Karlsruhe Institute of Technology

Berechnung simpler Routen mit Distanzgarantie

Bachelorarbeit
von

Heiner Zille

An der Fakultät für Informatik
Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

Erstgutachterin:	Prof. Dr. D. Wagner
Betreuender Mitarbeiter:	Andreas Gemsa

Bearbeitungszeit: 17. Juni 2011 – 30. September 2011

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, und die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht.

Ort, Datum	Unterschrift

Abstract. Diese Arbeit beschäftigt sich mit der Berechnung möglichst einfacher Routen als Alternativen zu kürzesten Routen. Im Gegensatz zu früheren Arbeiten sollen einfachste Routen unter der Nebenbedingung gefunden werden, dass diese eine gewisse Maximallänge im Vergleich zum kürzesten Weg nicht überschreiten. Nachdem das Problem modelliert und durch geeignete Konstrukte dargestellt wurde, wird der Algorithmus zum Finden solcher besten Kompromisse zwischen Distanz und Einfachheit vorgestellt. Die anschließende Evaluation zeigt, dass solche Routen im Gegensatz zu einfachsten Routen in der Realität sinnvolle Alternativen zu kürzesten Wegen darstellen können.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **3**

2.1 Graphen 3

2.2 Dijktras Algorithmus 4

2.3 Bisherige Arbeiten 5

3 Simple Routen **9**

3.1 Definition der Einfachheit 10

3.2 Der Kantengraph 13

3.3 Der Erweiterte Kantengraph 16

3.4 Kostenfunktionen 18

3.5 Berechnung von simplen Routen mit Distanzgarantie 23

3.5.1 Modellierung 23

3.5.2 Funktionsweise des Algorithmus' 25

3.5.3 Ausschlusskriterien für Pfade 30

3.5.4 Analyse 34

4 Implementierung und Evaluation **39**

4.1 Datengrundlage 40

4.2 Experimente und Analyse 41

4.3 Fallbeispiele 55

5 Zusammenfassung und Ausblick **69**

Literaturverzeichnis **71**

1. Einleitung

Das Problem, von A nach B zu gelangen, ist so alt wie die Menschheit selbst. Über die Jahrhunderte wurden jedoch die Entfernungen, die es zurückzulegen galt, immer größer. Durch die Bildung von Städten im Mittelalter wuchsen zusätzlich auch die logistischen Probleme innerhalb kleinerer Räume. In modernen Zeiten der Globalisierung stellen sich diese Herausforderungen noch in erhöhtem Maße. Der Überlegung, auf welche Weise Menschen oder Waren am schnellsten an ihr Ziel kommen, ist dadurch mit der Zeit immer mehr Bedeutung zugekommen. Messungen von Routen erfolgten und erfolgen dabei meist in Distanz oder Zeit. In den letzten Jahren lagen Beschleunigungstechniken für Routenplanung im Fokus wissenschaftlicher Untersuchungen. Bekannte Techniken sind dabei das Transit Node Routing [BFM08], SHARC [BD10] und Contraction Hierarchies [GSSD08]. Eine Übersicht über dieses Themengebiet findet sich in [DSSW09].
Dabei weitestgehend unbeobachtet blieb das Problem, wie kompliziert so ein schnellster, beziehungsweise kürzester Weg eigentlich ist. Die schnellste Route mag im Warenverkehr unverzichtbar sein, im privaten Personenverkehr stellen sich jedoch unter Umständen andere Anforderungen an eine Route. Navigationsgeräte sind heutzutage fast in jedem neuen Automobil vorhanden oder können optional erworben werden. Die Intention dahinter ist klar: Die Navigation soll für den Menschen einfacher werden. Eine Zielsetzung, die bisher wenig erforscht wurde, ist jedoch, dass nicht nur die Route schnell und einfach auf dem Monitor erscheint, sondern dass diese Route auch einfach zu fahren ist. Bei komplizierten Straßenführungen kann eine einfache Route Stress reduzieren und Fehler des Fahrers vermeiden. In der Vergangenheit haben sich darum verschiedene Autoren mit der Frage auseinandergesetzt, was die Einfachheit einer Route ausmacht und wie man einfache Routen finden kann. Ihre Ergebnisse sind jedoch in weiten Teilen für die Praxis ungeeignet, da einfachste Routen oft viel zu lang sind, um sinnvolle Alternativen darzustellen.

Das Ziel dieser Arbeit ist es daher, das Konzept der einfachen Routen für die Praxis anwendbar zu machen.
Als Erstes werden einige Grundlagen erläutert und ein kurzer Überblick über bisherige Arbeiten auf diesem Gebiet gegeben. In Kapitel 3 wird dann zunächst eine Definition des klassischerweise subjektiven Begriffs der Einfachheit vorgestellt. Es wird ein Konstrukt aufgezeigt, sowie dessen Erweiterung erarbeitet, mit dessen Hilfe eine Modellierung einfacher und kurzer Routen gleichzeitig möglich ist. Anschließend werden beispielhaft verschiedene Methoden zur Messung der Einfachheit erläutert. Es wird ein Algorithmus entworfen, vorgestellt und analysiert, der in der Lage ist, einfachste Routen unter Einhaltung bestimmter Kriterien für die Länge der Route zu berechnen. Kapitel 4 wird sich schließlich mit der Implementierung des Algorithmus befassen. Testergebnisse werden vorgestellt und analysiert. Schließlich wird eine kurze Zusammenfassung sowie ein Ausblick auf mögliche zukünftige Arbeiten auf diesem Gebiet gegeben.

2. Grundlagen

In diesem Kapitel sollen zunächst einige Grundlagen erläutert werden, die für das Verständnis der folgenden Kapitel von Bedeutung sind.

2.1 Graphen

Um Netzwerke der realen Welt informationstechnisch darzustellen, modelliert man diese oft als Graphen. Ein Graph in der Informatik besteht aus Knoten und Kanten. Man kann bestimmte Identitäten, beispielsweise Städte, Personen oder Zustände, mit einem Knoten beschreiben. Besteht zwischen zwei Identitäten eine direkte Verbindung, so spannt man eine Kante zwischen ihnen. Formal ist ein Graph ein Tupel $G = (V, E)$, wobei V die Menge der Knoten und E die Menge der Kanten ist. Eine gerichtete Kante $e \in E$ ist dann das Paar (u, v), mit $u \in V$ als Start- und $v \in V$ als Endknoten. Die Anzahl der Knoten und Kanten bezeichen wir mit $n := |V|$ und $m := |E|$

Man unterscheidet zwischen gerichteten und ungerichteten Graphen. In einem gerichteten Graph besteht eine Kante nur in eine Richtung, und wird grafisch meist durch einen Pfeil repräsentiert. Die Kante besitzt dadurch eine „Vorwärts-“ und eine „Rückwärtsrichtung“. Im Gegensatz dazu besitzt eine Kante in einem ungerichteten Graphen keinen eindeutigen Start- und Endknoten, sondern bezeichnet eine Verbindung von beiden Richtungen aus.

Um eine Route von einem Ort zum anderen zu bestimmen, muss das Straßennetz als Graph modelliert werden. Hierbei werden in der Regel bestimmte Punkte, beispielsweise Kreuzungen oder Abbiegungen, als Knoten dargestellt, und diese mit Straßen als Kanten verbunden. Auf diese Weise bekommt man eine brauchbare und dennoch realitätsnahe Abbildung des wirklichen Straßennetzes. Diese Abbildung bezeichnen

wir kurz als *Straßengraphen*.
In der Algorithmik sind oft bestimmte Eigenschaften eines Graphen von Bedeutung. Im Gebiet der Routenplanung wäre es zum Beispiel wünschenswert, dass der Graph zusammenhängend ist, da man von jedem Punkt aus möglichst zu jeder anderen Adresse des Landes kommen möchte. Vor allem ist aber die Frage von Relevanz, welcher der schnellste Weg in dem Graphen von einem Knoten zu einem anderen ist. Da Kanten die Straßen abbilden, bekommen diese ein „Gewicht“ zugewiesen, das beispielsweise der Länge der Straße, beziehungsweise des Straßenabschnittes, entspricht. Da aber die tatsächliche Entfernung in Zeiten moderner Transportmittel oft nur zweitrangig ist, legt man statt der Entfernung üblicherweise die Zeit zugrunde, die man braucht, um diese Straße zu durchqueren. Daher können Autobahnen aufgrund ihrer hohen Durchschnittsgeschwindigkeit beispielsweise *kürzer* sein als Bundesstraßen, auch wenn dies nicht der Realität entspricht.
Einen Weg von einem Startpunkt zu einem Ziel beschreiben wir im Straßengraphen durch eine Folge von Knoten (oder auch dazugehörigen Kanten), die man passieren muss, um vom Start- zum Zielknoten zu gelangen. Wir bezeichnen dies als *Route* oder auch *Weg*.

2.2 Dijktras Algorithmus

Ein in der Praxis häufig verwendeter Algorithmus, um die kürzesten Wege zwischen zwei Knoten in einem Netzwerk zu finden, ist Dijkstras Algorithmus. Dieser wurde 1959 von dem niederländischen Informatiker Edsger Wybe Dijkstra veröffentlicht [Dij59]. Die Grundidee beruht darauf, beginnend mit einem gegebenen Startknoten alle Entfernungen zu den jeweiligen Nachbarknoten zu ermitteln. Jeder Knoten speichert dabei seine aktuelle Entfernung. Wenn ein kürzerer Weg gefunden wird, wird die Entfernung aktualisiert. Auf diese Weise werden in der Reihenfolge der aktuell kleinsten Distanzen der Knoten nacheinander alle Knoten abgearbeitet.

Die Zeitkomplexität des Dijkstra-Algorithmus' hängt von der verwendeten Datenstruktur ab. Wir bezeichnen mit $T_{decreaseKey}$, $T_{deleteMin}$ und T_{insert} die Zeitkomplexitäten für eine Ausführung der entsprechenden Operationen in der genutzten Datenstruktur. Die Zeitkomplexität liegt dann in

$T_{Dijkstra} = \mathcal{O}(\, m \cdot T_{decreaseKey}(n) + n \cdot (\, T_{deleteMin}(n) + T_{insert}(n)\,)\,)$ [MS08].

Naive Implementierungen von Dijkstras Algorithmus (mit zwei Arrays) erreichen somit eine Laufzeit von $T_{DijkstraNaiv} = \mathcal{O}(\, m + n^2\,)$. Schnellere Implementierungen, beispielsweise mit Fibonacci Heaps, erreichen eine theoretische Laufzeit von $T_{DijkstraFibonacci} = \mathcal{O}(\, m + n \log(n)\,)$.

2.3 Bisherige Arbeiten

Verschiedene Arbeiten haben sich in der Vergangenheit mit simplen Routen beschäftigt. Diese betrachteten auch verschiedene Auffassungen von der „Einfachheit“ einer Route.
In den Jahren 2001 und 2002 beschrieb Winter eine Methode, die sich mit der Einfachheit im Bezug auf die Abbiegungen innerhalb einer Route auseinandersetzt [Win01, Win02]. Er ging dabei davon aus, dass eine Route komplizierter mit ihren Abbiegungen wird. Jede Abbiegung, die man nehmen muss, ist ein Entscheidungspunkt, den sich der Nutzer merken muss, beziehungsweise an dem er sich orientieren und konzentrieren muss, um die Anweisung richtig auszuführen und keinen Fehler zu machen. Er sprach daher von Abbiegungskosten, die nach verschiedenen Kriterien für eine Abbiegung vergeben werden können. Die Entscheidung, welche Kante als nächstes besucht wird, könnte sich zum Beispiel nach dem kleinsten zu durchfahrenden Winkel richten, oder auch danach, ob eine Straße an einer Kreuzung einfach geradeaus führt, und somit gar kein Abbiegungsmanöver darstellt. Je nach Art der Zielvorstellung können die Kosten für eine Abbiegung unterschiedlich modelliert werden. Unterschiede in den verschiedenen Kostenfunktionen ergeben sich insbesondere daraus, ob man am Ende die Route mit den wenigsten Abbiegungen erhalten möchte oder die semantische Routenbeschreibung möglichst kurz halten möchte. Beschreibungen wie „Folge der Hauptstraße bis ...“ enthalten unter Umständen implizit das Überqueren mehrer Kreuzungen, die auf diese Weise selbst nicht als Abbiegungen betrachtet werden, obwohl die Straße nicht notwendigerweise nur geradeaus verlaufen muss.
Eine andere Zielvorstellung, die Winter beschreibt, ist, die Gesamtsumme der gefahrenen Winkel zu minimieren. Weiterhin erwähnt er die Anwendung einer *least angle heuristic*, bei der sich die ungefähre Richtung auf das Ziel zu nicht wesentlich ändert. Dies kann insbesondere eine sinnvolle Zielvorgabe sein, wenn man annimmt, dass bei schwer zu befolgenden Routen, beispielsweise in Großstädten, ein falsches Abbiegen weniger „Schaden“ im Sinne von Verzögerung anrichtet. Detailliertere Untersuchungen zu solchen *least-angle* Strategien wurden später von Hochmair und Karlsson durchgeführt [HK05].

Da solche Abbiegungskosten allerdings nicht direkt auf den Knoten oder Kanten eines Graphen angebracht werden können, beschreibt Winter zur Modellierung der Abbiegungskosten die Konstruktion eines zweiten Graphen, der aus dem eigentlichen Straßengraphen erstellt werden kann, und den er als *Pseudo-Dualen Graphen* bezeichnet. Diese Idee wurde ursprünglich von Caldwell 1961 aufgeworfen [Cal61]. Dieser beschäftige sich in einem kurzen Aufsatz mit Strafkosten für Abbiegungen. Solche Bestrafungen von Abbiegungen wurden auch in jüngster Zeit von anderen

Autoren wieder aufgegriffen [GV11]. Der pseudo-duale Graph ist heute auch unter anderen Namen bekannt. Zum Aufbau eines solchen Graphen siehe Kapitel 3.2.

Eine andere Art, die Kosten für eine Abbiegung zu definieren, wurde bereits 1986 von David Mark vorgestellt [Mar86]. Er schlug vor, die Kosten für eine Abbiegung aus einem Framework zu generieren, das darauf beruhte, wie kompliziert es ist, eine Anweisung mündlich oder schriftlich wiederzugeben. Basierend darauf, wie viele Straßennamen und Landmark-Namen in einer Anweisung enthalten waren, vergab er Slots von bis zu neun Punkten. Eine Abbiegung an einer Kreuzung war demnach schwerer zu beschreiben, und somit kostspieliger, als eine Abbiegung an einer T-Kreuzung oder einfaches Geradeausfahren. Marks Algorithmus suchte dann den besten Weg innerhalb des Graphen anhand eines Gesamtgewichts, das sich aus gewichteten Entfernungs- und Abbiegungskosten zusammensetze. Die Entscheidung, welche Kante als nächste besucht wird, war also schon während der Berechnung durch die Strafe für ein vorheriges Abbiegen beeinflusst, allerdings wurde die Unterscheidung zwischen Einfachheit und Länge einer Route durch die Kombination beider Werte nicht berücksichtigt.

Basierend auf Marks Arbeit, der das System nur theoretisch beschrieb, eine Implementierung an realen Daten aber offen ließ, veröffentlichten 2003 Duckham und Kulik eine Arbeit, die die Idee Marks erweiterte und implementierte [DK03]. Sie verallgemeinerten die Idee dahingehend, dass sie für bestimmte Typen von Entscheidungspunkten eine bestimmte Menge an Strafkosten vergaben. Eine Klassifizierung der Abbiegungen, angelehnt an die Arbeiten von Mark, Duckham und Kulik, ist später in dieser Arbeit in Abbildung 3.8 zu sehen.

Um diese Kosten für jedes Paar von inzidenten Kanten in den Graphen einzufügen, bedienten sie sich Winters Idee des pseudo-dualen Graphen. Ihr Algorithmus suchte dann im Gegensaz zu Marks Arbeit die simpelste Route einzig anhand der Minimierung der Gesamtabbiegungskosten im pseudo-dualen Graphen. Die eigentliche, reale Länge der zu fahrenden Route berücksichtigten sie nicht. Die Ergebnisse ihrer Tests zeigten, dass in ihrem Modell, bestehend aus Algorithmus, Kostenzuweisungen und den ausgewählten Daten, ein simpelster Pfad im Durchschnitt 15,8 % länger war als der kürzeste Pfad. Von allen Pfaden waren ca. 93 % der Pfade weniger als 50 % länger als der korrespondierende kürzeste Weg. Allerdings gab es auch Wege, die im Extremfall bis zu 75 % länger wurden als der kürzeste Weg.

Einen weiteren Ansatz zum Finden der einfachsten Route benutzten Richter und Duckham in ihrer Arbeit von 2008 [RD08]. Sie gingen in ihrer Arbeit davon aus, dass

die einfachste Route diejenige ist, die die kürzeste textutelle Beschreibung hat. Routen lassen sich durch Instruktionen besonders gut beschreiben, wenn man mehrere Schritte zu einem zusammenfassen kann (*„Gearadeaus und an der dritten Kreuzung links abbiegen“* anstatt *„Geradeaus, an der ersten Kreuzung geradeaus fahren, an der zweiten Kreuzung geradeaus fahren und dann an der nächsten Kreuzung links abiegen“*). Solche Zusammenfassungen und Vereinfachungen von Instruktionen, auch durch Orientierungshilfen (Landmarks) wie Postämter oder Bahnhöfe unterstützt, die für jede Route anders sind, nennt man kontext-spezifische Anweisungen. Sie basieren auf den Knoten die vor einem anderen Knoten passiert wurden. Einen Prozess, genannt GUARD (Generation of Unambiguopus, Adapted Route Directions), der solche kontext-spezifischen Beschreibungen aus einer gegebenen Route erstellt, wurde von Richter bereits vorher entwickelt [Ric07]. Richter und Duckham entwarfen nun einen Algorithmus, der basierend auf GUARD und den vorher erwähnten Arbeiten zu simplen Routen (angelehnt an die Modellierung der Kosten aus [DK03]) die Route nun schon von vornherein so berechnete, dass die herauskommende Route von GUARD möglichst kurz beschreibbar war. Die entsprechenden Routen ihres „simplest Instruction“ Algorithmus in ihren Tests waren im Durchschnitt 13,31% länger als die dazugehörigen kürzesten Pfade, und unterschieden sich damit in der Länge kaum von den Pfaden des reinen „simpelste Pfade“ Algorithmus' von Duckham und Kulik.

3. Simple Routen

Ziel dieser Arbeit ist es nun, eine Methode zu entwickeln, die eine möglichst einfache Route von einem Start- zu einem Zielknoten findet. Die Begriffe „einfach“ und „simpel“ werden dabei in dieser Arbeit synonym verwendet.
Wie oben beschrieben, haben bisherige Arbeiten auf diesem Gebiet den Nachteil, dass durchschnittlich zwar ganz „gute“ (kurze) Routen berechnet werden, in einigen Fällen aber die Routen auch über 50 und bis zu 75 % länger wurden, als die zugehörigen kürzesten Wege. Dies ist nicht wünschenswert, denn verschiedene Faktoren, wie etwa der Treibstoffverbrauch oder die benötigte Zeit, würden eine solche Route, trotz Einfachheit, in der Praxis unbrauchbar machen. Allerdings kann davon ausgegangen werden, dass es durchaus sinnvoll sein kann, eine gewisse Verlängerung der Route in Kauf zu nehmen, um dafür möglichst „einfach“ an sein Ziel zu kommen. Fährt man beispielsweise zum ersten Mal in eine neue, einem unbekannte Stadt, ist aufgrund mangelnder Ortskenntnisse die Gefahr, sich zu verfahren, recht hoch. Eine allzu lange und komplizierte Wegbeschreibung ist dort wenig förderlich, auch wenn dies normalerweise auf dem schnellsten Wege zum Ziel führen würde. Komplizierte Abbiege-Instruktionen können während einer Fahrt falsch verstanden werden. Hinzu kommt, dass Fahrer sich Routen (oder Teile davon) während einer Fahrt merken müssen. Untersuchungen haben ergeben, dass Menschen, die nach dem Weg gefragt werden, oft möglichst einfache Routen als Antwort geben, anstatt schnellster Routen [Mar86]. Die erhaltenen Antworten sind so leichter zu merken und für den Fahrer leichter zu befolgen.

Der Unterschied in dieser Aufgabenstellung zu bisherigen Arbeiten auf diesem Gebiet besteht daher darin, dass der Benutzer der Route hier nur bereit ist, eine gewisse Verlängerung seines Weges (im Vergleich zum kürzesten Weg) in Kauf zu nehmen, um eine einfachere Route zu erhalten. Er möchte allerdings dabei eine Obergrenze

festlegen, die die simple Route keinesfalls übersteigt. Unter dieser Restriktion können die Konzepte, die in der Vergangenheit beschrieben wurden, nicht mehr ohne Weiteres angewendet werden. Eine simpelste Route zu berechnen und hinterher zu überprüfen, ob der Umweg klein genug ist, um ihn zu akzeptieren, ist im Allgemeinen keine sehr effektive Methode, da im negativen Fall keine Alternativen zur Verfügung stehen. Stattdessen geht es darum, von vornherein eine Garantie dafür zu geben, dass der Weg möglichst einfach wird, und dabei eine vorgegebene Länge (im Vergleich zum kürzesten Weg) nicht überschreitet.

3.1 Definition der Einfachheit

Um nun eine möglichst einfache Route zu berechnen, ist es zunächst einmal notwendig, sich darüber klar zu werden, wie der Begriff „einfach“ für diese Aufgabe definiert werden soll.
Bisherige Arbeiten beschäftigten sich mit unterschiedlichen Definitionen. Im Wesentlichen lassen sich jedoch die Ansätze von Caldwell, Mark, Winter, Duckham und Kulik dahingehend verallgemeinern, dass sie alle Einfachheit definieren als eine leichtere Art, die Strecke letztendlich zu fahren [Win01, Win02, Cal61, Mar86, DK03]. Im Gegensatz dazu verstand die Arbeit von Richter und Duckham Einfachheit als möglichst kurze textuelle Beschreibung [RD08]. Diese beiden Ziele stehen sich zwar prinzipiell nicht im Wege und ergaben, wie in Experimenten gezeigt wurde, auch vergleichbare Ergebnisse, trotzdem ist die Route mit der kürzesten Beschreibung nicht zwangsläufig auch die beim Fahren am einfachsten zu befolgende und umgekehrt.
In dieser Arbeit wollen wir uns nun darauf konzentrieren, welche Route am einfachsten zu fahren ist.
Was also macht eine Route kompliziert? Offensichtlich muss sich der Fahrer der Route jedesmal dann besonders konzentrieren, wenn er eine Entscheidung darüber zu treffen hat, welche Aktion er als nächstes ausführt. Gibt ihm die Route eine Entscheidung vor (beispielweise „Rechts abbiegen“), so ist der Fahrer derjenige, der sich orientieren und seine Handlung so anpassen muss, dass er die geforderte Aktion durchführt.
Bei einer geraden isolierten Strecke zwischen zwei Punkten gibt es zuallererst keinerlei Entscheidungen, die getroffen werden müssen, und keine Aktionen, mit denen man seine Fahrtrichtung ändern kann. Existieren entlang der Strecke Kreuzungen, muss man wissen, dass man geradeaus zu fahren hat. Muss man irgendwo abbiegen, ist gegebenenfalls das Einordnen auf der richtigen Fahrspur nötig oder Ähnliches. Maßgeblich für die Komplexität einer Route sind also deren *Entscheidungspunkte*. Auf welche Weise sie eine Route komplizierter machen, also nach Anzahl, Art oder konkreter Aktion, ist hierbei noch nicht gesagt. Allen diesen Varianten gemeinsam

ist, dass die Entscheidungspunkte deren Kern bilden, und über sie das Verhalten und die Komplexität der Route bestimmt wird. (Bei dieser Betrachtung werden Ereignisse wie Staus oder Baustellen, die außerhalb der eigenen Entscheidungsgewalt liegen, und somit keinen Entscheidungspunkt bilden, nicht berücksichtigt. Diese sind aber in den meisten Fällen temporär, und in statischen Straßengraphen nicht modelliert. Die Verfahren, die in dieser Arbeit vorgestellt und verwendet werden, können aber auch auf diese Situationen angewendet werden.)

Von diesem Gedanken ausgehend ist eine Modellierung möglich, die jedem Entscheidungspunkt einen Betrag zuweist, den dieser zu der Route beiträgt, und zwar abhängig auch davon, welche Entscheidung dort getroffen wird. Die Frage ist, was genau ein Entscheidungspunkt ist. Einfluss auf den weiteren Verlauf des Weges hat man nämlich immer dort, wo sich Straßen treffen. Dort treffen sich zwei oder mehrere Kanten an einem Knoten.
Als Beispiel dient hier folgende Einbettung eines Testgraphen:

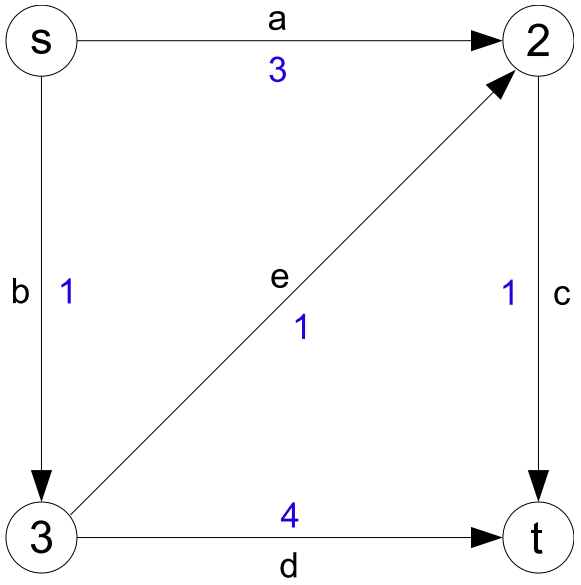


Abbildung 3.1: Eingebetteter Graph mit einer kürzesten und zwei einfacheren Routen vom Start (s) zum Ziel (t)

Die Gewichte an den Kanten geben ihre Länge an. Der kürzeste Weg vom Start (s) zum Ziel (t) führt über die Kanten b, e und c mit einer Länge von 3 Entfernungseinheiten. Dabei muss man an zwei Knoten (Nummer 2 und Nummer 3) abbiegen und auf eine neue Straße wechseln. Wenn wir annehmen, dass eine Route mit weniger Abbiegungen einfacher wird, gibt es in diesem Graphen aber außerdem noch zwei

weitere, einfachere Routen über die Kanten a und c sowie b und d. Diese beinhalten jeweils nur eine Abbiegung an einem Knoten, haben aber eine größere Länge.

