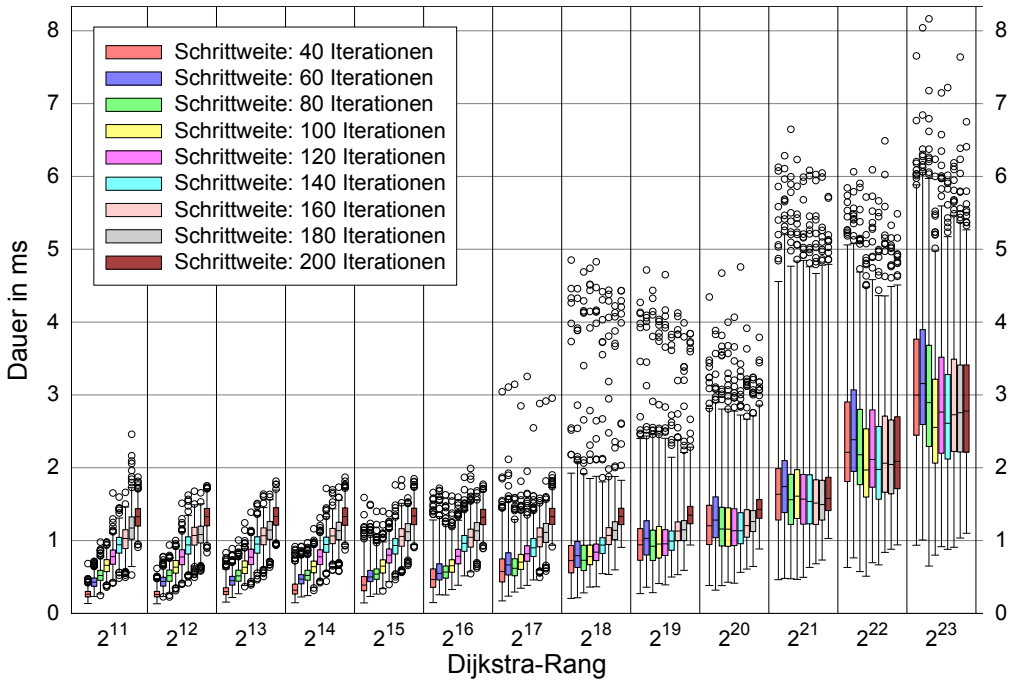


(b) Suchdauer

Abbildung B.18: Benchmark für parallele, adaptive Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen (parallele Ausführung von Verfahren 22, S. 226)
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.29, S. 258.



(a) Benötigte Iterationen

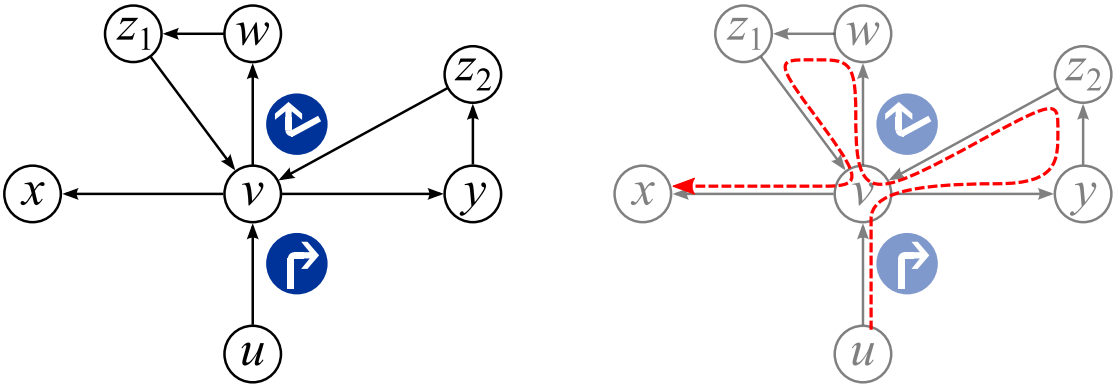


(b) Suchdauer

Abbildung B.19: Benchmark für parallele, adaptive A*-Suchen nach schnellsten Wegen in Contraction Hierarchies mit Abbiegebeschränkungen (basierend auf Verfahren 22, S. 226)
Das zugehörige Diagramm ohne Antennen zeigt Abbildung 6.30, S. 259.

Anhang C

Beispiel für Weg mit Mehrfach-Zyklus



(a) Beispielgraph

(b) Einziger zulässiger und daher optimaler Weg von Knoten u nach x

Abbildung C.1: Beispiel für optimalen Weg mit Mehrfach-Zyklus
Anstelle von Abbiegeverboten ist die Situation der Übersichtlichkeit halber durch Abbiegegebote dargestellt.
Pfeilbewertungen sind für dieses Beispiel unerheblich und daher nicht mit angegeben.

Literaturverzeichnis

[Abraham u. a. 2010a] Abraham, Ittai ; Delling, Daniel ; Goldberg, Andrew V.: A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In: *Proceedings of the 10th International Symposium on Experimental Algorithms SEA11* (2010), Nr. v

[Abraham u. a. 2011] Abraham, Ittai ; Delling, Daniel ; Goldberg, Andrew V. ; Werneck, Renato F.: A hub-based labeling algorithm for shortest paths in road networks. In: Pardalos, Panos M. (Hrsg.) ; Rebennack, Steffen (Hrsg.): *Experimental Algorithms*. Berlin : Springer, 2011. – ISBN 978–3–642–20661–0, S. 230–241

[Abraham u. a. 2010b] Abraham, Ittai ; Fiat, Amos ; Goldberg, Andrew V. ; Werneck, Renato F.: Highway dimension, shortest paths, and provably efficient algorithms. In: Charikar, Moses (Hrsg.): *Proceedings of the twenty-first annual ACM SIAM Symposium on Discrete Algorithms*. New York : ACM Press, 2010. – ISBN 0898716985, 782–793

[Alter 1980] Alter, Steven: *Decision support systems: Current practice and continuing challenges*. Reading, Mass : Addison-Wesley, 1980 (Addison-Wesley series on decision support). – ISBN 0201001934

