technische universität
dortmund

Green Logistics – Optimierung von multimodalen Logistiknetzwerken

Matthias Woste

Algorithm Engineering Report
TR10-1-004
Mai 2010
ISSN 1864-4503

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

technische universität
dortmund

Diplomarbeit

Green Logistics - Optimierung von multimodalen Logistiknetzwerken

Matthias Woste

3. Dezember 2009

Betreuer:
Frau Prof. Dr. Petra Mutzel
Herr Dr.-Ing. Giovanni Prestifilippo

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren ist ein deutlich gestiegenes Interesse für Umwelteinflüsse zu verzeichnen. Dies beschränkt sich nicht nur auf die Gesellschaft. Auch Unternehmen überprüfen ihren Einfluss auf die Umwelt und analysieren ihre Prozesse in Bezug auf Umweltverträglichkeit und Nachhaltigkeit. Am Beispiel des Kyoto-Protokolls lässt sich sogar erkennen, dass auch Staaten daran interessiert sind, die Schädigung des Planeten zu bekämpfen.

Technologien, die umweltschonender sind, haben jedoch meist höhere Investitionskosten. Hierbei wird oft übersehen, dass diese höheren Investitionskosten sich auf lange Sicht auszahlen werden. Aufgrund der gestiegenen Sensibilität der Gesellschaft gegenüber der Umwelt können Unternehmen durch nachhaltige Prozesse unter Umständen ihre Reputation steigern, was sich wiederum positiv auf die Kundenentwicklung auswirkt.

Die Informatik kann Unternehmen dabei unterstützen, umweltfreundlicher zu werden. Viele Probleme der Logistik sind eng mit Problemstellungen der Informatik verknüpft. Beispiele hierfür sind Standortoptimierung, Tourenplanung, Arc-Routing oder Flussprobleme. Diese Fragestellungen sind in ihrer ursprünglichen Version schon lange im Einsatz. Als Resultat werden zumeist ökonomische Kosten reduziert, indem Prozesse effektiver gestaltet werden. Bei der Tourenplanung z. B. resultiert dies meist in kürzeren Wegen, die ein LKW fahren muss. Dies führt bereits zu einer Reduzierung der Schadstoffausstoße. Diese Optimierungen sind jedoch nur der Anfang. Um beim Beispiel Tourenplanung zu bleiben, muss unter anderem darauf geachtet werden, dass LKW optimal ausgelastet und bepackt sind, dass die Flotte möglichst neue, umweltfreundliche Motoren besitzt und dass die Ergebnisse der Optimierung auch umgesetzt werden. Eine wichtige Variante der Tourenplanung ist die dynamische Ausrichtung der Fragestellung. Hierbei wird versucht, die Komponente Fahrtzeit auf einer Straße in Abhängigkeit der Tageszeit zu berücksichtigen. Hintergrund dieser Erweiterung sind Staus. Betrachtet man die mittlere Fahrtzeit, die benötigt wird, um eine Straße zu in der Rush-Hour oder um Mitternacht zu befahren, so können sich

große Unterschiede ergeben. Die Vermeidung von Staus bringt zum einen eine genauere Zeitplanung der Transporte, aber auch eine weitere Verringerung der Schadstoffausstoße mit sich, da ein LKW in einem Stau Schadstoffe produziert obwohl er nicht fährt.

In dieser Diplomarbeit wird ein anderer Ansatz gewählt, um den Schadstoffausstoß zu reduzieren. Mittels multimodalen Transporten sollen Sendungen transportiert werden, dass für einen Transport mehrere Verkehrsträger zum Einsatz kommen. Durch eine geeignete Auswahl von Verkehrsmitteln und eine entsprechende Aufteilung des Transportweges ist es möglich Schadstoffe einzusparen.

1.2 Verwandte Literatur

Zu dem in dieser Arbeit behandeltem Themenkomplex gibt es eine Vielzahl von verschiedenen Arbeiten. Sie lassen sind grob in zwei Kategorien aufteilen: Arbeiten, die sich generell mit dem Thema Transportlogistik beschäftigen und Artikel, die verschiedene Ansätze algorithmischer Art für praktische Anwendungen vorstellen.

1.2.1 Transportlogistik

Wie bereits in der Einleitung erwähnt, stellt die Informatik ein wichtiges Werkzeug dar, wenn es um die Lösung von transportlogistischen Problemstellungen geht. Crainic und Laporte haben sich mit derartigen Problemen beschäftigt und stellen in [23] eine Reihe von Methoden vor, mit denen sie gelöst werden können. Dabei unterscheiden sie Probleme nach ihrem jeweiligen Planungshorizont: kurz-, mittel- oder langfristig. Einen ähnlichen Ansatz verfolgen auch Macharis and Bontekoning [49]. Ihre Zusammenfassung von Modellierungsproblemen fokussiert sich auf multimodale Transporte. Neben der zeitlichen Einteilung unterscheiden sie außerdem nach Akteuren wie z. B. Leitern von Terminals, Disponenten oder Netzwerkplanern. Beide Artikel sind sich jedoch einig, dass auf dem Gebiet der multimodalen Transportprobleme weiter geforscht werden müsse. Dies liegt an der Vielzahl von ineinander greifenden Prozessen, wie z. B. der Transport an sich, der Umschlag oder die Pufferung von Containern.

1.2.2 Anwendungen

Für die Planung von multimodalen Transporten existieren mehrere Herangehensweisen. Für die Bestimmung eines kosten-minimalen Routings einer Sendung schlagen Barnhart und Ratliff [16] einen kürzesten Wege Ansatz für den Fall, dass die Kosten per Container ermittelt werden und einen Matching Ansatz für den Fall, dass die Kosten per Zugwagon gelten, vor. Min [51] hingegen konstruiert ein *chance constrained goal programming* Modell um in einem intermodalen Transportnetzwerk eine günstigste Auswahl an Routen in Bezug auf verschiedene Kostenfunktionen zu finden. Einen ähnlichen Ansatz verfolgen