Einen Knoten allein als eine Entscheidungssituation zu betrachten, und diesem dann Kosten zuzuweisen, reicht aber in der Realität nicht aus, da die verschiedenen Ausprägungen der Entscheidung (beispielsweise „links“, „rechts“ oder „geradeaus“) berücksichtigt werden müssen, und somit die Entscheidungssituation aus unterschiedlichen Richtungen leichter oder schwerer ausfallen kann (Beispiel: T-Kreuzung). Der Wert, den ein Entscheidungspunkt zur Einfachheit einer Route beiträgt, kann davon abhängen, von welcher Kante man sich nähert, welche besonderen Eigenschaften (beispielsweise Verkehrsbedingungen) an einem Knoten herrschen, wie hoch der Knotengrad ist, welche Winkel die Kanten zueinander haben, und auf welcher der Kanten man den Knoten letztendlich wieder verlässt. Charakterisiert wird eine Entscheidung also durch die Angabe:

$$(\textbf{\textit{von}}\textit{Kante, }\textbf{\textit{über}}\textit{Knoten, }\textbf{\textit{zu}}\textit{Kante})$$

In Abbildung 3.2 lautet diese Entscheidung beispielsweise „von Kante a über Knoten v zu Kante b“, also das Tripel (a, v, e). Nun können wir mit dieser Modellierung jeder dieser Entscheidungen einen Wert zuweisen, der der Komplexität der Route hinzugefügt wird, wenn diese Entscheidung an diesem Punkt unter diesen Umständen getroffen wird. Verschiedene Vorstellungen von Einfachheit kann man nun einfach dadurch umsetzen, dass man die Werte nach einer zur Zielsetzung passenden Regel vergibt. Diese Vergabekriterien bezeichnen wir als *Kostenfunktionen*.

Verschiedene Zielsetzungen könnten nun beispielsweise sein:

- die Anzahl der zu fahrenden Abbiegungen zu minimieren
- Abbiegungen bevorzugen, an denen sich nur wenige Straßen treffen
- möglichst langes Fahren auf einer Straße

Verschiedene Kombinationen dieser Ziele sind genauso denkbar. Geht man davon aus, dass Instruktionen vom Fahrer leicht falsch verstanden werden können, wäre außerdem eine Variante möglich, die die ungefähre Gesamtrichtung auf das Ziel zu erhält, so dass ein versehentliches Abweichen von der Route geringe Schwierigkeiten mit sich bringt, um wieder zu ihr zurück zu finden.

Zur genaueren Umsetzung einzelner Zielsetzungen und Definition verschiedener Kostenfunktionen siehe Abschnitt 3.4.

3.2 Der Kantengraph

Um die Einfachheit einer Abbiegung zu modellieren, muss man sich überlegen, wie man diese Information in einem Graphen repräsentiert. Als eine Abbiegung bezeichnen wir, wie oben bereits erwähnt, den Übergang von einer Kante zur nächsten auf unserem Weg. Dieser Übergang findet immer an einem Knoten statt. Die Information über die Einfachheit lässt sich aber weder einer Kante noch einem Knoten eindeutig zuordnen. Die Information über die Einfachheit ist also in unserem normalen Straßengraphen nicht darstellbar.
Um dieses Problem zu lösen, wurde in der Vergangenheit die Idee eines zweiten Graphen verwendet, der aus dem ursprünglichen Straßengraphen erzeugt werden kann. Diese Idee wurde in diesem Kontext zuerst von Winter im Jahre 2001 präsentiert [Win01], der den Namen „Pseudo-Dualer Graph“ prägte. Pseudo-dual deshalb, weil er nicht die eigentliche duale Abbildung zum ursprünglichen Graphen darstellt (in dem neue Knoten aus Facetten gebildet werden), jedoch auf ähnliche Weise gebildet wird.
Andere Autoren bezeichnen diesen Graphen auch als „Line-Graph“ oder „Kantengraph“ [Vol96]. Der Kantengraph wird durch eine Transformation aus dem Originalgraphen erzeugt.
Sei unser Originalgraph das Tupel $G = (V, E)$ mit $|V| = n$ und $|E| = m$, wobei V die Menge der Knoten und E die Menge der Kanten ist. Der dazugehörige Kantengraph ist ein Tupel $\mathcal{L}(G) = (LV, LE)$ mit $\eta := |LV|$ und $\psi := |LE|$. Dabei ist $\eta = |E| = m$ und ψ ist gleich der Anzahl der Pfade der Länge 2 im Graphen. Der Kantengraph wird gebildet durch eine Abbildung $\Phi : E \to LV$, die zunächst alle Kanten des Ursprungsgraphen auf je einen neuen Knoten abbildet. Es wird also ein neuer Graph $\mathcal{L}(G)$ erstellt und für jede Kante e im Originalgraphen ein Knoten $\Phi(e)$ in den Kantengraphen eingefügt. Bei der Transformation wird nun zwischen diesen neuen Knoten immer eine gerichtete Kante im Kantengraphen eingefügt, wenn die beiden originalen Kanten in G adjazent sind und einen gerichteten Pfad in G bilden. Die Richtung der neuen Kante verläuft in Richtung dieses Pfades (der Länge 2). Die Transformation kann folgendermaßen verdeutlicht werden:

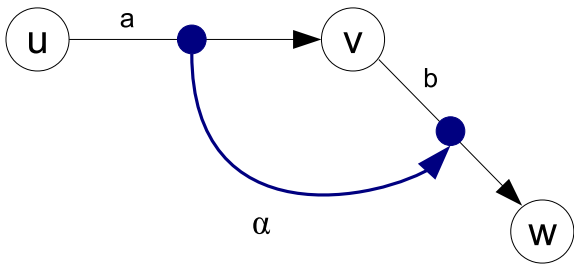


Abbildung 3.2: Erzeugung von Knoten und Kanten des Kantengraphen (blau) aus einem Ursprungsgraphen (schwarz)

Zur Unterscheidung bezeichnen wir von nun an die Knoten und Kanten des Ursprungsgraphen als „Originalknoten“ bzw. „Originalkanten“. Mit den Worten „Knoten“ und „Kanten“ werden dagegen fortan diejenigen des Kantengraphen bezeichnet. Mit dem Kantengraphen sind wir in der Lage, für jedes Tripel (Originaleingangskante, Orginalknoten, Originalausgangskante) Kosten zu vergeben, da jede Kante im Kantengraphen den Übergang von einer Originalkante zu einer anderen im ursprünglichen Graphen darstellt. Weiterhin kann ein gefundener Weg im Kantengraphen durch Rückwärtszuordnung der Knoten zu den jeweiligen Originalkanten wieder in einen Weg im Originalgraphen überführt werden.

Betrachten wir erneut unseren Beispielgraphen aus Abbildung 3.1. In diesem gibt es drei Pfade von Originalknoten s zu Originalknoten t. Die kürzeste Route in diesem Graphen über die Originalkanten (b, e, c) besteht aus zwei Abbiegungen an Originalknoten 2 und 3, die beiden einfacheren Routen (a, c) und (b, d) jeweils aus einer dieser Abbiegungen. Da man sich bei jeder Abbiegung in die richtige Spur einordnen und die Verkehrsführung im Auge behalten muss, wird jede Abbiegung in unserem Beispiel nun mit Kosten bestraft. An Originalknoten 2 vergeben wir für die einfache Linkskurve von Originalkante b zu Originalkante d Kosten von 1. Da die engere Kurve zu Originalkante e schwieriger zu fahren ist, wird diese Abbiegung mit Kosten von 2 bestraft. Aufgrund besonders komplizierter und verwirrender Ampelschaltungen an der Kreuzung an Originalknoten 3 sind die beiden Abbiegungen dort sogar jeweils noch höher bestraft (Kosten von 3 und 4). In dieser beispielhaften Situation sieht der Kantengraph zu unserem Beispielgraphen 3.1 dann folgendermaßen aus:

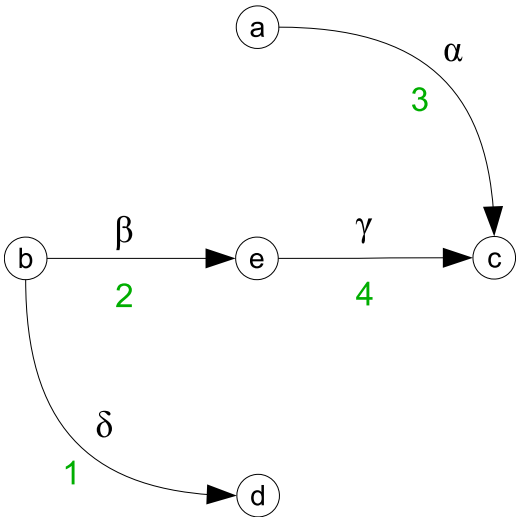


Abbildung 3.3: Kantengraph zum Ursprungsgraphen aus Abbildung 3.1

Leider ist in diesem Graphen die ursprüngliche Information über die Längen der Originalkanten nicht mehr enthalten. Diese kann in diesem Graphen nicht vollständig gespeichert werden, da es sonst zwangsläufig zu einem Informationsverlust käme. Würde man beispielsweise die Distanzen einer Originalkante zusätzlich jeweils den ausgehenden Kanten des entsprechenden Knotens zuweisen, ginge die Distanzinformation verloren, wenn ein Originalknoten keine ausgehenden Kanten mehr hat. Dessen eingehende Originalkanten hätten dann keine „Nachfolger“ mehr mit denen sie einen Pfad bilden könnten, so dass die entsprechenden Knoten im Kantengraphen keine ausgehenden Kanten hätten, denen Distanzinformationen zugerechnet werden könnten. Analog verhält es sich mit den eingehenden Kanten. Grafik 3.4 verdeutlicht dies.

Der Kantengraph kann also nur zur Berechnung simpelster Routen verwendet werden (wie z.B. in [Win01]), aber nicht zur gleichzeitigen Berechnung von Distanzen. Um aber bei der Berechnung einer Route sowohl Distanz als auch Einfachheit einbeziehen zu können, wird ein neues Konstrukt benötigt, dass es in früheren Arbeiten auf dem Gebiet der simplen Routen nicht gab, und in dem beide Informationen vorhanden sind. Dies erreichen wir mit einer Modifikation des Kantengraphen.

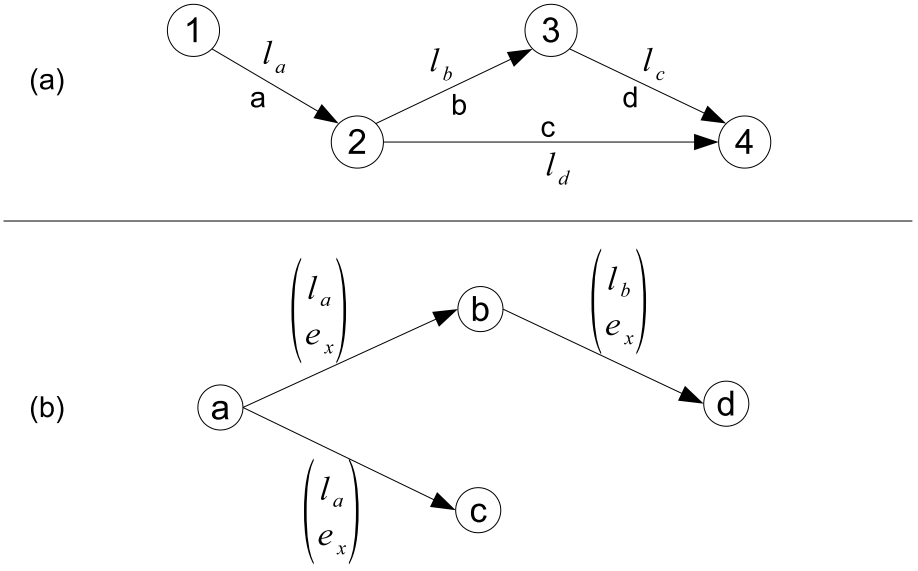


Abbildung 3.4: Negativbeispel für verlustbehaftete Transformation der Distanzen: Originalgraph (a) und Transformation mit Distanzen an ausgehenden Kanten (b)

3.3 Der Erweiterte Kantengraph

Der Kantengraph wird zum *erweiterten Kantengraphen*, indem wir ihn um weitere Knoten ergänzen. Der Unterschied zum normalen Kantengraphen besteht darin, dass bei der Transformation aus dem Originalgraphen nun nicht mehr für jede Originalkante nur ein Knoten angelegt wird, sondern zwei Stück. Die Transformation Φ zerfällt damit in zwei Transformationsfunktionen:

$$\begin{aligned} \Phi_E &: E \to PV \\ \Phi_A &: E \to PV \end{aligned} \tag{3.1}$$

Diese beiden neuen Knoten können gewissermaßen als Ein- und Ausgangsknoten der Originalkante gesehen werden. Sie sind verbunden durch eine Kante, die den Wert für die Distanz der Originalkante als Kostenelement trägt. Um zwischen Distanz und Einfachheit unterscheiden zu können, werden Kosten als zweidimensionaler Vektor modelliert, mit der Entsprechung $\binom{Distanz}{Einfachheit}$. Kosten in Form von Vektoren wurden in der Vergangenheit beispielsweise in [CM85] verwendet. Es gibt nun zwei Sorten von Kanten. Zum Einen diejenigen, die zwischen zwei Knoten im erweiterten Kantengraphen verlaufen, die dieselbe Kante des Originalgraphen repräsentieren. Sie modellieren die Kosten der ursprünglichen Kanten, also die Distanz, die zurückgelegt werden muss. Demzufolge erhalten sie Kosten von $\binom{Distanz}{0}$. Zum anderen die

Kanten, die die Übergänge zwischen zwei verschiedenen Kanten des Originalgraphen an einem Knoten repräsentieren, also einen Entscheidungspunkt in der Form $(vonKante\,,\,überKnoten\,,\,zuKante)$ darstellen. Diese Kanten tragen ähnlich wie im normalen Kantengraphen die Kosten der Einfachheit der Abbiegung als entspechenden Vektor $\binom{0}{Einfacheit}$.

Wir betrachten erneut die Einbettung des schon bekannten Testgraphen:

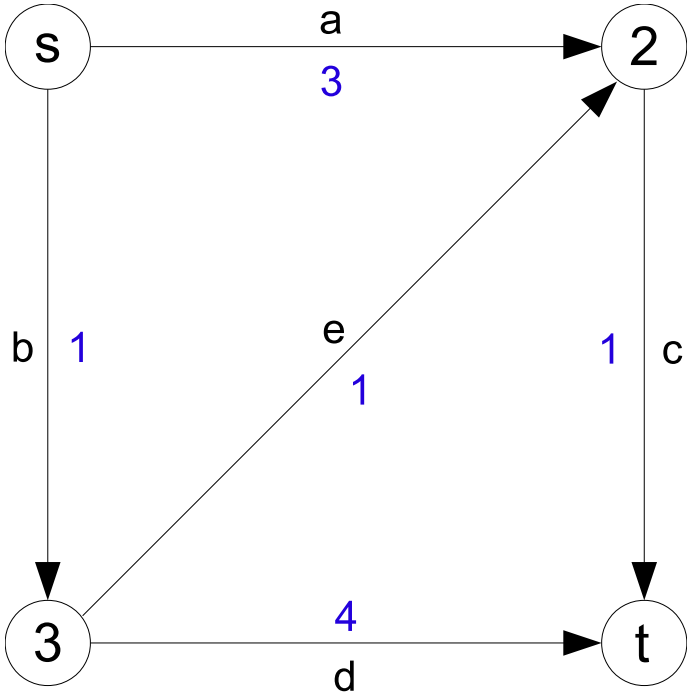


Abbildung 3.5: Einbettung eines Graphen mit einer kürzesten und zwei einfacheren Routen vom Start (s) zum Ziel (t)

Wendet man die erweiterte Transformation entsprechend wieder auf diesen Beispielgraphen, beziehungsweise seinen Kantengraphen aus Abbildung 3.3, an, erhält man schließlich einen Graphen mit seinen entsprechenden Kostenvektoren wie in Abbildung 3.6 gegeben.

Somit lassen sich im erweiterten Kantengraphen sowohl Werte für Einfachheit als auch für die Distanz einer Route modellieren.
Ein Algorithmus, der nur die kürzeste Route zwischen zwei Punkten sucht, müsste nichts weiter tun, als den kürzesten Weg bezogen auf die erste Komponente aller Vektoren an den Kanten zu bestimmen. Da alle Kanten, die lediglich Kosten von Abbiegungen modellieren, in der Entfernungskomponente ihres Kostenvektors den Wert Null haben, wäre ihre Relaxierung eine neutrale Operation, die das Ergebnis

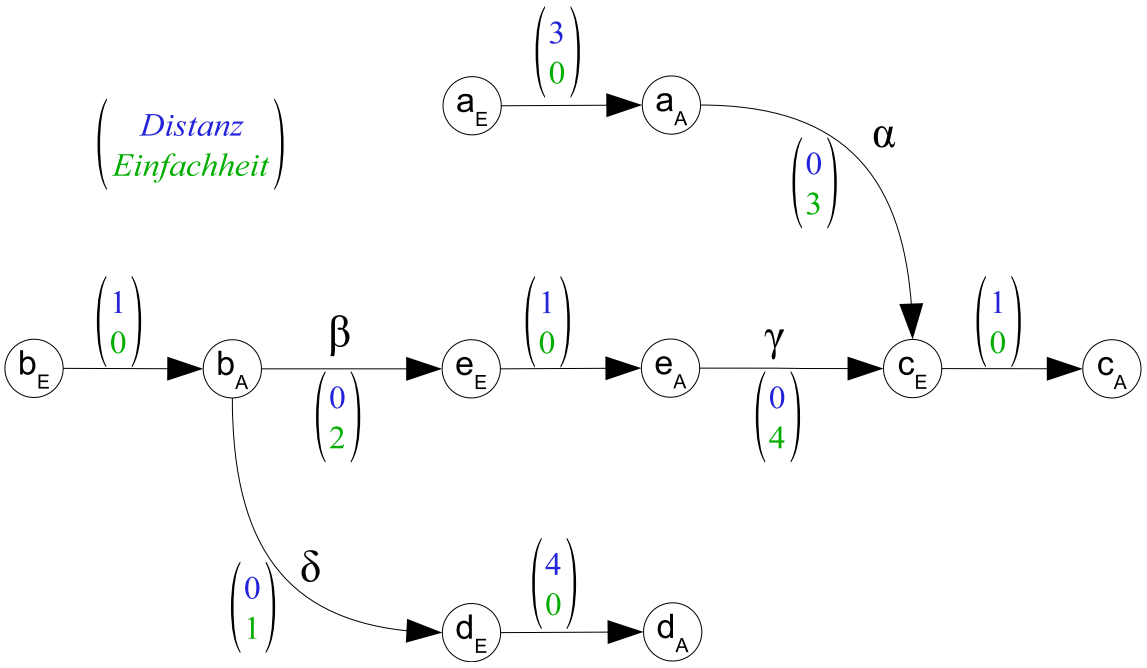


Abbildung 3.6: Erweiterter Kantengraph zum Ursprungsgraphen aus Abbildung 3.1 und zum Kantengraphen aus Abbildung 3.3

nicht verändern würde, welches der Entfernung des kürzesten Weges im Originalgraphen entspricht. Der zugehörige Pfad im Ursprungsgraphen kann leicht durch Zuordnung der Knoten zu ihren Originalkanten gefunden werden.
Auf die gleiche Weise kann ein simpelster Weg zwischen zwei Punkten gefunden werden, wenn man den kürzesten Weg im erweiterten Kantengraphen nur im Bezug auf die zweite Komponente der Kostenvektoren berechnet. Dieser stimmt dann mit dem simpelsten Weg überein, der sich durch alleinige Verwendung des Kantengraphen - ohne Erweiterung - ergeben hätte.

Die Informationen des Originalgraphen bleiben also im erweiterten Kantengraphen erhalten, sowie zusätzlich die Informationen über die Einfachheit der Abbiegungen. Der erweiterte Kantengraph kann nun also als Grundlage für den Algorithmus dienen, der sowohl Einfachheit als auch Distanz berücksichtigt, um den besten Kompromiss an Einfachheit bei Einhaltung einer Distanzgrenze zu finden.

3.4 Kostenfunktionen

Mit dem erweiterten Kantengraphen haben wir nun ein Kostrukt, mit dem es uns möglich ist, viele beliebige Definitionen der Einfachheit zugrunde zu legen, und dabei Routen sowohl im Bezug auf Distanz als auch auf Einfachheit zu berechnen und zu

messen. Als nächstes gilt es, die oben angedeuteten Kriterien der Einfachheit so zu modellieren, dass sie in dem erweiterten Kantengraphen verwendet werden können. Wie wir bereits gesehen haben, arbeitet der erweiterte Kantengraph auf zweidimensionalen Kostenvektoren. Die Distanzen des Originalgraphen befinden sich auf den Kanten, die zwei Knoten mit derselben Entsprechung im Originalgraphen verbinden. Wie bei der Transformation oben schon vorweg genommen, wollen wir uns in dieser Anwendung darauf beschränken, Kosten nur für Abbiegungen, d.h. für den Übergang von einer Kante auf eine andere an einem Knoten, zu vergeben. Theoretisch denkbar wären aber auch Modellierungen, die den Wert der Einfachheit auf den „Distanz-Kanten“ anstatt mit Null mit einem positiven Wert besetzen, beispielsweise falls es Baustellen auf der Straße gibt, die das Fahren auf der Straße anstrengender machen können. Wie bereits vorher erwähnt, lassen sich die Verfahren dieser Arbeit leicht auch auf solche Modelle anwenden.

Oben haben wir bereits verschiedene Kriterien gesehen, die die Einfachheit einer Abbiegung für den Menschen ausmachen können. Diese und andere Kriterien müssen nun als Kosten modelliert werden. Hierzu definieren wir uns Kostenfunktionen, die ein Tripel $(vonKante\,,\,überKnoten\,,\,zuKante)$ abbildet auf einen Wert $k \geq 0$. Die Kante zwischen dem Ausgangsknoten der Ursprungskante und dem Eingangsknoten der originalen Zielkante erhält dann als Kostenelement den Vektor $\binom{k}{0}$.
Nachfolgend sind verschiedene mögliche Kostenfunktionen definiert, die vorstellbare Zielsetzungen beschreiben.

Konstante Kosten. Die erste recht naheliegende Methode ist die, einfach jede Abbiegung mit konstanten Kosten $k \geq 0$ zu versehen. Auf diese Weise wird die Route gesucht, welche die wenigsten Entscheidungspunkte benötigt. Da bei der Modellierung des Straßennetzes aber eine Straße aus vielen Kanten im Originalgraphen bestehen kann, auch wenn es keine Kreuzungen auf dem Weg gibt (beispielsweise wenn eine Straße geschwungen verläuft), wird mit diesem Ansatz auch das Geradeausfahren auf einer langen Straße unter Umständen bestraft.
Um diesem Verhalten vorzubeugen, kann man lange Pfade im Originalgraphen, die nur aus Knoten mit Grad 2 (genauer gesagt einer Eingangs- und einer Ausgangsoriginalkante) bestehen, zu einer einzigen Kante zusammenfassen.

Winkel der Abbiegung. Der zweite Ansatz sieht vor, jede Abbiegung entsprechend des zu fahrenden Winkels zu bestrafen. Die Kosten sind dabei linear fallend mit der Größe des Winkels. Der größte Winkel von 180 Grad entspräche dann den niedrigsten Kosten, in diesem Fall Kosten von 0, da direktes Geradeausfahren am

einfachsten ist, während der kleinste Winkel von 0 Grad eine 180 Grad Wendung des Fahrers erfordern würde, und somit am höchsten bestraft wird.
Außerdem bietet es sich hierbei an, eine Quantisierung der Winkel in Intervalle vorzunehmen, damit kleine Schwankungen der geographischen Anordung von Straßen keine unterschiedlichen Kosten verursachen, die nicht im Verhältnis zur tatsächlich größeren oder kleineren Einfachheit zweier Abbiegungen stehen. Die folgende Grafik veranschaulicht die Einteilung.

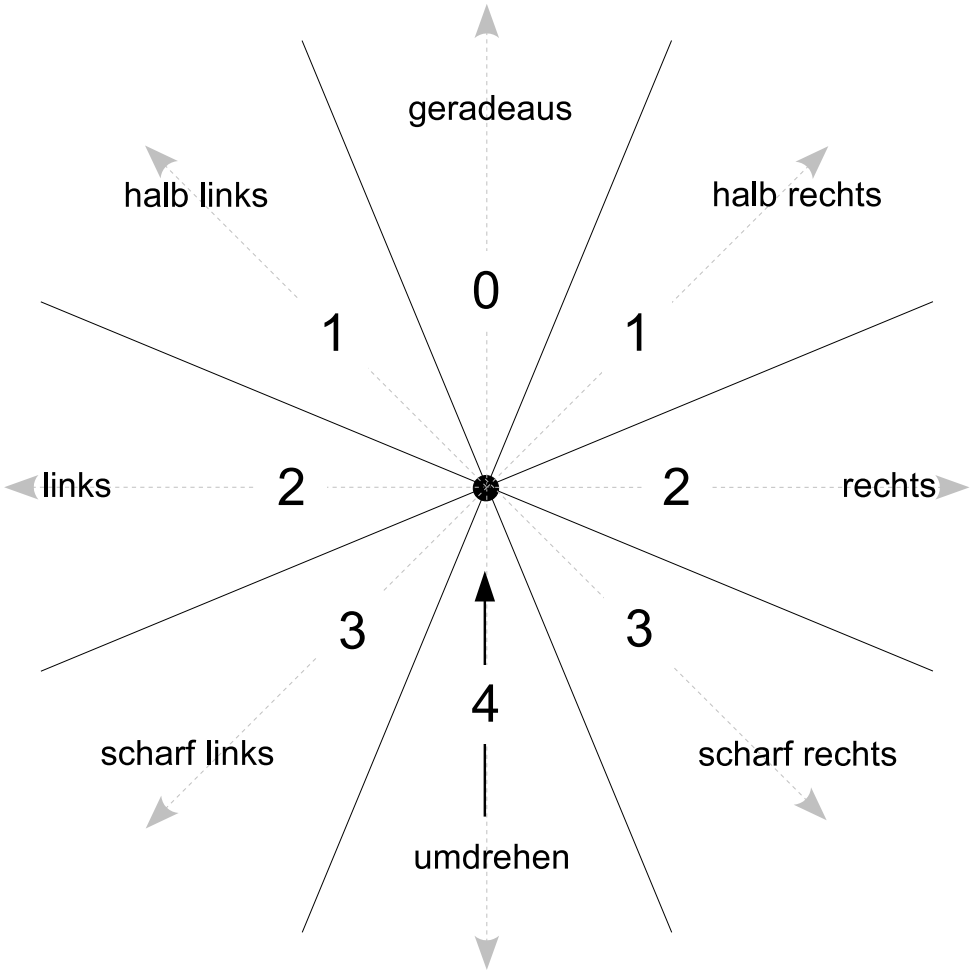


Abbildung 3.7: Einteilung des 360° Intervalls in 8 Teile zu je 45° mit zugehörigen Kosten

Knotengrad des passierten Knotens. Weiterhin interessant ist der Grad eines Knotens im Originalgraphen, an dem die Entscheidung stattfindet. Wie schon gesehen, findet an einer Kreuzung mit Knotengrad 2 normalerweise keine Entscheidung statt, da es sich hier nur um einen inneren Originalknoten in einer längeren, eventuell aber geschwungenen Straße handelt. Je mehr Straßen sich aber treffen, desto

unübersichtlicher wird die Kreuzung, da Verkehrsführung und Ampelschaltungen umfangreicher werden. Man stelle sich hierzu den Unterschied vor zwischen einer T-Kreuzung und einer Kreuzung, an der sich fünf Straßen treffen, und es für verschiedene Abbiegungen dann mehr Spuren zum Einordnen gibt und Ähnliches. Die Vergabe von Kosten nach dem Knotengrad vermeidet somit große, unübersichtliche Kreuzungen und zieht Abbiegungen an kleineren Kreuzungen vor.