[ArbZG 2012] ArbZG: *Arbeitszeitgesetz vom 6. Juni 1994 (BGBl. I S. 1170, 1171), das zuletzt durch Artikel 3 Absatz 6 des Gesetzes vom 20. April 2013 (BGBl. I S. 868) geändert worden ist*. `http://www.gesetze-im-internet.de/arbzg/BJNR117100994.html`. Version: 2012, Abruf: 28.12.2013

[Archetti u. a. 2006] Archetti, C. ; Speranza, M. G. ; Hertz, A.: A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem. In: *Transportation Science* 40 (2006), Nr. 1, S. 64–73. `http://dx.doi.org/10.1287/trsc.1040.0103`. – DOI 10.1287/trsc.1040.0103

[Bachmann 1894] Bachmann, P.G.H: *Die analytische Zahlentheorie*. Teubner, 1894 (Zahlentheorie. Versuch einer Gesammtdarstellung dieser Wissenschaft in ihren Haupttheilen). `http://openlibrary.org/books/OL22878421M/Die_analytische_Zahlentheorie`.

[Bast u. a. 2007a] Bast, H. ; Funke, S. ; Matijevic, D. ; Sanders, R. ; Schultes, Dominik: In Transit to Constant Shortest-Path Queries in Road Networks. In: *Proceedings of ALENEX 2007, SIAM* (2007), 46–59. `http://algo2.iti.kit.edu/1051.php`

[Bast u. a. 2007b] Bast, H. ; Funke, S. ; Sanders, P. ; Schultes, D.: Fast Routing in Road Networks with Transit Nodes. In: *Science* 316 (2007), Nr. 5824, S. 566.

http://dx.doi.org/10.1126/science.1137521. – DOI 10.1126/science.1137521. – ISSN 0036–8075

[Bast u. a. 2009] Bast, Holger ; Funke, Stefan ; Matuevic, Domagoj: Ultrafast Shortest-Path Queries via Transit Nodes. In: Demetrescu, Camil (Hrsg.) ; Goldberg, Andrew V. (Hrsg.) ; Johnson, David S. (Hrsg.): *The shortest path problem.* Providence : American Mathematical Society, 2009. – ISBN 0821843834, S. 175–192

[Bauer u. a. 2008] Bauer, Reinhard ; Delling, Daniel ; Sanders, Peter ; Schieferdecker, Dennis ; Schultes, Dominik ; Wagner, Dorothea: Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In: MacGeoch, Catherine C. (Hrsg.): *Experimental algorithms.* Berlin [etc.] : Springer, 2008. – ISBN 9783540685487, S. 303–318

[Bauer u. a. 2010] Bauer, Reinhard ; Delling, Daniel ; Sanders, Peter ; Schieferdecker, Dennis ; Schultes, Dominik ; Wagner, Dorothea: Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In: *ACM Journal of Experimental Algorithmics* 2010 (2010), Nr. 15, 2.3:2.1–2.3:2.31. http://algo2.iti.kit.edu/1328.php

[Bayer 1972] Bayer, Rudolf: Symmetric binary B-Trees: Data structure and maintenance algorithms. In: *Acta Informatica* 1 (1972), Nr. 4, S. 290–306. http://dx.doi.org/10.1007/BF00289509. – DOI 10.1007/BF00289509. – ISSN 0001–5903

[Bonczek u. a. 1981] Bonczek, R. H. ; Holsapple, C. W. ; Whinston, Andrew B.: *Foundations of decision support systems.* New York : Academic Press, 1981 (Operations research and industrial engineering). – ISBN 0–12–113050–9

[Caldwell 1961] Caldwell, Tom: On finding minimum routes in a network with turn penalties. In: *Communications of the ACM* 4 (1961), Nr. 2, S. 107–108. http://dx.doi.org/10.1145/366105.366184. – DOI 10.1145/366105.366184. – ISSN 00010782

[Clarke u. Wright 1964] Clarke, G. ; Wright, J. W.: Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. In: *Operations Research* 12 (1964), Nr. 4, S. 568–581. http://dx.doi.org/10.1287/opre.12.4.568. – DOI 10.1287/opre.12.4.568

[Continental Automotive GmbH 2007] Continental Automotive GmbH: *Mit großen Schritten in die digitale Zukunft.* http://www.vdo.de/generator/www/de/de/vdo/main/press/Archiv/commercial_vehicles/2007/sv_200704_002_d_de.html. Version: 2007, Abruf: 28.12.2013 (Pressemitteilung vom 10. April 2007)

[Continental Automotive GmbH 2008] Continental Automotive GmbH: *Modularer Tachograph: MTCO 1324.* http://www.vdo.de/generator/www/de/de/vdo/main/products_solutions/commercial_vehicles/tachographs/analogue_tachographs/download/flc_mtco1324_de_v2_2008_de.pdf. Version: 2008, Abruf: 28.12.2013 (Datenblatt des MTCO 1324)

[Cormen u. a. 2001] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to algorithms*. 2. Aufl. Cambridge, Mass, New York : MIT Press, 2001. – ISBN 0–262–03293–7

[Dantzig 1960] Dantzig, George B.: On the Shortest Route Through a Network. In: *Management Science* 6 (1960), Nr. 2, S. 187–190

[Delling u. a. 2009a] Delling, Daniel ; Sanders, Peter ; Schultes, Dominik ; Wagner, Dorothea: Engineering Route Planning Algorithms. In: Lerner, Jürgen (Hrsg.) ; Wagner, Dorothea (Hrsg.) ; Zweig, Katharina A. (Hrsg.): *Lecture Notes in Computer Science* Bd. 5515. Berlin, Heidelberg : Springer, 2009. – ISBN 978–3–642–02093–3, S. 117–139

[Delling u. a. 2009b] Delling, Daniel ; Sanders, Peter ; Schultes, Dominik ; Wagner, Dorothea: Highway Hierarchies Star. In: Demetrescu, Camil (Hrsg.) ; Goldberg, Andrew V. (Hrsg.) ; Johnson, David S. (Hrsg.): *The shortest path problem*. Providence : American Mathematical Society, 2009. – ISBN 0821843834, S. 141–169