Boardman et al. [18]. Sie schlagen ein Entscheidungsunterstützungssystem vor, dass auf der Grundlage von k kürzesten Wege Algorithmen arbeitet. Ein ähnliches System wurde von Tadashi [70] auf der Grundlage von genetischer lokaler Suche entwickelt. Ziliaskopoulos und Wardell präsentieren in [72] eine Methode um einen optimalen intermodalen Pfad in einem Netzwerk zu finden, wenn sowohl Fahrtzeiten als auch Transferzeiten dynamisch sind. Diese Methodik lässt sich sowohl auf Gütertransporte als auch auf Personenverkehre anwenden. Die Arbeiten von Chang [20] und Moccia et al. [52] sind dieser Diplomarbeit thematisch am ähnlichsten. Sie berechnen Routen in multimodalen Transportnetzwerken für mehrere Sendungen gleichzeitig. Dabei unterstützen beide Zeitfenster und abschnittsweise definierte Kostenfunktionen. Während Chang jedoch eine Lagrange Relaxion zur Lösung des Problems verwendet, lösen Moccia et al. das Problem durch einen Column Generation Ansatz. Ein weiterer Unterschied besteht darin, dass Chang eine Aufspaltung der einzelnen Sendungen erlaubt, während Moccia et al dies explizit verbietet.

1.3 Problembeschreibung

In dieser Diplomarbeit wird eine Flussproblemvariante betrachtet. Hierbei geht es darum, eine Menge an Sendungen, die jeweils einen Start- und Zielort sowie eine bestimmte Warenmenge beinhalten, so durch ein Transportnetzwerk zu schicken, das möglichst geringe CO_2-Emissionen ausgestoßen werden. Sollen Schadstoffemissionen gesenkt werden, so kann entweder versucht werden durch obige Methoden, wie z. B. Erneuerung des Fuhrparks, Erfolge zu erzielen. Ein anderer Ansatz, der in dieser Arbeit untersucht wird, ist die Benutzung mehrerer Verkehrsträger. Eisenbahnen sind Massengut taugliche Transportmittel. Dies bedeutet, dass der Energieaufwand des Transports einer großen Menge an Gütern über eine ausreichend lange Strecke im Vergleich zu anderen Verkehrsmitteln wesentlich geringer ausfällt. Die Idee ist nun, einen Transport so in einen Vor-, Haupt- und Nachlauf aufzuteilen, dass in jedem Transportabschnitt das Verkehrsmittel gewählt wird, dessen Eigenschaften am besten zu den entsprechenden Anforderung passen. Optimal ist eine Sammlung bzw. Verteilung der Güter im Vor- und Nachlauf durch den flexiblen Einsatz von LKW, während im Hauptlauf die Güter gebündelt über eine längere Strecke effizient, zum Beispiel per Güterzug, transportiert werden.

Abbildung 1.1 veranschaulicht vereinfacht die grundlegende Struktur der Transporte nach der Optimierung. Findet ein Transport unter Benutzung mehrerer Verkehrsträger statt, so spricht man auch von *kombiniertem*, *intermodalem* oder *multimodalem* Transport.

Um dieses Problem zu lösen existieren verschiedenste Ansätze. In dieser Diplomarbeit wird ein deterministischer Approximationsalgorithmus zum Einsatz kommen, der ein *minimum cost multicommodity flow* Problem löst. Die Vorteile dieses Algorithmus sind seine theoretisch gute Laufzeit und eine einfache Einbindung von mehreren und/oder sendungsspezifischen Kostenfunktionen. Dies ist besonders für transportlogistische Fragestellungen

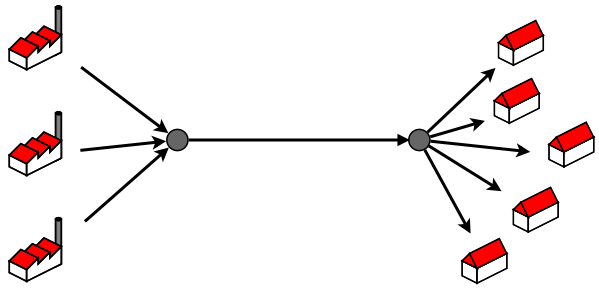

Abbildung 1.1: Elemente eines multimodalen Transports. Sendungen werden von den Fabriken eingesammelt, gebündelt über eine große Distanz transportiert und schließlich an die Endkunden verteilt.

interessant. Denkbar ist z. B. eine unterschiedliche Kostenstruktur für verschiedene Warengruppen sowie die multikriterielle Optimierung in Bezug auf Schadstoffemissionen und ökonomische Kosten.

1.4 Aufbau der Diplomarbeit

Nachdem in diesem Kapitel das zu bearbeitende Problem vorgestellt wurde und eine Übersicht über verwandte Literatur gegeben wurde, stellt Kapitel 2 die in dieser Arbeit zugrunde liegenden Themenbereiche vor. Hierzu gehören elementare Begriffe der Logistik, insbesondere der Transportlogistik, eine kurze Einführung in Flussalgorithmen mit besonderem Augenmerk für *multicommodity* Flussalgorithmen sowie einen Überblick über luftgetragene Schadstoffe und ihre Auswirkungen auf die Umwelt. In diesem Kapitel werden zudem Entscheidungen bezüglich der Algorithmenauswahl getroffen.

Nachdem alle Grundlagen und Begriffe vorgestellt worden sind, wird in Kapitel 3 ein Modell entwickelt mit dem es möglich ist multimodale Transporte auf der Straße und Schiene abbilden zu können. Es werden alle Elemente des Modell sowie die Konstruktion umfangreich vorgestellt. Im zweiten Teil des Kapitels werden die Berechnungsvorschriften für CO_2-Emissionen und ökonomische Kosten für Straßen- und Schienengüterverkehre hergeleitet bzw. eingeführt.

Das Kapitel 4 stellt das in dieser Arbeit verwendete Algorithmenframework vor und beschreibt und analysiert detailliert den verwendeten Algorithmus. Darüber hinaus werden Anmerkungen zur Implementierung gegeben. Da ein kürzester-Wege-Algorithmus die Kernkomponente des verwendeten *multicommodity* Algorithmus darstellt, werden zum Abschluss dieses Kapitels verschiedene *single source shortest path* Algorithmen vorgestellt.