Eine mögliche Erweiterung könnte hier, wie auch bei anderen Kostenfunktionen, sein, dass man Originalknoten, die Grad 2 haben, also es nur eine Möglichkeit gibt, sich dieser Kreuzung zu nähern und sie zu verlassen, niedriger bewertet (bis hin zu Kosten von Null).

Minimale Winkelauflösung. Die minimale Winkelauflösung beeinflusst, ebenso wie der Knotengrad, die Kompliziertheit einer Kreuzung. Anders als beim Knotengrad zählt hier aber nicht die Anzahl der Straßen, die sich treffen, sondern die Art, auf die sie sich treffen. Es gibt Kreuzungen, an denen zwei Straßen zur gleichen Seite, zum Beispiel nach rechts, abgehen. Die Anweisung „nach rechts abbiegen“ könnte dabei durchaus missverstanden werden. Je weiter die beiden Straßen jedoch auseinander liegen, desto klarer und schneller ist für den Menschen der Unterschied zwischen „rechts“, „scharf rechts“ und „halb rechts“ auszumachen. Die minimale Winkelauflösung betrachtet die beiden benachbarten Kanten zu der Kante, in die man einbiegen will. Der kleinste Abstand zu einem der beiden Nachbarn entscheidet über die Kosten (wobei die Kante, von der man kommt, natürlich nicht berücksichtigt wird, da man in sie nicht aus Versehen einbiegen wird). Enger zusammengelegene Straßen bei der Abbiegung erhalten höhere Kosten, während hohe Winkelauflösungen nur leicht bestraft werden.

Straßennamen. Eine weitere Überlegung ist es, für das Weiterfahren auf der gleichen Straße gar keine Kosten zu vergeben, und ansonsten bei einem Wechsel der Straße eine andere Kostenfunktion, beispielsweise die Winkelfunktion, anzuwenden. Dies hat, wie auch die Winkelfunktion und der Knotengrad, den Vorteil, dass Geradeausfahren, beziehungsweise dem Straßenverlauf zu folgen, als am einfachsten erachtet wird.

Klassifikation von Abbiegungen. Die nun vorgestellte Kostenfunktion konzentriert sich auf die Klassifikation verschiedener Abbiegungstypen. Sie ist angelehnt an die Kosten aus bisherigen Arbeiten von Mark, Duckham und Kulik [Mar86, DK03]. Aus diesem Grund werden sie später in dieser Arbeit auch als „historische Kosten“

bezeichnet. Die Höhe der Kosten orientiert sich an einem von Mark entwickelten Framework, dass für bestimmte Typen von Abbiegungen Sätze zur textuellen Beschreibung der Aktion vorsieht. Die Kosten werden danach vergeben, wie viele Wörter in der Instruktion variabel sein müssen, um Straßennamen und Richtung anzugeben. In der Arbeit von Mark und Kulik wurden die Kosten ebenfalls an dieses Framework angelehnt. Die verschiedenen Typen von Instruktionen sind in folgender Abbildung gegeben:

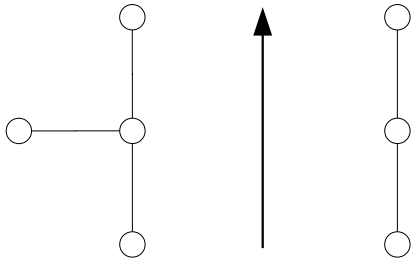
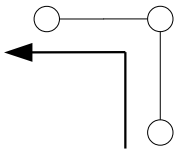
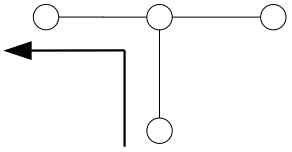
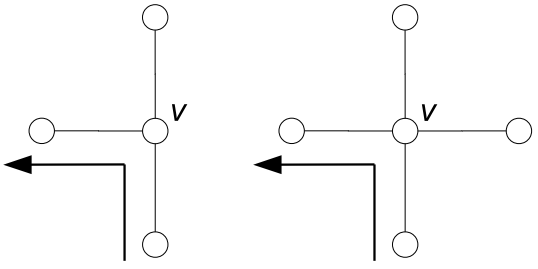
Geradeaus		1
Kurve (keine Kreuzung)		4
Links oder rechts Abbiegen an T-Kreuzung		6
Links oder rechts Abbiegen an anderer Kreuzung		5 + *deg*(v)

Abbildung 3.8: Klassifikation von Abbiegungen nach [DK03] und [Mar86] mit dazugehörigen Kosten (rechte Spalte)

Die Entscheidung, ab wann eine Abbiegung als Rechtsabbiegung zählt, und bis zu welchem Winkel sie noch als geradeaus bezeichnet werden kann, kann dabei aufgrund des Winkels der Kanten zueinander getroffen werden.

3.5 Berechnung von simplen Routen mit Distanzgarantie

Wir haben bisher mit dem erweiterten Kantengraphen ein Konstrukt kennengelernt, mit dem uns die Abbildung von Distanz- und Einfachheitsinformationen in einem Graphen möglich ist. Weiterhin haben wir den Begriff der Einfachheit definiert und verschiedene Methoden kennengelernt, diese Einfachheit zu messen. Wie in früheren Arbeiten zu sehen ist, sind einfachste Routen jedoch in der Praxis teilweise zu lang, um für einen Fahrer brauchbar zu sein.
In diesem Teil der Arbeit soll nun der Algorithmus vorgestellt werden, der in der Lage ist, möglichst einfache Routen zu berechnen, die eine vorgegebene Distanzgrenze nicht überschreiten.

3.5.1 Modellierung

Bisher ist es mit dem erweiterten Kantengraphen möglich, die kürzeste sowie die einfachste Route zwischen zwei gegebenen Knoten zu bestimmen. Da im erweiterten Kantengraphen die Kostenvektoren an den Kanten die Distanz und die Einfachheit symbolisieren, ist der kürzeste Weg derjenige, der im erweiterten Kantengraphen den kürzesten Abstand zwischen einem Start- und einem Zielknoten hat. Entsprechend verhält es sich auch mit der einfachsten Route.
Der erweiterte Kantengraph ist jedoch nur eine Hilfskonstruktion. Start und Ziel sind weiterhin als Knoten im Originalgraphen vorgegeben. Um die kürzeste (einfachste) Route zwischen diesen Punkten zu finden, muss also im erweiterten Kantengraphen die kürzeste (einfachste) Route zwischen den Entsprechungen von Start- und Zielknoten gefunden werden.
Da jede Originalkante einen Knoten darstellt, kann jede Originalkante, die aus dem originalen Startknoten heraus führt (Startkanten), die erste des gewünschten Weges sein. Analog verhält es sich bei den eingehenden Originalkanten des originalen Endknotens (Endkanten). Man erhält dadurch statt eines 1-zu-1-Knoten Problems nun ein Problem mit vielen Start- und vielen Endknoten, bei dem alle Wege von Knoten aus der Menge der Start-Originalkanten zu Knoten aus der Menge der Ziel-Originalkanten Wege zwischen Start- und Ziel darstellen.
Um dieses Problem zu lösen, wird zunächst ein temporärer Startknoten im erweiterten Kantengraphen eingefügt, der ausgehende Kanten mit Kosten $\binom{0}{0}$ zu allen Startknoten (= originale Startkanten) hat. Mit den Endpunkten verfährt man analog. Nun lassen sich kürzeste und simpelste Wege im Originalgraphen finden, indem man kürzeste und simpelste Wege im erweiterten Kantengraphen findet. Dies ist, wie schon angedeutet, mit geringen Modifikationen von Dijkstras Algorithmus möglich.

Da man nun für jeden Weg von einem Start- zu einem beliebigen anderen Knoten sowohl Distanz als auch Einfachheit angeben kann, definieren wir nun einen Pfad als eine Möglichkeit, einen bestimmten Knoten mit bestimmter Distanz und Einfachheit zu erreichen.

Definition 3.5.1. Pfad.
Ein Pfad p, mit dem ein Knoten erreicht werden kann, ist ein 3-Tupel $(\, d_p \,,\, e_p \,,\, u_p \,)$ *mit der Entsprechung* $(\, Distanz \,,\, Einfachheit \,,\, Vorgängerknoten \,)$.

Ein Pfad, der einem Knoten zugeordnet ist, gibt also genau an, welche Distanz und welche Einfachheit ein Weg zu diesem Knoten hat, und von welchem anderen Knoten man sich auf diesem Weg nähert.

Der nächste Schritt ist, die Distanzgarantie zu modellieren. Diese wird dargestellt durch einen Grenzwert ε. Dieser Wert gibt die prozentuale Verlängerung des Weges an, die man, dass heißt der Anwender, bereit ist, in Kauf zu nehmen, um eine einfachere Route zu fahren. Die maximal erlaubte Länge der gewünschten Route ergibt sich dann aus

$$(\, 1 + \varepsilon \,) \cdot Länge_{kürzesterWeg} \quad (:= \text{Distanzgrenze}\,) \tag{3.2}$$

Eine Route, die diese Distanzgrenze einhält, bezeichnen wir als *gut*. Der beste Kompromiss, das heißt die einfachste Route, die noch die Distanzgrenze einhält, bezeichnen wir als die *beste* Route.
Um jetzt die beste Route, also den besten Kompromiss, zu finden, bietet es sich zunächst an, den zweit-, dritt-, und viertkürzesten Pfad und so weiter zum Zielknoten zu suchen, bis der erste Weg die Distanzgrenze überschreitet. Der einfachste der gefundenen Wege wäre dann der beste. Von der anderen Seite verhält es sich ähnlich: Eine Enumeration der schrittweise komplizierter werdenden Wege liefert uns irgendwann den ersten Pfad, der nicht mehr länger ist als die Distanzgrenze. Es ergibt sich also folgende Situation:

$$\begin{array}{ccccc} \underline{\text{kürzester Weg}} & & \underline{\text{bester Weg}} & & \underline{\text{einfachster Weg}} \\ \text{Länge} & \leq & \text{Länge} & \leq & \text{Länge} \\ \text{Einfachheit} & \geq & \text{Einfachheit} & \geq & \text{Einfachheit} \end{array} \tag{3.3}$$

Da eine Enumeration aufgrund der exponentiell großen Zahl an Pfaden nicht in

Betracht kommt, bedienen wir uns eines anderen Ansatzes. Gleichung 3.3 liefert aber dennoch wertvolle Informationen, die später im Algorithmus und zu dessen Beschleunigung verwendet werden können.

Im Wesentlichen folgt der nun vorgestellte *simple-and-short* Algorithmus der Idee, „alle“ Pfade zu berechnen, die vom Start zum Ziel führen, und am Ende den besten Kompromiss auszuwählen. Bei der Berechnung beschränkt sich der Algorithmus allerdings auf die relevanten Pfade. Die Relevanz eines Pfades ergibt sich in erster Linie aus dem Dominanzkriterium für Pfade:

Definition 3.5.2. Dominazkriterium.
Ein Pfad $p = (d_p, e_p, u_p)$, mit dem man einen Knoten erreicht, dominiert einen anderen Pfad $q = (d_q, e_q, u_q)$ genau dann, wenn folgende Bedingungen erfüllt sind:

$$\begin{aligned} &1. \quad d_p \leq d_q \\ &2. \quad e_p \leq e_q \end{aligned} \tag{3.4}$$

Dem liegt die Idee zugrunde, dass ein Pfad, der sowohl kürzer, als auch einfacher ist als ein anderer, diesem immer vorgezogen werden kann, da mit beiden der gleiche Knoten erreicht werden kann. Es muss also nur noch der bessere von beiden weiterhin betrachtet werden. Der dominierte Pfad kann verworfen werden.

3.5.2 Funktionsweise des Algorithmus’

Die Berechnung funktioniert nun wie in Algorithmus 1 gegeben.

Der Ausgangspunkt ist der erweiterte Kantengraph, in dem jeder Knoten unerreicht ist. Alle relevanten Pfade, mit denen ein Knoten erreicht werden kann, werden in einer Pfadliste an jedem Knoten gespeichert. Initial sind alle diese Listen also leer, da kein Knoten mit irgendeinem Pfad erreicht worden ist.

Wie wir gesehen haben, gibt es aufgrund der Transformation (Kapitel 3.3) nun mehrere Startknoten. Als Vorbereitung wird der Algorithmus daher einen temporären Startknoten einfügen und diesen mittels Kanten - mit Kosten $\binom{0}{0}$ - mit allen möglichen Startknoten verbinden (Zeilen 2 - 4). Der temporäre Startknoten erhält einen Initialpfad mit Distanz und Einfachheit von jeweils 0. Als Vorgängerknoten wird das Dummy Element -1 verwendet, da jeder Weg hier beginnt und keine weiteren Vorgänger mehr hat.

Algorithmus 1 Simpelste Route mit Distanzgarantie

Eingabe: Startknoten s, Zielknoten t, Distanzgrenze d_{max},
Erweiterter Kantengraph $\mathcal{L}(G)$

Ausgabe: Einfachste Route, die die Distanzgrenze einhält

1: Temporärer Startknoten c mit Pfadliste $P = \langle\ (0, 0, -1)\ \rangle$
2: **for all** ausgehende Originalkante k von s **do**
3: erstelle temporäre Kante $(c, \Phi_E(k))$ mit Kosten $(0, 0)$
4: **end for**
5: PriorityQueue Q
6: $Q.insert(c)$
7: **while** Q ist nicht leer **do**
8: Knoten $u = Q.getMin$
9: Pfad $p = u.simplestNotPushedPath$;
10: **for all** ausgehende Kanten k von u **do**
11: //Relaxierung von Kante k
12: Knoten $v = k.targetNode$
13: $join(p, v.pfade)$ //Inklusive Dominanz- und Ausschlusskriterien
14: **if** Pfadliste an v wurde verändert **then**
15: **if** Q enthält v **then**
16: $Q.decreaseKey(v, e_{v.simplestNotPushedPath})$
17: **else**
18: $Q.insert(v)$
19: **end if**
20: **end if**
21: **end for**
22: markiere p als gepusht
23: $p = u.simplestNotPushedPath$
24: **if** $p = \emptyset$ **then**
25: $Q.remove(u)$
26: **else**
27: $Q.increaseKey(u, e_p)$
28: **end if**
29: **end while**

Der Algorithmus verwendet eine *Priority Queue*, um die abzuarbeitenden Knoten zu speichern. Dabei handelt es sich um eine adressierbare Prioritätsliste, die die Operationen *getMin* (Zugriff auf das Element mit kleinsten *Key*), *insert*, *remove*, *decreaseKey* und *increaseKey* unterstützt. Mit einem *Fibonacci-Heap* benötigt die *remove*-Operation amortisiert logarithmische Zeit, während die Operationen *getMin*, *insert* und *decreaseKey* in konstanter Zeit ablaufen [MS08]. Die *increaseKey*-Operation kann durch eine *remove* und anschließende *insert*-Operation realisiert werden und benötigt damit ebenfalls amortisiert logarithmische Laufzeit.

Im Wesentlichen handelt es sich bei diesem Algorithmus um eine Modifikation von Dijkstras Algorithmus, die sich vor allem im Abbruchkriterium der Schleife und in der Relaxierung der Kanten unterscheidet. Daher sind auch hier nur nicht negative Kosten zugelassen. Der temporäre Startknoten wird als erstes in die Prority Queue eingefügt. In der Schleife des Algorithmus' (Zeilen 7 - 29) wird immer derjenige Knoten mit dem kleinsten Schlüssel in der Priority Queue betrachtet. Diesen aktuell betrachteten Knoten nennen wir u. Es bietet sich hierbei an, als Schlüssel eines Knotens immer den Wert der Einfachheit des kleinsten, noch nicht abgearbeiteten Pfades an dem Knoten zu benutzen. Da wir prinzipiell daran interessiert sind, die Einfachheit zu minimieren (unter gewissen Nebenbedingungen), liefert dies eine sinnvolle Abarbeitungsreihenfolge.

Ein aktuell betrachteter Knoten u besitzt eine Pfadliste bestehend aus Pfadelementen $(\, d_i \,,\, e_i \,,\, vorgänger_i \,)$. Zusätzlich wird für jedes dieser Pfadobjekte ein Zustand $\in \{\text{gepusht}, \text{nicht gepusht}\}$ gespeichert, der angibt, ob ein Pfad bereits betrachet wurde. Ähnlich wie beim Dijkstra werden nun die ausgehenden Kanten von u relaxiert. Das bedeutet hier, dass der einfachste, noch nicht weitergegebene (nicht gepushte) Pfad an u an alle seine Nachfolgerknoten v „weitergegeben“ wird. Der Pfad $(\, d_i \,,\, e_i \,,\, vorgänger_i \,)$ wird dabei über eine ausgehende Kante k mit Kosten $(\, d_k \,,\, e_k \,)$ weitergegeben als $(\, d_i + d_k \,,\, e_i + e_k \,,\, u \,)$.

Die Weitergabe wird realisiert durch eine *join* Operation (Zeile 13). Der Pfad p wird dabei mit der vorhandenen Pfadliste an v vereint, wenn er gewisse Bedingungen erfüllt, die im Folgenden erläutert werden. Eine wichtige Forderung ist an dieser Stelle der Ausschluss von allen Routen, die die Distanzgrenze von $(\, 1 + \varepsilon \,)$ überschreiten. Da die Distanzgrenze das Maximum festlegt, das der Anwender des Algorithmus' bereit ist, in Kauf zu nehmen, ist eine weitere Betrachtung längerer Pfade irrelevant. Wichtig ist außerdem, dass bei der *join*-Operation dominierte Pfade eliminiert werden. So wird sichergestellt, dass eine Pfadliste immer nur aus sich nicht dominierenden Pfaden bestehen kann.

An dieser Stelle des Algorithmus' werden auch Ausschlusskriterien mit in die Berechnung einbezogen (näheres dazu unter „Ausschlusskriterien für Pfade“ in Abschnitt 3.5.3). Obige Forderung des Einhaltens der Distanzgrenze kann zwar ebenfalls als Ausschlusskriterium angesehen werden, ist jedoch für den Arbeitsaufwand und die reale Laufzeit des Algorithmus' von erheblich größerer Bedeutung und stellt daher eher einen Teil des Algorithmus' an sich dar.

Diese *join* Operation liefert immer eine sortierte Liste aus nicht dominierten Pfaden und hat mehrere mögliche Ausgänge.
Als erstes kann der Pfad p durch die vorherige Addition der Gewichte der Kante als nicht mehr relevant erachtet werden, wenn er die Distanzgrenze überschreitet oder ein anderes Ausschlusskriterium zutrifft.
Ist dies nicht der Fall, wird als nächses mittels binärer Suche die richtige Einfügeposition für p in der Pfadlste von v bestimmt. Dabei kann es sein, dass es einen Pfad in der Liste von Knoten v gibt, der p dominiert. In diesem Fall wird p auch nicht eingefügt. Trifft dies nicht zu, wird p an der richtigen Stelle in die Pfadliste eingefügt, und überprüft, ob p einen oder mehrere bereits vorhandene Pfade dominiert. Die dominierten Pfade werden dann aus der Liste gelöscht.

Das folgende Beispiel verdeutlicht eine exemplarische *join*-Operation. Zur Übersicht sind hier die Vorgängerknoten nicht eingezeichnet. Der neue, weitergegebene Pfad $(3, 4)$ wird in die Liste eigefügt und dominiert dort drei vorhandene Pfade. Die resultierende Liste rechts enthält dann nur noch vier Pfade.

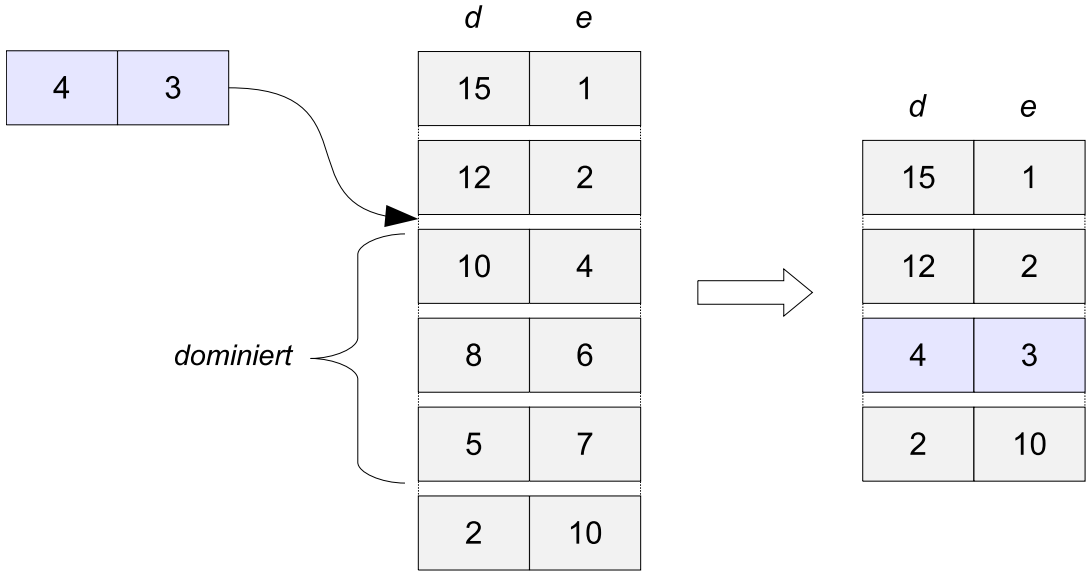


Abbildung 3.9: Beispiel einer *join*-Operation. Die linke Spalte d gibt die Distanz an, die rechte Spalte e die Einfachheit

Sollte p nicht eingefügt werden, weil er dominiert wird oder durch Ausschlusskriterien nicht mehr relevant ist, können dann die Zeilen 14 bis 20 des Codes übersprungen werden. Dies verhält sich ähnlich wie bei Dijkstras Algorithmus, wenn beim Relaxieren einer Kante kein kürzerer Weg gefunden wurde als der bereits vorhandene. Die Relaxierung der Kante hatte in diesem Fall keine Auswirkungen, weshalb sich Position oder Vorhandensein von v in der Queue nicht ändern. Andernfalls wird entweder die Position von v in der Queue aktualisiert (da der neu erhaltene Pfad nun der einfachste an v sein könnte), beziehungsweise v neu in die Queue eingefügt. Für die Aktualisierung der Position genügt die *decreaseKey* Operation, da der neue simpelste, noch nicht gepushte Pfad maximal den gleichen Wert haben kann wie der vorherige, beziehungsweise ansonsten kleiner ist.

Nach der Relaxierung der ausgehenden Kanten wird der nun weitergegebene Pfad p als *gepusht* markiert. Dieser muss später im Algorithmus nicht mehr betrachtet werden, verbleibt aber als Pfadobjekt in der Liste, da er für die Information, über welche Knoten die gefundene Route gelaufen ist, unter Umständen notwendig ist. Sollte u danach keine nicht gepushten Pfade mehr in seiner Pfadliste besitzen, kann er aus der Priority Queue entfernt werden. Andersfalls wird der Schlüssel von u in der Queue auf die Einfachheit des einfachsten, noch nicht gepushten Pfades erhöht. Durch diese *increaseKey* Operation und die mögliche *decreaseKey* Operation bei der Relaxierung ist sichergestellt, dass der nächste abzuarbeitende Knoten immer derjenige ist, der den aktuell simpelsten Pfad zur Verfügung stellt.

Wichtig bei der *join*-Operation ist, dass bei Gleichheit zweier Pfade (also gleiche Werte sowohl in Distanz als auch in Einfachheit) der bereits in der Liste vorhandene Pfad dominiert, also Vorrang erhält, und dies somit nicht dazu führen kann, dass die Liste als verändert angesehen wird. Auf diese Weise wird verhindert, dass auch bei Kosten von Null an Kanten, welche bei vielen Kostenfunktionen durchaus auftreten, keine Kreise entstehen können, und somit der Algorithmus nicht in eine Endlosschleife laufen kann.

Da der Algorithmus alle relevanten Pfade verfolgt (Näheres zur Relevanz unter „Ausschlusskriterien für Pfade“), erhält man am Ende an allen erreichbaren Knoten, die im Originalgraphen Eingangskanten des Zielknotens darstellen, Listen, über welche Pfade diese erreicht werden können. Die Vereinigung dieser Listen - unter Berücksichtigung von Dominanzen - liefert eine Menge von sich gegenseitig nicht dominierenden Pfaden, mit denen der originale Zielknoten erreicht werden kann. Der einfachste, der die Distanzgrenze einhält, ist der gesuchte beste Kompromiss und damit der beste Pfad. Alle nachfolgenden Pfade allerdings halten ebenfalls die Distanzgrenze ein.

Sie sind also ebenfalls Kompromisse, die zwar komplizierter werden, sich jedoch in ihrer Distanz dem kürzesten Pfad annähern (siehe Gleichung 3.3). Der letzte Kompromiss stellt wieder die kürzeste Route dar. Die Einfachheit des kürzesten Weges ist am Ende des Algorithmus' also ebenfalls bekannt, was nützlich ist, da man um die Distanzgrenze auszurechnen nur seine Länge mit einem normalen Dijkstra zu berechnen braucht, die Einfachheit dadurch normalerweise aber nicht kennt.
Alle diese *guten* Alternativen zu dem besten Pfad bis hin zum kürzesten ergeben sich bei der Durchführung des Algorithmus' also automatisch mit, und können so je nach Anwendung vom Benutzer ebenfalls in Betracht gezogen werden.

3.5.3 Ausschlusskriterien für Pfade

Kommen wir nun zu den bereits erwähnten Ausschlusskriterien für Pfade. Sie verkürzen die praktische Laufzeit und beeinflussen das Verhalten des Algorithmus' so, dass irrelevante Pfade nicht länger verfolgt werden. Die Idee dahinter ist, sogenannte Ausschlusskriterien für Pfade zu definieren, um die Berechnung zu beschleunigen. Ausschlusskriterien eliminieren bestimmte Pfade in den Pfadlisten an den Knoten, wenn diese bestimmte Anforderungen nicht mehr erfüllen. Das Ganze passiert direkt innerhalb der *join*-Operation.