[Dijkstra 1959] Dijkstra, Edsger W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), Nr. 1, S. 269–271. `http://dx.doi.org/10.1007/BF01386390`. – DOI 10.1007/BF01386390. – ISSN 0029–599X

[Domschke 1997] Domschke, Wolfgang: *Rundreisen und Touren*. 4. Aufl. München [u.a.] : Oldenbourg, 1997. – ISBN 3–486–24273–3

[Domschke 2007] Domschke, Wolfgang: *Logistik: Transport: Grundlagen, lineare Transport- und Umladeprobleme*. 5. Aufl. München [u.a.] : Oldenbourg, 2007. – ISBN 9783486582901

[Domschke u. Drexl 2007] Domschke, Wolfgang ; Drexl, Andreas: *Einführung in Operations Research: Mit 63 Tabellen*. 7. Aufl. Berlin : Springer, 2007 (Springer-Lehrbuch). – ISBN 978–3540709480

[Drexl 2012] Drexl, Michael: Rich vehicle routing in theory and practice. In: *Logistics Research* 5 (2012), Nr. 1-2, S. 47–63. `http://dx.doi.org/10.1007/s12159-012-0080-2`. – DOI 10.1007/s12159–012–0080–2. – ISSN 1865–035X

[Dreyfus 1969] Dreyfus, Stuart E.: An Appraisal of Some Shortest-Path Algorithms. In: *Operations Research* 17 (1969), Nr. 3, S. 395–412

[Dubois u. Semet 1995] Dubois, Nicolas ; Semet, Frédéric: Estimation and determination of shortest path length in a road network with obstacles. In: *European Journal of Operational Research* 83 (1995), Nr. 1, S. 105–116. `http://dx.doi.org/10.1016/0377-2217(94)00015-5`. – DOI 10.1016/0377–2217(94)00015–5. – ISSN 03772217

[Elmasri u. Navathe 2009] Elmasri, Ramez ; Navathe, Sham: *Grundlagen von Datenbanksystemen: Bachelorausgabe*. 3. Aufl. München [u.a.] : Pearson Studium, 2009 `http://www.worldcat.org/oclc/316310566`. – ISBN 386894012X

[Europäisches Parlament 2006] Europäisches Parlament: Verordnung (EG) Nr. 561/2006 des Europäischen Parlaments und des Rates vom 15. März 2006 zur Harmonisierung bestimmter Sozialvorschriften im Straßenverkehr und zur Änderung der Verordnungen (EWG) Nr. 3821/85 und (EG) Nr. 2135/98 des Rates sowie zur Aufhebung der Verordnung (EWG) Nr. 3820/85 des Rates: VO (EG) 561/2006. In: *Amtsblatt der Europäischen Union* (2006), Nr. L 102, 1–14. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006R0561:DE:HTML

[FPersV 2005] FPersV: *Fahrpersonalverordnung vom 27. Juni 2005 (BGBl. I S. 1882), die zuletzt durch Artikel 2 der Verordnung vom 19. Dezember 2011 (BGBl. I S. 2835) geändert worden ist.* http://www.gesetze-im-internet.de/fpersv/BJNR188210005.html. Version: 2005, Abruf: 28.12.2013

[Fredman u. Tarjan 1987] Fredman, Michael L. ; Tarjan, Robert E.: Fibonacci heaps and their uses in improved network optimization algorithms. In: *Journal of the ACM* 34 (1987), Nr. 3, S. 596–615. http://dx.doi.org/10.1145/28869.28874. – DOI 10.1145/28869.28874. – ISSN 00045411

[Gaul u. Both 1990] Gaul, W. ; Both, Martin: *Computergestütztes Marketing.* Berlin, New York : Springer, 1990. – ISBN 3–540–52310–3

[Geisberger 2008] Geisberger, Robert: *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.* Karlsruhe, Universität Karlsruhe, Diplomarbeit, 2008

[Geisberger u. a. 2008] Geisberger, Robert ; Sanders, Peter ; Schultes, Dominik ; Delling, Daniel: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)* 5038 of Lecture Notes in Computer Science (2008), 319–333. http://i11www.iti.uni-karlsruhe.de/extra/publications/gssd-chfsh-08.pdf

[Geisberger u. Vetter 2011] Geisberger, Robert ; Vetter, Christian: Efficient Routing in Road Networks with Turn Costs. In: Pardalos, Panos M. (Hrsg.) ; Rebennack, Steffen (Hrsg.): *Experimental Algorithms.* Berlin : Springer, 2011. – ISBN 3642206611, S. 100–111

[Goel 2010] Goel, A.: Truck Driver Scheduling in the European Union. In: *Transportation Science* 44 (2010), Nr. 4, S. 429–441. http://dx.doi.org/10.1287/trsc.1100.0330. – DOI 10.1287/trsc.1100.0330

[Goldberg u. Harrelson 2005] Goldberg, Andrew V. ; Harrelson, Chris: Computing the shortest path: A* search meets graph theory. In: *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms.* Philadelphia : Society for Industrial and Applied Mathematics, 2005 (SODA '05). – ISBN 0–89871–585–7, 156–165

[Google Inc 2009] Google Inc: *Terms of Service für Google Maps/Google Earth API (Stand 27. Mai 2009).* http://code.google.com/intl/de-DE/apis/maps/terms.html. Version: 2009, Abruf: 11.03.2010

[Gorry u. Scott Morton 1971] Gorry, G. A. ; Scott Morton, Michael S.: A Framework for Management Information Systems. In: *Sloan Management Review* (1971), S. 21–36

[Hahne 2000] Hahne, Felix: *Kürzeste und schnellste Wege in digitalen Straßenkarten.* Hildesheim, Universität Hildesheim, Dissertation, 2000

[Hahne u. a. 2008] Hahne, Felix ; Nowak, Curt ; Ambrosi, Klaus: Acceleration of the A*-Algorithm for the Shortest Path Problem in Digital Road Maps. In: Kalcsics, Jörg (Hrsg.) ; Nickel, Stefan (Hrsg.): *Operations Research Proceedings 2007 – Selected Papers of the Annual International Conference of the German Operations Research Society (GOR).* Berlin, Heidelberg : Springer, 2008. – ISBN 3540779027, S. 455–460

