Karlsruhe Institute of Technology

Routing mit Violations

Bachelorarbeit
von

Matthias Tangemann

An der Fakultät für Informatik
Institut für theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. rer. nat. Peter Sanders
Betreuende Mitarbeiter:	Dipl.-Inform. Ben Strasser

Bearbeitungszeit: 1. August 2014 – 30. November 2014

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 13. Januar 2015

Deutsche Zusammenfassung

Bei der Routenberechnung für LKW muss beachtet werden, dass einige Straßenabschnitte aufgrund von Restriktionen nicht befahren werden dürfen. So gelten bei vielen Brücken Gewichtseinschränkungen und bei Unterführungen ist die Höhe der Fahrzeuge beschränkt. Klassische Routing-Algorithmen, die Kanten ignorieren auf denen eine Einschränkung verletzt ist, liefern in Fällen in denen keine Route ohne Verletzung existiert, keine Lösung. In dieser Arbeit wird untersucht, wie sich die Algorithmen zur Routenberechnung anpassen lassen, um in solchen Fällen eine Route mit einer *minimalen* Verletzung zu erhalten. Dazu wird zunächst beschrieben, wie sich eine Route mit minimalen Verletzungen formal definieren lässt und anschließend Algorithmen vorgestellt, die entsprechende Routen berechnen. Abschließend werden die Laufzeiten dieser Algorithmen in der Praxis experimentell untersucht.

Inhaltsverzeichnis

1. Einleitung

Eines der ältesten Probleme der Informatik ist es, kürzeste Wege in Graphen bezüglich eines Kantengewichtes zu berechnen. Dieses Problem hat Anwendungen in vielen Feldern, unter Anderem in der Netzwerktechnik und bei der Lösung einiger Planungsprobleme. Im Rahmen dieser Arbeit wird die Routenberechnung für LKW betrachtet. Dazu werden alle Straßen einer Karte als Kanten und Kreuzungen als Knoten eines Straßengraphen betrachtet. Als Kantengewicht wird dann üblicherweise die Distanz oder die mittlere Fahrzeit der entsprechenden Kante verwendet.

Bei der Routenberechnung für LKW muss jedoch zusätzlich beachtet werden, dass auf vielen Straßen Restriktionen für LKW gelten: Es gelten zum Beispiel Gewichtsbeschränkungen für viele Brücken und Tunnel dürfen in der Regel nicht befahren werden, wenn der Lastwagen explosive Stoffe geladen hat. Aufgrund dieser Restriktionen gibt es Fälle, in denen keine Route berechnet werden kann, weil auf jeder Route eine Einschränkung verletzt wäre. Der klassische Routing-Algorithmus, der verletzte Kanten ignoriert, findet in diesem Fall keine Route. Sinnvoller wäre es jedoch, wenn der Algorithmus stattdessen eine Route mit einer minimalen Verletzung und eine entsprechende Warnmeldung liefern würde.

In der Praxis sind einige Fälle denkbar, in denen dieses Problem relevant ist. In manchen Fällen muss eine Restriktion verletzt werden, zum Beispiel wenn ein Lastwagen ein Geschäft in einer Innenstadt beliefern soll, die für LKW gesperrt ist. In solchen Fällen lässt sich eine Ausnahmegenehmigung beantragen. Weiterhin relevant ist das Problem, wenn keine genauen Endpunkte angegeben werden. Soll zum Beispiel nur ermittelt werden, wie weit die Strecke von Karlsruhe nach Berlin in etwa ist, wählt das verwendete Programm die Endpunkte in den Städten selbständig. Werden diese ungeschickt gewählt sodass keine gültige Route existiert, liefert das Programm auch in diesem Fall keine Lösung. Nicht zuletzt ist ein Einsatz für komplexere Planungsalgorithmen denkbar: Ein Algorithmus, der für ein Transportproblem eine Menge von Fahrzeugen wählen soll, kann die differenzierten Informationen zu den verletzten Restriktionen möglicherweise verwenden um die Menge der eingesetzten Fahrzeuge zu optimieren.

In dieser Arbeit wird deshalb untersucht, wie sich eine Route mit einer *minimalen* Verletzung formal definieren lässt und Algorithmen vorgestellt, mit denen sich solche Routen berechnen lassen.

1.1 Inhalt

In Kapitel 2 wird zunächst ein Überblick über die Entwicklungen im Bereich Routenberechnung gegeben. Es werden die verwendeten Basisalgorithmen und Techniken zur Beschleunigung sowie Variationen des klassischen Kürzeste-Wege-Problems kurz vorgestellt.

Kapitel 3 fasst die nötigen Grundlagen, die für das Verständnis dieser Arbeit nötig sind, zusammen. Des Weiteren werden Konventionen, die für die gesamte Arbeit gelten, sowie die verwendete Notation beschrieben. Leser, die mit dem Themen Routenberechnung und Pareto-Optimierung bereits vertraut sind, können diese Kapitel zunächst überspringen und später bei Bedarf darauf zurückkommen.

Das Problem Routing mit Violations ergibt sich direkt aus Problemfällen, die in der Praxis auftreten. In Kapitel 4 wird eine Formalisierung der Problemstellung beschrieben. Diese Formalisierung ist Grundlage für alle weiteren Betrachtungen des Problems. Im letzten Abschnitt des Kapitels wird die Komplexität des Problems untersucht und gezeigt, dass der zur Lösung nötige Berechnungsaufwand exponentiell mit der Knotenanzahl des Graphen steigen kann.

In Kapitel 5 wird eine Verallgemeinerung des Algorithmus von Dijkstra vorgestellt, mit der sich das Problem Routing mit Violations lösen lässt. Die Korrektheit des Algorithmus, bezüglich der im vorherigen Kapitel beschriebenen, formalen Problemstellung, wird bewiesen und Techniken zur Beschleunigung des Algorithmus vorgestellt.

Bei der Vorstellung der formalen Problemstellung wurde eine Familie von Graphen definiert, für die der Berechnungsaufwand exponentiell mit der Größe des Graphen steigt. In Kapitel 6 werden die Ergebnisse von Experimenten vorgestellt, die aufzeigen, wie sich der Algorithmus in der Praxis verhält. Weiterhin wird geprüft, wie hoch die Beschleunigung durch die im vorherigen Kapitel beschriebenen Techniken bei realen Eingabedaten ausfällt.

Im letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst und die Anwendbarkeit auf die Praxis diskutiert. Weiterhin werde Ansätze genannt, mit denen sich die vorgestellten Algorithmen noch weiter beschleunigen lassen könnten.

1.2 Danksagung

Danken möchte ich besonders Ben Strasser, der mich bei der Ausarbeitung dieser Arbeit stets mit wertvollen Tipps unterstützt hat. Weiterhin möchte ich den Kollegen bei der Firma PTV Group danken, die mir die Arbeit in einer optimalen Umgebung ermöglicht haben und mir bei Schwierigkeiten stets zur Seite standen. Nicht zuletzt möchte ich meiner Familie und meinen Freunden danken, die mir, insbesondere in den arbeitsintensiven Abschnitten meiner Arbeit, stets die nötige Zeit gelassen haben.

2. Verwandte Arbeiten

Das Problem, in einem Graphen kürzeste Pfade zu berechnen, ist eines der ältesten der Informatik. Es hat Anwendungen in vielen Feldern, im Rahmen dieser Arbeit wird die Routenberechnung auf Straßennetzwerken betrachtet. Einen guten Überblick über die Algorithmen in diesem Themenfeld und Varianten des Problems bietet das Übersichtspaper [DSSW09] oder [BDG+14].

Im einfachsten Fall werden in einem Graphen kürzeste Wege bezüglich einer statischen, nicht-negativen Kantengewichtsfunktion berechnet. Werden die kürzesten Routen von einem Startknoten zu allen anderen Knoten gesucht (*Single Source Shortest Path*, SSSP), wird klassischer Weise der Algorithmus von Dijkstra verwendet (vgl. Abschnitt 5.1).

Da Straßengraphen nicht selten einige Millionen Kanten und Knoten enthalten, wurden in den letzten Jahrzehnten einige Beschleunigungstechniken für den klassischen Algorithmus von Dijkstra entwickelt. Mit diesen ist eine Beschleunigung des Basisalgorithmus um einen Faktor von bis zu einer Million möglich. Das ist insbesondere nötig, wenn als Teil eines komplexeren Planungsproblems viele kürzeste Wege berechnet werden müssen.