Das erste Kriterium, das bereits oben als Teil des Algorithmus' erläutert wurde, ist das Verwerfen von Pfaden, die die Distanzgrenze überschreiten. Da negative Kosten nicht zugelassen sind, kann ein Pfad weder in Distanz, noch in Einfachheit zu einem späteren Zeitpunkt wieder kürzer oder einfacher werden. Ein Pfad, der also einmal länger als $(1 + \varepsilon) \cdot Länge_{kürzesterWeg}$ geworden ist, kann durch Weitergabe an folgende Knoten niemals wieder relevant werden, und kann somit aussortiert werden. Allerdings gibt es noch weitere Einschränkungen, die unter bestimmten Voraussetzungen getroffen werden können, die irrelevante Pfade während der Berechnung identifizieren und ausschließen können.
Wenn man sich Gleichung 3.3 noch einmal anschaut, fällt auf, dass die Länge eines besten Pfades kleiner oder gleich der des einfachsten Pfades sein muss (da sonst der einfachste der beste wäre). Damit lässt sich ein weiteres Ausschlusskriterium definieren. Weiterhin haben wir gesehen, dass neben dem besten Pfad weitere gute Alternativen bis hin zum kürzesten Weg bei der Ausführung des Algorithmus' automatisch mit berechnet werden, da alle Pfade weiter verfolgt werden, bis sich keinerlei Änderungen mehr ergeben. Im Laufe des Algorithmus' werden aber einige der Endknoten (also der Knoten, die Abbildungen von Eingangskanten des originalen Zielknotens sind) früher erreicht als andere. Dies bedeutet, dass damit schon Wege zum Ziel gefunden wurden, der Algorithmus aber noch weitere Routen, die das Ziel

noch nicht erreicht haben, weiter verfolgt, da diese ebenfalls relevante gute Wege darstellen. Wenn man aber bereit ist, auf die Alternativen zu verzichten und nur die beste Route zu finden, kann man Pfade verwerfen, deren Einfachheit höher ist als die bisher kleinste Einfachheit eines Weges, der das Ziel erreicht hat. Dies funktioniert, da alle Pfade, die überhaupt gespeichert werden, die Distanzgrenze nicht überschreiten und somit zulässig sind.
Den aktuell kleinsten Wert speichert man bei der Ausführung des Algorithmus' temporär und aktualisiert diesen, wenn ein Weg mit geringerer Einfachheit gefunden wurde (der initiale Wert bei unerreichten Zielknoten ist auf Unendlich gesetzt). Dies schließt weitere Pfade aus und sorgt dafür, dass im Anschluss nur noch einfachere Pfade gefunden werden.

Diese drei Ausschlusskriterien lassen sich wie folgt formulieren:

Ein Pfad $p = (\, d_p \,,\, e_p \,,\, u_p \,)$ muss nicht weiter betrachtet werden, wenn eine der folgenden Bedingungen erfüllt ist:

$$\begin{aligned} &1. \quad d_p > (\, 1 + \varepsilon \,) \cdot d_{kürzesterWeg} \\ &2. \quad d_p > d_{simpelsterWeg} \\ &3. \quad e_p > e_{bisherEinfachsterWeg} \end{aligned} \tag{3.5}$$

Kommen wir nun noch einmal zu unserem Eingangsbeispiel zurück. Wir betrachten erneut den Beispielgraphen und seinen erweiterten Kantengraphen (Abbildung 3.10). Wie bereits früher erwähnt gibt es in diesem originalen Graphen drei Wege vom Startknoten s zum Zielknoten t. Diese sind:

- $(\, s \,,\, 3 \,,\, 2 \,,\, t)$ mit Distanz 3 und Einfachheit 6
- $(\, s \,,\, 2 \,,\, t)$ mit Distanz 4 und Einfachheit 3
- $(\, s \,,\, 3 \,,\, t)$ mit Distanz 5 und Einfachheit 1

Sei nun ein hypothetischer Fahrer bereit, eine bis zu 50 % längere Strecke in Kauf zu nehmen, um eine einfachere Route zu bekommen. Der kürzeste Weg ist 3 Entfernungseinheiten lang. Die Distanzgrenze beträgt somit $1{,}5 \cdot 3 = 4{,}5$ Einheiten. Man sieht, dass die Strecke über Knoten 2 diese Distanzgrenze einhält.

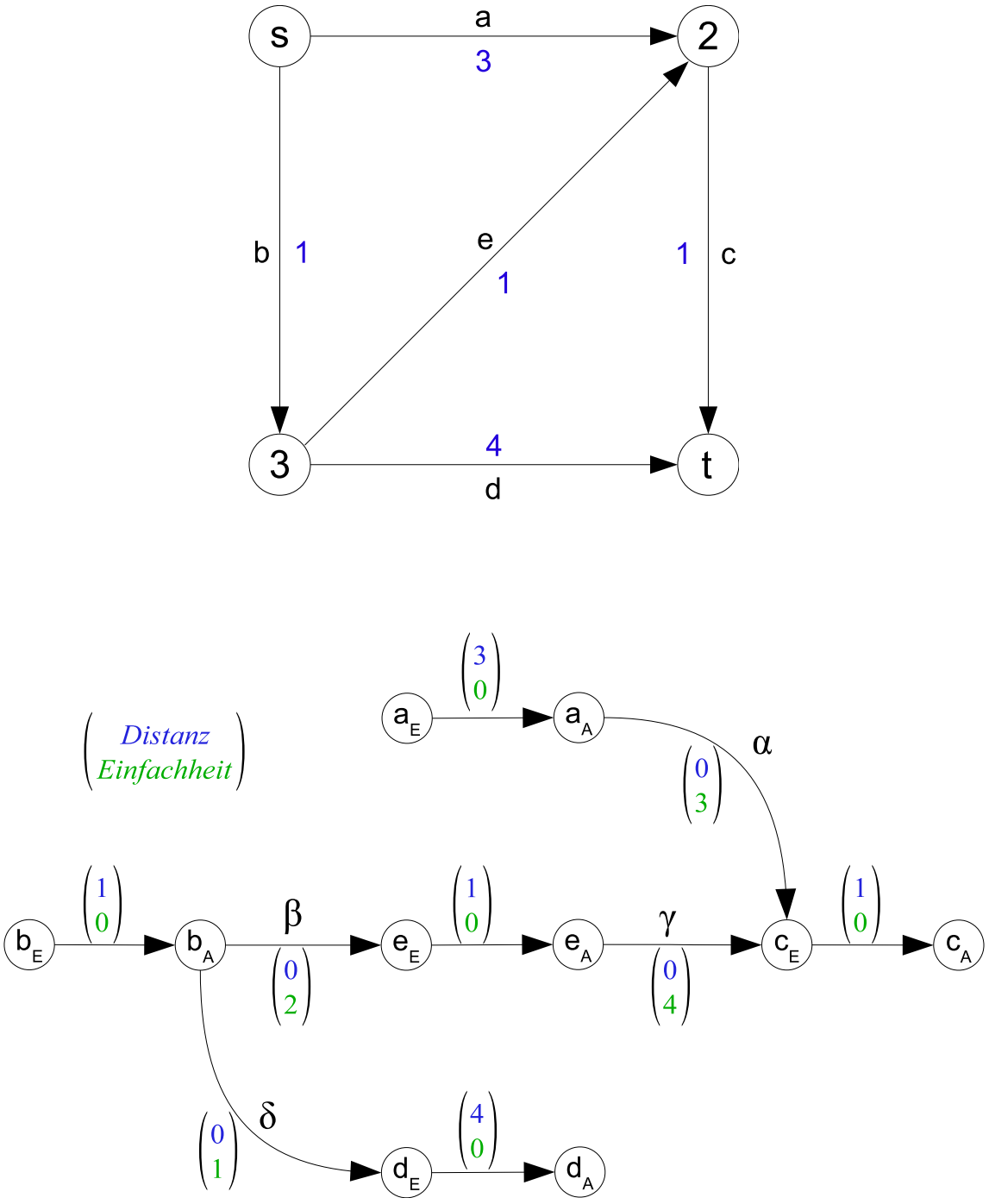


Abbildung 3.10: Beispielgraph und dazugehöriger erweiterter Kantengraph

Der Algorithmus beginnt an einem temporären Startknoten x im erweiterten Kantengraphen, der Kanten zu den Knoten a_E und b_E besitzt. Von dort aus finden Relaxierungen statt. Zu der ersten interessanten Situation kommt es, wenn der Pfad $(1, 1, b_A)$ am Knoten d_E an dessen Nachfolger d_A weiter gepusht werden soll. Bei d_A kommt der Pfad als $(5, 1, d_E)$ an. Ein Vergleich mit der Distanzgrenze sorgt dafür, dass der Pfad an dieser Stelle verworfen wird.
Die nächste nicht-triviale Situation entsteht bei der Relaxierung der Kante $(0, 4)$ zwischen den Knoten e_A und c_E. Zu diesem Zeitpunkt existiert in der Pfadliste des Knotens c_E bereits der Pfad $(3, 3, a_A)$. Der an Knoten e_A vorhandene Pfad $(2, 2, e_E)$ wird als $(2, 6, e_A)$ an den Knoten c_E weiter gepusht. Da keiner der beiden Pfade den anderen dominiert, wird der neue Pfad in die Liste eingefügt. Anschließend werden beide Pfade noch an c_A weiter gegeben.
Als Resultat des Algorithmus' erhält man folgende Situation:

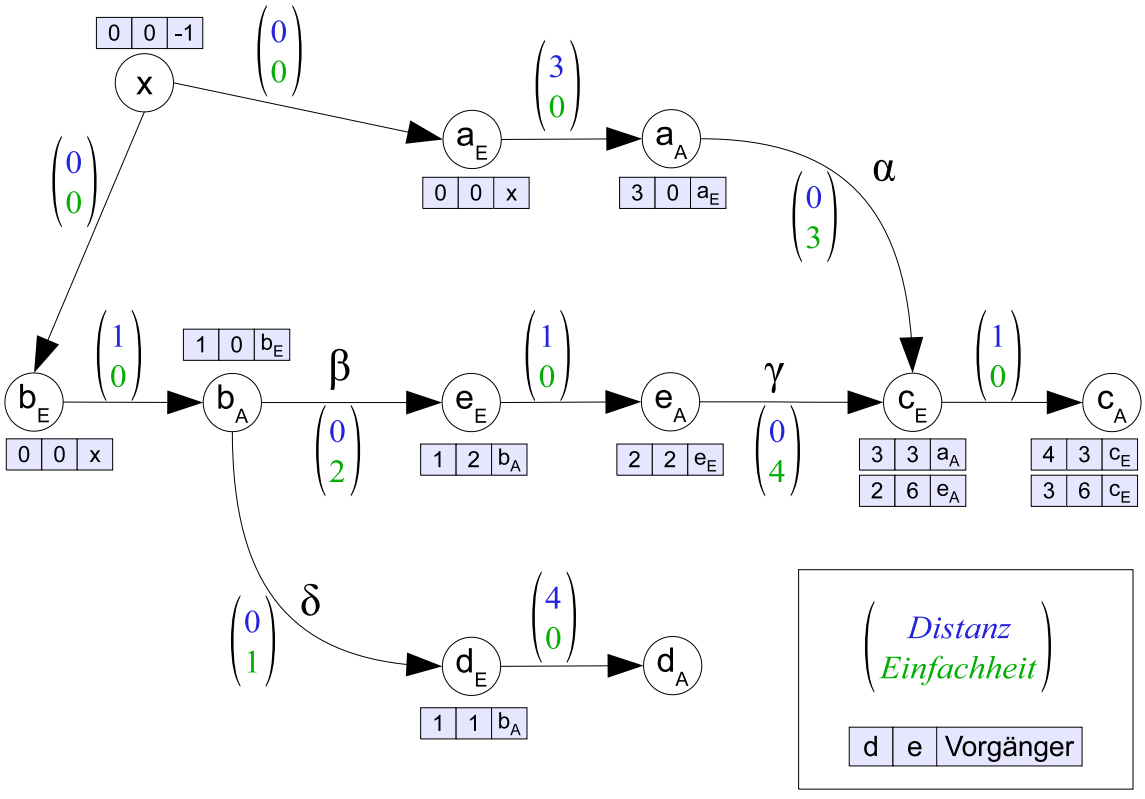


Abbildung 3.11: Erweiterter Kantengraph nach Ausführung des *simple-and-short* Algorithmus'

Der Knoten d_A wurde nicht erreicht, was bedeutet, dass es keinen Pfad gibt, der die Originalkante d benutzt und der die Distanzgrenze einhält. Am Knoten c_A befinden sich zwei Pfade. Der Pfad $(4, 3, c_E)$ ist der gesuchte beste Kompromiss. Der Pfad $(3, 6, c_E)$ stellt den kürzesten Weg dar.

3.5.4 Analyse

Bei dem Algorithmus handelt es sich um eine Erweiterung von Dijkstras Algorithmus. Aus diesem Grund bleiben auch wesentliche Eigenschaften des Algorithmus' erhalten.

Lemma 3.5.1. *Keine bei der Ausführung des simple-and-short Algorithmus' berechnete Route kann einen Kreis enthalten.*

Beweis. Angenommen es entstünde ein Kreis. Der Algorithmus arbeitet ausschließlich mit nicht negativen Kosten (in beiden Dimensionen: Distanz und Einfachheit). Bei Relaxierungen von Kanten können bestehende Pfade in beiden Dimensionen daher nur gleich bleiben oder größer werden. Ein Pfadobjekt q an einem Knoten, das Teil eines Kreises ist, muss mit der Pfadliste an diesem Knoten vereint worden sein. Da der gleiche Knoten im Kreis jedoch schon einmal passiert wurde, existiert dort mindestens ein anderes Pfadobjekt r, dessen Distanz und Einfachheit kleiner oder gleich denen von q sind. Per Definition dominiert bei Gleichheit zweier Pfade in beiden Dimensionen der bereits in der Pfadliste vorhandene. Folglich müsste der Pfad r den Pfad q beim Vereinen der Listen dominiert haben. Der Kreis kann also nicht entstanden sein. □

Im Folgenden soll nun der für die Ausführung des Algorithmus' erforderliche Speicherplatz sowie die Laufzeit untersucht werden. Bei diesen theoretischen Untersuchungen gehen wir von diskreten, ganzzahligen Werten für die Einfachheit aus. Diese Annahme ist mit allen in dieser Arbeit vorgestellten Kostenfunktionen vereinbar und stellt auch für weitere denkbare Kostenfunktionen keine signifikante Einschränkung dar, da eine solche Einteilung leicht durch Quantisierung der Werte erreicht werden kann.
Für die Analyse bezeichnen wir den maximal vorkommenden Wert der Einfachheit, der einer Kreuzung, und damit einer Kante, durch die verwendete Kostenfunktion zugewiesen werden kann mit $maxEinfachheit$. Weiterhin bezeichnen wir den maximalen, in einem Graphen vorkommenden Ausgangsgrad eines Knotens mit $maxDegree$.

Analyse des Platzverbrauchs

Theorem 3.5.2. *Gegeben sei der erweiterte Kantengraph $\mathcal{L}(G) = (KV, KE)$ mit ganzzahligen Kosten für die Einfachheit und $\eta := |KV|$. Der Platzverbrauch des simple-and-short Algorithmus' liegt in $\mathcal{O}(\eta^2 \cdot maxEinfachheit)$.*

Beweis. Bei der Ausführung des Algorithmus' ergibt sich der Platzverbrauch durch die zu speichernden Pfade an jedem Knoten. Durch das Dominanzkriterium kann zu jedem Wert an Einfachheit an jedem Knoten nur maximal ein Pfad existieren. Jeder andere Pfad mit dem gleichen Wert würde entweder dominiert werden oder selbst dominieren.

Nach Lemma 3.5.1 kann bei der Ausführung des Algorithmus' kein Kreis entstehen. Folglich kann jeder Weg im Graphen maximal aus $(\eta - 1)$ Knoten bestehen. Im schlechtesten Fall wird an jedem dieser Knoten der größte Wert an Einfacheit zu der Einfachheit der Route hinzu addiert. Aus diesem Grund ist der maximale Wert für die Einfachheit, den es im Graphen geben kann, nach oben beschränkt durch $(\eta \cdot maxEinfachheit)$. Für alle Pfadlisten erhält man also insgesamt einen Speicherplatzbedarf von $\mathcal{O}(\eta^2 \cdot maxEinfachheit)$.

Die Anzahl der möglichen Kanten im erweiterten Kantengraphen liegt in $\mathcal{O}(\eta^2)$. Für die gesamte Ausführung des Algorithmus' mit η Knoten ergibt sich somit ein Platzverbrauch für die zu speichernden Pfade von $\mathcal{O}(\eta^2 \cdot maxEinfachheit)$. □

Aus dem Beweis von Theorem 3.5.2 ergibt sich das folgende Lemma.

Lemma 3.5.3. *Bei ganzzahligen Werten für die Einfachheit ist die Anzahl der Pfade an jedem einzelnen Knoten nach oben beschränkt durch* $(\eta \cdot maxEinfachheit)$.

Analyse der Worst-Case Laufzeit

Theorem 3.5.4. *Gegeben der erweiterte Kantengraph* $\mathcal{L}(G) = (KV, KE)$ *mit* $\eta := |KV|$. *Es bezeichne* $maxEinfachheit$ *den größten Wert, der durch die verwendete Kostenfunktion vergeben wird, und* $maxDegree_{\mathcal{L}(G)}$ *den größten vorkommenden Ausgangsgrad eines Knotens in* $\mathcal{L}(G)$. *Die theoretische Laufzeit für die Ausführung des simple-and-short Algorithmus' ist gegeben durch* $\mathcal{O}(\eta^3 \cdot maxEinfachheit^2 \cdot maxDegree_{\mathcal{L}(G)})$.

Beweis. Ein Knoten im *simple-and-short* Algorithmus wird nicht wie beim normalen Dijkstra nach seiner Abarbeitung als abgeschlossen markiert. Dies ist darauf zurückzuführen, dass die Anzahl, wie viele Pfade pro Knoten gespeichert werden, nicht auf 1 festgelegt ist, wie dies beim Dijkstra der Fall ist. Ein Knoten kann demnach mehrfach in die Priority Queue aufgenommen und abgearbeitet werden. Die Anzahl der Abarbeitungen pro Knoten ist aber dennoch beschränkt. Unabhängig davon, wie oft der Knoten zwischendurch aus der Priority Queue entfernt und wieder eingefügt wird, kann jeder Pfad, der an diesem Knoten vorhanden ist, nur einmal

weiter gepusht werden. Da immer erst alle einfachsten Pfade abgearbeitet werden, kann später kein Pfad mit gleicher Einfachheit einen bereits gepushten Pfad an einem Knoten dominieren und ersetzen. Aus diesem Grund kann jeder Knoten nur maximal so oft bearbeitet werden, wie er maximal nicht dominierte Pfade besitzen kann. Dies sind nach Lemma 3.5.3 maximal $\eta \cdot maxEinfachheit$. Die Anzahl der Schleifendurchläufe der äußeren *while*-Schleife des Algorithmus' liegt damit insgesamt in $\mathcal{O}(\eta^2 \cdot maxEinfachheit)$.

Wird ein Knoten besucht, werden wie beim Dijkstra alle ausgehenden Kanten relaxiert. Hierbei wird jedesmal eine *join*-Operation ausgeführt, um die Pfadliste des Zielknotens mit dem aktuell weiterzugebenen Pfad zu kombinieren. Bei den Pfadlisten handelt es sich um nach Einfachheit aufsteigend sortierte Listen. Aufgrund des Dominanzkriteriums sind diese daher dann in der Distanzkomponente absteigend sortiert. Das Kombinieren dieser zweidimensionalen Liste mit dem neuen Pfad läuft dabei in Linearzeit der Listengröße, da der neue Pfad theoretisch beliebig viele vorhandene Pfade dominieren kann. Das Einfügen in die Liste kann zwar in logarithmischer Zeit vorgenommen werden, allerdings benötigt das Eliminieren dominierter Pfade lineare Zeit. Die Überprüfung, ob eine Änderung der Liste stattgefunden hat, wird dabei implizit mit ausgeführt. Somit läuft auch die gesamte *join*-Operation in Linearzeit der Listengröße, die sich aus Lemma 3.5.3 ergibt. Es folgt daraus:

Lemma 3.5.5. *Eine join-Operation und damit die Relaxierung einer Kante benötigt maximal* $\mathcal{O}(\eta \cdot maxEinfachheit)$ *Zeit.*

Die gesamte Laufzeit des Algorithmus' setzt sich im Worst Case also zusammen aus $\mathcal{O}(\eta \cdot maxEinfachheit)$ Betrachtungen pro Knoten, wobei jedesmal so viele Relaxierungen vorgenommen werden, die dem Ausgangsgrad des Knotens entsprechen. Eine theoretische Schranke hierfür bildet der größte im erweiterten Kantengraphen vorkommende Ausgangsgrad $maxDegree_{\mathcal{L}(G)}$. Es folgt, dass pro Knoten maximal $\eta \cdot maxEinfachheit \cdot maxDegree_{\mathcal{L}(G)}$ Relaxierungen, und damit *join*-Operationen, vorgenommen werden können.

Für den gesamten Graphen ergibt sich damit eine Laufzeit von

$$\mathcal{O}(\eta \cdot \underbrace{\eta \cdot maxEinfachheit \cdot maxDegree_{\mathcal{L}(G)}}_{\text{Anzahl Relaxierungen}} \cdot \underbrace{\eta \cdot maxEinfachheit}_{\text{Laufzeit einer Relaxierung}})$$

Damit folgt Theorem 3.5.4. □

Analyse der Laufzeit bezogen auf den Originalgraphen

Um nun die theoretische Laufzeit des Algorithmus' in Größen des Originalgraphen statt in denen des transformierten auszudrücken, wollen wir folgendes Theorem beweisen.

Theorem 3.5.6. *Gegeben sei der Originalgraph $G = (V, E)$ mit $m := |E|$. Es bezeichne $maxEinfachheit$ den größten Wert, der durch die verwendete Kostenfunktion vergeben wird, und $maxDegree_G$ den größten vorkommenden Ausgangsgrad eines Knotens in G. Die theoretische Laufzeit zur Berechnung der simpelsten Route, die die Distanzgrenze einhält, liegt in $\mathcal{O}(m^3 \cdot maxEinfachheit^2 \cdot maxDegree_G)$.*

Hierzu beweisen wir zunächst ein Hilfslemma.

Lemma 3.5.7. *Gegeben sei der Originalgraph $G = (V, E)$ und dessen erweiterter Kantengraph $\mathcal{L}(G) = (KV, KE)$. Für die maximal vorkommenden Ausgangsgrade gilt: $maxDegree_{\mathcal{L}(G)} = maxDegree_G$.*

Beweis. Wir betrachten die Veränderung des $maxDegree_G$ bei der Transformation. Dafür nehmen wir an, dass unser originaler Graph schlingenfrei ist und keine Mehrfachkanten besitzt, was bei einem Straßengraphen als realistisch angesehen werden kann. Es gilt für den Knotengrad eines Knotens in einem ungerichteten (nicht erweiterten) Kantengraphen: $deg(\Phi(k)) = deg(x) + deg(y) - 2$ [Vol96]. Dabei bezeichnet $k = (x, y)$ eine Kante im Orignalgraphen und $\Phi(k)$ die Abbildung einer Originalkante auf einen Knoten im Kantengraphen. Diese Gleichung trifft auf unseren erweiterten Kantengraphen jedoch nicht zu. Dieser ist gerichtet und erzeugt auch nur dann eine Kante zwischen zwei Originalkanten, wenn diese einen Pfad der Länge 2 bilden. Ein Knoten im normalen Kantengraphen entspricht einer Kante im Originalgraphen. Am Ende einer Kante entsteht an deren Zielknoten eine Entscheidungssituation, in der in jede ausgehende Kante dieses Knotens eingebogen werden kann. Durch die Unterscheidung zwischen gerichteten und ungerichteten Kanten entspricht also der Ausgangsgrad eines Knotens im Kantengraphen gerade dem Ausgangsgrad des Originalknotens, in den die zugehörige Originalkante hineinführt. Es gilt $deg(\Phi(k)) = deg(y)$. Der Umstand, dass wir statt mit dem normalen Kantengraphen mit dem erweiterten Kantengraphen arbeiten, und daher für jede Originalkante immer zwei Knoten entstehen, wirkt sich auf den maximalen Knotengrad nicht aus. Es entstehen immer ein Knoten mit Ausgangsgrad 1 und einer, dessen Grad der eben genannten Gleichung entspricht. □

Beweis zu Theorem 3.5.6. Bei der Erzeugung des erweiterten Kantengraphen entstehen duch die Transformation für jede Originalkante zwei Knoten. Die Knotenzahl η des erweiterten Kantengraphen entspricht somit $2 \cdot m$ des Originalgraphen.

Zusammen mit Lemma 3.5.7 kann die Laufzeit des Algorithmus' in Bezug auf den Originalgraphen somit ausgedrückt werden als $\mathcal{O}((2 \cdot m)^3 \cdot maxEinfachheit^2 \cdot maxDegree_G)$. □

Weitere Überlegungen zur Laufzeit

Die Laufzeiten des Algorithmus', sowohl im Bezug auf den Originalgraphen, als auch im Bezug auf den erweiterten Kantengraphen, sind polynomiell in der Anzahl der Kanten, beziehungsweise der (transformierten) zugehörigen Knoten. Ebenso sind sie polynomiell in der Größe $maxEinfachheit$. Dadurch ist die Laufzeit aber exponentiell in der Länge der binären Repräsentation des Wertes $maxEinfachheit$. Der vorgestellte Algorithmus besitzt aus diesem Grund eine pseudopolynomielle Laufzeit. Durch realistische Abschätzungen kann man jedoch sehen, dass dieser Umstand für die Praxis von geringer Relevanz ist.

Wir betrachten nun erneut den maximalen Ausgangsgrad. Straßengraphen sind im Allgemeinen recht dünn besetzte Graphen. Dieser Umstand ändert sich auch durch die Transformation zum erweiterten Kantengraphen nicht. Damit liegt die theoretische Laufzeit zwar über $\mathcal{O}(m^3)$, allerdings nicht in dem Maße, wie es bei allgemeinen Graphen theoretisch vorkommen könnte, wo der $maxDegree$ $(n-1)$ erreichen könnte. Der maximale Ausgangsgrad eines Straßengraphen ist dagegen im Allgemeinen recht klein (< 10). Da wir bei der Erweiterung des Kantengraphen einen Knoten des normalen Kantengraphen auf zwei Knoten im erweiterten Kantengraphen abbilden, zwischen denen dann aber nur eine Kante verläuft, gilt außerdem, dass die Hälfte aller Knoten im erweiterten Kantengraphen den Ausgangsgrad 1 besitzt. Die Anzahl der Relaxierungen verringert sich hierdurch enorm.