[Hart u. a. 1968] Hart, Peter ; Nilsson, Nils ; Raphael, Bertram: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems Science and Cybernetics* 4 (1968), Nr. 2, S. 100–107. – ISSN 0536–1567

[IN tIME Express Logistik GmbH 2013] IN tIME Express Logistik GmbH: *IN tIME: Zahlen & Fakten.* `http://www.intime.de/ueber-intime/zahlen-fakten.html`. Version: 2013, Abruf: 21.06.2013

[Jünemann u. a. 1989] Jünemann, Reinhardt ; Daum, M. ; Piepel, U. ; Schwinning, St.: *Materialfluss und Logistik: Systemtechnische Grundlagen mit Praxisbeispielen.* Berlin : Springer, 1989 (Logistik in Industrie, Handel und Dienstleistungen). – ISBN 9780387512259

[Kimchi 2010] Kimchi, Gur: *Bing engages open maps community: Bing Maps Blog: Blog-Eintrag vom 23.11.2010.* `http://www.bing.com/community/site_blogs/b/maps/archive/2010/11/23/bing-engages-open-maps-community.aspx`. Version: 2010, Abruf: 22.02.2011. – Blogeintrag

[Knuth 1974] Knuth, Donald E.: Structured Programming with go to Statements. In: *ACM Computing Surveys* 6 (1974), Nr. 4, S. 261–301. `http://dx.doi.org/10.1145/356635.356640`. – DOI 10.1145/356635.356640. – ISSN 03600300

[Knuth 1976] Knuth, Donald E.: Big Omicron and big Omega and big Theta. In: *ACM SIGACT News* 8 (1976), Nr. 2, S. 18–24. `http://dx.doi.org/10.1145/1008328.1008329`. – DOI 10.1145/1008328.1008329. – ISSN 01635700

[Köhler u. a. 2009] Köhler, Ekkehard ; Möhring, Rolf H. ; Schilling, Heiko: Fast Point-to-Point Shortest Path Computations with Arc-Flags. Version: 2009. `www.dis.uniroma1.it/~challenge9/papers/kohler.pdf`. In: Demetrescu, Camil (Hrsg.) ; Goldberg, Andrew V. (Hrsg.) ; Johnson, David S. (Hrsg.): *The shortest path problem.* Providence : American Mathematical Society, 2009. – ISBN 0821843834, 41–72

[Kopfer u. Meyer 2009] Kopfer, Herbert ; Meyer, Christoph M.: A Model for the Traveling Salesman Problem including the EC Regulations on Driving Hours. In: Fleischmann, Bernhard (Hrsg.): *Operations Research Proceedings 2008 – Selected*

papers of the annual international conference of the German Operations Research Society (GOR). Berlin, Heidelberg : Springer, 2009. – ISBN 3642001416, S. 289–294

[Kraftfahrt-Bundesamt 2005] Kraftfahrt-Bundesamt ; Kraftfahrt-Bundesamt (Hrsg.): *Ausgabestellen für Fahrerkarten: Stand vom 14.07.2005.* `http://www.kba.de/cln_005/nn_124592/SharedDocs/FAQ/ZKR/Fahrerkarte/ausgabestellen__Fahrerkarten,templateId=raw,property=publicationFile.pdf/ausgabestellen_Fahrerkarten.pdf`. Version: 2005, Abruf: 26.01.2011

[Leuker u. Daniel 2007] Leuker, Reinhard ; Daniel, Anja: *Lenk- und Ruhezeiten im Straßenverkehr: Infobroschüre mit Arbeitszeitgesetz (ArbZG).* 4. Aufl. Verkehrs-Verlag Fischer, 2007. – ISBN 9783878413080

[LiteratePrograms 2009] LiteratePrograms: *Skip list (Java) – LiteratePrograms: Wiki-Seite vom Stand des 15. Januar 2009 (22:34).* `http://en.literateprograms.org/index.php?title=Skip_list_(Java)&oldid=15959`. Version: 2009, Abruf: 21.06.2013

[Neis u. a. 2012] Neis, Pascal ; Zielstra, Dennis ; Zipf, Alexander: The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007–2011. In: *Future Internet* 4 (2012), Nr. 1, S. 1–21. `http://dx.doi.org/10.3390/fi4010001`. – DOI 10.3390/fi4010001. – ISSN 1999–5903

[Nicholson 1966] Nicholson, T. A. J.: Finding the Shortest Route between Two Points in a Network. In: *The Computer Journal* 9 (1966), Nr. 3, S. 275–280

[Nowak u. a. 2009] Nowak, Curt ; Ambrosi, Klaus ; Hahne, Felix: An Application for Simulations at Large Pickup and Delivery Service Providers. In: Fischer, Stefan (Hrsg.): *Im Focus das Leben.* Bonn : Gesellschaft für Informatik, 2009. – ISBN 978–3–88579–248–2, S. 3620–3632

[Nowak u. a. 2012] Nowak, Curt ; Hahne, Felix ; Ambrosi, Klaus: Contraction Hierarchies with A* for digital road maps. In: Klatte, Diethard (Hrsg.) ; Lüthi, Hans-Jakob (Hrsg.) ; Schmedders, Karl (Hrsg.): *Operations Research Proceedings 2011 – Selected papers of the International Conference on Operations Research.* Berlin, Heidelberg : Springer, 2012. – ISBN 3642292097, S. 311–316

[Nowak u. a. 2014] Nowak, Curt ; Hahne, Felix ; Ambrosi, Klaus: Contraction Hierarchies with Turn Restrictions. In: Helber, Stefan (Hrsg.) ; Breitner, Michael (Hrsg.) ; Rösch, Daniel (Hrsg.) ; Schön, Cornelia (Hrsg.) ; Schulenburg, Johann-Matthias Graf von d. (Hrsg.) ; Sibbertsen, Philipp (Hrsg.) ; Steinbach, Marc (Hrsg.) ; Weber, Stefan (Hrsg.) ; Wolter, Anja (Hrsg.): *Operations Research Proceedings 2012 – Selected Papers of the International Annual Conference of the German Operations Research Society (GOR).* Cham, Heidelberg, New York, Dordrecht, London : Springer, 2014 (Operations Research Proceedings, GOR (Gesellschaft für Operations Research e.V.)). – ISBN 978–3–319–00794–6, S. 569–575