2.1 Beschleunigungstechniken

Entscheidend für die Laufzeit des Algorithmus von Dijkstra, ist die Größe des Suchraumes. Der Suchraum ist dabei die Anzahl der Knoten die betrachtet werden, bevor ein kürzester Weg gefunden wurde. Der Algorithmus sucht standardmäßig kreisförmig ausgehend vom Startknoten, bis der Zielknoten gefunden wird. Bei der *bidirektionalen Suche* wird abwechselnd vom Start- und Zielknoten ausgehend gesucht und die Teilpfade der Suchen kombiniert. Dadurch entspricht der Suchraum zwei kleineren Kreisen um die beiden Endknoten, wodurch dieser gegenüber dem Standardalgorithmus verkleinert wird.

Eine andere Möglichkeit ist die Verwendung des *A*-Algorithmus*. Die Idee dabei ist, Kanten die auf das Ziel zuführen künstlich zu verkürzen und Kanten die vom Ziel wegführen zu verlängern. Dadurch sucht der Algorithmus nicht mehr kreisförmig sondern stärker in Richtung des Zielknotens. Aus diesem Grund wird diese Technik auch *Geometric Goal Directed Search* genannt. Der A*-Algorithmus lässt sich, wie in Abschnitt 5.3.2 beschrieben, auch auf das vorliegende Problem anwenden.

Um den Algorithmus von Dijkstra weiter zu beschleunigen, lässt sich ausnutzen, das Straßengraphen annähernd planar sind. Wie bei vollständig planaren Graphen, lassen sich

dadurch gute Separatoren, d.h. Knotenteilmengen durch deren Wegnahme der Graph in Komponenten zerfällt, finden. Dies kann so ausgenutzt werden, dass zur Routenberechnung nur noch die Komponenten, in denen die Endknoten liegen, sowie der Separator betrachtet werden muss (*Multi-Level Techniques*).

Eine weitere Methode zur Beschleunigung, ist die Ausnutzung des hierarchischen Aufbaus von Straßengraphen: Einige Knoten, zum Beispiel die Knoten von Autobahnen, werden wesentlich öfter in kürzesten Wegen verwendet als andere. Ein verbreitete Methode, diese Eigenschaft auszunutzen, ist die Berechnung von *Contraction Hirarchies (CHs)* [GSSD08]. Dabei wird zunächst eine Ordnung der Knoten nach Wichtigkeit berechnet und die Knoten in dieser Ordnung kontrahiert (d.h. aus dem Graph entfernt, wobei Kanten zwischen je zwei benachbarten Knoten eingefügt werden) und in einen Suchgraph eingefügt. Anfragen nach kürzesten Routen können dann auf diesem Suchgraphen, in wesentlich kürzerer Zeit, berechnet werden. Die aufwändigere Berechnung der Contraction Hierarchy ist unabhängig vom Start- und Zielpunkt und muss somit für jeden Graphen nur einmal ausgeführt werden.

Es existieren einige weitere Verfahren, die den Algorithmus von Dijkstra durch Vorberechnungen beschleunigen. Insbesondere sind drei-stufige Verfahren verbreitet: In einem ersten Schritt werden Vorberechnungen basierend auf der Struktur des Graphen durchgeführt. Diese werden im zweiten, schnelleren Schritt um die Gewichtsfunktion erweitert. Im dritten Schritt wird dann eine kürzeste Route zwischen zwei Knoten berechnet. Besonders vorteilhaft ist dieses Verfahren bei Verwendung mehrerer Gewichtsfunktionen, da der erste, aufwändigste Schritt nur auf der Struktur des Graphen basiert und somit auch dann nur einmal durchgeführt werden muss. Auch Contraction Hierarchies lassen sich in diese drei-stufige Form bringen (*Customizable Contraction Hierarchies*, [DSW14]).

2.2 Varianten des Problems

In der Praxis treten oft Varianten des klassischen Kürzesten-Wege-Problems auf. Für viele von diesen wurden in der Vergangenheit bereits effiziente Algorithmen entwickelt bzw. angepasst.

Treten auch negative Kantengewichte auf, kann der Graph negative Zyklen enthalten. In diesem Fall können Pfadkosten beliebig klein werden, sodass kein kürzester Weg existiert. Der Algorithmus von Dijkstra würde in diesem Fall nicht terminieren. Der Bellmann-Ford-Algorithmus dagegen terminiert in jedem Fall und es lässt sich nach der Ausführung leicht erkennen ob der Graph negative Zyklen enthält. Ebenfalls verwendet wird dieser Algorithmus, wenn die kürzesten Wege zwischen allen Paaren von Knoten und nicht nur ausgehend von einem Startknoten gesucht werden.

Weitere Problemstellungen ergeben sich auch, wenn mehrere Transportnetze betrachtet werden (*Intermodales Routing*), wenn Abbiegekosten oder *zeitabhängige* Faktoren berücksichtigt werden müssen. In diesen Fällen reicht eine statische, skalare Kantengewichtsfunktion zur Modellierung des Problems nicht aus.

Auch in dieser Arbeit wird eine multikriterielle Gewichtsfunktion betrachtet, d.h. es existiert kein eindeutiger optimaler Pfad zwischen zwei Knoten mehr. Aus diesem Grund lassen sich die meisten bisher entwickelten Beschleunigungstechniken nicht auf das betrachtete Problem anwenden. In [DSSW09] wird deshalb die Adaption der Techniken auf den multikriteriellen Fall als wichtiges Problem der näheren Zukunft bezeichnet ("The adaption of a fast method to this scenario is one of the main challenges in the near future.", S.128).

3. Vorbemerkungen

3.1 Graphentheorie

Im Rahmen dieser Arbeit werden nur gerichtete Graphen betrachtet, sei deswegen $G = (V, E \subseteq V \times V)$ stets ein einfacher, gerichteter Graph. Weiterhin bezeichne $n := n(G) := |V|$ die Anzahl der Knoten und $m := m(G) := |E|$ die Anzahl der Kanten des Graphen.

Ein *Weg* $W = (v_0, v_1, \ldots, v_l)$ ist eine Folge von nicht zwingend verschiedenen Knoten $v_i \in V, 0 \leq i \leq l$, wobei alle auf einander folgenden Knoten v_i, v_{i+1} adjazent sind (also $(v_i, v_{i+1}) \in E$). Die Anzahl Knoten l heißt die *Länge* des Weges. Ein Weg für den alle Knoten paarweise verschieden sind heißt *Pfad* $P = (v_0, v_1, \ldots, v_l)$.

Eine Funktion $f : E \to \mathbb{R}$, die jeder Kante von G einen Wert zuordnet heißt *Kantengewicht*. Weiterhin sei das Gewicht eines Weges $W = v_0, \ldots, v_l$ definiert als die Summe der Kantengewichte der enthaltenen Kanten:

$$f(W) := \sum_{i=1}^{l} f((v_{i-1}, v_i))$$

Eine Zone $Z \subseteq V$ ist eine Knotenteilmenge des Graphen. Jede Zone induziert eine Subgraph $G_Z = (Z, E_Z = E \cap Z \times Z)$. Eine Zone heißt zusammenhängend, wenn der durch sie induzierte Subgraph zusammenhängend ist.

3.2 Pareto-Optimierung

Die Formalisierung, die in Kapitel 4 vorgestellt wird, verwendet eine multikriterielle Gewichtsfunktion. Die Kosten eines Weges sind damit ein Vektor. Zum Vergleich der [...].

Ein Vektor a dominiert einen Vektor b, geschrieben $a \prec b$, falls alle Komponenten von a kleiner oder gleich als die von b sind, und eine Komponente echt kleiner ist. Die Notation $a \preceq b$ wird verwendet wenn auch $a = b$ gelten darf. Die entgegengesetzten Operatoren $\succ$ und $\succeq$ werden entsprechend definiert.

Eine Menge von Vektoren heißt pareto-optimal, wenn diese keine zwei Vektoren enthält, so dass einer den anderen dominiert. Im Falle einer multikriteriellen Gewichtsfunktion beim Routing gibt es damit keine eindeutige optimale Distanz, sondern eine Menge von pareto-optimalen Distanzen.

4. Formalisierung der Problemstellung

Wie in der Einleitung beschrieben, ergibt sich das betrachtete Problem direkt aus Problemfällen aus der Praxis. Für die Entwicklung und Bewertung eines Algorithmus ist jedoch eine formale Problemdefinition nötig, welche in diesem Kapitel beschrieben wird. Dazu wird zunächst die komplette Formalisierung vorgestellt und anschließend die einzelnen Elemente näher beschrieben.