Eine ähnlich kleine Dimensionierung wird sich bei den meisten Anwendungen auch für den maximalen Wert der Einfachheit ergeben. Die Klassifizierung von Abbiegungen in mehr als 10 bis 20 verschiedene Werte ist recht unwarscheinlich, beziehungsweise wenig sinnvoll. Die in dieser Arbeit vorgestellten Kostenfunktionen nehmen bei realistischer Anwendung allesamt Einteilungen in weniger als 10 verschiedene Werte vor, wobei der maximale Wert der Einfachheit zwischen 4 und 11 liegt. Auch für das Quadrat dieser Werte sind zur Kodierung wenige Bits ausreichend.

Die Faktoren $maxEinfachheit^2$ und $maxDegree_G$ in der Laufzeit des Algorithmus' sind daher in realen Anwendungen klein gegen die Anzahl der Kanten im Graphen.[1]

[1] Der Straßengraph von Luxemburg von 2006 besitzt über 82.000, der von Deutschland über 10 Millionen Kanten.

4. Implementierung und Evaluation

Nachdem der Algorithmus zur Berechnung simpelster Routen mit Distanzgarantie nun vorgestellt und theoretisch untersucht wurde, soll nun durch Experimente die Anwendung in der Praxis untersucht werden.

Hierzu wurden die folgenden Algorithmen implementiert und getestet:

- Die Transformation eines Straßengraphen zu seinem erweiterten Kantengraphen
- Dijkstras Algorithmus zur Berechnung der kürzesten Route
- Dijstras Algorithmus mit Anpassung an den erweiterten Kantengraphen zur Berechnung der simpelsten Route (durch alleinige Betrachtung der Einfachheits-Komponente des Kostenvektors)
- Der beschriebene *simple-and-short* Algorithmus zur Berechnung der simpelsten Route mit Distanzgarantie

Die Implementierung erfolgte in der Prgrammiersprache C++.

Da für den Vergleich der besten gefundenen Routen mit den zugehörigen simpelsten Routen auch Länge und Einfachheit der simpelsten Routen relevant sind, werden diese mit Hilfe des modifizierten Dijkstras auf dem erweiterten Kantengraphen berechnet. Dies wird vor der Ausführung des *simple-and-short* Algorithmus' getan, so dass der simpelste Pfad zur Verfügung steht und somit zur Beschleunigung der Berechnungen das Ausschlusskriterium Nummer 2 (Gleichung 3.5) ebenfalls benutzt

werden kann.
Wie schon erwähnt berechnet der *simple-and-short* Algorithmus auch alle nicht dominierten Alternativen mit absteigender Distanz bis hin zum kürzesten Weg automatisch mit. Für den Vergleich interessiert uns hier jedoch nur der beste gefundene Weg, so dass wir auf die Alternativen verzichten, und damit Ausschlusskriterium Nummer 3 (Gleichung 3.5) ebenfalls anwenden können.
Da Dijkstras Algorithmus für den kürzesten Weg auf dem Originalgraphen ausgeführt wird, wird die Einfachheit des kürzesten Weges separat noch einmal ausgerechnet, indem die verwendete Kostenfunktion auf alle Abbiegungen entlang des Weges noch einmal angewendet wird. Dies ist nötig, da beim Verzicht auf Alternativen auch der kürzeste Weg nicht mehr als letzte Alternative mit berechnet wird.

4.1 Datengrundlage

Als Datengrundlage während der Implementierung wurde das Straßennetz von Luxemburg und das von Deutschland der PTV AG aus dem Jahre 2006 verwendet [1].
Für die Tests wurden Routen innerhalb großer Städte / Metropolregionen in Deutschland gewählt. Pro Region wurde eine *Bounding Box* mittels Längen- und Breitengraden für die linke, untere und die rechte, obere Ecke definiert. Innerhalb dieser Koordinatenintervalle wurden zufällig und gleichverteilt Start- und Zielknoten gewählt, zwischen denen Routen berechnet wurden. Die Regionen sowie die Anzahl der berechneten Routen sind in folgender Tabelle gegeben:

Gebiet	Anzahl Routen	Länge und Breite der Bounding Box
Berlin	20.000	12 km · 10 km
München	20.000	15 km · 12 km
Essen	5.000	40 km · 38 km
Erweitertes Ruhrgebiet	2.000	100 km · 115 km

Tabelle 4.1: Berechnete Routen und Größen der Bounding Boxen der Gebiete

Die Tests wurden berechnet auf einer Maschine bestehend aus 48 AMD Opteron Prozessoren mit einer Taktfrequenz von jeweils 2,1 GHz. Das Betriebssystem ist Linux in der Version 2.6.34.8-0.2. Die Maschine verfügt insgesamt über 256 GB Arbeitsspeicher. Die Implementierung wurde kompiliert mit dem Compiler GCC Version 4.5.0 mit Optimierungsstufe 3.

[1] PTV AG, www.ptv.de

4.2 Experimente und Analyse

Die 47.000 berechneten Start-Ziel Paare der verschiedenen Tests teilen sich in vier Gebiete ein. Um Aussagen über Routen unterschiedlicher Längen treffen zu können, wurden die Gebiete so gewählt, dass sie im Durchschnitt verschieden lange Routen erzeugen. Die Länge einer Route wurde in benötigter Zeit zum Zurücklegen der Strecke gemessen. Die durchschnittlichen Längen der kürzesten Routen sowie ihre Extrema sind wie folgt:

Gebiet	Durschnittl. Länge	Minimum	Maximum
Berlin	8 Min. 21 Sek.	0,86 Sek.	20 Min. 16 Sek.
München	9 Min. 39 Sek.	0,95 sek.	25 Min. 19 Sek.
Essen	21 Min. 40 Sek.	5,8 Sek.	49 Min. 49 Sek.
Ruhrgebiet	44 Min. 27 Sek.	3 Min. 8 Sek.	1 Std. 39 Min. 41 Sek.

Tabelle 4.2: Längen der kürzesten Wege

Da es sich hierbei um die Längen der kürzesten Wege handelt, sind diese unabhängig von der verwendeten Kostenfunktion und auch unabhängig vom verwendeten ε.

Bei den Tests wird für alle 47.000 Routen jeweils ein Tripel aus drei Wegen erzeugt. Dabei wird wie folgt vorgegangen:
Zuerst wird Dijkstras Algorithmus auf dem Originalgraphen ausgeführt, um die Länge des kürzesten Weges zu berechnen. Die Einfachheit dieses Weges wird dann seperat durch Anwendung der Kostenfunkion ermittelt. Im Anschluss daran wird der modifizierte Dijkstra auf dem erweiterten Kantengraphen ausgeführt, und so der simpelste Weg berechnet. Dessen Länge wird dann ebenfalls seperat errechnet durch Rücktransformation des Weges in den Originalgraphen. Nun wird der *simple-and-short* Algorithmus ausgeführt. Dabei kommen alle erwähnten Ausschlusskriterien aus Abschnitt 3.5.3 zum Einsatz. Als Ergebnis erhält man jeweils Distanz und Einfachheit des kürzesten, simpelsten und besten Weges. Diese werden gespeichert und alle Änderungen und Informationen an berührten Knoten beider Graphen wieder gelöscht, so dass das nächste Routentripel berechnet werden kann.

Alle Routen wurden jeweils für die historische Kostenfunktion wie oben beschrieben berechnet, sowie für eine kombinierte Kostenfunktion. Letztere setzt die Kosten zusammen aus einer Konvexkombination mehrerer Kostenfunktionen.

Dabei geht ein:

- zu 10% der Knotengrad einer Abbiegung
- zu 30% wiederum die historische Klassifizierung von Abbiegungen
- zu 60% die Kosten der oben beschriebenen Winkeleinteilung (Abbildung 3.7), wobei die Wertigkeiten der Abbiegungen wie folgt erhöht sind, um eine vernünftige Größe im Vergleich zu den historischen Kosten zu erhalten.

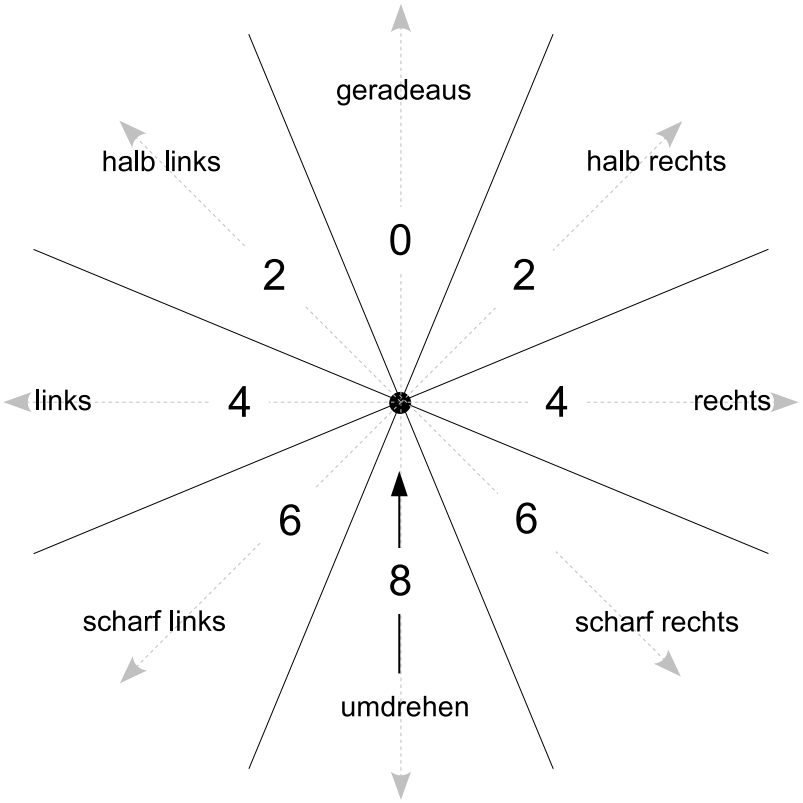


Abbildung 4.1: Für die Tests verwendete Winkeleinteilung mit zugehörigen Wertigkeiten

Untersuchung der einfachsten Routen

Um sich nun einen Überblick über die erhaltenen Routen zu verschaffen, soll zunächst der Unterschied zwischen kürzester und einfachster Route angeschaut werden. Wir betrachten die durchschnittliche Verlängerung des einfachsten Weges gemessen am kürzesten durch die relative prozentuale Abweichung (RPD [2]) nach der Formel:

$$RPD_{s,k} = \frac{d_s - d_k}{d_k} \tag{4.1}$$

Die relativen prozentualen Abweichungen der einfachsten Routen sind unabhängig vom gewählten ε, hängen aber von der gewählten Kostenfunktion ab. Für die kombinierte Kostenfunktion ergeben sich die Abweichungen in Distanz und Einfachheit wie folgt:

	Durchschnittliche Längen		
Gebiet	kürzeste	einfachste	RPD
Berlin	8 Min. 21 Sek.	9 Min. 29 Sek.	13,95 %
München	9 Min. 39 Sek.	10 Min. 46 Sek.	11,69 %
Essen	21 Min. 40 Sek.	23 Min. 12 Sek.	7,38 %
Ruhrgebiet	44 Min. 27 Sek.	47 Min. 6 Sek.	6,06 %

	Durchschnittliche Einfachheit		
Gebiet	kürzeste	einfachste	RPD
Berlin	68,2	59,2	-12,26 %
München	73,0	64,0	-10,97 %
Essen	136,4	123,9	-8,46 %
Ruhrgebiet	231,5	212,3	-7,75 %

Tabelle 4.3: Vergleich von kürzesten und einfachsten Routen für kombinierte Kosten

Dabei bezeichnet die Spalte RPD den Mittelwert der Abweichungen über alle Routen, und nicht die Abweichung der Mittelwerte.

Innerhalb von Berlin ist also eine einfachste Route durchschnittlich 13,95 % länger als die dazugehörige kürzeste Route. Gleichzeitig ist die einfachste Route aber auch 12,26 % einfacher als die kürzeste.

[2] engl: **r**elative **p**ercentage **d**ifference

Bei der historischen Kostenfunktion ergibt sich ein ähnliches Bild:

	Durchschnittliche Längen		
Gebiet	kürzeste	einfachste	RPD
Berlin	8 Min. 21 Sek.	9 Min. 46 Sek.	17,30 %
München	9 Min. 39 Sek.	10 Min. 51 Sek.	12,91 %
Essen	21 Min. 40 Sek.	23 Min. 39 Sek.	9,73 %
Ruhrgebiet	44 Min. 27 Sek.	47 Min. 41 Sek.	7,63 %

	Durchschnittliche Einfachheit		
Gebiet	kürzeste	einfachste	RPD
Berlin	119,1	99,5	-15,14 %
München	126,5	109,6	-12,17 %
Essen	240,3	213,9	-10,41 %
Ruhrgebiet	413,8	375,3	-8,97 %

Tabelle 4.4: Vergleich von kürzesten und simpelsten Routen für historische Kosten

Die Ergebnisse der historischen Kostenfunktion bestätigen die Ergebnisse früherer Autoren, die diese ebenfalls verwendeten. In der Arbeit von Duckham und Kulik [DK03] war ein einfachster Pfad im Durchschnitt $15,8\,\%$ länger als der kürzeste. Betrachtet man alle 47.000 Routen dieser Arbeit, so egibt sich eine mittlere Abweichung von circa $14{,}22\,\%$.

Unsere Testergebnisse sollen nun exemplarisch für die kombinierte Kostenfunktion grafisch veranschaulicht werden. Die Distanzen der kürzesten Routen sind auf der X-Achse abgetragen. Die zugehörigen Distanzen der simpelsten Routen auf der Y-Achse. Die untere, schwarze, durchgezogene Linie zeigt die Gerade mit Steigung 1.

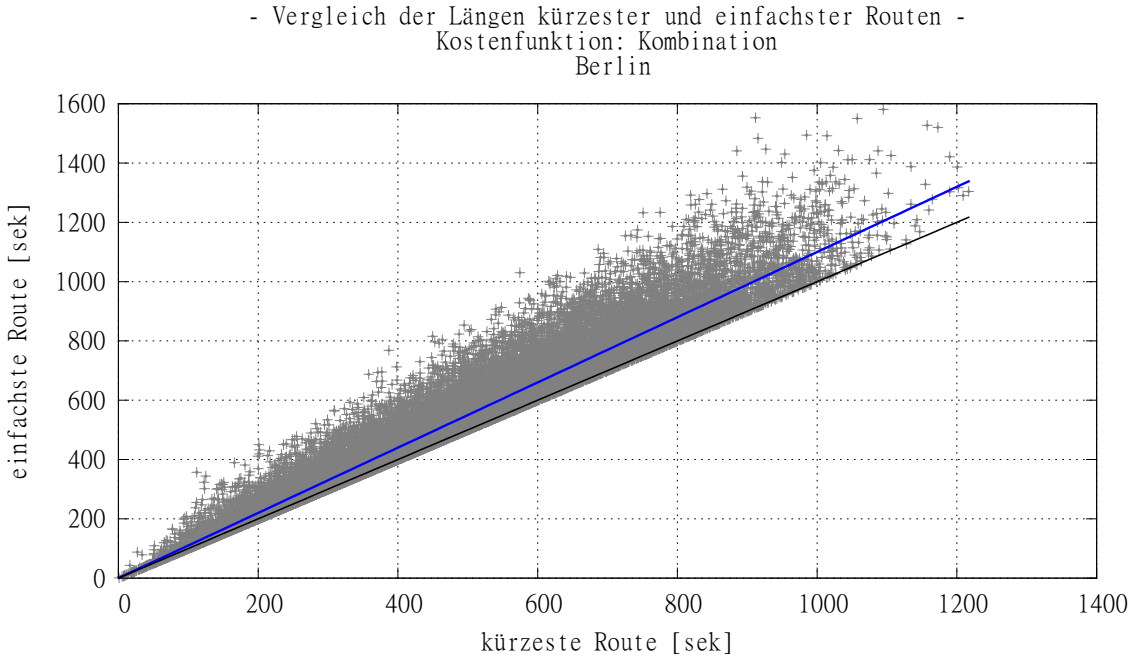


Abbildung 4.2: Längen der kürzesten und zugehörigen einfachsten Routen in Berlin für kombinierte Kosten

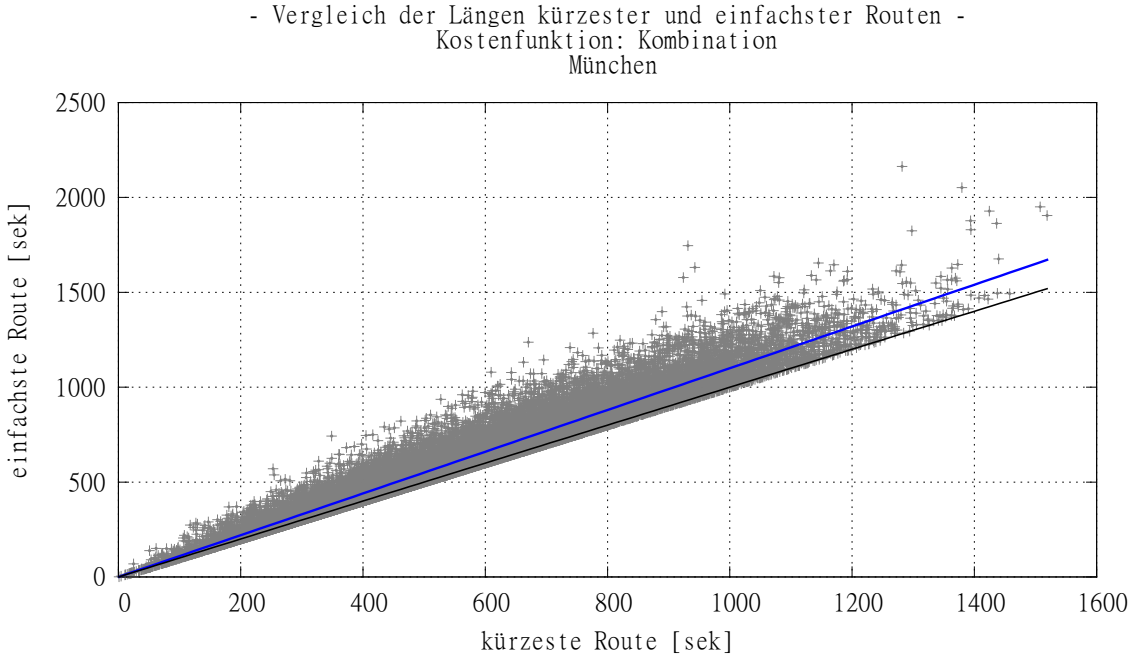


Abbildung 4.3: Längen der kürzesten und zugehörigen einfachsten Routen in München für kombinierte Kosten

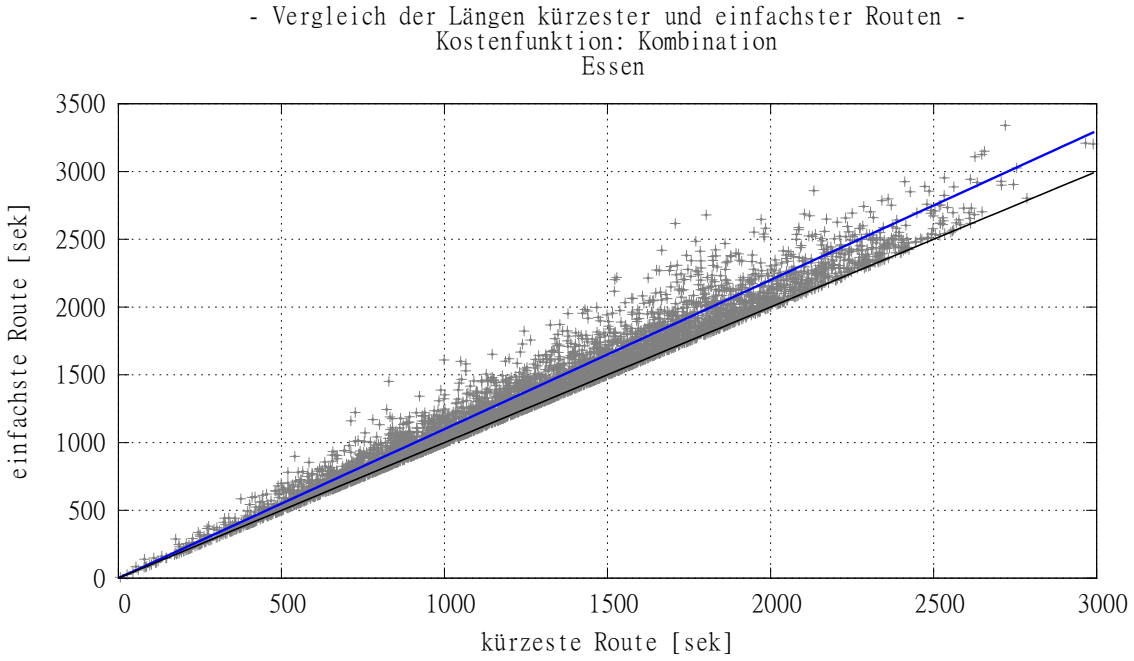


Abbildung 4.4: Längen der kürzesten und zugehörigen einfachsten Routen in Essen für kombinierte Kosten

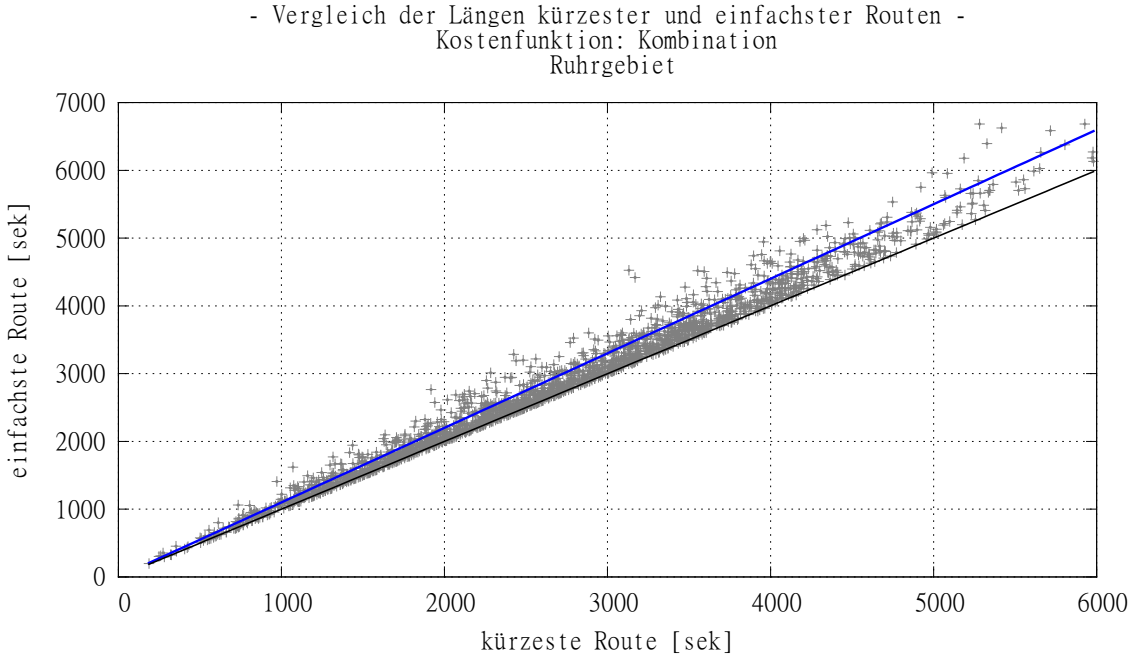


Abbildung 4.5: Längen der kürzesten und zugehörigen einfachsten Routen im Ruhrgebiet für kombinierte Kosten

Logischerweise kann ein simpelster Pfad niemals kürzer sein als der kürzeste, weshalb sich alle Messpunkte oberhalb dieser Linie befinden müssen. Die obere blaue Linie besitzt die Steigung 1,1. Das bedeutet, dass für alle Punkte oberhalb dieser Linie gilt, dass die einfachste Route mehr als 10 % länger ist als die zugehörige kürzeste. Wie viele Routen tatsächlich eine Verlängerung über einem bestimmten Prozentsatz aufweisen, sieht man in folgender Aufstellung:

Gebiet	Anteil der einfachsten Routen mit Verlängerung					
	≥ 5 %	≥ 10 %	≥ 20 %	≥ 30 %	≥ 50 %	≥ 75 %
Berlin	65,6 %	49,8 %	26,3 %	13 %	2,8 %	0,5 %
München	59,5 %	42,9 %	20,9 %	9,2 %	1,7 %	0,3 %
Essen	47,4 %	27,6 %	8,4 %	2,5 %	0,3 %	0,02 %
Ruhrgebiet	43,9 %	19,7 %	4,2 %	0,7 %	0,05 %	0 %

Tabelle 4.5: Verlängerungen der einfachsten Routen für kombinierte Kosten

Wie man sowohl an den Grafiken als auch in der Tabelle sieht, ist der Anteil der Routen, die beispielweise über 10 % liegen, relativ hoch. In Berlin und München ist der Anteil außerdem deutlich höher als in den anderen Gebieten. Im Ruhrgebiet beträgt dieser Anteil 30 % weniger als in Berlin. Dieser Umstand wird später unter „Betrachtung unterschiedlicher Distanzen“ noch einmal näher untersucht.

Untersuchung der besten Routen

Als nächstes sehen wir uns die Ergebnisse der besten Routen an. Alle diese Routen wurden mit ε-Werten von 0,1 und 0,05 berechnet. Die Längen und Einfachheiten der besten Routen, ebenfalls wieder im Vergleich zu den kürzesten, sehen für die kombinierte Kostenfunktion aus wie in den Tabellen 4.6 und 4.7 dargestellt.

Wie an den Daten für $\varepsilon = 0{,}1$ zu sehen ist, beträgt die durchschnittliche Verlängerung der simpelsten Route um die 4 %, obwohl die Distanzgrenze 10% beträgt. Dies ist darauf zurückzuführen, dass all die Routen, die vorher über 10 % länger waren als die kürzeste Route, nun auf 10 % „gestutzt“ werden. Die vorher aber schon unter 10 % längeren Routen behalten aber natürlich ihren Wert. Der Durchschnitt liegt damit dann logischerweise niedriger. Ebenso verhält es sich für $\varepsilon = 0{,}05$.