Die Evaluation findet in Kapitel 5 statt. Zunächst werden die untersuchten Testinstanzen beschrieben und Informationen zur Implementierung und Architektur des Testsystems gegeben. Anschließend werden sowohl die kürzesten Wege Algorithmen als auch verschiede-

ne *multicommodity* Algorithmen des Frameworks auf verschiedenen Graphklassen getestet. Der Fokus liegt hierbei auf der Überprüfung der theoretischen Laufzeit sowie auf der Darstellung algorithmenspezifischer Eigenschaften. Nach einem vergleichendem Test mit einem anderen *multicommodity* Algorithmus werden die Ergebnisse der Transportnetzwerkinstanzen in Bezug auf die zentrale Fragestellung aus Kapitel 1 analysiert.

Die Zusammenfassung der Ergebnisse sowie ein Ausblick auf weiterführende Forschungsthemen wird in Kapitel 6 gegeben. Im Anhang A finden sich die Resultate der durchgeführten Tests aus Kapitel 5 in tabellarischer Form.

Kapitel 2

Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Zusammenhänge erläutert. Dabei wird sowohl auf logistische als auch auf informatische Aspekte eingegangen, die in dieser Arbeit Anwendung finden. Im Bereich der Logistik liegt der Fokus auf der Transportlogistik. Es werden Strukturen und Elemente, wie zum Beispiel verschiedene Transportnetzwerke oder Güterverkehrszentren vorgestellt. Als Grundlage für diese Ausführungen dienen die Bücher von Arnold et al. [14], Gudehus [39] und Jünemann [42]. Darüber hinaus wird auf den Begriff „Green Logistics“ näher eingegangen. Im Zuge dessen gibt es auch eine Einführung in die Schadstoffe, die im Straßenverkehr auftreten.

Der zweite Teil dieses Kapitels ist dem Thema Flussprobleme gewidmet. Hierbei werden zunächst verschiedene Problemvarianten vorgestellt. Schließlich folgt eine Übersicht über werden verschiedene Algorithmen, die geeignet sind, das in Kapitel 1 beschriebene Problem zu lösen.

2.1 Logistische Grundlagen

Eine allgemeine Definition der Logistik wird durch Arnold et al. [14] gegeben: „Logistik bedeutet die Gestaltung logistischer Systeme sowie die Steuerung der darin ablaufenden logistischen Prozesse“. Dabei umfasst der Begriff der logistischen Prozesse alle diejenigen Prozesse, die den Transport, die Lagerung sowie deren Unterprozesse betreffen. Dazu gehören insbesondere die Be- und Entladung, die Ein- und Auslagerung und die Kommissionierung. Ein logistisches System wiederum ist für die Ausführung von Prozessen verantwortlich. Ein solches System hat eine Netzwerkstruktur. Nimmt man den Fall eines Transportnetzwerkes, so würden die Knoten Standorte und die Kanten Verbindungen zwischen diesen darstellen. Der Fluss in einem solchen Netzwerk würde dann Verkehr zwischen den Standorten repräsentieren.

Diese kurze Definition umfasst jedoch nicht alle Facetten der Logistik. Vor allem die Betrachtung von Informationen ist ein sehr wichtiges Merkmal. Zu jedem logistischen System

gehört ein Informations- und Kommunikationssystem welches eine wichtige Voraussetzung für die Prozesssteuerung ist. Da stets eine Vielzahl unterschiedlicher Prozesse gleichzeitig betrachtet werden, muss ein ganzheitlicher Denkansatz verfolgt werden und Wechselwirkungen erkannt und im besten Fall ausgenutzt werden. Hierzu zählen auch interdisziplinäre Herausforderungen.

Die Aufgaben, die aus der obigen Definition hervorgehen, sind Planungsaufgaben. Sie dienen der Ermittlung und Festlegung zukünftiger Aktivitäten, die der Zielsetzung eines Unternehmens dienen. Da meist mehrere Prozesse ineinander verschachtelt sind oder aufeinander aufbauen, umfasst die Planung auch gewisse koordinative Aspekte. Generell können Planungsaufgaben in zwei Kategorien aufgeteilt werden. Die erste ist der zeitliche Rahmen der Aufgabe. Hier wird zwischen langfristig/strategisch, mittelfristig/taktisch und kurzfristig/operativ unterschieden. Die zweite Kategorie beschreibt die Prozesszugehörigkeit einer Aufgabe, z. B. Beschaffung, Distribution, etc.. Betrachtet man beides zusammen, so entsteht die so genannte Planungsmatrix, dargestellt in Abbildung 2.1.

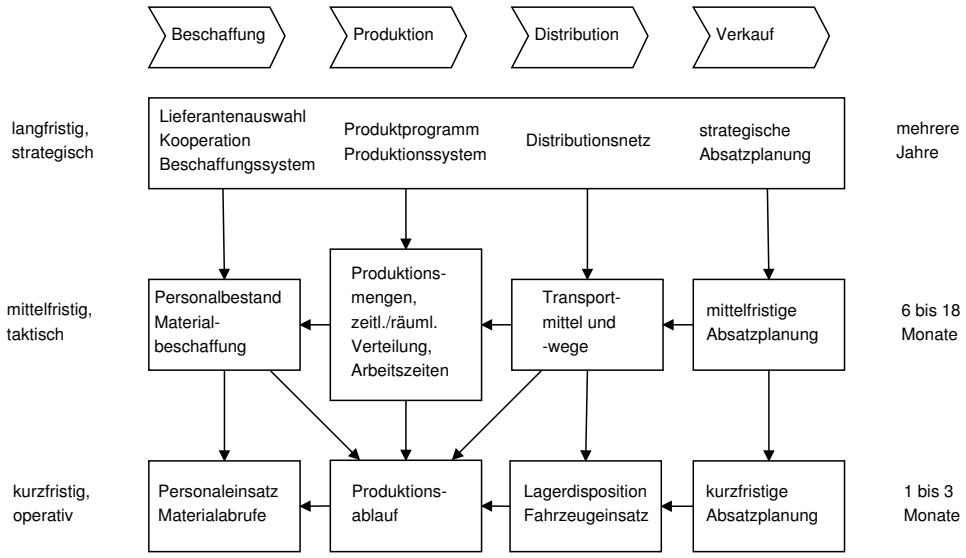

Abbildung 2.1: Logistische Planungsmatrix nach Arnold et al. [14].