Im folgenden sei $G = (V, E)$ stets ein einfacher, gerichteter Graph mit $n := |V|$ Knoten und $m := |E|$ Kanten. Weiterhin sei $R \in \mathbb{N}^+$ die Anzahl betrachteter Restriktionstypen und $p \in \mathbb{R}^R$ ein Vektor, der das beim Routing betrachtete Fahrzeug bezüglich aller Restriktionstypen beschreibt. Details zur Modellierung der Restriktionen und des Fahrzeugprofils finden sich in Abschnitt 4.1.2.

4.1 Formale Problemdefinition

Für die Formalisierung wurde ein multikriterieller Ansatz gewählt: Anstatt einer skalaren Gewichtsfunktion wird für das Routing eine mehrdimensionale Funktion gewählt. Die Kosten eines Weges sind damit ein Vektor. Die letzte Komponente ist dabei die Distanz wie sie beim klassischen Routing verwendet wird, die anderen Kriterien sind Strafkosten für die Verletzung von Restriktionen. Dadurch existiert zwischen zwei Knoten des Graphen keine eindeutige, optimale Route mehr, sondern eine Menge von pareto-optimalen Routen. Genauer ist ein Pfad genau dann pareto-optimal, wenn sich durch Inkaufnahme von Violations die räumliche Distanz oder die Strafkosten einer anderen Restriktion verringern lassen.

Dadurch ergibt sich direkt das Problem, zu gegeben Start- und Zielknoten, die Menge aller pareto-optimalen Distanzen zu berechnen. Das Problem *Routing mit Violations* besteht dann darin, zu jeder dieser Distanzen eine Route zu berechnen:

> Gegeben sei ein einfacher, gerichteter Graph $G = (V, E)$, eine nichtnegative Distanzfunktion $w : E \to \mathbb{R}_0$ sowie $R \in \mathbb{N}_0$ Restriktionen. Zu jeder Restriktion i $(1 \leq i \leq R)$ sei ein Kantengewicht $r_i : E \to \mathbb{R}_0 \cup \{\infty\}$ sowie ein Bewertungsvektor (c_i, z_i, d_i, k_i) gegeben.
>
> Die Kosten eines Pfades P sind $k(P) = (q_C(P), \ldots, q_1(P), w(P))^\top$, wobei $q_j(P)$ die Summe der Strafkosten aller Violations von Restriktion i aus Klasse j sind (d.h. $c_i = j$).

Die Kosten einer einzelnen Violation $X \subseteq P$ ist dabei die Summe der Zonen-, Distanz- und Kapazitätskosten, wobei die Gewichtung der Anteile für jede Restriktion einzeln definiert ist:

$$q_i(X) := z_i + d_i \cdot w(X) + k_i \cdot \delta(X)$$

Zu zwei Knoten $s, t \in V$ wird dann zu jeder pareto-optimalen Distanz von s nach t ein Pfad mit dieser Distanz gesucht.

4.1.1 Restriktionsunabhängige Distanzen

Die oben beschriebene Problemstellung ist eine Erweiterung des Kürzeste-Wege-Problems. Damit gehen wie beim klassischen Problem restriktionsunabhängige Kosten in die Distanz eines Pfades ein: Das Kantengewicht $w : E \to \mathbb{R}_0$ ist ein nicht-negatives Distanzmaß in diesem Sinn (z.B. Länge in Metern) und geht als letzte Komponente in den Kostenvektor einer Route ein.

4.1.2 Restriktionen

Restriktionen werden als Kantengewichte $r_i : E \to \mathbb{N}_0 \cup \{\infty\}$, $1 \leq i \leq R$ modelliert. R sei dabei die Anzahl verschiedener Restriktionstypen. Die Gewichte weisen jeder Kante die zulässige Kapazität der entsprechenden Restriktion zu. Gilt die Einschränkung auf der Kante nicht, so wird $r_i(e) = \infty$ gesetzt:

$$r_i(e) := \begin{cases} m & \text{frei bis Maximalwert } m \\ \infty & \text{keine Einschränkung} \end{cases}$$

Für einige Restriktionen, wie zum Beispiel Verbote von Anhängern, muss kein Maximum definiert werden. Für dies wird $m = 0$ festgelegt:

$$r_i(e) := \begin{cases} 0 & \text{Kante eingeschränkt} \\ \infty & \text{keine Einschränkung} \end{cases}$$

Für die Routenberechnung ist dann ein Fahrzeugprofil nötig, anhand dessen geprüft wird, ob eine Restriktion verletzt ist. Die Kapazitäten zu jedem Restriktionstyp sind als die Werte $p_i \in \mathbb{N}_0$, $1 \leq i \leq R$ des Fahrzeugsprofils definiert. Für Einschränkungen i ohne Maximum sei $p_i = 1$.

4.1.3 Violations

Jede Restriktionsfunktion r_i definiert implizit Zonen auf dem Straßengraphen. Für jeden Wert c, der von r_i angenommen wird, bilden alle Knoten die zu einer Kante e mit $r_i(e) = c$ adjazent sind, eine Zone.

Auf einer Kante $e \in E$ ist eine Restriktion i genau dann verletzt, wenn $r_i(e) < p_i$ gilt. Auf einem Knoten genau dann, wenn eine adjazente Kante verletzt ist. Als Violation X wird ein inklusionsmaximaler, zusammenhängender Teilpfad einer Route, der innerhalb einer verletzten Zone verläuft. Es muss also insbesondere die Kapazität aller enthalten Kanten gleich sein. Da bereits die Verwendung eines Knotens eine Zone verletzt, kann eine Violation auch ein Pfad mit 0 Kanten, also ein Knoten, sein.

Für jede Violation werden Strafkosten berechnet, welche jeweils die Summe aus Zonen-, Distanz- und Kapazitätskosten sind.

Die Verwendung eines Knotens aus einer verletzten Zonen, wird mit den konstanten Zonenkosten z_i bestraft. Diese werden für eine Violation stets genau einmal zu den gesamten Strafkosten addiert, auch wenn die Violation mehr Knoten enthält. Weiterhin ist eine Violation als Pfad definiert und hat damit eine (räumliche) Distanz $w(X)$. Diese geht mit dem Faktor d_i in die Strafkosten der Violation ein. Die Kapazitätskosten berechnen sich, in dem der Wert $\delta(X)$, um den die zulässige Kapazität überschritten wird, mit einem Faktor k_i multipliziert wird. Auch diese Kosten werden bereits addiert, wenn nur ein Knoten aus einer verletzten Zone verwendet wird.

Insgesamt berechnen sich die Strafkosten einer Violation damit nach folgender Formel:

$$q_i(X) := z_i + d_i \cdot w(X) + k_i \cdot \delta(X)$$

4.1.3.1 Klasseneinteilung

Die Verletzung von manchen Restriktionstypen ist deutlich schwerwiegender als bei anderen. So sollen zum Beispiel eine Route, auf der eine physikalische Einschränkungen (z.B. Höhenbeschränkungen) verletzt ist, stets schlechter bewertet werden als eine Route auf der nur rechtliche Restriktionen verletzt sind. Weiterhin soll sichergestellt werden, dass eine Route ohne Violations stets am Besten bewertet wird.

Um dies zu garantieren, ist es nicht möglich, die Strafkosten einfach zur Distanz zu addieren: Ein hinreichend langer Umweg ohne Verletzungen würde so schlechter bewertet als eine kurze Route mit Verletzung und eine hinreichend große Anzahl rechtlicher Einschränkungen würde schlechter bewertet als die Verletzung einer physikalischen Einschränkung.

Stattdessen werden die Restriktionen in $C \in \{1 \dots R\}$ Klassen eingeteilt. Sei $1 \leq c_i \leq C$ die Klasse der Restriktion i. Die Kosten eines Pfades P ist dann ein Vektor $(q_C(P), \dots, q_1(P), w(P))$. Dabei sei $q_i(P)$ die Summe aller Strafkosten von Violations aus Klasse i.

4.1.4 Ordnung der pareto-optimalen Routen

In vielen Fällen werden nicht alle pareto-optimalen Routen sondern nur eine optimale Route gefordert. Dazu werden die Distanzen der berechneten Routen lexikalisch geordnet und so eine beste Route bestimmt. Diese Ordnung führt aufgrund der Klasseneinteilung dazu, dass eine Route ohne Violations stets am besten bewertet wird. Existiert keine solche Route, werden zunächst Routen bevorzugt, die nur Violations aus Klasse 1 enthalten, danach Routen mit Violations aus den Klassen 1 und 2 usw.