[Nowak u. a. 2008] Nowak, M. A. ; Özlem, E. ; White III, Chelsea C.: Pickup and Delivery with Split Loads. In: *Transportation Science* 42 (2008), Nr. 1, S. 32–43

[Oertel 2000] Oertel, Peter: *Routing with Reloads*. Köln, Universität zu Köln, Dissertation, 2000. http://e-archive.informatik.uni-koeln.de/423/

[OpenStreetMap contributors 2011] OpenStreetMap contributors: *OpenStreetMap Statistics*. http://www.openstreetmap.org/stats/data_stats.html. Version: 2011, Abruf: 22.02.2011

[Parragh u. a. 2008] Parragh, Sophie N. ; Doerner, Karl F. ; Hartl, Richard F.: A survey on pickup and delivery problems. In: *Journal für Betriebswirtschaft* 58 (2008), Nr. 2, S. 81–117. http://dx.doi.org/10.1007/s11301-008-0036-4. – DOI 10.1007/s11301–008–0036–4. – ISSN 0344–9327

[Pfohl 2004] Pfohl, Hans-Christian: *Logistikmanagement: Konzeption und Funktionen*. 2. Aufl. Berlin [u.a.] : Springer, 2004. – ISBN 3540004688

[Pisinger u. Ropke 2007] Pisinger, David ; Ropke, Stefan: A general heuristic for vehicle routing problems. In: *Computers & Operations Research* 34 (2007), Nr. 8, S. 2403–2435. http://dx.doi.org/10.1016/j.cor.2005.09.012. – DOI 10.1016/j.cor.2005.09.012

[Pugh 1990] Pugh, William: Skip lists: a probabilistic alternative to balanced trees. In: *Communications of the ACM* 33 (1990), Nr. 6, S. 668–676. http://dx.doi.org/10.1145/78973.78977. – DOI 10.1145/78973.78977. – ISSN 00010782

[Salzmann 2011] Salzmann, Stefan: *Neue Verfahren zur Suche nach optimalen Wegen in Graphen*. Hildesheim, Universität Hildesheim, Seminararbeit, 2011

[Sanders u. Schultes 2005] Sanders, Peter ; Schultes, Dominik: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal, Gerth S. (Hrsg.) ; Leonardi, Stefano (Hrsg.): *Lecture Notes in Computer Science* Bd. 3669. Berlin, Heidelberg : Springer, 2005. – ISBN 978–3–540–29118–3, S. 568–579

[Sanders u. Schultes 2006] Sanders, Peter ; Schultes, Dominik: Engineering Highway Hierarchies. In: Azar, Yossi (Hrsg.) ; Erlebach, Thomas (Hrsg.): *Lecture Notes in Computer Science*. Berlin, Heidelberg : Springer, 2006. – ISBN 978–3–540–38875–3, S. 804–816

[Savelsbergh u. Sol 1995] Savelsbergh, M. W. P. ; Sol, M.: The General Pickup and Delivery Problem. In: *Transportation Science* 29 (1995), Nr. 1, S. 17–29

[Schaechterle u. Braun 1977] Schaechterle, Karl-Heinz ; Braun, Jürgen: *Forschung Straßenbau und Straßenverkehrstechnik*. Bd. 222: *Vergleichende Untersuchung vorhandener Verfahren für Verkehrsumlegungen unter Verwendung elektronischer Rechenanlagen*. Bonn : Typo Verlagsgesellschaft mbH, 1977

[Schmid 2001] Schmid, Wolfgang: *Berechnung kürzester Wege in Straßennetzen mit Wegeverboten*. Stuttgart, Universität Stuttgart, Dissertation, 2001

[Schultes 2008] Schultes, Dominik: *Route Planning in Road Networks*. Karlsruhe, Universität Fridericiana zu Karlsruhe, Dissertation, 2008

[Schultes u. Sanders 2007] Schultes, Dominik ; Sanders, Peter: Dynamic Highway-Node Routing. In: Demetrescu, Camil (Hrsg.): *Experimental algorithms*. Berlin : Springer, 2007. – ISBN 978–3–540–72844–3, S. 66–79

[Scott Morton 1971] Scott Morton, Michael S.: *Management decision systems: Computer-based support for decision making*. Boston : Division of Research, Graduate School of Business Administration, Harvard University, 1971. – ISBN 0875840906

[Seidler 2009a] Seidler, Patrick: *Abbildung einer Tachoscheibe für einen mechanischen Tachographen*. `http://upload.wikimedia.org/wikipedia/commons/3/37/Wiki_Tachoscheibe.jpg`. Version: 2009, Abruf: 28.12.2013

[Seidler 2009b] Seidler, Patrick: *Ausschnitt einer Tachoscheibe mit Erläuterungen*. `http://upload.wikimedia.org/wikipedia/commons/a/a8/Ausschnitt_Tachoscheibe.jpg`. Version: 2009, Abruf: 28.12.2013

[Solomon 1987] Solomon, M. M.: Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. In: *Operations Research* 35 (1987), Nr. 2, S. 254–265. `http://dx.doi.org/10.1287/opre.35.2.254`. – DOI 10.1287/opre.35.2.254

[Stiege 2006] Stiege, Günther: *Graphen und Graphalgorithmen*. Aachen : Shaker, 2006 (Berichte aus der Informatik). – ISBN 3–8322–5113–8

[Sun Microsystems 2006] Sun Microsystems, Inc: *Memory Management in the Java HotSpot™ Virtual Machine*. `http://www.oracle.com/technetwork/java/javase/tech/memorymanagement-whitepaper-1-150020.pdf`. Version: 2006, Abruf: 24.09.2012

[Tukey 1977] Tukey, John W.: *Exploratory data analysis*. Reading, Mass : Addison-Wesley, 1977. – ISBN 9780201076165

[Vahrenkamp 2003] Vahrenkamp, Richard: *Quantitative Logistik für das Supply Chain Management*. München : Oldenbourg, 2003. – ISBN 9783486273694