Nach dieser Schematik fällt das in dieser Diplomarbeit betrachtete Problem sowohl in die strategische als auch in die mittelfristige Planung der Distributionsprozesse. So können bereits bei der Planung der Standorte mögliche Transportwege ermittelt werden. Bei bereits bestehenden Netzen ist eine Analyse der bereits vorhandenen Transportwege möglich. Als Einsatzgebiet für das in dieser Arbeit entwickelte Modell eignen sich besonders Zulieferund Distributionsnetze.

2.1.1 Transportnetze

Zuliefernetz

Ein Zuliefernetz dient der laufenden Versorgung eines Produktionsbetriebes. Hierbei ist eine zuverlässige Lieferung ausschlaggebend für einen reibungslosen Ablauf der Produktion. Es existieren eine Vielzahl verschiedener Ausprägungen eines solchen Netzes. Sie unterscheiden sich in der Häufigkeit der zu liefernden Waren sowie in den angewendeten Transportkonzepten. So können zyklische Lieferungen im Abstand von Tagen oder Wochen angeliefert werden und müssen beim Produzenten gelagert werden. Erfolgt die Lieferung tagesgenau, kann meist auf eine Lagerung verzichtet werden. Statt dessen wird das ankommende Material über einen kurzen Zeitraum gepuffert. Die stundengenaue Anlieferung wird auch *Just in Time (JIT)* genannt. Hierbei werden die gelieferten Waren direkt an den Verbrauchsort im Werk geliefert. Falls die angelieferten Teile bereits in der Reihenfolge vorsortiert sind, in der sie am Band benötigt werden, spricht man auch von *Just in Sequence (JIS)*. Dieses Konzept ist insbesondere in der Automobilindustrie weit verbreitet.

Hinsichtlich der Transportkonzepte existieren zwei Ausprägungen, die für das betrachtete Problem aus Kapitel 1 interessant sind. Bei einem direkten Transport werden Güter entweder durch den Lieferanten direkt geliefert oder durch den Abnehmer abgeholt. Beim Gebietsspediteur-Konzept werden Lieferungen von vielen Lieferanten eingesammelt, an bestimmten Stellen gebündelt und dann direkt zum Werk gebracht. Für den gebündelten Transport wird bei entsprechender Eignung auch die Bahn eingesetzt.

Distributionsnetz

Die Struktur eines Distributionsnetzes unterscheidet sich grundlegend von der eines Zuliefernetzes. Ziel ist es, Waren von einer geringen Anzahl an Werken zu einer sehr viel größeren Anzahl an Kunden zu liefern. Je nach Einzelfall können dies zwischen 500 und 5000 Kunden sein. Diese Form des Netzes spielt vor allem bei Konsumgüterherstellern eine große Rolle. Typischerweise sind die zu transportierenden Mengen eher klein und die Lieferfristen kurz. In den letzten 30 Jahren hat sich das Konzept der Zentralläger durchgesetzt [14]. Diese werden an strategisch günstig gelegenen Orten errichtet, um von dort aus die Kunden zu beliefern.

Abbildung 2.2 stellt ein solches Netz exemplarisch dar, welches dem in Kapitel 1 vorgestellten abstrakten Beispiel sehr ähnelt. In diesem konkreteren Fall werden zunächst Güter aus verschiedenen Werken in einem Zentrallager konsolidiert. Anschließend werden die Waren gebündelt und über eine große Distanz zu so genannten Transshipmentpunkten transportiert. Schließlich werden die Güter am Transshipmentpunkt in einem Radius von ca. 100 km zu den Kunden transportiert.

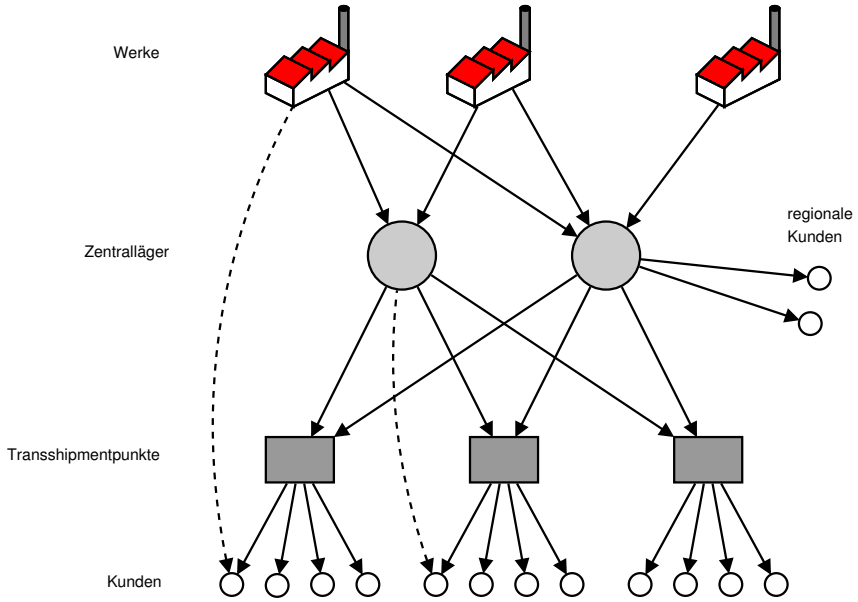

Abbildung 2.2: Struktur eines Distributionsnetzes. Gestrichelte Kanten stehen für große Sendungen, die direkt zum Kunden gefahren werden.

2.1.2 Green Logistics

Unter dem Stichwort *Green Logistics* werden Maßnahmen zusammengefasst, die das Ziel haben, Prozesse nachhaltiger und damit umweltverträglicher zu gestalten [61]. Dies steht nicht zwangsläufig im Gegensatz zum bisherigen Vorgehen, ausschließlich die ökonomischen Kosten minimieren zu wollen. In einigen Fällen können beide Absichten Hand in Hand gehen. Ein Beispiel hierfür ist die Verpackungsoptimierung. Hierbei wird weniger Verpackungsmaterial verbraucht, was zur Konsequenz hat, dass das Verpackungsvolumen sinkt und so die Transportkapazitäten besser ausgenutzt werden können. Unabhängig davon wird auf jeden Fall weniger Verpackungsmaterial produziert werden müssen.