4.2 Alternative Ansätze

Ein alternativer Ansatz wäre, die Strafkosten für Violations direkt zum Distanzmaß des Routings zu addieren. Allerdings sollte dies nicht für jede verletzte Kante passieren, da sonst die Anzahl verletzter Kanten minimiert wird. So würde zum Beispiel eine Stadt, die für LKW gesperrt ist, umfahren, wenn sich durch Betreten von der anderen Seite die Anzahl verletzter Kanten verringern lässt. Da in der Praxis in der Regel kein Zusammenhang zwischen Kantenanzahl und Distanz der Route besteht, erhält man so oft unerwünschte Ergebnisse.

Besser ist es deshalb, die Strafkosten nur beim Betreten der Zone in der die Restriktion gilt, zu addieren. Um jedoch zu garantieren, dass eine Route ohne Violations stets bevorzugt wird, müssten die Strafkosten sehr hoch gewählt werden (mindestens so groß wie der

Tabelle 4.1: Bewertung von Violations

Restriktion	Klasse	Zonenkosten	Distanzkosten	Kapazitätskosten
Gewicht	1	50	0	10
Achslast	1	50	0	10
Fahrzeug-Verbot	1	0	1	0
LKW-Verbot	1	0	1	0
Anhängerverbot	1	90	0	0
Umweltzonen	1	100	0	0
Nachtfahrverbot	1	80	0	0
Länge	2	200	0	1
Steigung	2	200	1	5
Tunnel Restriction Code	2	100	1	0
Gefahrgut	2	100	1	0
Explosive Stoffe	2	100	1	0
Wassergefährdende Stoffe	2	100	1	0
Höhe	3	1000	0	1
Breite	3	1000	0	1

Durchmesser des Graphen). Sollen zusätzlich Violations eines bestimmten Typs garantiert bevorzugt werden, müssten die Kosten für einige Typen noch einmal deutlich größer gewählt werden. Dies ist ein der Praxis normalerweise nicht möglich. Da die Fehleranzahl bei geringeren Kosten vermutlich sehr gering ist, könnte ein solcher Ansatz für die Praxis dennoch relevant sein. Im Rahmen dieser Arbeit wird dies jedoch nicht weiter verfolgt.

4.3 Beispiel für die Bewertung von Restriktionen

Welche Restriktionen konkret betrachtet werden hängt vom verwendeten Kartenmaterial ab und kann sich, zum Beispiel durch Änderung von Gesetzen, ändern. Die Problemstellung wurde deshalb so Allgemein formuliert, dass beliebig viele Restriktionen berücksichtigt werden können. Weiterhin lässt sich die Berechnung der Strafkosten durch die Klasseneinteilung und die Gewichtung der Zonen-, Distanz- und Kapazitätskosten die Bewertung der Restriktionen flexibel einstellen.

Im Rahmen dieser Arbeit wird Kartenmaterial der Firma *PTV Group*[1] verwendet. Tabelle 4.1 zeigt, wie die Klasseneinteilung und Gewichtung für die, in diesen Kartendaten zur Verfügung stehenden, Restriktion gewählt wurden.

Die Restriktionen, die der 1. Klasse zugeordnet sind, sind leicht zu umgehen und sollen deswegen bevorzugt verletzt werden. Dies umfasst zum einen rechtliche Einschränkungen, wie zum Beispiel Verbote von LKWs oder Anhängern. Außerdem wurden die Gewichtseinschränkungen dieser Klasse zugeordnet. Diese betreffen in der Regel Brücken und bestehen, um eine hohe Abnutzung der Brücke durch Schwerlastverkehr zu vermeiden. Muss die Brücke aber benutzt werden, um das Ziel zu erreichen, lässt sich normalerweise problemlos eine entsprechende Genehmigung beantragen.

In Klasse 2 wurden Restriktionen eingeteilt, bei denen eine Verletzung zwar schwierig aber möglich ist. Das betrifft zum Beispiel alle Einschränkungen aufgrund von Gefahrgut. Auch Längenbeschränkungen lassen sich durch Mitbenutzung der Gegenfahrbahn oft umgehen.

[1] Basierend auf Kartendaten von Nokia Here (ehemals Navteq) aus dem 4. Quartal 2013

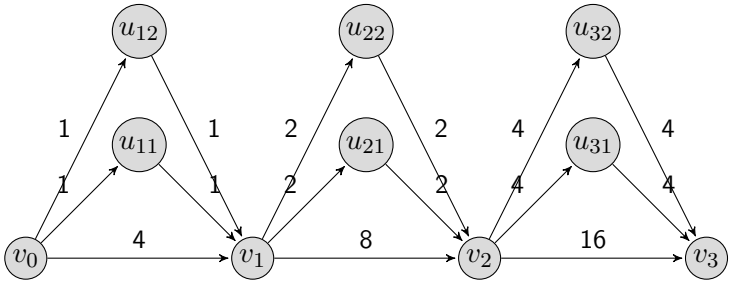

Abbildung 4.1: Worst-Case-Graph G_3

Klasse 3 enthält schließlich Höhen- und Breitenbeschränkungen, die in der Praxis nicht verletzt werden können. Da eine Überschreitung um wenige Zentimeter noch möglich sein könnte, werden die Kapazitätskosten auch berücksichtigt, damit kleine Überschreitungen etwas besser bewertet werden.

4.4 Komplexität des Problems

Abschließend soll die Komplexität des Problems beurteilt werden. Dazu wird die Anzahl von Lösungen des Problems in Abhängigkeit von der Knoten- und Kantenzahl des Graphen sowie der Anzahl von Restriktionsklassen C betrachtet. Die Klassenanzahl C sei im Folgenden beliebig aber fest.

Betrachtet wird die Familie von Graphen $\{G_k | k \in \mathbb{N}^+\}$. Die Struktur der Graphen G_k sei wie folgt definiert (vgl. G_3 in Abbildung 4.1):

$$\begin{aligned} V(G_k) :=& \{v_i | i = 0 \dots k\} \\ & \cup \{u_{ij} | i = 1 \dots k, j = 1 \dots C\} \end{aligned}$$

$$\begin{aligned} E(G_k) :=& \{(v_{i-1}, v_i) | i = 1 \dots k\} \\ & \cup \{(v_{i-1}, u_{ij}) | i = 1 \dots k, j = 1 \dots C\} \\ & \cup \{(u_{ij}, v_i) | i = 1 \dots k, j = 1 \dots C\} \end{aligned}$$

Weiterhin sei zu jedem Graphen eine Distanzfunktion $w_{C,s}$ wie folgt definiert:

$$w_k(v, w) := \begin{cases} 2(i+1) & \text{falls } \exists i : v = v_{i-1} \wedge w = v_i \\ i & \text{falls } \exists i, j : v = v_{i-1} \wedge w = u_{ij} \\ & \text{oder } \exists i, j : v = u_{ij} \wedge w = v_i \end{cases}$$

Zu jeder Restriktionsklasse j sei eine Restriktion r_j definiert:

$$r_j(v, w) := \begin{cases} 0 & \text{falls } \exists i : v = u_{ij} \vee w = u_{ij} \\ \infty & \text{sonst} \in \end{cases}$$

Als Fahrzeugprofil wird $p = (1, 1, \dots) \in \mathbb{R}^C$ gewählt, damit alle Restriktionen im Graph verletzt sind. Die Start- und Zielknoten des Routings seien $s = v_0$ und $t = v_k$.

Im Graphen G_k gibt es somit zwischen v_i und v_{i+1} genau $C + 1$ verschiedene Wege: Einen direkten Weg über die Kante (v_i, v_{i+1}) und C Wege über die Knoten $u_i j$. Auf jedem der

Wege über einen Knoten $u_i j$ ist die Restriktion j verletzt, auf dem direkten Weg ist keine Restriktion verletzt. Insgesamt gibt es in G_k also genau $(C+1)^k$ verschiedene v_0-v_k-Wege.

Jeder dieser Pfade ist pareto-optimal. Um das zu zeigen wird für jede der Alternativen zwischen zwei aufeinander folgenden Knoten v_i, v_{i+1} ein Bitvektor betrachtet, wobei das i-te Bit genau dann gesetzt ist, wenn der entsprechende v_i, v_{i+1}-Teilpfad benutzt wird. Jedem s-t-Pfad kann so eindeutig eine Menge von Bitvektoren zugeordnet werden. Die Kosten des Pfades ergeben sich direkt aus den Vektoren: Der Wert jedes Bitvektors entspricht dem halben Kosten des entsprechenden Kriteriums. Da die Distanz für jede Kante größer 0 ist, kommt bei diesem Kriterium ein Offset von 2^{k-1} hinzu. Da in jedem Segment eine der Alternativen gewählt werden muss, ist die Summe der Bitvektoren konstant. Daraus geht direkt hervor, dass wenn für eine Route der Wert eines Kriteriums echt kleiner ist, der Wert eines anderen Kriteriums echt größer sein muss. Es kann also kein Pfad einen anderen dominieren. Da der Wert des Bitvektors eindeutig ist, existieren auch keine zwei Pfade mit den selben Kosten.