[VDO 2008] VDO ; Continental Trading GmbH (Hrsg.): *Remote Download Device (DLD®) – schnell, komfortabel, kostensparend: Datenübertragung per WLAN/GPRS*. `http://www.xmatik.ch/remotedownload.htm`. Version: 2008, Abruf: 9.11.2012 (Datenblatt des Remote Download Device)

[VDO 2011] VDO: *VDO - DTCO 1.4: Neue Version stellt attraktive Funktionen für Fahrer und Flottenbetreiber bereit*. `http://www.vdo.de/generator/www/de/de/vdo/main/press/releases/commercial_vehicles/2011/20110305_dtco_1_4_de.html`. Version: 2011, Abruf: 09.11.2012 (Pressemitteilung)

[Vetter 2009] Vetter, Christian: *Parallel Time-Dependent Contraction Hierarchies*. Karlsruhe, Universität Karlsruhe, Studentische Projektarbeit, 2009. `algo2.iti.kit.edu/download/vetter_sa.pdf`

[Wassan u. Osman 2002] Wassan, N. A. ; Osman, I. H.: Tabu Search Variants for the Mix Fleet Vehicle Routing Problem. In: *Journal of the Operational Research Society* 53 (2002), Nr. 7, S. 768–782

[Wilson u. Kesselman 2000] Wilson, Steve ; Kesselman, Jeff: *Java platform performance: Strategies and tactics*. Boston : Addison-Wesley, 2000. – ISBN 9780201709698

[Winkler 2007] Winkler, Dietmar: Die Tachodaten im Griff. In: *VerkehrsRUNDSCHAU* 27 (2007), Nr. 27, 28–31. `http://www.iavproducts.com/_downloads/aktuelles/2007/PCS-VR-27.2007.pdf`

[Winter 2002] Winter, Stephan: Modelling Costs of Turns in Route Planning. In: *GeoInformatica* 6 (2002), Nr. 4, S. 345–361. `http://dx.doi.org/10.1023/A:1020853410145`. – DOI 10.1023/A:1020853410145. – ISSN 13846175

Danksagung

Beim Schreiben dieser Arbeit habe ich von mehreren Seiten Hilfe erhalten. An dieser Stelle möchte ich all denjenigen meinen Dank aussprechen, die mich im Laufe der Zeit unterstützten.

Ich möchte mich sehr herzlich bei meinem Doktorvater Herrn Prof. Dr. K. Ambrosi für die Betreuung dieser Arbeit bedanken. Die gemeinsamen Diskussionen und seine Ratschläge schärften das Profil der Arbeit und trugen zu ihrem Gelingen bei. Die vertrauensvolle und konstruktive Unterstützung werde ich stets in guter Erinnerung behalten.

Ich danke auch Herrn Prof. Dr. U. Tüshaus für die Übernahme des Zweitgutachtens und viele hilfreiche Verbesserungsvorschläge zur Arbeit und deren Satz in LaTeX.

Mein besonderer Dank gilt Herrn D. Bartels für die aus der Praxis entstandene, ursprüngliche Idee und die Initiative, mit seinem Unternehmen ein Forschungsprojekt zur Entscheidungsunterstützung mit dem Institut für Betriebswirtschaft und Wirtschaftsinformatik der Universität Hildesheim durchzuführen. Der IN tIME Express Logistik GmbH danke ich ausdrücklich für die mir zur Verfügung gestellten operativen Vergangenheitsdaten und die Einblicke, die mir in die Unternehmensabläufe und speziell in die Disposition gewährt wurden.

Ich bedanke mich ebenfalls bei meinen Kollegen für viele Anregungen, Diskussionen und Hilfestellungen. Ausdrücklich möchte ich Herrn Dr. F. Hahne für so manchen vertiefenden, „algorithmischen“ Gedankenaustausch danken.

Den größten Rückhalt gab mir meine Familie. Ich bin meinen Eltern und meiner Schwester zutiefst dankbar für ihre wunderbare Unterstützung und ihre fortwährende Motivation. Und ich danke meiner Frau Dorothea dafür, dass sie mich so glücklich macht, auf mich acht gibt und Geduld mit mir hat, auch über diese Promotion hinaus.

Hannover, im April 2014 *Curt Nowak*

Anmerkung
Der Quellcode für sämtliche dieser Arbeit zugrunde liegenden Implementierungen ist auf Anfrage unter sekretar@bwl.uni-hildesheim.de erhältlich.

Index

Symbolverzeichnis

A	Menge aller Aufträge bei einem Tourenplanungsproblem	265
a	Platzhalter für einen Auftrag aus A bei einem Tourenplanungsproblem	265
AB	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach *Abbiegebeschränkungs-Flags*	164
b	Platzhalter für einen Auftrag aus A bei einem Tourenplanungsproblem	286
B_u^k	Zeitpunkt der Ankunft bei Ort $u \in V$ durch das Fahrzeug $k \in F$ bei einem Tourenplanungsproblem; zur Modellierung von Wartezeiten an Kundenstandorten, falls Fahrzeug k vor dem entsprechenden Zeitfenster Ort u erreicht	268
C	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach den *Kontraktionskosten*	163
c	Kantenbewertungsfunktion in einem ungerichteten Graphen bzw. Pfeilbewertungsfunktion in einem gerichteten Graphen	30
$c(p)$	Kostenbewertung eines Weges p	30
$c(u, v)$	Kostenbewertung des Pfeils $(u, v) \in E$	30
C^k	Kapazität des Fahrzeugs $k \in F$ bei einem Tourenplanungsproblem	265
c_{uv}^k	Kosten einer Leerfahrt von Fahrzeug $k \in F$ über Pfeil $(u, v) \in E$ bei einem Tourenplanungsproblem	268
$\hat{c}_{uv}^k$	Kosten einer Lastfahrt (mit Fracht an Bord) von Fahrzeug $k \in F$ über Pfeil $(u, v) \in E$ bei einem Tourenplanungsproblem	268
$\mathcal{C}_G(V', s)$	Menge der Abdeckungsknoten vom Graphen G aus der Knotenmenge V' bezüglich des Knotens s beim Highway-Node-Routing-Verfahren	114
c_P	Pfeilbewertungsfunktion in einem Pseudographen	209
D	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach der *Anzahl der kontrahierten Nachbarknoten*	162
$\mathrm{dist}(u, v)$	Kostenbewertung eines kürzesten Weges von u nach v	30