Ein anderes Beispiel sind Förderbänder, die, anstatt im Dauerbetrieb eingesetzt zu sein, angehalten werden, wenn sie nicht benötigt werden. Dadurch werden sowohl der Verschleiß als auch die Betriebskosten bzw. der Energieverbrauch gesenkt. Auch kombinatorische Optimierungsalgorithmen können dazu beitragen, die Umweltbelastungen zu reduzieren [61]. Eine Klasse dieser Algorithmen sind dynamische Graphalgorithmen, die Transporte nicht nur auf dem kürzesten Weg zum Ziel führen, sondern auch in der schnellsten Zeit. Der dynamische Aspekt findet sich in der Betrachtung von Staus wieder. Befindet sich ein LKW in einem Stau, so werden Emissionen verursacht, obwohl er sich nur langsam fortbewegt. Durch die Anwendung von dynamischen Algorithmen wird sowohl die Fahrtzeit als auch die Standzeit und somit der Schadstoffausstoß reduziert.

Im Rahmen der vorliegenden Arbeit werden durch die Berechnung eines geeigneten *multicommodity minimum cost flow* Problems die Schadstoffemissionen eines Transport-

netzwerks optimiert. Ziel ist es, Transporte von der Straße auf die Schiene zu verlagern. Aufgrund des wesentlich geringeren Schadstoffausstoßes von ausreichend ausgelasteten Güterzügen bedeutet dies eine erhebliche Verminderung der Transportemissionen.

Die Betrachtung von Straßengüterverkehren mit Blick auf Schadstoffemissionen ist besonders wichtig, da im Straßenverkehr sowohl Emissionen bei der Herstellung des Kraftstoffes, als auch bei der Verbrennung durch den Motor ausgestoßen werden. Hinzu kommt, dass gerade der Straßengüterverkehr überproportional zunimmt. Hierzu stellt das Bundesministerium für Verkehr, Bau und Stadtentwicklung jährlich Statistiken bereit [65].

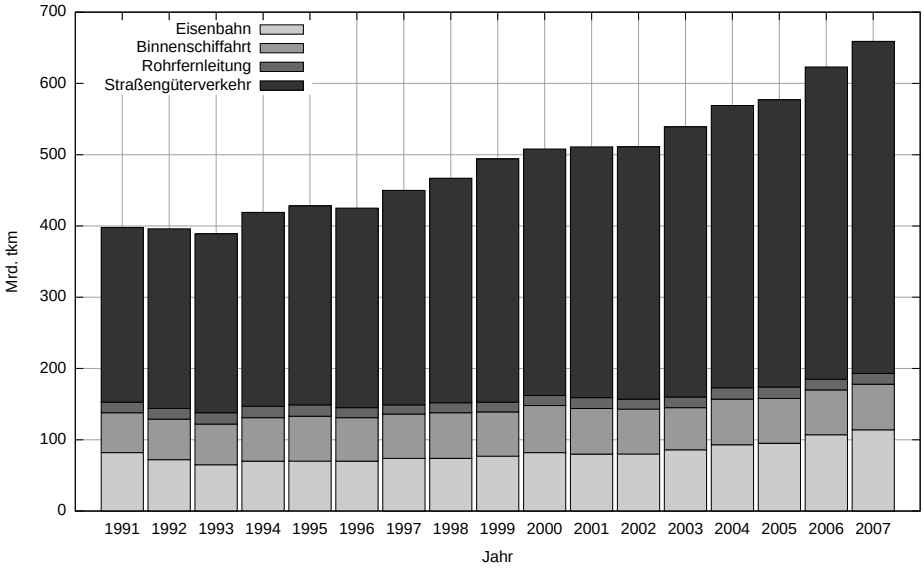

Abbildung 2.3: Modal Split des Verkehrsaufkommens im Güterverkehr in absoluten Zahlen. (Quelle: Verkehr in Zahlen 2008/09 [65]).

Wie die Abbildungen 2.3 und 2.4 zeigen, hat sich das Straßenverkehrsgüteraufkommen von 1991 bis 2007 fast verdoppelt. Dahingegen konnte der Eisenbahngüterverkehr sein Aufkommen nur um ca. 39%, die Binnenschifffahrt nur um 16% steigern. Dies hat mehrere Gründe: Der oben erwähnte Trend von einer dezentral organisierten Lagerhaltung hin zu einer zentralen, mag Vorteile in Bezug auf die durch der Lagerung entstehenden Kosten nach sich ziehen. Ein Nachteil ist jedoch die Tatsache, dass durch eine Zentralisierung die Verkehre stark zunehmen, da sich die Transportwege verlängern. Außerdem erfolgt durch die stetige Globalisierung und die somit einhergehende Vergrößerung des Absatzmarktes eine generelle Zunahme der Verkehre. Hinzu kommen außerdem die verkehrsträgerspezifischen Eigenschaften eines LKW: Dieser ist wesentlich flexibler einsetzbar und oft kostengünstiger als Eisenbahn oder Binnenschiff und wird damit häufig als Transportmittel eingesetzt. Dies wirkt sich jedoch negativ auf die Umwelt aus, da der Transport auf der Straße wesentlich mehr Schadstoffe pro transportiertem Gut freisetzt, als die alternativen, Massengut fähigen

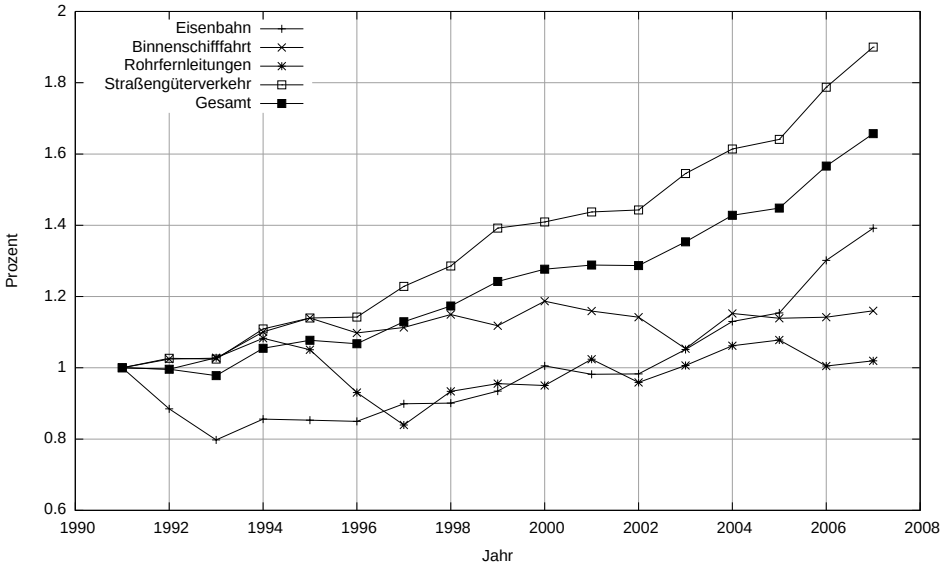

Abbildung 2.4: Prozentuale Steigerung des Verkehrsaufkommens im Güterverkehr. Basiswert 1991 entspricht 100%. (Quelle: Verkehr in Zahlen 2008/09 [65]).