Durchschnittliche Längen

Gebiet	kürzeste	beste	RPD
Berlin	8 Min. 21 Sek.	8 Min. 44 Sek.	4,10 %
München	9 Min. 39 Sek.	10 Min. 4 Sek.	3,86 %
Essen	21 Min. 40 Sek.	22 Min. 30 Sek.	3,69 %
Ruhrgebiet	44 Min. 27 Sek.	46 Min. 19 Sek.	4,04 %

Durchschnittliche Einfachheit

Gebiet	kürzeste	beste	RPD
Berlin	68,2	61,2	-9,02 %
München	73,0	66,0	-8,30 %
Essen	136,4	125,5	-7,25 %
Ruhrgebiet	231,5	213,6	-7,21 %

Tabelle 4.6: Vergleich von kürzesten und besten Routen für kombinierte Kosten mit $\varepsilon = 0{,}1$

Durchschnittliche Längen

Gebiet	kürzeste	beste	RPD
Berlin	8 Min. 21 Sek.	8 Min. 32 Sek.	1,79 %
München	9 Min. 39 Sek.	9 Min. 51 Sek.	1,71 %
Essen	21 Min. 40 Sek.	22 Min. 8 Sek.	1,97 %
Ruhrgebiet	44 Min. 27 Sek.	45 Min. 35 Sek.	2,38 %

Durchschnittliche Einfachheit

Gebiet	kürzeste	beste	RPD
Berlin	68,2	62,8	-6,79 %
München	73,0	67,8	-6,15 %
Essen	136,4	127,6	-5,84 %
Ruhrgebiet	231,5	216,3	-6,08 %

Tabelle 4.7: Vergleich von kürzesten und besten Routen für kombinierte Kosten mit $\varepsilon = 0{,}05$

Durch das Einhalten der Distanzgrenze werden die errechneten besten Routen zwar wieder kürzer im Vergleich zur simpelsten, aber auch wieder komplizierter (siehe Gleichung 3.3). Hervorzuheben ist hier, dass zum Beispiel in Berlin die durchschnittliche Abweichung der Distanz von circa 14 % bei der simplesten Route um 10 % auf circa 4 % sinkt, die damit verbundene notwendige Steigerung der Komplexität aber nur um 3 % steigt, von circa −12 % bei der simpelsten auf circa −9 % bei der besten Route. Hier ist es also möglich, relativ viel Distanzersparnis zu erzielen, bei vergleichsweise wenig Verlust von Einfachheit.

Sieht man sich nun statt der einfachsten die besten Routen im Vergleich zu den kürzesten an, ergibt die grafische Betrachtung (aller Routen) folgendes:

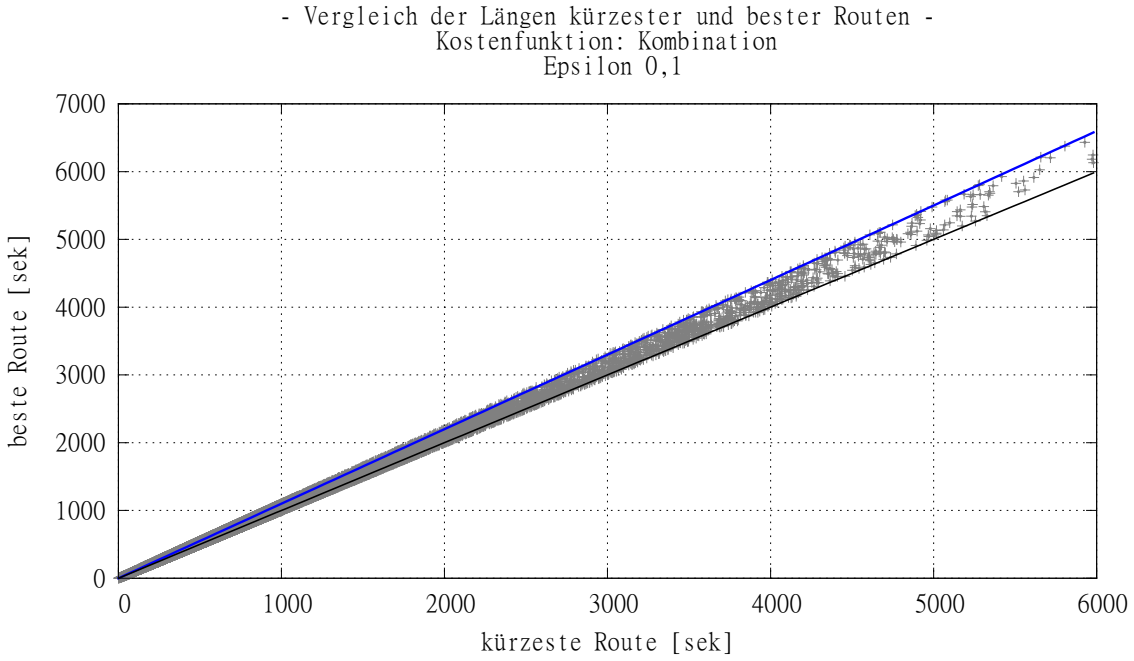


Abbildung 4.6: Längen der kürzesten und zugehörigen besten Routen (alle vier Gebiete) für kombinierte Kosten und eine Distanzgrenze von 10 %

Die blaue (obere) durchgezogene Linie besitzt hier wieder eine Steigung von 1,1, was der Distanzgrenze entspricht. Wie man sieht, sind alle einfachsten Wege, die vorher zu lang waren, nun auf die Distanzgrenze „gestutzt“, und halten diese ein.

Anhand der Routen in Berlin wollen wir uns nun noch die Veränderung der Einfachheit ansehen, wenn man statt der einfachsten die besten Routen benutzt. Die Einfachheit der einfachsten (besten) Route wird im Vergleich zur Einfachheit der kürzesten Route dargestellt:

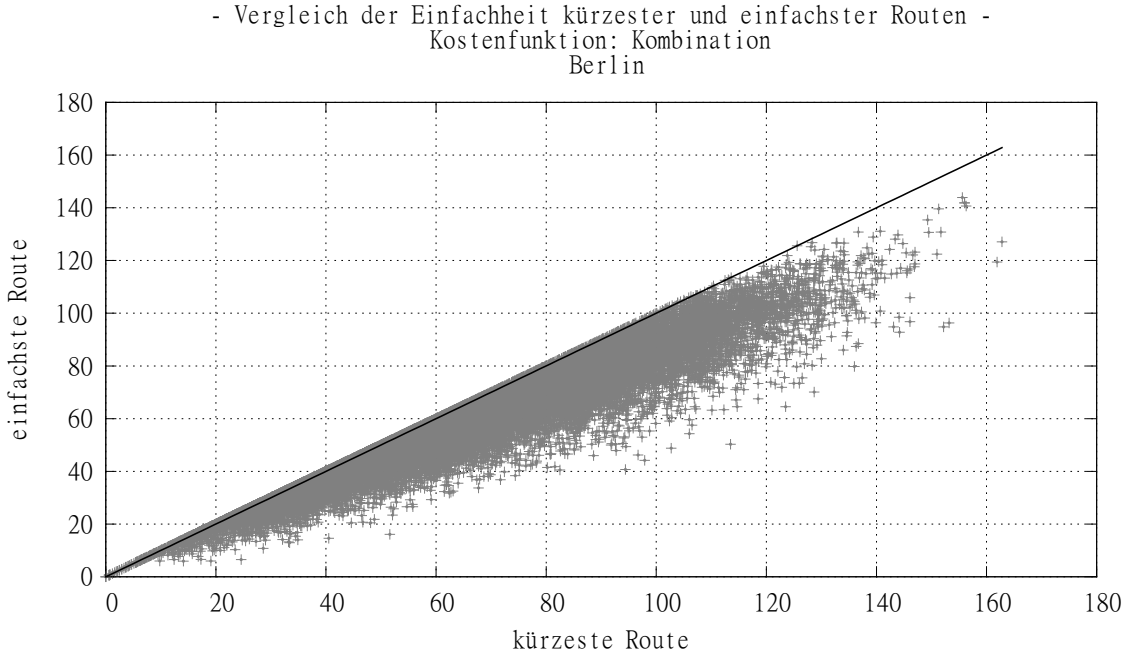


Abbildung 4.7: Einfachheit der kürzesten und zugehörigen einfachsten Routen in Berlin für kombinierte Kosten

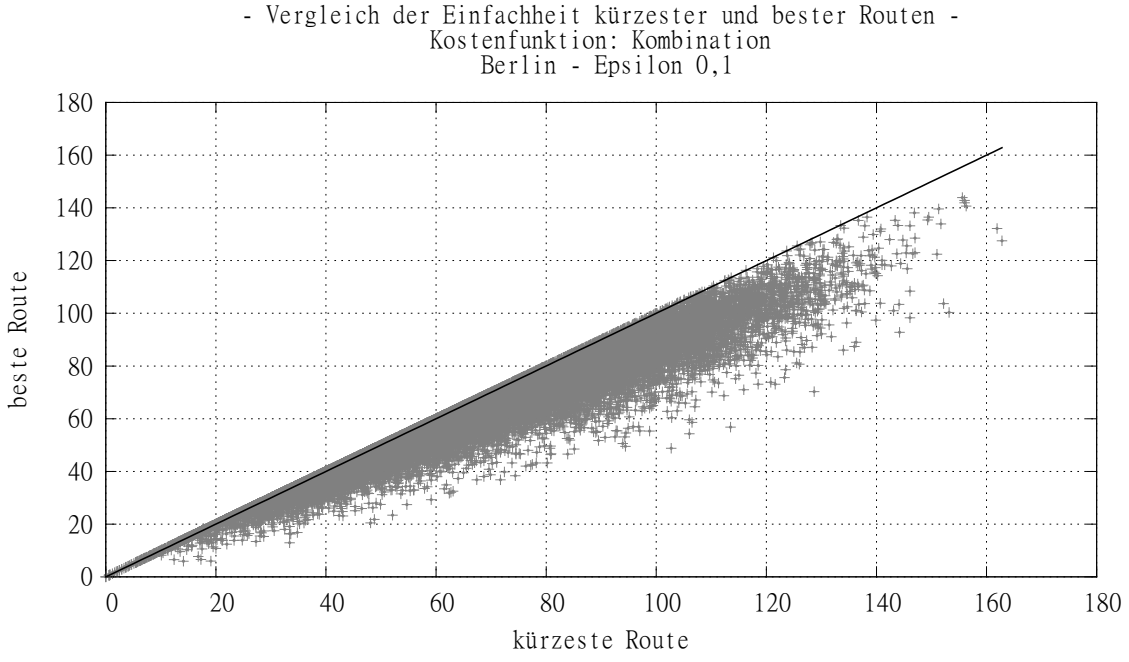


Abbildung 4.8: Längen der kürzesten und zugehörigen besten Routen in Berlin für kombinierte Kosten und eine Distanzgrenze von 10 %

Man sieht hierbei, dass sich trotz der Einhaltung der Distanzgarantie in der Masse kein deutlicher Rückgang der Einfachheit zu erkennen ist. Dies deckt sich mit den durchschnittlichen Abweichungen der Einfachheit in obigen Tabellen. Zwar sind beste Routen komplizierter als einfachste, der Unterschied ist jedoch wesentlich kleiner als in der Distanzkomponente. Im Vergleich zu kürzesten können beste Routen für wenig Mehrdistanz vergleichsweise viel Einfachheit erlangen.

Betrachtung unterschiedlicher Distanzen

Wie wir gesehen haben, ist der Anteil der einfachsten Routen, die im Vergleich zur kürzesten Route eine gewisse prozentuale Verlängerung übersteigen, in den einzelnen Gebieten sehr unterschiedlich. In Tabelle 4.5 wurde dies dargestellt. Diese Anteile werden mit den Gebieten Berlin, München, Essen und dem Ruhrgebiet in dieser Reihenfolge für alle Prozentzahlen stetig kleiner. Wie aus Tabelle 4.2 zu entnehmen ist, steigt die durschnittliche Länge einer Route in den Gebieten in der gleichen Reihenfolge. Dieser Umstand deutet darauf hin, dass das Optimierungspotential des *simple-and-short* Algorithmus' mit steigender Länge der Route abnimmt, da ein größerer Anteil der Routen eine gewünschte Distanzgrenze schon von vornherein mit der einfachsten Route einhält.

Diese Tatsache ist auf die Beschaffenheit des Straßennetzes zurückzuführen. Mit steigender Länge einer kürzesten Route zwischen zwei Punkten steigt in Deutschland die Wahrscheinlichkeit, dass innerhalb der Route eine Autobahn oder andere Schnellstraße benutzt werden kann. Nach den meisten Kostenfunktionen, und insbesondere den hier verwendeteten, werden Fahrten entlang von Autobahnen relativ niedrig bewertet. Sie haben meist keine größeren Knicke und führen geradeaus, weshalb die Winkelfunktion dafür niedrige Kosten vergibt. Der Knotengrad entspricht größtenteils zwei, da es vergleichsweise wenige Abbiegungen gibt. Auch die historische Kostenfunktion bewertet eine Autobahn größtenteils als simples Geradeausfahren. Da eine Autobahn außerdem die schnellste Straßenkategorie im deutschen Straßennetz ist, ist sie sowohl einfach als auch kurz. Sie wird also in beiden Dimensionen bevorzugt.
Wenn also auf der kürzesten Route ein Teil des Weges auf einer Autobahn verläuft, wird wahrscheinlich auch die simpelste Route entlang dieser Autobahn verlaufen, wodurch sich die wesentlichen Unterschiede nur noch in Start- und Zielbereich, das heißt innerhalb von Städten und Orten, ergeben. Da bei längeren Strecken dann ein Großteil der Strecke aus einer Autobahn besteht, nimmt der prozentuale Unterschied zwischen kürzestem und einfachstem Weg mit steigender Länge ab.
Um diese Hypothese zu verdeutlichen betrachten wir alle 47.000 Routen auf einmal

exemplarisch anhand der kombinierten Kostenfunktion und sortieren diese nach der Länge des kürzesten Weges. Diese Liste wird in 94 Teile zu je 500 Datensätzen geteilt. Berechnet werden jeweils die Mittelwerte sowie die durchschnittliche Abweichung der Distanz von einfachster zu kürzester Route. In folgender Grafik erkennt man, dass die mittlere prozentuale Verlängerung der einfachsten zur kürzesten Route mit steigender Entfernung zwischen Start- und Endpunkt der Route deutlich abnimmt.

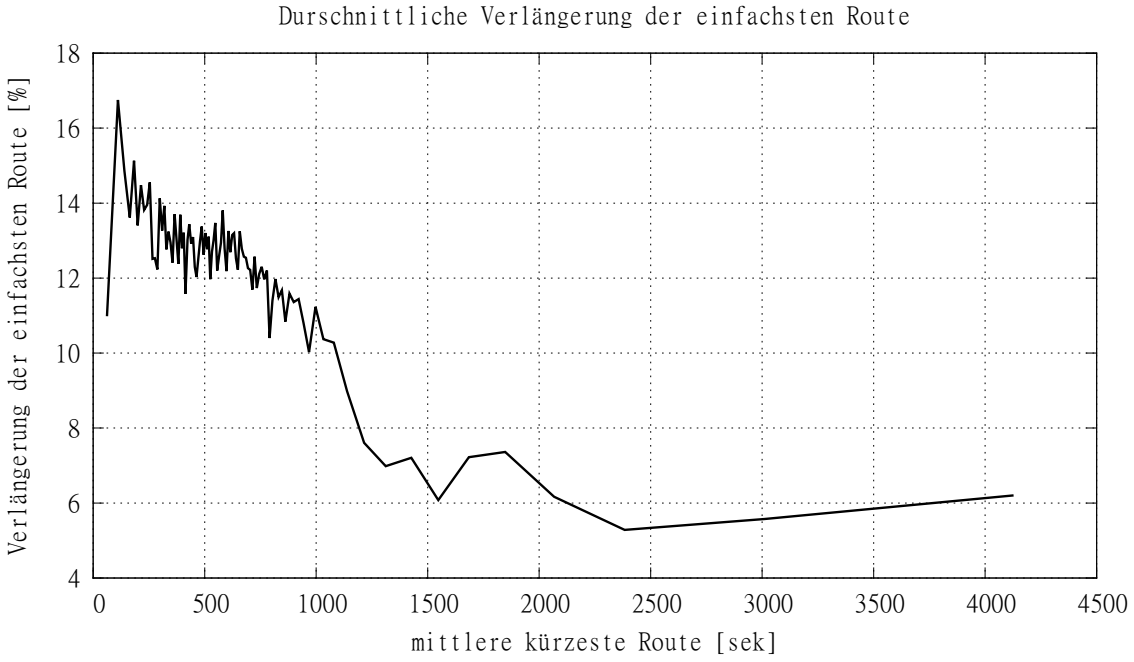


Abbildung 4.9: Mittlere prozentuale Abweichungen der einfachsten zu kürzesten Routen (Quantisierung: 500 Routen)

Untersuchung des Platzverbrauchs und der Laufzeit

Wie oben gezeigt, liegt die theoretische Anzahl Pfadelemente pro Knoten im erweiterten Kantengraphen in $\mathcal{O}(\eta \cdot maxEinfachheit)$ (Lemma 3.5.3). Es zeigt sich jedoch, dass durch das Dominanzkriterium und die Ausschlusskriterien die reale Anzahl Pfade, die pro Knoten gespeichert werden müssen, deutlich niedriger ist. Wie bereits erwähnt, tragen auf diese Weise Ausschlusskriterien wesentlich zur Laufzeit des Algorithmus' bei. Sie verhindern, dass Knoten, die sich am anderen Ende des Landes befinden, überhaupt betrachtet werden, wenn eine Route innerhalb einer Stadt berechnet werden soll. Mit zunehmender Länge des kürzesten Pfades wächst die Anzahl der Knoten, die in die Berechnung mit einbezogen werden.

Der hier verwendete originale Deutschlandgraph besteht aus 5.137.910 Knoten und 10.806.191 Kanten. Durch die vorgestellte Transformation entstehen so im erweiterten Kantengraphen 21.612.382 Knoten und 39.341.110 Kanten. Bei den Experimenten wurde die Anzahl betrachteter Knoten während der Laufzeit des Algorithmus', sowie die Anzahl der Pfade pro Knoten gespeichert. Für eine Distanzgrenze von $\varepsilon = 0{,}1$ ergibt sich die durchschnittliche Anzahl Pfadobjekte pro angefasstem Knoten wie folgt:

Gebiet	Durchschnittlich betrachtete Knoten	Durschnittliche Anzahl Pfade pro betrachtetem Knoten
Berlin	44.158	2,96
München	52.817	2,90
Essen	437.362	3,93
Ruhrgebiet	1.705.020	6,46

Tabelle 4.8: Durschnittliche Anzahl Pfade pro angefasstem knoten für die kombinierte Kostenfunktion bei $\varepsilon = 0{,}1$

Wie man sieht, steigt die betrachtete Anzahl Knoten mit steigender Entfernung sehr stark an. Als Beispiel ist eine beste Route in Essen durchschnittlich etwa 157 % länger als eine in Berlin, die zu betrachtende Knotenanzahl um diese Routen zu berechnen steigt jedoch um 890 %. Dieses Verhalten entspricht allerdings auch dem Verhalten des normalen Dijkstra-Algorithmus', bei dem die betrachtete Anzahl Knoten in Essen um 876% im Vergleich zu Berlin steigt.
Im Gegensatz dazu steigt jedoch die Anzahl der Pfadobjekte pro Knoten relativ langsam an (hier nur um 32 %). Diese Zahlen sind von der theoretischen Zahl des Platzverbrauchs für die Pfade um viele Größenordnungen entfernt.

Da die Größen der Pfadlisten auch in der Laufzeit einen wesentlichen Faktor darstellen, verhält es sich dort ebenso. Die durchschnittlichen Laufzeiten von Dijkstras Algorithmus sowie des *simple-and-short* Algorithmus' sind in Tabelle 4.9 aufgelistet. Die Zeit zum Extrahieren der Routen per Rückwärtsdurchlaufen nach dem Ausführen der Algorithmen kann dabei vernachlässigt werden. Sie ist in allen Tests etwa gleich hoch und im Durchschnitt $< 0{,}001$ Sekunde.

Gebiet	Dijkstra	*simple-and-short*	RPD
Berlin	0,070 Sek.	0,440 Sek.	756 %
München	0,080 Sek.	0,477 Sek.	680 %
Essen	1,026 Sek.	3,023 Sek.	223 %
Ruhrgebiet	4,802 Sek.	14,125 Sek.	229 %

Tabelle 4.9: Durchschnittliche Laufzeiten für die kürzeste Route (Dijkstra) und die beste Route (*simple-and-short* Algorithmus) für kombinierte Kosten mit $\varepsilon = 0{,}1$

Die Zeitmessungen ergeben, dass der *simple-and-short* Algorithmus bei kurzen Routen im Vergleich zum Dijkstra recht viel Zeit benötigt. Die Laufzeit erhöht sich jedoch bei längeren Routen weitaus weniger, als dies bei Dijkstras Algorithmus der Fall ist. Diese sehr langsam steigende Laufzeit lässt sich mit der langsam steigenden Anzahl Pfadelemente pro Knoten erklären.

Verhalten bester Routen in der Visualisierung

Da einfache Routen, und damit auch die besten, für den Benutzer leichter zu befolgen sein sollten, wäre es ebenfalls wüschenswert, wenn die Visualisierungen von einfacheren Routen ebenfalls einfacher wären als die der kürzesten. Um dies zu untersuchen wurden die berechneten Routen mit einem Schematisierungstool [GNPR11] visualisiert. Interessante Faktoren sind hierbei die Anzahl der Kanten, die zu visualisieren sind. Da das Schematisierungstool eine interne Optimierung vornimmt, kann die Anzahl dieser Kanten sowohl vor als auch nach dieser Optimierung untersucht werden. Ein weiterer interessanter Faktor, der in der Visualisierung von Bedeutung ist, ist die dafür benötigte Zeit. Die mittleren Werte für die kombinierte Kostenfunktion sowie einen Epsilonwert von 0,05 sind in Tabelle 4.10 gegeben. Um eine Verfälschung der Zeitmessung zu vermeiden, wurden für die Mittelwerte der Zeit die oberen sowie die unteren 5% der Werte nicht berücksichtigt.

Wenn man sich die Tabellen anschaut, fällt auf, dass einfachere Routen (sowohl beste als auch einfachste) schneller zu visualisieren sind als kürzeste Routen. Weiterhin kann man erkennen, dass die durchschnittlichen Werte für die zu optimierenden und zu visualisierenden Kanten sich zueinander ungefähr so verhalten, wie Gleichung 3.3 angibt. Damit ist gemeint, dass die kürzeste Route am meisten zu bearbeitende Eingabekanten aufweist, während diese Anzahl bei der einfachsten Route am niedrigsten ist, und die beste Route im Durchschnitt Werte dazwischen annimmt. Daraus resultierend hat auch die zu visualisierende Route nach der Optimierung bei der einfachsten Route weniger Kanten als bei der kürzesten.

kürzeste Route			
Gebiet	Kanten vorher	Kanten nachher	Zeit (ms)
Berlin	94,4	30,2	601,43
München	99,9	33,2	688,2
Essen	163,9	27,1	681,7
Ruhrgebiet	307,2	30,2	728,7
einfachste Route			
Gebiet	Kanten vorher	Kanten nachher	Zeit (ms)
Berlin	90,0	31,7	617,9
München	92,9	26,1	617,7
Essen	158,8	24,8	632,6
Ruhrgebiet	298,9	28,6	701,5
beste Route			
Gebiet	Kanten vorher	Kanten nachher	Zeit (ms)
Berlin	91,1	29,5	577,0
München	96,0	29,5	661,2
Essen	160,1	25,3	604,3
Ruhrgebiet	300,2	29,2	704,6

Tabelle 4.10: Daten zur Visualisierung der Routen für kombinierte Kosten und $\varepsilon = 0{,}05$

Man sieht, dass im Vergleich zur kürzesten Route sowohl bei der einfachsten als auch bei der besten Route in den meisten Fällen weniger Zeit benötigt wird und weniger Eingabekanten zu verarbeiten sind. Die entstehenden visualisierten Routen besitzen ebenfalls weniger Kanten. Einfache Routen (mit Distanzgarantie), sind also für den Anwender schneller bildlich darzustellen und bieten das Potential, für den Menschen leichter erfassbare und übersichtliche Routen zu zeichnen.

4.3 Fallbeispiele

Schließlich wollen wir uns in diesem Teil der Arbeit anhand von Beispielen einige Routen ansehen. Mit Hilfe des schon erwähnten Schematisierungstools [GNPR11] sollen zunächst drei Fallbeispiele visualisiert und angesehen werden. Es handelt sich dabei um Routen innerhalb von München, die mit der kombinierten Kostenfunktion berechnet wurden.

Beispiel 1. Die folgenden drei Bilder zeigen zu einem Start- und Zielknotenpaar die kürzeste, die einfachste und die beste Route. Es handelt sich um Routen vom Norden in den Süden Münchens, die mit einem Epsilonwert von 0,1 berechnet wurden. Die Länge der kürzesten Route beträgt 15 Minuten und 17 Sekunden.