E	Kantenmenge eines ungerichteten Graphen, bzw. Pfeilmenge eines gerichteten Graphen	29
e_a^+	Zeitpunkt des frühesten Beginns der Beladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
e_a^-	Zeitpunkt des frühesten Beginns der Entladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
e_1	Funktion, die jeden Shortcut-Pfeil einer Contraction Hierarchy auf den ersten von ihm überbrückten Pfeil abbildet und jeden Original-Pfeil auf sich selbst abbildet	134
e_2	Funktion, die jeden Shortcut-Pfeil einer Contraction Hierarchy auf den zweiten von ihm überbrückten Pfeil abbildet und jeden Original-Pfeil auf sich selbst abbildet	134
ED	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach der *Pfeildifferenz*	161
E_P	Pfeilmenge eines gerichteten Pseudographen	209
ε	Exzentrizität einer Ellipse; einheitenloser Wert der sich aus dem Verhältnis des Abstands der Länge der Strecke zwischen Brenn- und Mittelpunkt zur Länge der großen Halbachse berechnet und die Form der Ellipse beschreibt	152
EQ	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach dem *Pfeilquotient*	163
F	Zur Verfügung stehende Fahrzeugflotte bei einem Tourenplanungsproblem	265
G	Allgemeine Bezeichnung eines Graphen; $G = [V, E]$ ist ein ungerichteter und $G = (V, E)$ ein gerichteter Graph	29
g_u^+	Positiver Knotengrad von Knoten u; entspricht $\lvert\mathcal{N}(u)\rvert$	30, 206
g_u^-	Positiver Knotengrad von Knoten u; entspricht $\lvert\mathcal{V}(u)\rvert$	30, 206
G_{ch}	Graph einer Contraction Hierarchy	134, 226, 229, 233
G_{KS}	Durch Knotensplitting (KS) entstandener gerichteter Graph mit Knotenmenge V_{KS}, Pfeilmenge E_{KS} und Pfeilbewertungsfunktion c_{KS}, in dem Standard-Wegsucheverfahren auch ohne Anpassung Abbiegebeschränkungen berücksichtigende Wege finden	204
G_P	Gerichteter Pseudograph mit Knotenmenge V_G, Pfeilmenge E_P und Pfeilbewertungsfunktion c_P, der zur Wegsuche mit Abbiegebeschränkungen verwendet werden kann	209
g_u	Knotengrad von Knoten u; entspricht $\lvert\mathcal{NB}(u)\rvert$	30
h	Laufindex	140
i	Laufindex	30, 140
j	Laufindex	108, 140

K	Menge aller fahrzeugbezogenen Orte bei einem Tourenplanungsproblem; entspricht $K^+ \cup K^-$	266
k	Laufindex für Fahrzeuge bei einem Tourenplanungsproblem mit $k \in F$	265
K^+	Menge aller aktuellen Fahrzeugpositionen bei einem Tourenplanungsproblem	266
K^-	Menge aller Zielorte der Fahrzeuge bei einem Tourenplanungsproblem	266
$\mathcal{L}$	Entscheidungsfunktion (*Lokalitätsfilter*) beim Transit-Node-Routing-Verfahren	117
l	Laufindex über die Ebenen bei hierarchischen Wegsucheverfahren	110, 112, 114, 133, 134
l_a^+	Zeitpunkt des spätesten Beginns der Beladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
l_a^-	Zeitpunkt des spätesten Beginns der Entladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
L_i	i-te Landmarke beim ALT-Verfahren	107
$L^D_{u,\text{rück}}$	Menge der Distanzen zu allen Knoten aus dem Rückwärtslabel von u beim Hub-Based-Labeling-Verfahren; $L^D_{u,\text{rück}}(v)$ entspricht $\text{dist}(v,u), v \in L^K_{u,\text{rück}}$	124
$L^K_{u,\text{rück}}$	Knotenmenge des Rückwärtslabels für Knoten u beim Hub-Based-Labeling-Verfahren. Basiert das Verfahren auf Contraction Hierarchies, dann sind hier alle Knoten enthalten, die im Rahmen der CH-Rückwärtssuche ausgehend von u erreicht werden können; sinnvollerweise nach Knoten-Id sortiert	124
$L^D_{u,\text{vor}}$	Menge der Distanzen zu allen Knoten aus dem Vorwärtslabel von u beim Hub-Based-Labeling-Verfahren; $L^D_{u,\text{vor}}(v)$ entspricht $\text{dist}(u,v), v \in L^K_{u,\text{vor}}$	124
$L^K_{u,\text{vor}}$	Knotenmenge des Vorwärtslabels für Knoten u beim Hub-Based-Labeling-Verfahren. Basiert das Verfahren auf Contraction Hierarchies, dann sind hier alle Knoten enthalten, die im Rahmen der CH-Vorwärtssuche ausgehend von u erreicht werden können; sinnvollerweise nach Knoten-Id sortiert	124
M	Menge der Landmarken beim ALT-Verfahren	107
m	Platzhalter für Brückenknoten bei bidirektionalen Wegsuchen	98, 124, 150, 157, 225
$\mathcal{N}(u)$	Menge der Nachfolgerknoten zu Knoten u	29
$\mathcal{NB}(u)$	Menge der Nachbarknoten zu Knoten u; entspricht $\mathcal{V}(u) \cup \mathcal{N}(u)$	30, 161