Transportmittel Eisenbahn und Binnenschiff. Diese Entwicklung erklärt den Anstieg der Gesamtemissionen des Straßengüterverkehrs seit 1991 um ca. 38%. Allerdings konnte der Dieselkraftstoffverbrauch und damit auch die Emissionen pro Verkehrsaufwand (Tonnenkilometer) deutlich gesenkt werden. Seit 1991 ist der spezifische Kraftstoffverbrauch um ca. 28% gefallen [65]. Dies begründet sich in den verbesserten Motoren sowie den Regularien bezüglich des zulässigen Schadstoffausstoßes. Beide Verläufe werden in Abbildung 2.5 dargestellt.

Die gezielte Verlagerung der Transporte weg von der Straße bringt sowohl eine Verminderung der Umweltbelastung mit sich, als auch eine mögliche Ausnutzung von verkehrsträgerspezifischen Kostenvorteilen.

2.1.3 Verkehrslogistik

Straßengüterverkehr

Ein modernes Straßennetz stellt die Grundlage für den Straßengüterverkehr dar. Der Ausbau und die Instandhaltung dieser Infrastruktur wird in Deutschland maßgeblich von Bund, Ländern und Kommunen betrieben. Im Straßenfernverkehr werden überwiegend Bundesautobahnen benutzt. Aufgrund der Lage Deutschlands in Europa nehmen sie auch international eine zentrale Rolle ein.

LKW sind im Straßengüterverkehr das überwiegend zum Einsatz kommende Verkehrsmittel. Es gibt sie in unzähligen Varianten. Eine Gliederung ist anhand verschiedener Eigenschaften, wie z. B. zulässiges Gesamtgewicht, Abmessungen oder Aufbauten möglich.

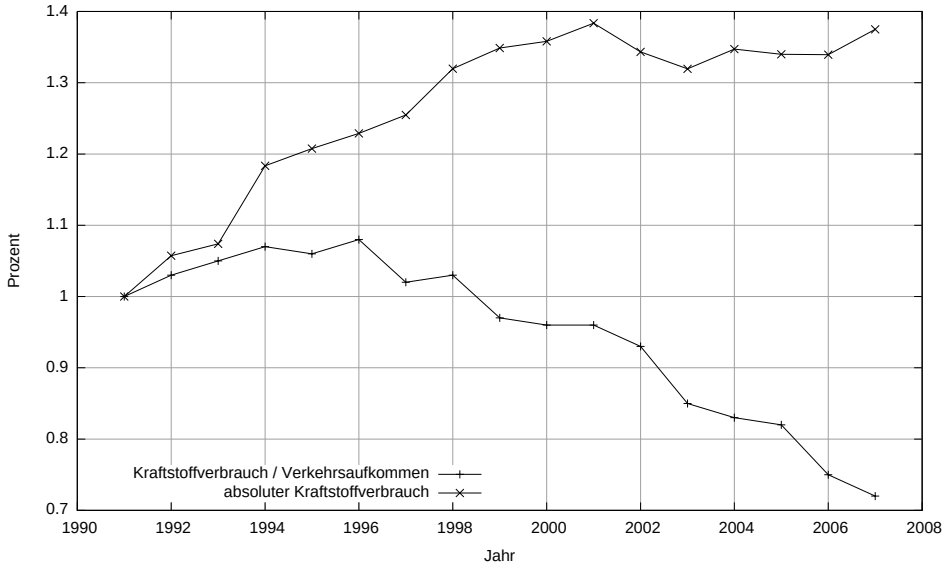

Abbildung 2.5: Prozentuale Veränderung des absoluten Kraftstoffverbrauchs des Güterverkehrs und des Kraftstoffverbrauchs des Güterverkehrs pro Verkehrsaufkommen in Tonnenkilometern. Basiswert 1991 entspricht 100%. (Quelle: Verkehr in Zahlen 2008/09 [65]).

Fahrzeuge mit einem zulässigen Gesamtgewicht von 2,8 Tonnen werden als Transporter bezeichnet. Bei einem Gewicht von unter 7,5 Tonnen spricht man von einem leichten, darüber hinaus von einem schweren LKW.

Im operativen Einsatz spielen neben dem Gesamtgewicht auch Betriebskosten, Beladungsmöglichkeiten sowie Flexibilität eine große Rolle. Gerade bei den Beladungsmöglichkeiten existieren eine Vielzahl von speziellen Lösungen für die unterschiedlichsten Anforderungen.

Eisenbahngüterverkehr

In Deutschland ist die Bahn trotz eines steigenden Güterverkehrsaufkommens nur unterdurchschnittlich vertreten. Diesen Umstand verdeutlicht Abbildung 2.3. Ein Grund hierfür ist die im Vergleich zum Straßengüterverkehr verminderte Flexibilität. Allerdings haben Züge vor allem energetische Vorteile gegenüber dem LKW. Aufgrund der geringen Rollreibung von Rad und Schiene verbraucht der Transport großer Mengen auf der Schiene weit weniger Energie je Transporteinheit als auf der Straße. Dies gilt jedoch nur, falls der Zug ausreichend ausgelastet ist. Allerdings können Züge nicht beliebig viele Güter aufnehmen. So ist die Zuglänge auf 700 Meter und die Zuglast auf ca. 2000 Tonnen begrenzt. Gründe hierfür sind die Dimensionen von Überholgleisen sowie die Bruchlast der Schraubenkupplung.