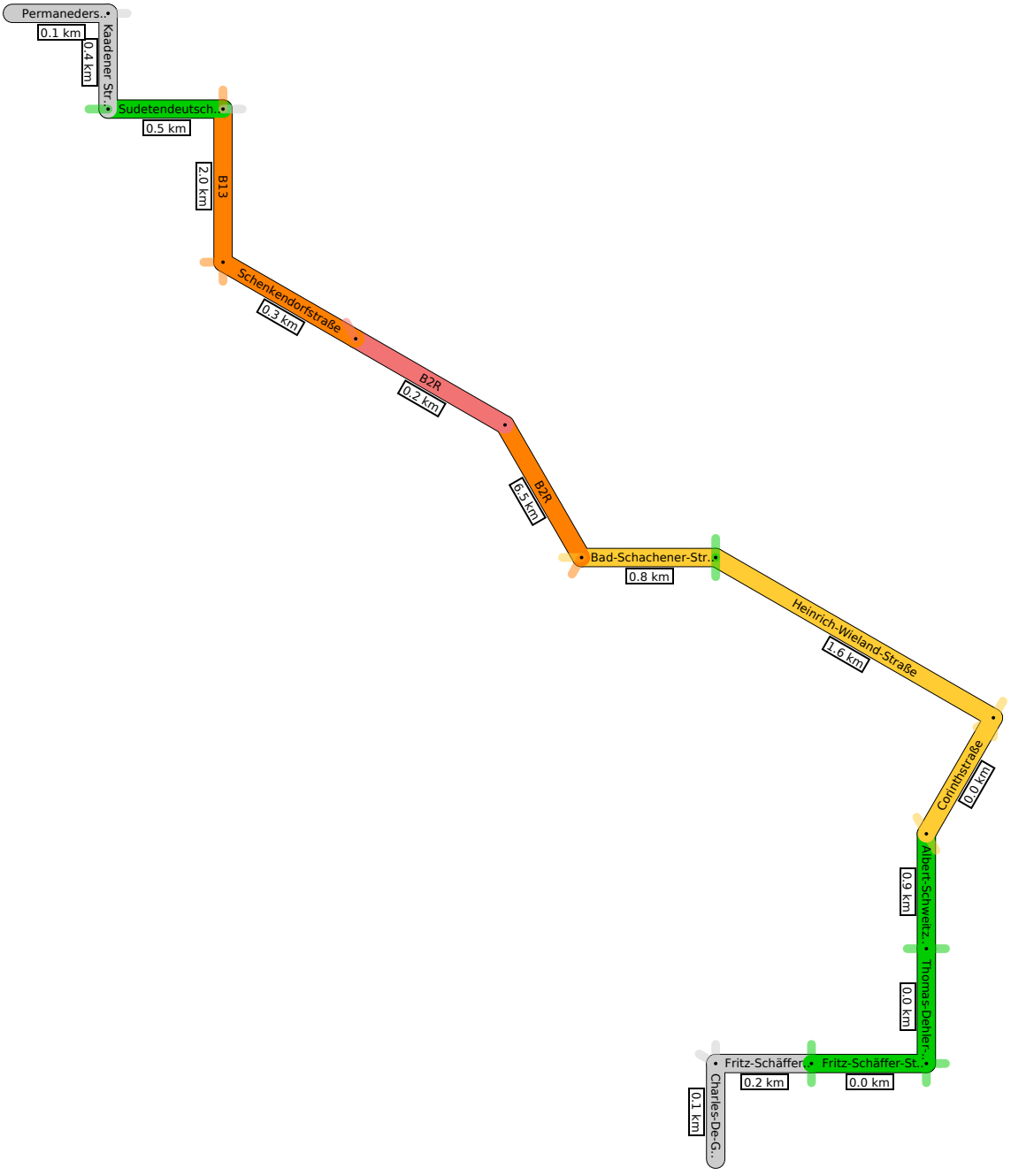


Abbildung 4.10: (Beispiel 1) Kürzeste Route. Länge 15 Minuten und 17 Sekunden. Einfachheit: 110,9. Berechnungsdauer: 0,03 Sekunden.

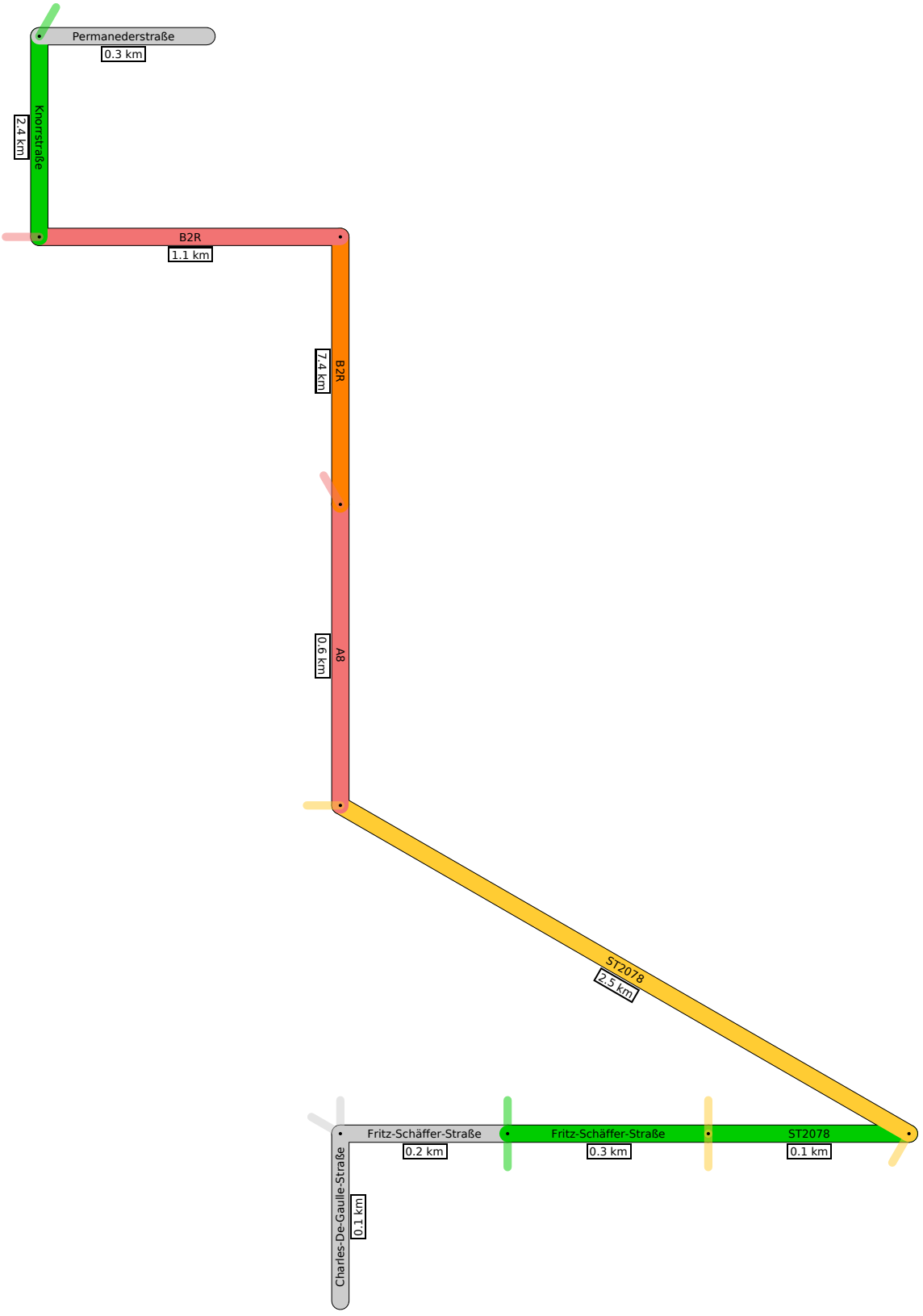


Abbildung 4.11: (Beispiel 1) Einfachste Route. Länge 17 Minuten und 8 Sekunden. Einfachheit: 103. Berechnungsdauer: 0,26 Sekunden.

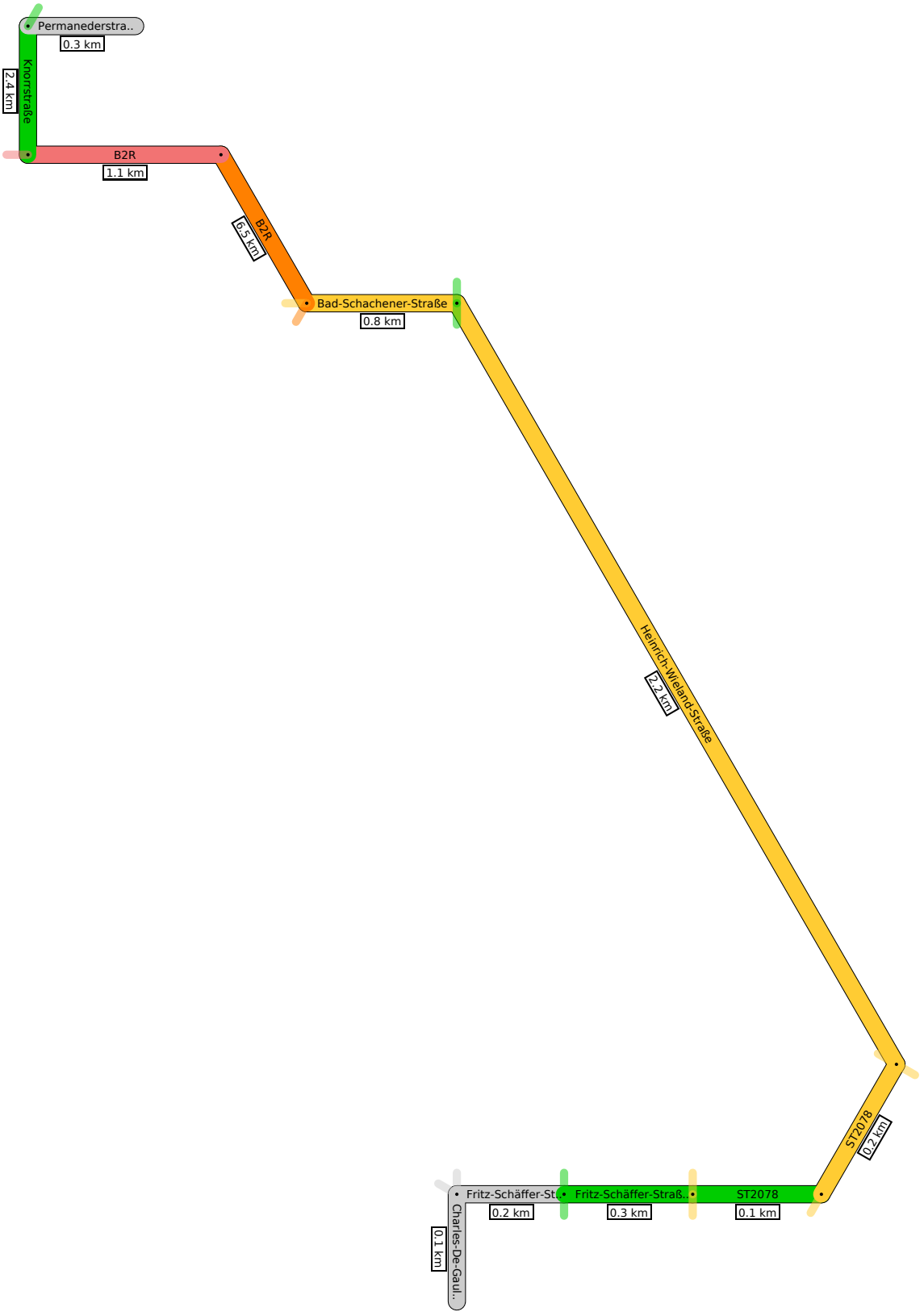


Abbildung 4.12: (Beispiel 1) Beste Route. Länge 16 Minuten und 42 Sekunden. Einfachheit: 103,7. Berechnungsdauer: 0,57 Sekunden.

Man kann an den drei Routen sehr gut erkennen, dass intuitiv die einfachste Route am leichtesten zu befolgen erscheint. Die beste Route scheint hier auch grafisch einen Kompromiss zu bilden. Hervorzuheben ist, dass die einfachste Route in diesem Fall 12,1 % länger ist als die kürzeste, die beste nur noch 9,2 %. Gleichzeitig ist die beste Route aber nur um 0,7 % komplizierter als die einfachste. Der Unterschied von 2,8 % Distanz kann mit nur einem minimalen Mehraufwand in der Einfachheit erreicht werden.

Beispiel 2. Auch in diesem Beispiel beträgt die Distanzgrenze 10 %. Die Länge der kürzesten Route beträgt 15 Minuten und 44 Sekunden.

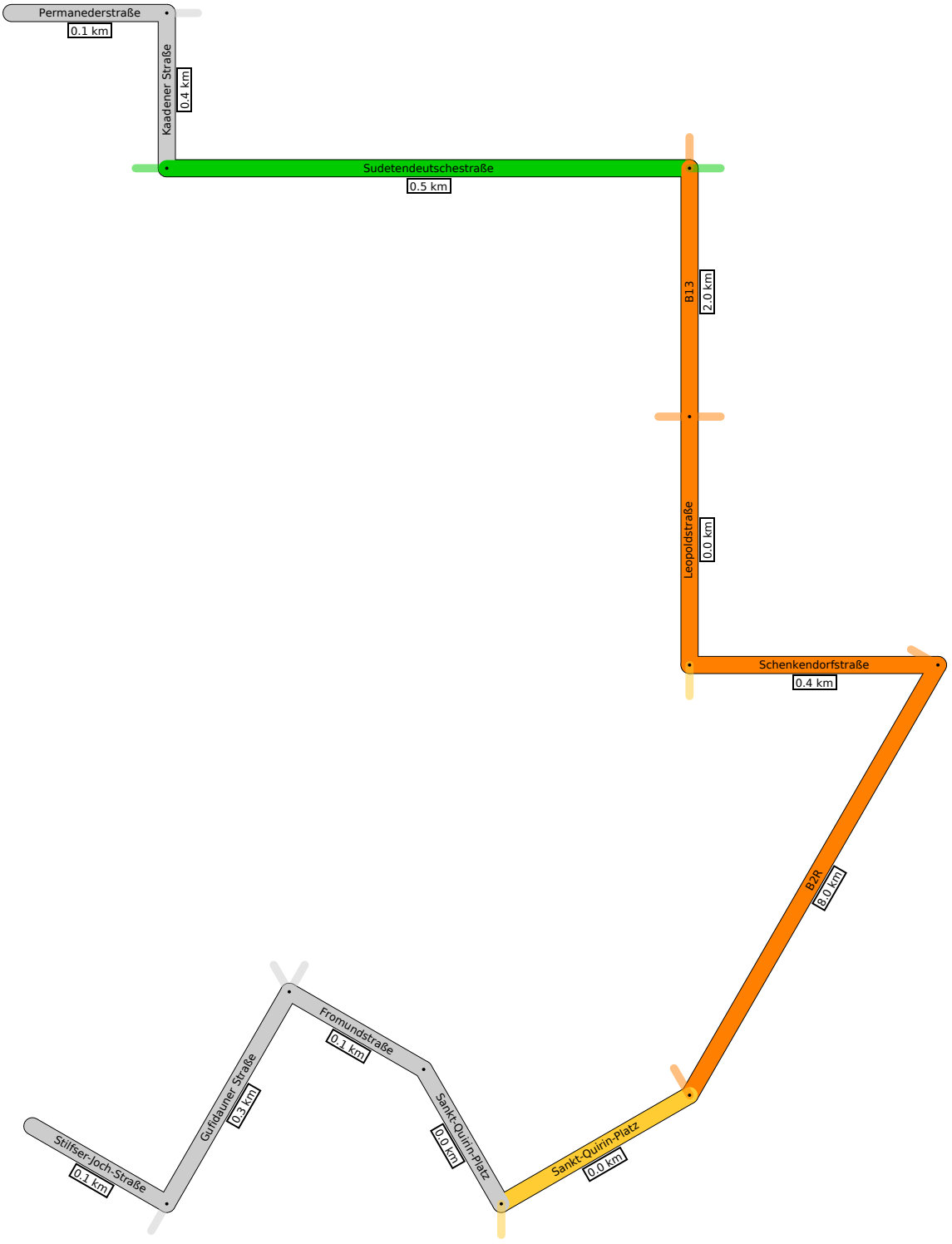


Abbildung 4.13: (Beispiel 2) Kürzeste Route. Länge 15 Minuten und 44 Sekunden. Einfachheit: 112,1. Berechnungsdauer: 0,07 Sekunden.

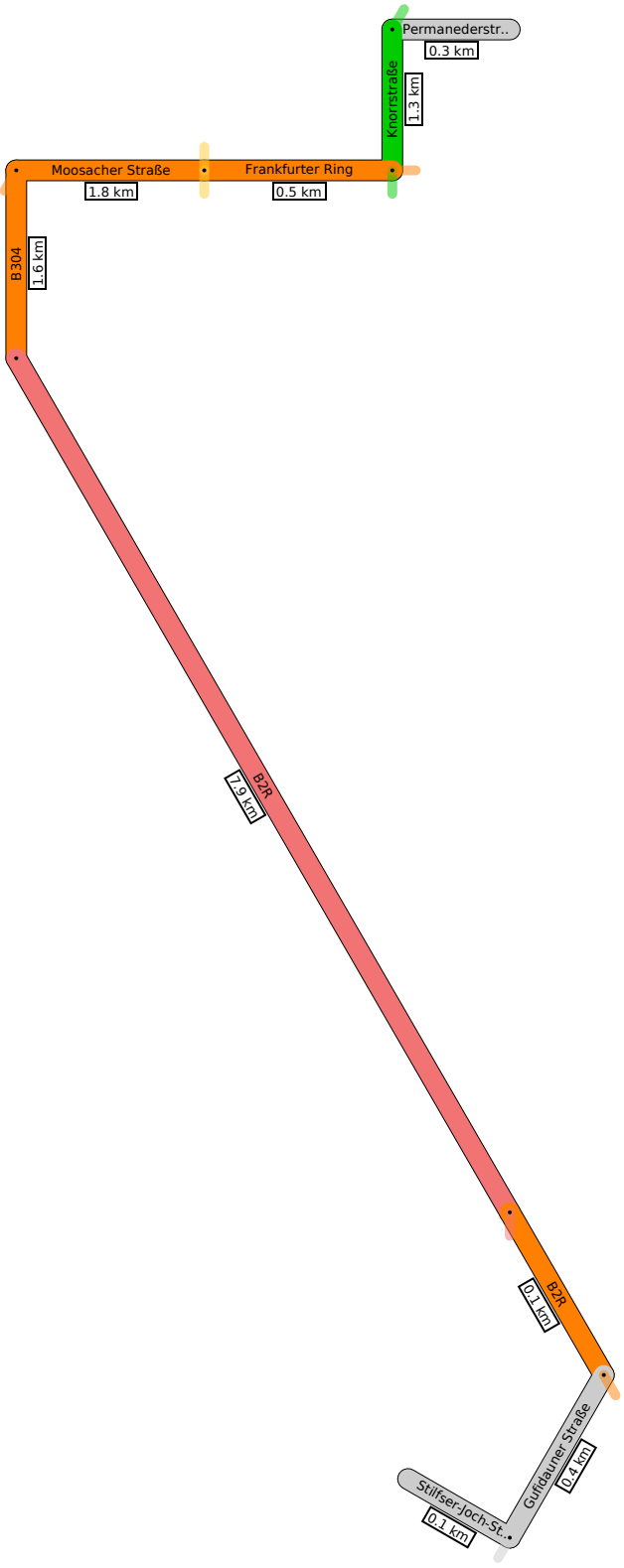


Abbildung 4.14: (Beispiel 2) Einfachste Route. Länge 18 Minuten und 58 Sekunden. Einfachheit: 97,9. Berechnungsdauer: 0,34 Sekunden.

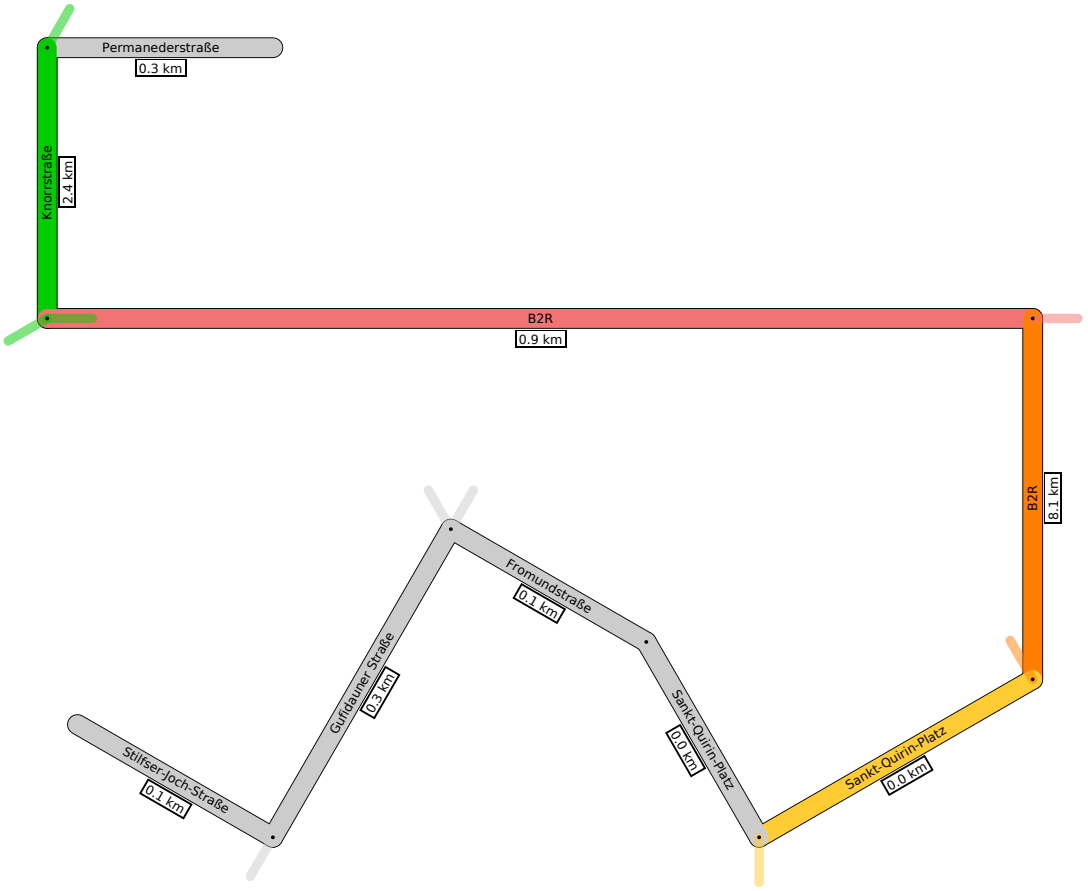


Abbildung 4.15: (Beispiel 2) Beste Route. Länge 16 Minuten und 46 Sekunden. Einfachheit: 108,9. Berechnungsdauer: 0,72 Sekunden.

Beispiel 3. Die folgenden drei Bilder zeigen wiederum zu einem Start- und Zielknotenpaar in München kürzeste, einfachste und beste Route. Der Epsilonwert für die beste Route ist 0,1. Die Länge der kürzesten Route beträgt 16 Minuten und 48 Sekunden.

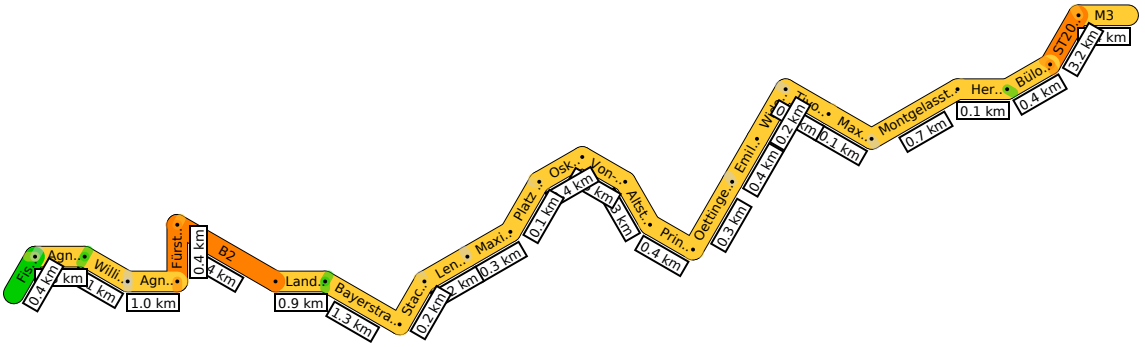


Abbildung 4.16: (Beispiel 3) Kürzeste Route. Länge 16 Minuten und 48 Sekunden. Einfachheit: 121,9. Berechnungsdauer: 0,12 Sekunden.

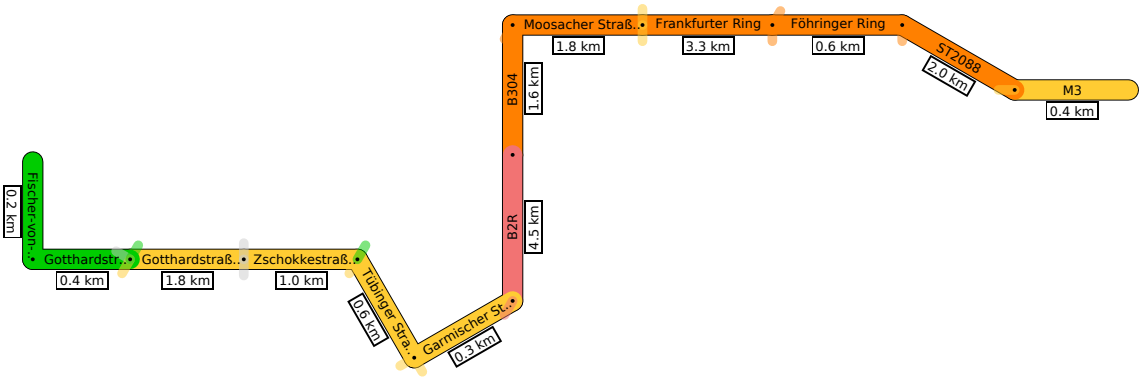


Abbildung 4.17: (Beispiel 3) Einfachste Route. Länge 18 Minuten und 16 Sekunden. Einfachheit: 96,1. Berechnungsdauer: 0,46 Sekunden.