In G_k existieren also $(C+1)^k$ pareto-optimale s-t-Pfade. Weiterhin enthält der Graph G_k $n = k \cdot (C+1) + 1 \in \Theta(k)$ Knoten und $m = k \cdot (2C+1) \in \Theta(k)$ Kanten. Damit steigt die Anzahl von pareto-optimalen s-t-Pfaden also exponentiell mit der Anzahl der Knoten des Graphen. Da für jeden dieser Pfade mindestens ein Eintrag zu S hinzugefügt wird, ist die Worst-Case-Laufzeit von jedem Algorithmus der das Problem Routing mit Violations löst exponentiell.

5. Algorithmus

5.1 Algorithmus von Dijsktra

Um kürzeste Wege in einem Graphen ohne negative Kantengewichte zu finden, kann der Algorithmus von Dijkstra verwendet werden (Algorithmus 5.1, siehe auch [CLRS01]). Für jeden Knoten $v \in V$ wird eine obere Schranke für die Distanz $v.d$ sowie der Vorgängerknoten $v.\pi$ auf einem Pfad dieser Länge gespeichert.

Zunächst wird für jeden Knoten $v.d = \infty$ und $v.\pi = \text{NIL}$ gesetzt. Der Startknoten s wird mit $s.d = 0$ und $s.\pi = \text{NIL}$ initialisiert. Anschließend werden alle Knoten in eine Warteschlange Q eingefügt. Solange Q nicht leer ist wird der Knoten u mit dem kleinsten $u.d$ aus der Warteschlange entnommen und alle adjazenten Knoten *relaxiert*. D.h. für jeden benachbarten Knoten wird geprüft, ob der Pfad über den Knoten u kürzer ist, als der beste bisher gefundene Pfad. Ist dies der Fall wird die Schranke für die Distanz angepasst und u als Vorgänger festgehalten.

Nach Ausführung des Algorithmus ist für alle Knoten $v.d$ die kürzeste Distanz aller Pfade von s nach v und $v.\pi$ der Vorgängerknoten auf einem kürzesten Pfad (möglicherweise existieren mehrere kürzeste Pfade mit der selben Distanz). Der kürzeste s-t-Pfad kann also ausgehend von t mit Hilfe der gespeicherten Vorgängerknoten rekonstruiert werden.

Dieser Algorithmus kann auch im multikriteriellen Fall verwendet werden, wenn nur eine Lösung bezüglich einer Ordnung berechnet werden soll. So werden beim Routing mit Violations die Lösungen lexikalisch sortiert, um Pfade mit keinen oder nur Violations aus geringen Klassen zu bevorzugen. Dies ist beispielsweise dann interessant, wenn das Ergebnis für komplexere Logistikprobleme weiterverwendet wird.

5.2 Multikriterieller Dijkstra-Algorithmus

Der Algorithmus von Dijkstra kann erweitert werden um alle pareto-optimalen Pfade bei Verwendung einer multikriteriellen Gewichtsfunktion zu berechnen (Algorithmus 5.2).

Im Unterschied zum klassischen Dijkstra ist die optimale Distanz zwischen zwei Knoten nicht eindeutig. In der Warteschlange Q werden deshalb keine Knoten, sondern Einträge $e = (v, d, \pi) \in V \times \mathbb{R}^N \times V$ gespeichert. Zur besseren Lesbarkeit werden die Komponenten des Tripels mit $e.v$, $e.d$ und $e.\pi$ bezeichnet. Jeder Eintrag entspricht dabei einem s-$e.v$-Pfad der Länge $e.d$; $e.\pi$ ist der Vorgängerknoten von $e.v$ auf diesem Pfad. Auf diese Weise können für jeden Knoten mehrere Pfade gespeichert werden.

Algorithm 5.1: DIJKSTRA

Input: Gerichteter Graph $G = (V, E \subseteq V \times V)$ mit Kantengewichtsfunktion $w : E \to \mathbb{N}$, Knoten $s \in V$

Output: $\forall v \in V$ Länge $v.d$ des kürzesten s-v-Weges und Vorgängerknoten $v.\pi$ von v auf einem s-v-Weg dieser Länge

```
// Initialisierung
for each vertex v ∈ G.V do
    v.d = ∞
    v.π = NIL
s.d = 0
Q = G.V
// Hauptschleife
while Q ≠ ∅ do
    u = EXTRACT-MIN(Q)
    for each vertex v ∈ G.Adj[u] do
        if u.d + w(u, v) < v.d then
            v.d = u.d + w(u, v)
            v.π = u
```

Zu Beginn befindet sich nur ein Eintrag für den Startknoten in der Warteschlange. Wie beim klassischen Dijkstra wird während jeder Iteration der while-Schleife ein Eintrag aus Q entnommen und alle ausgehenden Kanten des Entsprechenden Knotens relaxiert. Die Ordnung $<$ zur Bestimmung des kleinsten Elements in Q muss dabei so gewählt werden, dass für alle Vektoren $a, b \in \mathbb{R}^N : a \prec b \Rightarrow a < b$ gilt. Für das Problem Routing mit Violations wird die lexikalische Ordnung der Vektoren verwendet, welche die Bedingung erfüllt.

Ein Eintrag e', der bei der Relaxierung erzeugt wurde und von keinem anderen Eintrag dominiert wird, wird in die Warteschlange eingefügt. Ist dies der Fall, werden alle Eintrage aus Q gelöscht, die von e' dominiert werden. Aufgrund der Wahl der Ordnung $<$ kann in S kein Eintrag existieren, der von e' dominiert wird.

Der aus der Warteschlange entnommene Eintrag e entspricht einem pareto-optimalen s-$e.v$-Pfad und wird zur Menge S hinzugefügt. Nach Ausführung des Algorithmus enthält S einen Eintrag für jeden pareto-optimalen Pfad von s zu jedem Knoten in G.

5.2.1 Gewichtsfunktion für Routing mit Violations

Für den Algorithmus von Dijkstra wird eine Gewichtsfunktion benötigt, die jeder Kante einen Vektor mit der Distanz und den Strafkosten für Violations zuweist. Die restriktionsunabhängige Distanz ist direkt als Funktion über der Kantenmenge gegeben. Die Distanzkosten im Falle einer Violation können berechnet werden, indem der entsprechende Koeffizient mit der Länge multipliziert wird. Die Distanzkosten der Violation summieren sich dann korrekt auf.

Für die Zonen- und die Kapazitätskosten, muss jedoch sichergestellt werden, dass sie für jede Violation nur einmal berechnet werden, auch wenn die Violation aus mehreren Kanten besteht. Diese werden nur dann addiert, wenn der Knoten, von dem die Kante ausgeht außerhalb und der Zielknoten innerhalb einer verletzten Zone liegt. So werden diese Summanden der Strafkosten nur beim Betreten der Zone berechnet.

Algorithm 5.2: ParetoDijkstra

Input: Gerichteter Graph $G = (V, E \subseteq V \times V)$ mit Kantengewichtsfunktion $w : E \to \mathbb{N}$, Knoten $s \in V$

Output: Eintrag $e = (v, d, \pi)$ in S für jeden pareto-optimalen Pfad von s zu jedem Knoten $v \in V$

```
1  S = ∅
2  Q = {(s, 0, NIL)}
3  while Q ≠ ∅ do
4      e = Extract-Min(Q)
5      S = S ∪ e
6      for each vertex u ∈ G.Adj[e.v] do
7          d' = e.d + w(e.v, u)
8          if {e' ∈ Q ∪ S | e'.d ⪰ d' ∧ e'.v = u} = ∅ then
9              Q = Q \ {e' ∈ Q | e'.d ⪯ d' ∧ e'.v = u} ∪ {(u, d', e.v)}
```

5.2.2 Korrektheit

In diesem Abschnitt wird bewiesen, dass der vorgestellte Algorithmus das Problem Routing mit Violations löst. Der Beweis basiert auf der so genannten *Kürzester-Teilpfad-Eigenschaft*: Jeder Teilpfad eines pareto-optimalen Pfades ist wieder pareto-optimal bezüglich seines Start- und Endknotens. Diese Eigenschaft überträgt sich direkt aus dem eindimensionalen Fall, da die Zonen- und Kapazitätskosten bereits berechnet werden, wenn nur ein Knoten aus einer verletzten Zone verwendet wird.