Kombinierter Verkehr

Die Verbindung von Straßengüterverkehr und Eisenbahngüterverkehr wird kombinierter Verkehr (KV) genannt. Die Begriffe *Intermodaler Verkehr* bzw. *Multimodaler Verkehr* können synonym verwendet werden. Hierbei werden Güter von der Quelle zur Senke mittels verschiedener Verkehrsträger ohne längere Zwischenlagerung befördert. Ein solcher Transport kann auch als Transportkette angesehen werden.

Während des gesamten Transports wird der Transportbehälter beibehalten, d. h. die Ladung selbst muss nicht umgeschlagen werden. Dies ermöglicht es, die jeweiligen verkehrsspezifischen Eigenschaften und Vorteile optimal auszunutzen: im Vor- und Nachlauf transportiert ein LKW flexibel und schnell kleine Lademengen während für Massengütertransporte im Hauptlauf oft Massengut fähige Verkehrsträger wie Bahn oder Binnenschiff zum Einsatz kommen. Um einen Transport möglichst effizient und kostengünstig durchzuführen, müssen einige Einschränkungen gemacht werden. Der Transportbehälter muss standardisiert sein. Nur so wird ein reibungsloser Umschlag gewährleistet. 20- oder 40-Fuß Container, Wechselbehälter oder Sattelanhänger sind typische Beispiele für einen solchen Behälter. Bei den ersten beiden Container-Arten handelt es sich um ISO-Container, die zwar weltweit anerkannt sind und eingesetzt werden, jedoch nicht auf EURO-Paletten abgestimmt sind. Dies führt zu einem unvorteilhaft ausgelasteten Container. Als Alternative haben die europäischen Bahn-Gesellschaften einen Binnencontainer entwickelt. So konnte bei gleich bleibender Länge und gleichen Eckbeschlägen, die zum Handling wichtig sind, durch die Optimierung der Innenbreite eine bessere Raumausnutzung von Europaletten erreicht werden. Tabelle 2.1 verdeutlicht dies.

Container Typ		Europaletten		Chemiepaletten	
Container	Länge	Anzahl	Flächennutzungsgrad	Anzahl	Flächennutzungsgrad
Binnen-Container	40 Fuß	28	93,7%	22	91,8%
	20 Fuß	14	95,3%	10	84,9%
ISO-Container	40 Fuß	24	84,1%	21	91,8%
	20 Fuß	11	78,9%	10	98,4%

Tabelle 2.1: Charakteristika verschiedener Container [42]

Wird lediglich der Transportbehälter umgeschlagen, so spricht man von einem *unbegleiteten* Transport. Eine spezielle Form des KV ist der begleitete kombinierte Verkehr. Hierbei wird der gesamte LKW auf dem Zug verladen. Der LKW-Fahrer wird während des Transports auf der Schiene in einem Liegewagen untergebracht. Dieses Konzept wird auch als *Rollende Landstraße* bezeichnet. Die schwerwiegenden Nachteile dieses Systems

liegen im ungünstigen Nutzlast/Totlast-Verhältnis sowie im konstruktiven Aufwand der entsprechenden Wagons begründet.

Damit sich der Transport auf der Schiene auszahlt, müssen zwei wesentliche Punkte erfüllt sein: eine ausreichende Auslastung eines Zuges sowie eine gewisse zurückgelegte Entfernung. Als Faustregel gilt hier eine Distanz von ca. 300 km [39]. Innerhalb dieser Strecke ist ein LKW in der Regel ökonomisch vorteilhafter.

Nah- und Fernverkehr

Man spricht gewöhnlich von Nahverkehr, falls der Einsatzradius 150 km nicht übersteigt. Im Fernverkehr werden Komplettladungen über größere Entfernungen von einem Sender zu einem Empfänger transportiert.

Bei mehrgliedrigen Transporten, bestehend aus Vor-, Haupt- und Nachlauf, findet der Hauptlauf meist im Fernverkehr zwischen zwei Umschlagpunkten statt. Der Vorlauf hat die Aufgabe des Sammelns von Gütern im Nahverkehr, während der Nachlauf das Verteilen der Güter über die Fläche bewerkstelligt. Da es im Nahverkehr so gut wie keine Alternative zum Straßentransport gibt, stehen im Fernverker mit der Bahn oder dem Binnenschiff Alternativen zu Verfügung.

Güterverkehrszentren

So genannte KV-Terminals stellen die Hauptschnittstelle zwischen den Verkehrsträgern Straße und Schiene dar. Sie besitzen die nötigen Gerätschaften, um einen reibungslosen Umschlag zwischen LKW und Eisenbahn im kombiniertem Verkehr zu gewährleisten.

Eine weitere Schnittstelle zwischen Straßen- und Eisenbahngüterverkehren stellen Güterverkehrszentren (GVZ) dar. Es existiert keine einheitliche Definition dieses Begriffs, jedoch gelten für jedes GVZ eine Reihe von Eigenschaften. Ein GVZ ist eine Ansammlung von Firmen, die auf freiwilliger Basis ihre Waren bündeln und multimodale Transporte bilden. Ein GVZ ist nicht zu verwechseln mit einem Güterverteilzentrum, welches ein reiner Umschlagstandort ist. In einem GVZ hingegen werden zusätzliche Dienstleistungen angeboten. Das Spektrum reicht über die reine Zwischenlagerung bis hin zur Kommissionierung oder Etikettierung.