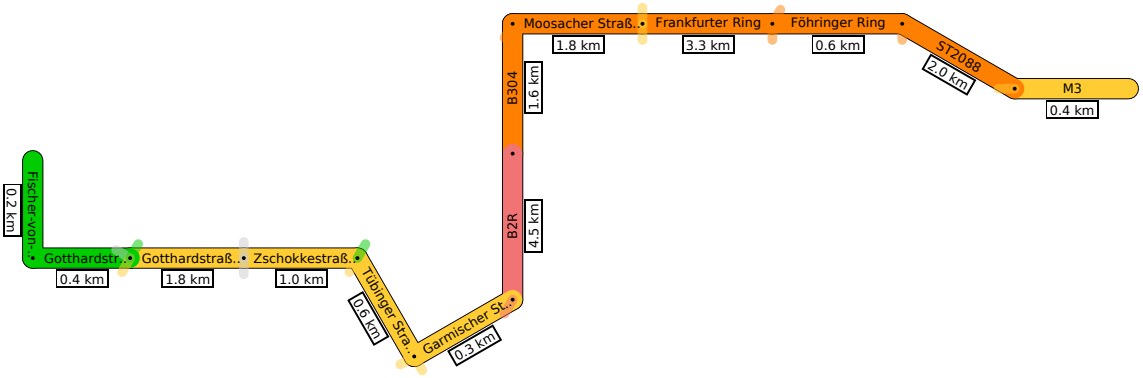
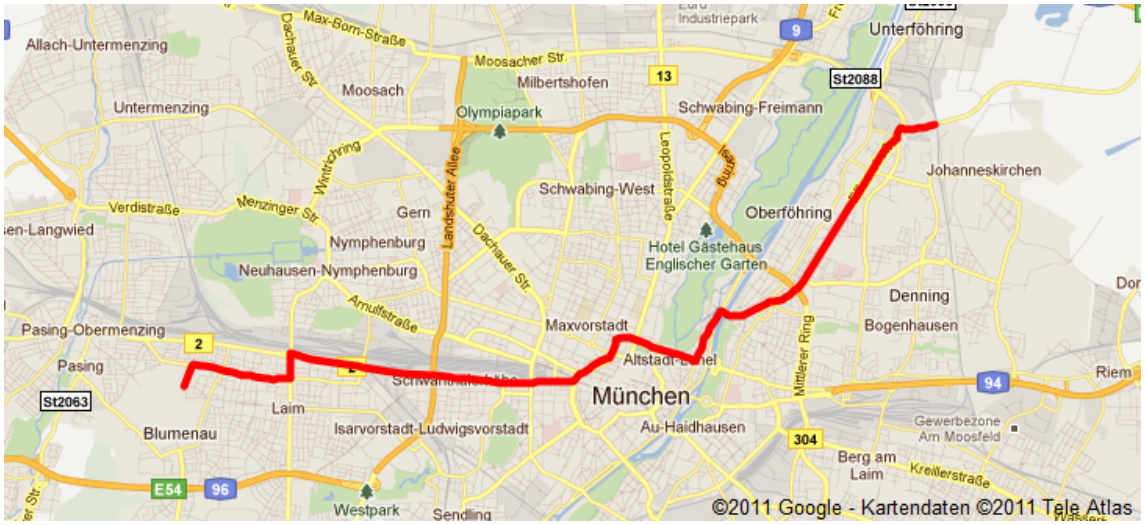
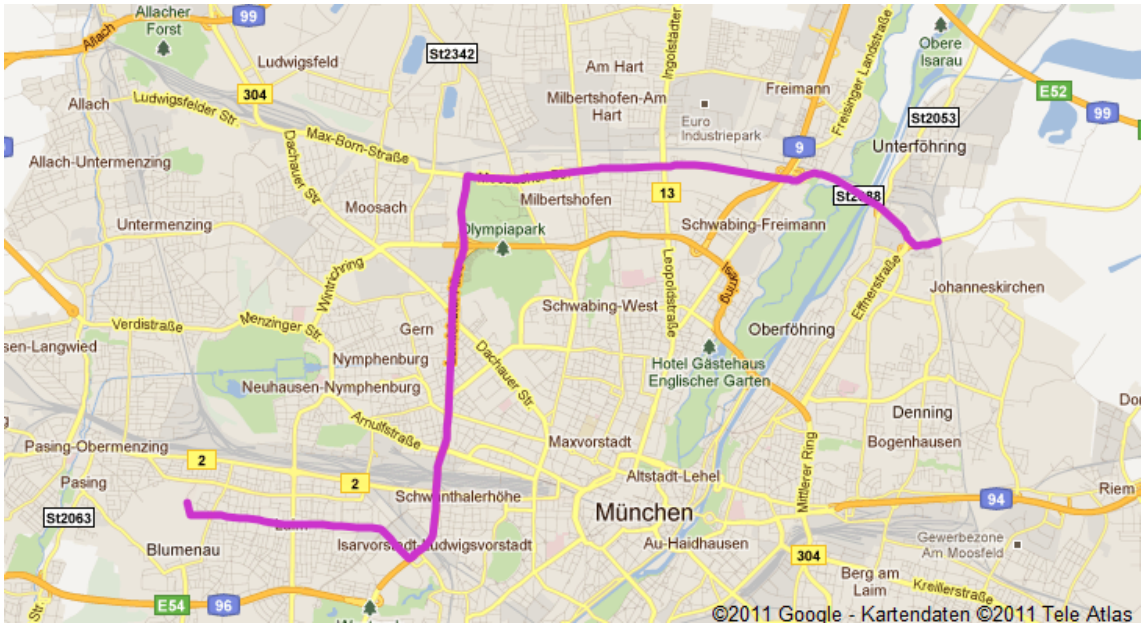


Abbildung 4.18: (Beispiel 2) Beste Route. Länge 18 Minuten und 16 Sekunden. Einfachheit: 96,1. Berechnungsdauer: 0,74 Sekunden.

Auch hier ist die intuitive Einfachheit anhand der Bilder eindeutig. In diesem Fall ist die einfachste Route außerdem nicht mehr als 10 % länger als die kürzeste, so dass die beste Route mit der einfachsten identisch ist. Dieses Beispiel wollen wir uns auch noch einmal anhand einer realen Straßenkarte ansehen:



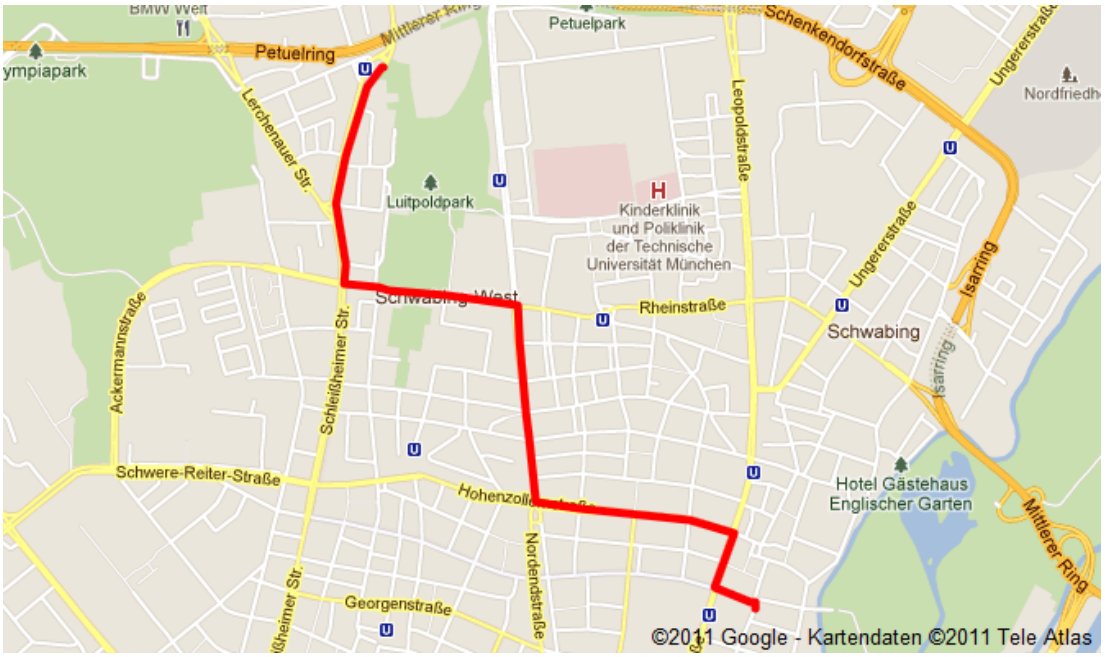
(a) Kürzeste Route



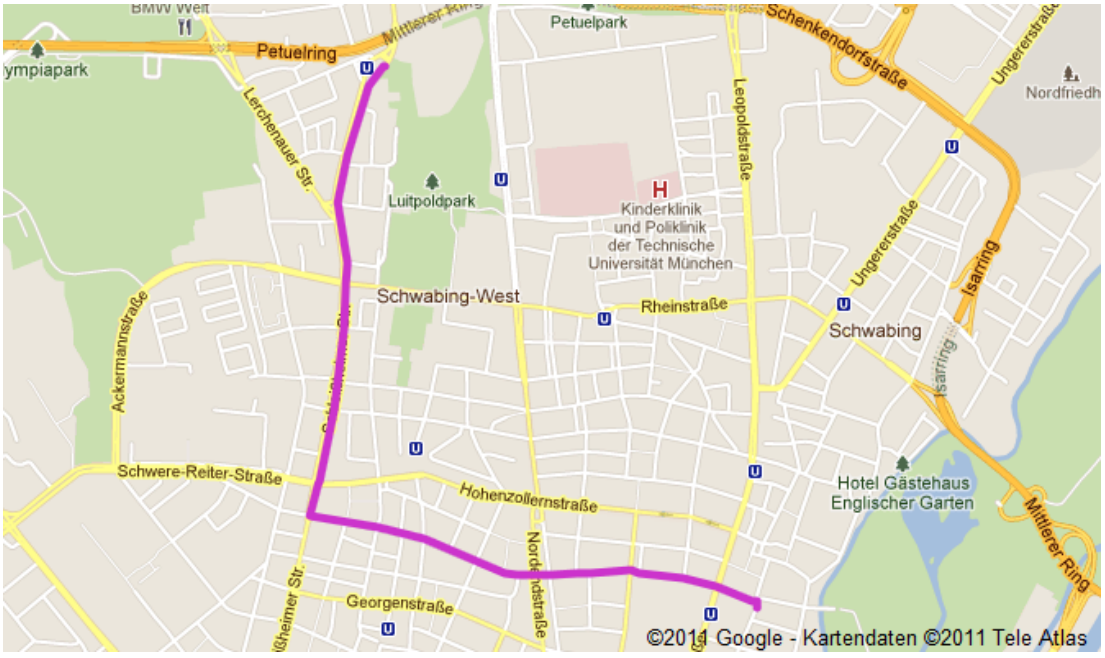
(b) Beste Route

Abbildung 4.19: (Beispiel 3) Darstellung in Google Maps.

Weitere Beispiele. Zum Schluss seien hier noch einige weitere Beispiele anhand von Straßenkarten gegeben.

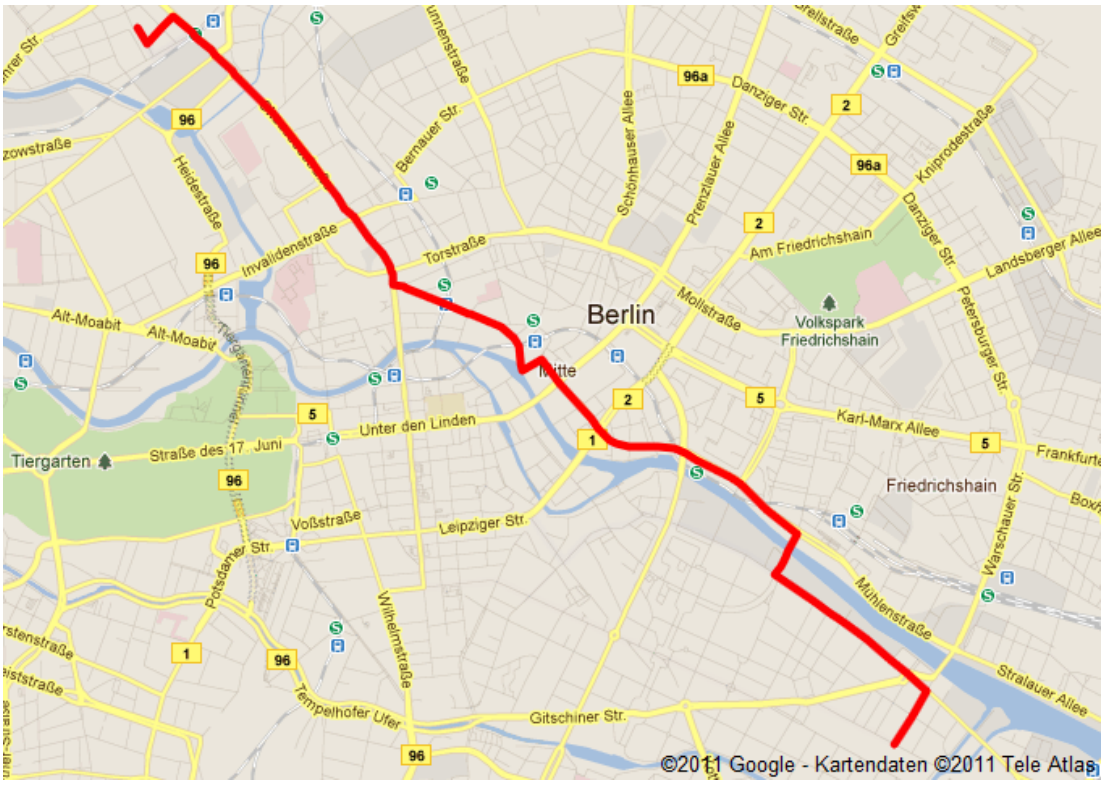


(a) Kürzeste Route. Länge 4 Minuten und 33 Sekunden. Einfachheit: 45,5. Berechnungsdauer: 0,02 Sekunden.



(b) Beste Route. Länge 4 Minuten und 39 Sekunden. Einfachheit: 31,8. Berechnungsdauer: 0,19 Sekunden.

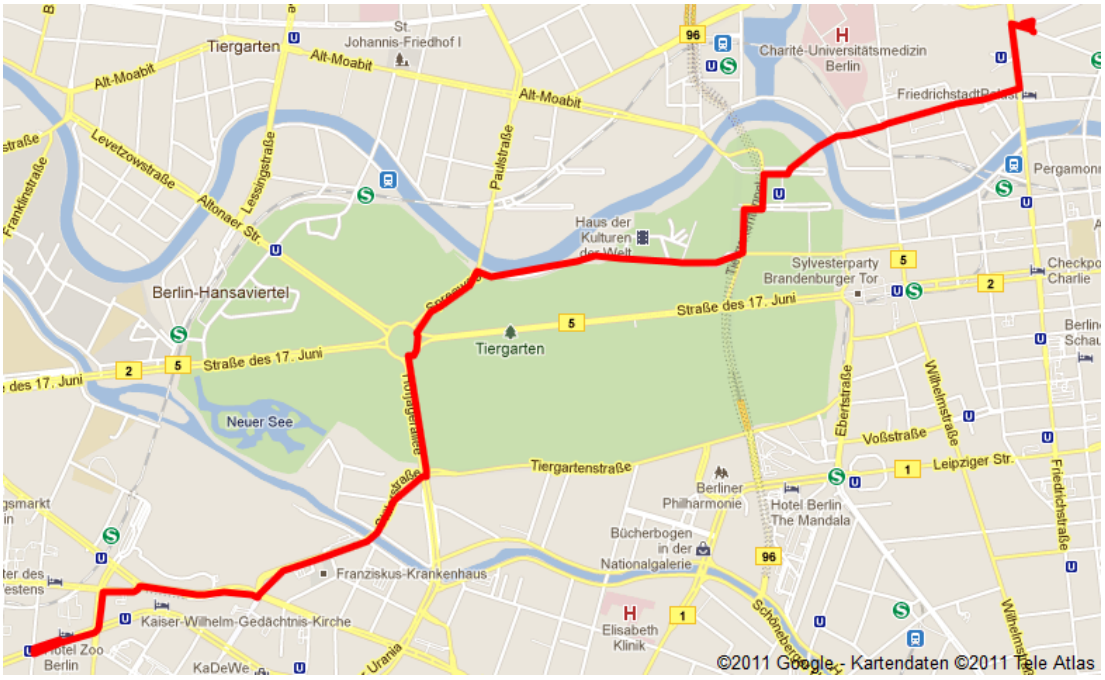
Abbildung 4.20: (Beispiel 4) Darstellung in Google Maps. Stadt: München. Epsilon: 0,1



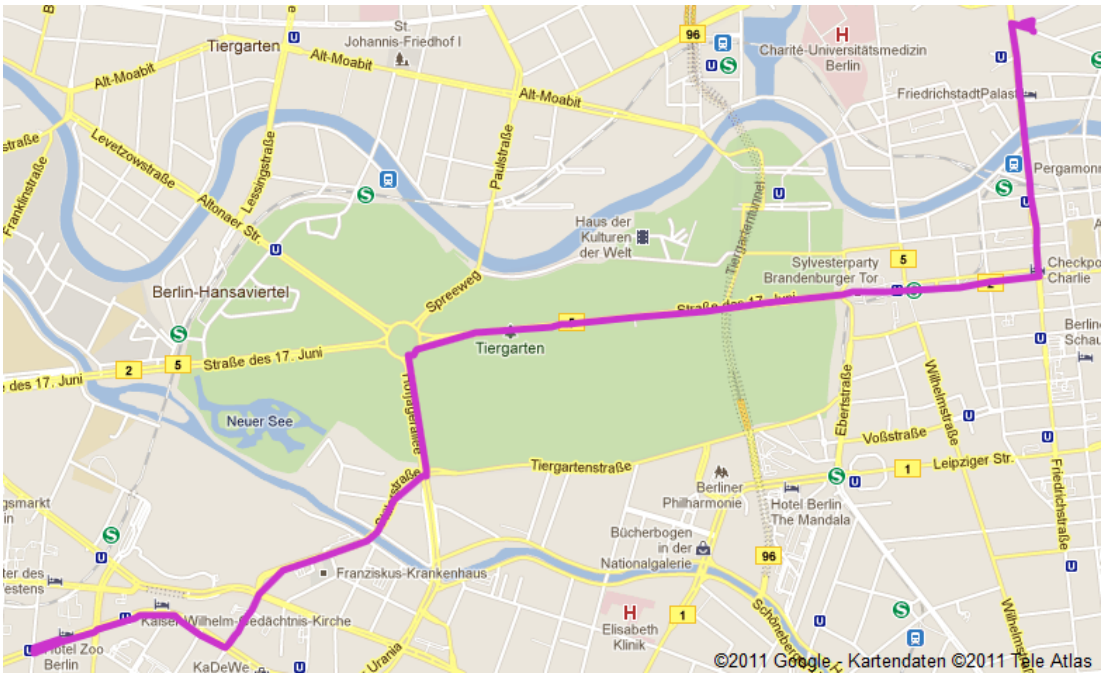
(a) Kürzeste Route. Länge 12 Minuten und 47 Sekunden. Einfachheit: 86,8. Berechnungsdauer: 0,05 Sekunden.

(b) Beste Route. Länge 13 Minuten und 21 Sekunden. Einfachheit: 77,3. Berechnungsdauer: 0,45 Sekunden.

Abbildung 4.21: (Beispiel 5) Darstellung in Google Maps. Stadt: Berlin. Epsilon: 0,1



(a) Kürzeste Route. Länge 7 Minuten und 44 Sekunden. Einfachheit: 104,1. Berechnungsdauer: 0,02 Sekunden.



(b) Beste Route. Länge 8 Minuten und 26 Sekunden. Einfachheit: 85,7. Berechnungsdauer: 0,3 Sekunden.

Abbildung 4.22: (Beispiel 6) Darstellung in Google Maps. Stadt: Berlin. Epsilon: 0,1

5. Zusammenfassung und Ausblick

Diese Arbeit beschäftigt sich mit der Möglichkeit, einfachere Alternativen zum kürzesten Weg in der Routenplanung zu finden. Die Ergebnisse bisheriger Arbeiten zu simplen Routen waren in der Praxis der Routenplanung überwiegend nicht anwendbar, da die einfachsten Routen keine Aussagen über die Länge des Weges treffen. Nachdem Grundlagen erläutert und bisherige Ansätze vorgestellt wurden, wurde in dieser Arbeit zunächst eine mögliche Definition des Begriffs Einfachheit vorgestellt. In Kapitel 3 wurde als Konstrukt zur Berechnung einfachster Routen der Kantengraph vorgestellt. Um zusätzlich Distanzgrenzen modellieren und in die Berechnung einbeziehen zu können, wurde das Konstrukt des erweiterten Kantengraphen eingeführt. Mit diesem können alle Informationen zu Distanz und Einfachheit eines Straßennetzes in einem Graphen modelliert werden. Anschließend wurden verschiedene Maße der Einfachheit in Form von Kostenfunktionen erläutert.
Um einfachste Routen zu finden, mit der zusätzlichen Nebenbedingung, dass diese eine gewisse Distanz nicht überschreiten, haben wir den *simple-and-short* Algorithmus kennen gelernt. Die Beziehungen zwischen kürzesten, einfachsten und besten Routen, sowie die Funktionsweise des Algorithmus' wurden erläutert und eine theoretische sowie eine realitätsnahe Analyse durchgeführt.
Kapitel 4 beschäftigte sich mit der Implementierung und Evaluation des Modells und des Algorithmus' an realen Daten. Anhand des deutschen Straßennetzes wurden Experimente durchgeführt und die kürzesten, einfachsten und besten Routen im Hinblick auf Länge, Einfachheit, Platz- und Zeitverbrauch der Berechnung sowie Verhalten in der Visualisierung untersucht. Die Ergebnisse bestätigten im Bezug auf einfachste Routen die Ergebnisse aus frühreren Arbeiten. Die berechneten besten Routen stellen für die Realität sinnvolle Alternativen zu kürzesten Wegen dar, da sie möglichst einfach, aber trotzdem nicht unverhältnismäßig länger sind als diese.

Da es sich bei dem hier vorgestellten Algorithmus um eine Modifikation des Dijkstra-Algorithmus' handelt, kann auch der *simple-and-short* Algorithmus nicht mit negativen Kantengewichten umgehen. Eine Anwendung, in der spätere Abbiegungen die Route wieder einfacher machen als sie durch vorherige war, ist durch das Konzept des erweiterten Kantengraphen modellierbar, jedoch stellen sich weitere Anforderungen an den Algorithmus.
Wie bereits erwähnt wurde, kann der *simple-and-short* Algorithmus ebenfalls mit Modellierungen umgehen, die auch das Entlangfahren einer Straße mit einem positiven Wert für die Einfachheit versehen. Hierbei könnten in der Zukunft beispielsweise temporäre Ereignisse wie Staus oder Baustellen in die Berechnung der Route einbezogen werden. Weitere Erweiterungsmöglichkeiten ergeben sich je nach Anwendung auch durch die Definition zusätzlicher Ausschlusskriterien, beispielweise nach persönlicher Neigung des Fahrers der Strecke.
Wie wir gesehen haben, werden Schnellstraßen wie Autobahnen sowohl in Distanz als auch Einfachheit bevorzugt, und das Optimierungspotential für beste Routen nimmt mit steigender Nutzung von Autobahnen ab. Um die Laufzeit für die Berechnung zu beschleunigen, ist deshalb eine Heuristik denkbar, die beste Routen vom Startpunkt bis zur Autobahnauffahrt und von der Autobahnabfahrt bis zum Ziel berechnet, und dazwischen den normalen Algorithmus von Dijkstra anwendet. Daneben existiert durch die Anwendung von Techniken wie der A^*-Suche oder Landmarks Potential zur Verbesserung der Laufzeit.

Literaturverzeichnis

[BD10] Reinhard Bauer and Daniel Delling. Sharc: Fast and robust unidirectional routing. Journal of Experimental Algorithmics, 14:4:2.4–4:2.29, January 2010.

[BFM08] Holger Bast, Stefan Funke, and Domagoj Matijevic. TRANSIT Ultrafast Shortest-Path Queries with Linear-Time Preprocessing. In: Demetrescu, Camil, Goldberg, Andrew V., Johnson, David S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, 2008.

[Cal61] Tom Caldwell. On finding minimum routes in a network with turn penalties. Communications of the ACM, 4:107–108, 1961.

[CM85] H. W. Corley and I. D. Moon. Shortest paths in networks with vector weights. Journal of Optimization Theory and Applications, 46:79–86, 1985. 10.1007/BF00938761.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959. 10.1007/BF01386390.

[DK03] Matt Duckham and Lars Kulik. "simplest" paths: Automated route selection for navigation. In Walter Kuhn, Michael Worboys, and Sabine Timpf, editors, Spatial Information Theory. Foundations of Geographic Information Science, volume 2825 of Lecture Notes in Computer Science, pages 169–185. Springer, 2003. 10.1007/978-3-540-39923-0_12.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina Zweig, editors, Algorithmics of Large and Complex Networks, volume 5515 of Lecture Notes in Computer Science, pages 117–139. Springer, 2009. 10.1007/978-3-642-02094-0_7.

[GNPR11] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. On d-regular schematization of embedded paths. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith Jefferey, Rastislav Královic, Marko

Vukolic, and Stefan Wolf, editors, SOFSEM 2011: Theory and Practice of Computer Science, volume 6543 of Lecture Notes in Computer Science, pages 260–271. Springer, 2011. 10.1007/978-3-642-18381-2_22.

[Gol95] Reginald Golledge. Path selection and route preference in human navigation: A progress report. In Andrew Frank and Werner Kuhn, editors, Spatial Information Theory A Theoretical Basis for GIS, volume 988 of Lecture Notes in Computer Science, pages 207–222. Springer, 1995. 10.1007/3-540-60392-1_14.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine McGeoch, editor, Experimental Algorithms, volume 5038 of Lecture Notes in Computer Science, pages 319–333. Springer, 2008. 10.1007/978-3-540-68552-4_24.

[GV11] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In Panos Pardalos and Steffen Rebennack, editors, Experimental Algorithms, volume 6630 of Lecture Notes in Computer Science, pages 100–111. Springer, 2011. 10.1007/978-3-642-20662-7_9.

[HK05] Hartwig Hochmair and Victoria Karlsson. Investigation of preference between the least-angle strategy and the initial segment strategy for route selection in unknown environments. In Christian Freksa, Markus Knauff, Bernd Krieg-Brückner, Bernhard Nebel, and Thomas Barkowsky, editors, Spatial Cognition IV. Reasoning, Action, Interaction, volume 3343 of Lecture Notes in Computer Science, pages 79–97. Springer, 2005. 10.1007/978-3-540-32255-9_5.

[Hoc00] Hartwig Hochmair. ”least angle” heuristic: Consequences of errors during navigation. In Proceedings of GIScience. University of California Regents, 2000.

[Mar86] David M. Mark. Automated route selection for navigation. In Aerospace and Electronic Systems Magazine, volume 1, pages 2–5. IEEE, 1986. 10.1109/MAES.1986.5005198.

[MS08] Kurt Mehlhorn and Peter Sanders. Algorithms and Data Structures - The Basic Toolbox. Springer, 2008.

[RD08] Kai-Florian Richter and Matt Duckham. Simplest instructions: Finding easy-to-describe routes for navigation. In Thomas Cova, Harvey Miller, Kate Beard, Andrew Frank, and Michael Goodchild, editors, Geographic Information Science, volume 5266 of Lecture Notes in Computer Science, pages 274–289. Springer, 2008. 10.1007/978-3-540-87473-7_18.

[Ric07] Kai-Florian Richter. A uniform handling of different landmark types in route directions. In Stephan Winter, Matt Duckham, Lars Kulik, and Ben Kuipers, editors, Spatial Information Theory, volume 4736 of Lecture Notes in Computer Science, pages 373–389. Springer, 2007. 10.1007/978-3-540-74788-8_23.

[RK05] Kai-Florian Richter and Alexander Klippel. A model for context-specific route directions. In Christian Freksa, Markus Knauff, Bernd Krieg-Brückner, Bernhard Nebel, and Thomas Barkowsky, editors, Spatial Cognition IV. Reasoning, Action, Interaction, volume 3343 of Lecture Notes in Computer Science, pages 58–78. Springer, 2005. 10.1007/978-3-540-32255-9_4.

[Vol96] Lutz Volkmann. Fundamente der Graphentheorie. Springer, 1996.

[Win01] Stephan Winter. Weighting the path continuation in route planning. In Proceedings of the 9th ACM international symposium on Advances in geographic information systems, GIS '01, pages 173–176. ACM, 2001.

[Win02] Stephan Winter. Modeling costs of turns in route planning. GeoInformatica, 6:345–361, 2002. 10.1023/A:1020853410145.