Im folgenden wird per vollständiger Induktion über die Kantenanzahl l gezeigt, dass jeder Pfad mit pareto-optimaler Distanz vom Algorithmus gefunden wird.

i.A.: Es existiert genau ein Pfad mit 0 Kanten: $p = (s)$ mit Gewicht 0. Dieser wird vom Algorithmus in der ersten Iteration der while-Schleife in S eingefügt.

i.V.: Für ein beliebiges aber festes $l \in \mathbb{N}$ gelte: Jeder pareto-optimale Pfad der genau l Kanten enthält wird vom Algorithmus gefunden.

i.S.: Sei $p = (s, v_1, \ldots, v_l, t)$ ein beliebiger pareto-optimaler Pfad mit genau $(l + 1)$ Kanten. Dann ist $p' = (s, v_1, \ldots, v_l)$ ein pareto-optimaler s-v_l-Pfad mit genau l Kanten. Nach Induktionsvorraussetzung wird dieser Pfad p' vom Algorithmus gefunden. Bei der anschließenden Relaxierung der Kante (v_l, t) wird dann auch der Pfad p gefunden.

Für jeden gefundenen, pareto-optimalen Pfad wird ein entsprechender Eintrag zu S hinzugefügt, falls S noch keinen Eintrag mit den selben Kosten enthält. Ein Eintrag der einmal zu S hinzugefügt wurde, wird später nicht mehr entfernt. Weiterhin wird zu S kein Eintrag hinzugefügt, der einem nicht pareto-optimalen Pfad entspricht: Betrachtet wird ein solcher Pfad p. Es existiert dann ein Pfad p', der diesen dominiert. Da die Kostenfunktion nicht-negativ ist und da die Ordnung der Warteschlange so gewählt wurde, dass eine Vektor, der einen anderen dominiert, kleiner als dieser ist, wird der pareto-optimale Pfad p' vom Algorithmus zuerst gefunden und zu S hinzugefügt. Der nicht optimale Pfad p wird später verworfen, da in S bereits ein Eintrag existiert, der diesen dominiert.

Damit liefert der Algorithmus das korrekte Ergebnis.

5.3 Beschleunigung

Die beim Routing verwendeten Straßengraphen umfassen nicht selten mehrere Millionen Kanten. Um auch in dieser Größenordnung effizient kürzeste Routen berechnen zu können,

wurden in der Vergangenheit einige Techniken entwickelt, um den Algorithmus von Dijkstra zu beschleunigen (siehe zu Beispiel [DSSW09]). Diese nutzen zum Beispiel spezielle Struktureigenschaften von Straßengraphen oder (vom Start- und Zielpunkt unabhängige) Vorberechnungen.

Auch das multikriterielle Routing soll so effizient wie möglich erfolgen. Im Folgenden werden deshalb zwei Möglichkeiten beschrieben, wie der Basisalgorithmus beschleunigt werden kann.

5.3.1 Abbruchkriterium

Im eindimensionalen Fall kann der Algorithmus von Dijkstra abgebrochen werden, sobald der Zielknoten erreicht wurde, da dann ein kürzester Pfad gefunden wurde. Beim multikriteriellen Dijkstra ist dies nicht so einfach möglich, da der Zielknoten mehrmals besucht wird, wenn mehrere pareto-optimale Pfade zu diesem existieren.

Jedoch können die bereits bekannten Pfade ausgenutzt werden, um die Ausführung zu beschleunigen: Wird bei der Relaxierung einer Kante ein Eintrag erzeugt, der von einem Eintrag am Zielknoten dominiert wird, ist dieser sicher nicht Teil eines optimalen Pfades und muss somit nicht weiter bearbeitet werden. In Algorithmus 5.2 muss dazu die Bedingung in Zeile 8 wie folgt verändert werden:

$$\{e' \in Q \cup S | e'.d \succeq d' \wedge (e'.v = u \vee \mathbf{e}'.\mathbf{v} = \mathbf{t})\} = \emptyset$$

Enthält der kürzeste Pfad zum Zielpunkt keine Violation, führt die Verwendung des Kriteriums dazu, dass der Suchraum ein Kreis um den Startpunkt bildet. Der Radius ist dabei die Länge des kürzesten s-t-Pfades. Aufgrund dieser Ähnlichkeit zum klassischen Dijkstra, wird diese Technik als verallgemeinertes Abbruchkriterium betrachtet. Jedoch wird hierdurch nur verhindert, dass nach Finden der kürzesten Route keine neuen Knoten mehr in die Queue aufgenommen werden. Die Knoten die sich bereits in der Queue befinden werden weiter bearbeitet.

5.3.2 A*

Eine weitere Methode zu Beschleunigung, die sich hier gut Anwenden lässt ist der A*-Algorithmus. Idee des Algorithmus ist es, unter Bewahrung der Korrektheit die Kantengewichte so anzupassen, das Kanten die auf den Zielpunkt zuführen verkürzt und Kanten die vom Zielpunkt weg führen verlängert werden. Dadurch sucht der Algorithmus nicht mehr kreisförmig vom Startpunkt ausgehend sondern stärker in Richtung des Zielpunkts, was zu einer Verkleinerung des Suchraums führt.

Benötigt wird eine Abschätzung $\phi(v)$, die jedem Knoten eine untere Schranke für die Distanz zum Zielpunkt zuweist. Um die Korrektheit des Algorithmus zu bewahren, muss dabei sichergestellt werden, dass die Distanz zum Zielpunkt nie überschätzt wird.

Klassischer Weise wird als Abschätzung die Luftliniendistanz verwendet. Im Fall Routing mit Violationsmacht die Verwendung einer präziseren Abschätzung Sinn: Vor der Ausführung des A*-Algorithmus wird der Algorithmus von Dijkstra ausgehend vom Zielpunkt ausgeführt wobei Restriktionen ignoriert werden. Dadurch erhält man eine untere Schranke für jeden Knoten, die genau dann scharf ist, wenn der kürzeste Weg zum Zielpunkt keine Restriktionen verletzt. In diesem Fall ist das Gewicht aller Kanten, die Teil eines kürzesten Weges sind, gleich 0. Da das Gewicht jedes kürzesten s-t-Weges gleich 0 ist, und das Gewicht aller nicht optimale s-t-Wege größer 0, betrachtet der Algorithmus dann genau die Knoten der Lösung und terminiert somit sehr schnell.

Da in der Praxis viele Pfade ohne Violations auskommen, ist die Abschätzung in der Regel sehr gut. Aus diesem Grund wird mit einer deutlichen Beschleunigung des A*Algorithmus gerechnet. Ausschlaggebend ob sich die Methode insgesamt lohnt hängt damit im wesentlichen von der Zeit ab, die benötigt wird um die Abschätzung zu berechnen. Eine komplette Rückwärtssuche wird insbesondere bei großen Karten zu teuer sein. Alternativ wäre denkbar die Rückwärtsuche nur bis zum Erreichen des Startknotens auszuführen, oder die Iterationen der Rückwärtsuche abwechselnd mit denen des A*-Algorithmus auszuführen. Für Knoten, die von der Rückwärtssuche nicht erreicht wurden, wird dann 0 als Abschätzung für die Kosten zum Ziel verwendet.

6. Experimentelle Evaluierung

Wie in Kapitel 4 gezeigt, kann die Anzahl pareto-optimaler Routen exponentiell mit der Größe des Graphen wachsen. Damit ist die Worst-Case-Laufzeit des in Kapitel 5 vorgestellten Algorithmus mindestens exponentiell, auch bei Verwendung von Beschleunigungstechniken. In diesem Kapitel soll daher experimentell untersucht werden, wie sich die Laufzeit des Algorithmus bei realen Eingabedaten verhält.

Neben der Größe der Karte wirken sich auch andere Faktoren auf die Laufzeit des Algorithmus aus. Wie bei der Untersuchung der Lösungsmenge gezeigt, spielt auch die Anzahl der Klassen, in die die Restriktionen eingeteilt werden, eine wichtige Rolle. Daneben geht vermutlich auch der Anteil der Kanten, auf denen eine Restriktion verletzt ist, in die Laufzeit ein. Dieser hängt neben der verwendeten Karte insbesondere vom gewählten Fahrzeugprofil ab.

6.1 Testumgebung

Alle Algorithmen wurden mit C++11 implementiert. Kompiliert wurde das Programm mit dem GNU C++ Compiler 4.8.2 (-std=c++11 -O3). Ausgeführt wurden die Experimente auf einem Rechner mit einem Core i7-3770K (Ivy-Bridge) mit 4×3,5Ghz und 16GB Arbeitsspeicher (DDR3, 1333MHz) unter Ubuntu 14.10.