$\mathcal{N}'_n(u)$	Menge der Knoten in der n-Nachbarschaft zu Knoten u mit $\mathcal{N}'_n(u) = \{v \in V \mid r_u(v) \leq n\}$ für $n \in \mathbb{N}$	112, 170
$\mathcal{N}_r(u)$	Menge der Knoten in der Radius-Nachbarschaft zu Knoten u beim Highway-Hierarchies-Verfahren; es gilt $\mathcal{N}_r(u) = \{v \in V \mid \text{dist}(u, v) \leq r\}$	112, 123
O	Menge aller auftragsbezogenen Orte bei einem Tourenplanungsproblem; entspricht $O^+ \cup O^-$	266
O^+	Menge aller Beladungsorte bei einem Tourenplanungsproblem	266
o_a^+	Beladungsort des Auftrags $a \in A$ bei einem Tourenplanungsproblem	265
O^-	Menge aller Entladungsorte bei einem Tourenplanungsproblem	266
o_a^-	Entladungsort des Auftrags $a \in A$ bei einem Tourenplanungsproblem	265
OEQ	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach dem *Originalen Pfeilquotient*	164
P	allgemeines Kürzel für eine Prioritätswarteschlange. Zugriff auf der erste Element, d. h. das Element mit höchster Priorität, erfolgt mit dem Ausdruck $\min\{P\}$	135, 160, 233
p	Platzhalter für einen Weg	30
ϕ	Bijektive Funktion bei der Definition eines Pseudographen, welche jedem Pfeil des Ursprungsgraphen einen Knoten des Pseudographen zuweist und umgekehrt.	209
ψ	Bewertungsfunktion, die während der Erzeugung einer Contraction Hierarchy jedem Knoten des Originalgraphen eine aktuelle *Wichtigkeit* zuweist. Die Knoten werden nacheinander aufsteigend nach ihrer Wichtigkeit kontrahiert.	159
$p^*_{s,t}$	Kürzester Weg zwischen Startknoten s und Zielknoten t	30
$\|p\|$	Anzahl der Pfeile im Weg p	30, 111, 201
Q	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach den *Wegsuchekosten*	163
q_a	Menge der zu Auftrag $a \in A$ gehörenden Fracht bei einem Tourenplanungsproblem	265
R	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach der *Anzahl notwendiger Shortcut-Pfeile*	164
r	Radius der Radius-Nachbarschaft im Rahmen des Highway-Hierarchies-Verfahrens	112
$\hat{r}$	Anzahl der Regionen, in welche die Knoten beim Arc-Flag-Verfahren eingeteilt werden	108

$RAND$	Bewertungsfunktion bei der Knotensortierung einer Contraction Hierarchy nach dem *zufälligen Bias*	164
$\mathrm{rd}(v)$	Schätzer für die Restdistanz, bzw. allgemeiner *Restkosten*, von einem Knoten v zum aktuellen Zielknoten im Rahmen einer Wegsuche	101, 107, 150, 156, 224, 229
reg	Funktion beim Arc-Flag-Verfahren, die jedem Knoten des Graphen den Index „seiner“ Region zuweist; es gilt daher $\mathrm{reg} : V \to \{1, \ldots, \hat{r}\}$	108
$r_s(v)$	Dijkstra-Rang von Knoten v bezüglich Startknoten s	94, 111, 112
s	Startknoten bei Wegsuchen	143
$\mathcal{T}$	Menge der Transitknoten beim Transit-Node-Routing-Verfahren	117
t	Zielknoten bei Wegsuchen	143
T	Menge der Trennlinien beim Trennlinien-Verfahren	105
τ	Platzhalter für eine Trennlinie beim Trennlinien-Verfahren	106
θ	Abbiegekostenfunktion für den Ursprungsgraphen $G = (V, E, c)$ bei der Definition eines Pseudographen $G_P = (V_P, E_P, c_P)$ mit $\theta : E \times E \to \mathbb{R} \cup \{\infty\}$	209
t_a^+	Dauer der Beladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
t_a^-	Dauer der Entladung von Auftrag $a \in A$ bei einem Tourenplanungsproblem	267
t_a^{z+}	Dauer der Zwischenbeladung von Auftrag $a \in A$ an einem Umladeort bei einem Tourenplanungsproblem	272
t_a^{z-}	Dauer der Zwischenentladung von Auftrag $a \in A$ an einem Umladeort bei einem Tourenplanungsproblem	272
TR	Menge aller Abbiegeverbote eines Graphen	201
t_{uv}^k	Fahrtdauer entlang Pfeil $(u, v) \in E$ für Fahrzeug $k \in F$ bei einem Tourenplanungsproblem	267
U	Menge aller potentiellen Umladeorte bei einem Tourenplanungsproblem	272
u	Platzhalter für einen bestimmten Knoten eines Graphen	29
$\ddot{U}((u, v), (v, w))$	Notation für den Übergang von Pfeil (u, v) nach Pfeil (v, w) und Grundlage der Menge der Abbiegeverbote TR	200
V	Knotenmenge eines Graphen	29
v	Platzhalter für einen bestimmten Knoten eines Graphen	29
V_P	Knotenmenge eines Pseudographen	209
$\mathcal{V}(u)$	Menge der Vorgängerknoten zu Knoten u	29
$\mathcal{W}_{u,v}$	Die Menge aller Knoten im eindeutigen kürzesten Weg von u nach v bei der Herleitung der Highway-Dimension	123

x	Knoten in Beispielgraphen zu den Wegsuchen	143
x_{uv}^{k}	Binäre Entscheidungsvariable bei einem Tourenplanungsproblem. Gibt an, ob Fahrzeug $k \in F$ über Pfeil $(u, v) \in E$ fährt (1) oder nicht (0)	268
y	Knoten in Beispielgraphen zu den Wegsuchen	31
y_{uv}^{ka}	Entscheidungsvariable aus $[0; 1]$ bei einem Tourenplanungsproblem. Gibt in Abschnitt 7.1.3 den Anteil der Fracht von Auftrag $a \in A$ an, den Fahrzeug $k \in F$ über Pfeil $(u, v) \in E$ transportiert. Ist in der vereinfachten Formulierung in den Abschnitten 7.1.1.2 und 7.1.2 noch eine Binärvariable.	268
Z	Zelle eines Graphen, über den ein Gitternetz gelegt wurde; bezieht sich auf die Ermittlung der Transitknoten beim Transit-Node-Routing-Verfahren	120
$\mathcal{Z}_v$	Menge der Zugangsknoten für den Knoten $v \in V$ beim Transit-Node-Routing-Verfahren	117