6.2 Testfälle

6.2.1 Karten

Die Experimente werden auf Karten von Luxemburg, der Schweiz und der Niederlande durchgeführt [1]. In Tabelle 6.1 sind die Knoten- und Kantenanzahlen sowie der Anteil Kanten mit einer Restriktion aufgeführt. Bei der Bewertung des Einflusses der Kartenanzahl muss beachtet werden, das die Karte von Luxemburg im Verhältnis fast doppelt so viele Restriktionen enthält.

Als Distanzmaß liegt allen Experimenten die mittlere Fahrzeit zu Grunde. Diese wurde aus der Länge und mittleren Geschwindigkeit der Kanten berechnet, welche in den Kartendaten enthalten waren.

[1] Alle Karten stammen von der Firma PTV Group und basieren auf Kartendaten von Nokia Here (ehemals Navteq) aus dem 4. Quartal 2013

Tabelle 6.1: Verwendete Karten

Karte	Anzahl Knoten	Anzahl Kanten	Anzahl Kanten mit Restriktion
Luxemburg	51 867	121 226	17 013 (14,0%)
Schweiz	887 673	2 080 034	163 148 (7,84%)
Niederlande	1 247 943	3 098 820	243 514 (7,86%)

Tabelle 6.2: Fahrzeugprofile

	Lieferwagen	LKW	Schwerlaster
Länge	5,5m	8m	16m
Breite	2m	2,4m	2,4m
Höhe	2m	3m	4m
Gewicht	3,4t	10t	35t
Achslast	2,2t	7t	15t
Steigung	15°	10°	5°
TRC	- (0)	- (0)	B (4)
Umweltzone	Grün (0)	Gelb (1)	Rot (2)
LKW	-	Ja	Ja
Anhänger	-	-	Ja
Gefahrgut	-	-	Ja
Brennbar	-	-	Ja
Wassergefährdend	-	Ja	Ja
Nacht	-	-	Ja

6.2.2 Fahrzeugprofile

Für die Tests wurden 3 Fahrzeugprofile verwendet, deren Parameter in Tabelle 6.2 aufgelistet sind. Die Parameter wurde dabei so gewählt, dass die verschiedenen Fahrzeuge unterschiedlich viele Restriktionen verletzen.

Der Lieferwagen hat ein Maximalgewicht von unter 3,5t. Damit gilt dieser innerhalb der EU nicht als LKW, sodass für diesen die oft in Innenstädten geltenden LKW-Verbote nicht verletzt sind. Das Profil des LKW orientiert sich an den Maßen von den Lastwägen, die in der Regel benutzt werden um Läden in Städten zu beliefern. Die Maße des Schwerlasters sind knapp unter den in der EU erlaubten Maxima für den Fernverkehr gewählt. Insgesamt wurde das Fahrzeugprofil so gewählt, das für den Schwerlaster alle in den Karten enthaltenen Restriktionen verletzt sind.

Tabelle 6.3 zeigt für jedes Fahrzeugprofile und jede Karte, bei wie vielen Kanten eine Restriktion verletzt ist. Dieser Wert ist auch für den Lieferwagen schon recht hoch. Das liegt daran, dass die Karten, wie im vorherigen Abschnitt beschrieben, viele für alle Fahrzeuge gesperrte Gebiete enthält. Diese Restriktion ist stets für alle betrachteten Fahrzeug verletzt.

Tabelle 6.3: Anzahl Kanten mit verletzter Restriktion

	Lieferwagen	LKW	Schwerlaster
Luxemburg	6 866	17 013	17 013
Niederlande	222 083	243 514	243 514
Schweiz	74 864	163 148	163 148

6.2.3 Bewertung der Violations

Sofern nicht explizit anders erwähnt, werden für die Evaluierung stets die gleiche Klasseneinteilung sowie die selben Koeffizienten für die Bewertung der Violations verwendet. Diese wurden bereits in Kapitel 4 in Tabelle 4.1 beschrieben.

6.2.4 Generierung zufälliger Anfragen

Die Algorithmen werden getestet, in dem zufällig Start- und Zielknoten gewählt und die Ergebnisse anschließend gemittelt werden. Die Knoten sollen dabei so gewählt werden, dass die Anfragen gleichmäßig über die gesamte Karte verteilt sind. Das kann durch gleichverteilt zufällige Wahl der Knoten sichergestellt werden. Zum Anderen soll jedoch auch die Lokalität der Anfragen, d.h. die Länge der resultierenden Routen, möglichst gleichverteilt sein.

Zur Bewertung der Lokalität einer Anfrage wird das *Dijkstra-Maß* verwendet. Die Reihenfolge, in der der Algorithmus die Knoten aus der Queue entnimmt und somit den kürzesten Weg gefunden hat, definiert eine Ordnung auf den Knoten. Sei $ord_s(v)$ die Nummer des Knotens bezüglich dieser Ordnung von Startknoten s. Der Dijkstra-Rank der Anfrage mit Startknoten s und Zielknoten t ist dann wie folgt definiert:

$$rank_s(t) = \lfloor \log_2 ord_s(t) \rfloor$$

Um eine gleichmäßige Verteilung des Dijkstra-Ranks, d.h. der Lokalität, der Anfragen zu erhalten, sollte der Zielknoten nicht gleichverteilt aus allen Knoten gezogen werden. Aufgrund der logarithmischen Skala des Dijkstra-Maßes würden so die Hälfte der Anfragen maximalen Dijkstra-Rank haben und es würden kaum lokale Anfragen getestet. Stattdessen wird nach der Wahl des Startknotens s der Dijkstra-Rank von allen Knoten bestimmt, wobei Restriktionen ignoriert werden. Als Zielknoten für das Routing werden dann die Knoten $t_i \in V$ mit $ord_s(t_i) = 2^i$ und $4 \leq i \leq \log_2 n$ gewählt. So wird gewährleistet, dass die Lokalität der Anfragen gleichmäßig verteilt ist.

Sofern nicht explizit anders erwähnt, wurden im Folgenden stets je 200 zufällige Anfragen zur Bestimmung jedes Mittelwerts und der Maxima ausgeführt.

6.3 Ergebnisse

6.3.1 Anzahl Routen

In Abschnitt 4.4 wurde gezeigt dass die Komplexität des Problems Routing mit Violations exponentiell mit der Größe des Graphen wächst. Grund dafür ist, das die Anzahl pareto-optimaler Routen exponentiell mit der Knotenanzahl des Graphen steigen kann. Tabelle 6.4 zeigt dagegen, dass die Routenzahlen in der Praxis weit geringer sind als theoretisch möglich: In den meisten Fällen erhält man nicht mehr als 3 Alternativen. Das liegt daran, dass bei längeren Strecken für große Abschnitte Autobahnen oder andere Schnellstraßen gewählt werden, auf denen nur in seltenen Fällen Restriktionen gelten. Dadurch steigt die Anzahl pareto-optimaler Routen kaum mit der Knotenanzahl des Graphen.

6.3.2 Laufzeit Pareto-Dijkstra

Tabelle 6.5 zeigt die mittleren (∅) und maximalen (m) Laufzeiten des ParetoDijsktra ohne Abbruchkriterium. Deutlich zu erkennen ist die höhere Laufzeit bei den größeren Karten der Schweiz und der Niederlande. Die Wahl des Fahrzeuges geht dagegen kaum in die Laufzeit ein. Weiterhin lässt sich beobachten, dass die Varianz der Laufzeiten sehr hoch sind: Bei allen Konfigurationen betragen die Maximalwerte ein Vielfaches der durchschnittlichen Laufzeiten.

Tabelle 6.4: Durchschnittliche Routenanzahl bei verschiedenen Karten und Fahrzeugen

	Lieferwagen	LKW	Schwerlaster
Luxemburg	2,04	2,58	3,24
Schweiz	2,38	2,47	2,29
Niederlande	2,11	2,14	3,19

Tabelle 6.5: Durchschnittliche Laufzeiten ParetoDijkstra ohne Abbruchkriterium

	Lieferwagen	LKW	Schwerlaster
Luxemburg	144ms (⌀)	240ms (⌀)	314ms (⌀)
	534ms (m)	425ms (m)	1327ms (m)
Niederlande	5867ms (⌀)	7825ms (⌀)	4819ms (⌀)
	13737ms (m)	13699ms (m)	9446ms (m)
Schweiz	3257ms (⌀)	5721ms (⌀)	3651ms (⌀)
	6674ms (m)	20514ms (m)	5758ms (m)

6.3.3 Beschleunigung mit Abbruchkriterium

In Abschnitt 5.3.1 wurde eine Verallgemeinerung des Abbruchkriteriums für den multikriteriellen Dijsktra vorgestellt. Wie Tabelle 6.6 zeigt, ist die mittlere Laufzeit des Algorithmus bei Verwendung des Abbruchkriteriums im Schnitt um etwa zwei Drittel geringer. Die maximalen Laufzeiten haben sich in manchen Fällen sogar verschlechtert. Letzteres Ergebnis sollte aufgrund der hohen Varianz jedoch vorsichtig interpretiert werden.

Tabelle 6.7 zeigt die Beschleunigung durch das Abbruchkriterium in Abhängigkeit vom Dijkstra-Rank der Anfrage. Wie aus den Messergebnissen hervorgeht, ist die Beschleunigung bei lokalen Anfragen nicht wesentlich höher als bei langen Routen. Das liegt daran, dass falls keine Route ohne Violation existiert, in jedem Fall zunächst der komplette Bereich der Karte abgesucht wird, der ohne Violations erreichbar ist.

6.3.4 Beschleunigung mit A*

Wie die Ergebnisse in Tabelle 6.8 zeigen, lässt sich der Algorithmus durch die Anpassung der Kantengewichte beim A* nicht beschleunigen. In vielen Fällen wird ist die durchschnittliche sogar schlechter als beim ParetoDijkstra. Problem ist die Berechnung der Abschätzung, für die ein kompletter Dijkstra vom Zielknoten ausgeführt wird, was insgesamt zu teuer ist.

Bessere Ergebnisse erhält man jedoch, wenn man den A*-Algorithmus mit dem Abbruchkriterium kombiniert, wie die Ergebnisse in Tabelle 6.9 zeigen. Die Ergebnisse sind sogar etwas besser, als bei der Verwendung des Abbruchkriteriums mit dem ParetoDijkstra. Besonders fällt auf, dass die Maximallaufzeiten deutlich verringert wurden.

Tabelle 6.6: Durchschnittliche Laufzeiten und Beschleunigung durch Abbruchkriterium

	Lieferwagen	LKW	Schwerlaster
Luxemburg	43ms, 30,5% (⌀)	85ms, 35,9% (⌀)	111ms, 35,7% (⌀)
	238ms, 44,7% (m)	30,2ms, 71,6% (m)	934ms, 70,6% (m)
Schweiz	1496ms, 46,6% (⌀)	3606ms, 63,5% (⌀)	2890ms, 80,1% (⌀)
	9565ms, 144,3% (m)	54779ms, 267,8% (m)	12663ms, 221,7% (m)
Niederlande	1222ms, 21,1% (⌀)	2561ms, 33,0% (⌀)	2431ms, 51,1% (⌀)
	5722ms, 41,9% (m)	30798ms, 226,0% (m)	18037ms, 192,2% (m)

Tabelle 6.7: Beschleunigung des Algorithmus durch das Abbruchkriterium in Abhängigkeit vom Dijkstra-Rank der Anfrage

Dijkstra-Rank	4	5	6	7	8	9
Laufzeit	1585ms	1571ms	1588ms	1573ms	1584ms	1602ms
	47,2%	46,8%	47,3%	46,9%	47,3%	47,8%
Dijksta-Rank	10	11	12	13	14	15
Laufzeit	1604ms	1588ms	1546ms	1592ms	1746ms	1531ms
	47,8%	47,3%	46,7%	48,1%	52,8%	46,2%
Dijkstra-Rank	16	17	18	19	20	
Laufzeit	2407ms	2047ms	2850ms	2326ms	5103ms	
	44,8%	38,1%	53,1%	43,4%	79,1%	

Tabelle 6.8: Durchschnittliche Laufzeiten und Beschleunigung durch A*

	Lieferwagen	LKW	Schwerlaster
Luxemburg	134ms, 95,0% (⌀)	276ms, 116,5% (⌀)	371ms, 119,3% (⌀)
	384ms, 72,2% (m)	612ms, 145,0% (m)	1482ms, 112,0% (m)
Schweiz	4677ms, 145,6% (⌀)	4276ms, 75,3% (⌀)	7537ms, 209,0% (⌀)
	16356ms, 246,8% (m)	7476ms, 36,5% (m)	17835ms, 312,2% (m)
Niederlande	5116ms, 88,2% (⌀)	6895ms, 88,9% (⌀)	5774ms, 121,4% (⌀)
	13383ms, 97,9% (m)	11707ms, 85,9% (m)	12757ms, 136,0% (m)

Tabelle 6.9: Durchschnittliche Laufzeiten und Beschleunigung durch A* und Abbruchkriterium

	Lieferwagen	LKW	Schwerlaster
Luxemburg	45ms, 31,3% (⌀)	50ms, 20,8% (⌀)	60ms, 19,1% (⌀)
	54ms, 10,1% (m)	101ms, 23,8% (m)	256ms, 19,3% (m)
Schweiz	873ms, 26,8% (⌀)	876ms, 15,3% (⌀)	872ms, 23,9% (⌀)
	912ms, 13,7% (m)	1196ms, 5,8% (m)	920ms, 16,0% (m)
Niederlande	1383ms, 23,6% (⌀)	1302ms, 16,6% (⌀)	1619ms, 33,6% (⌀)
	1454ms, 10,6% (m)	1543ms, 11,3% (m)	4355ms, 46,1% (m)

7. Ausblick

Im Rahmen dieser Arbeit wurde das Problem Routing mit Violations betrachtet. In Kapitel 4 wurde eine Formalisierung des Problems und in Kapitel 5 Algorithmen zur Lösung vorgestellt. In Kapitel 6 wurden die Laufzeiten der Algorithmen bei realen Eingabedaten experimentell ermittelt.

Da die Laufzeiten der Algorithmen, insbesondere bei größeren Karten, oft noch im Sekundenbereich liegen, ist eine Verwendung dieser in der Praxis noch nicht möglich. Insbesondere wenn die Routenberechnung als Teil eines komplexeren Planungsproblems sehr oft aufgerufen wird.

Weiter beschleunigen lässt sich der Algorithmus zum Beispiel in den Fällen, in denen keine Route ohne Violations existiert. In dem Fall wird nämlich zunächst der gesamte Bereich der Karte abgesucht, der ohne Verletzung von Restriktionen erreichbar ist, bevor eine Kante mit Strafkosten aus der Queue entnommen wird. Vor allem bei großen Karten wird die Routenberechnung so deutlich verlangsamt.

In den meisten Fällen existiert keine gültige Route, weil der Start- oder Zielknoten in einer verletzten Zone, oft zum Beispiel eine für LKW gesperrte Innenstadt. Liegt der Startknoten in einer solchen Zone tritt das Problem nicht auf, das der ohne Violations erreichbare Bereich sehr klein ist. Ob der Zielknoten in einer Zone liegt, könnte mit einer Rückwärtssuche schnell geprüft werden. Routet man in dem Fall vom Zielknoten zum Rand der umgebenden Zone, muss der Algorithmus nur solange ausgeführt werden, bis alle Randknoten erreicht wurden. So könnte in diesen Fällen ein Absuchen der kompletten Karte vermieden werden.

Letztendlich wird aber auch die Beschleunigung durch diese Methode vermutlich zu gering sein, um die in der Praxis gewünschten Laufzeiten von einigen Millisekunden zu erreichen. Um Laufzeiten in diesem Bereich zu erhalten, müssen effizientere Methoden, wie zum Beispiel Contraction Hierarchies, auf den multikriteriellen Fall angepasst werden.

Literaturverzeichnis

[BDG+14] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner und Renato F. Werneck: Route Planning in Transportation Networks. Technischer Bericht MSR-TR-2014-4, Microsoft Research, 2014. `http://research.microsoft.com/apps/pubs/?id=207102`.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: Introduction to Algorithms. MIT Press, 2nd Auflage, 2001.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes und Dorothea Wagner: Engineering Route Planning Algorithms. In: Jürgen Lerner, Dorothea Wagner und Katharina A. Zweig (Herausgeber): Algorithmics of Large and Complex Networks, Band 5515 der Reihe Lecture Notes in Computer Science, Seiten 117–139. Springer, 2009.

[DSW14] Julian Dibbelt, Ben Strasser und Dorothea Wagner: Customizable Contraction Hierarchies . Technischer Bericht, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2014. `http://arxiv.org/abs/1402.0402`, Technical Report on arXiv.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes und Daniel Delling: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: Catherine C. McGeoch (Herausgeber): Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), Band 5038 der Reihe Lecture Notes in Computer Science, Seiten 319–333. Springer, June 2008.