In dieser Diplomarbeit werden die Standorte von 29 GVZ exemplarisch als Umschlagstandorte festgelegt. Abbildung 2.6 zeigt ihre Verteilung in Deutschland. Sie besitzen alle eine gute Anbindung an das Schienennetz, da sie in unmittelbarer Nähe von großen Knotenpunkten des Schienennetztes liegen. Um Fahrpläne zwischen diesen Stationen zu erhalten, wurde die Fahrplanauskunft der DB Schenker Rail Deutschland AG [3] benutzt. Mittels einer automatisierten Abfrage wurden 891 Relationen zwischen den GVZ ermittelt. Diese umfassen Daten über den Abfahrtswochentag, die Abfahrtszeit, den Ankunftswochentag,

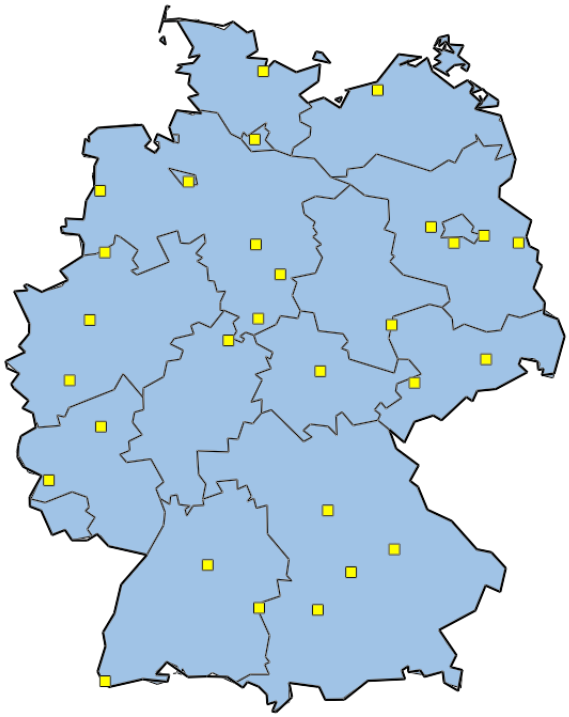

Abbildung 2.6: Standorte der in dieser Arbeit verwendeten GVZ in Deutschland.

die Ankunftszeit sowie die Länge der Strecke. Analog zu Verbindungen im Eisenbahnpersonenverkehr sind auch die Fahrpläne im Eisenbahngüterverkehr wöchentlich wiederkehrend.

2.2 Approximationsschemata

Viele Probleme innerhalb der Informatik gehören zur Klasse der NP-vollständigen Probleme. Für solche Probleme ist kein Algorithmus bekannt, der eine Lösung in polynomieller Zeit berechnen kann. Es wird sogar vermutet, dass für diese Probleme überhaupt kein Polynomialzeit-Algorithmus existiert. Nähere Informationen zur Komplexitätstheorie findet man z. B. bei Wegener [66].

Eine Möglichkeit NP-vollständige Probleme zu lösen besteht darin, sie zu approximieren. Oft ist es ausreichend eine Lösung zu finden, die nur unwesentlich schlechter als das Optimum ist. Ein Algorithmus, der eine solche, fast-optimale Lösung berechnet, heißt *Approximationsalgorithmus*.

Nach Cormen et al. [22] besitzt ein Algorithmus eine *Approximationsgüte* $\rho(n)$, falls er für eine beliebige Eingabe der Größe n eine Lösung mit Kosten C berechnet, die höchstens

um Faktor $\rho(n)$ von der optimalen Lösung C^* abweicht. Es muss also $\max\{\frac{C}{C^*}, \frac{C^*}{C}\} \leq \rho(n)$ gelten. Ein Algorithmus, der eine Approximationsgüte $\rho(n)$ berechnet, heißt auch $\rho(n)$-*Approximationsalgorithmus*.

Ein *Approximationsschema* beinhaltet neben der Eingabe für ein Problem einen Parameter $\epsilon > 0$, sodass für jedes feste ϵ eine $(1+\epsilon)$-Approximation berechnet wird. Falls für ein festes $\epsilon > 0$ das Approximationsschema eine Laufzeit besitzt, die polynomiell in der Eingabe n ist, so spricht man von einem *polynomial-time approximation scheme (PTAS)*. Ein PTAS kann eine exponentielle Laufzeit haben, falls ϵ im Exponenten vorkommt. Falls ein Approximationsschema für ein festes $\epsilon > 0$ eine Laufzeit besitzt, die sowohl polynomiell in n als auch in $1/\epsilon$ ist, so spricht man von einem *fully-polynomial approximation scheme*. In einem solchen Schema wirkt sich jede lineare Verkleinerung von ϵ auch nur linear auf die Laufzeit aus.

2.3 Flussprobleme

Flussprobleme bezeichnen eine Problemklasse, die in vielen praktischen Bereichen zum Einsatz kommen. Allen ist gemein, dass ein Fluss durch ein Flussnetzwerk geschickt werden muss. Anwendungsbeispiele finden sich bei der Simulation von Verkehren, Evakuierungsszenarien, Flüssigkeiten in Rohrsystemen, Stromverteilung in Stromnetzen oder Datenpaketrouting in Computernetzwerken. Besonders in der Logistik taucht diese Problemform häufig auf, da in der Logistik häufig Güter betrachtet werden, die von einer Quelle (Produktion) zu einer Senke (Kunde) fließen.

Obwohl es unzählige Flussproblemvarianten gibt, existieren gewisse Eigenschaften, die fast alle von ihnen besitzen. Hierzu gehört das zugrunde liegende Netzwerk. Nach Cormen et al. [22] ist ein Flussnetzwerk $G = (V, E)$ ein zusammenhängender, gerichteter Graph mit einer Kapazitätsfunktion $c : E \to \mathbb{R}^+$. Weiterhin gibt es zwei ausgezeichnete Knoten: eine Quelle $s \in V$ und eine Senke $t \in V$. Ziel ist es nun, so viel Fluss wie möglich von s nach t durch das Netzwerk zu schicken. Formal ist der Fluss eine Funktion $f : V \times V \to \mathbb{R}$ der die folgenden Eigenschaften erfüllt:

- *Kapazitätsbegrenzung:* $f(u, v) \leq c(u, v) \quad \forall u, v \in V$
- *Symmetrie:* $f(u, v) = -f(v, u) \quad \forall u, v \in V$
- *Flusserhaltung:* $\sum_{v \in V} f(u, v) = 0 \quad \forall u \in V \setminus \{s, t\}$

Der Wert eines Flusses ist definiert als $|f| = \sum_{v \in V} f(s, v)$.

Zu den grundlegenden Fragestellungen im Bezug auf Flussprobleme gehören *maximum flow* sowie *minimum cost flow* Probleme. Beide Problemarten werden im Folgenden kurz vorgestellt.

2.3.1 Maximum Flow

Bei diesem Problem soll der Flusswert $|f|$ maximiert werden. Eine einfache Abwandlung dieses Problems besteht darin, dass anstatt einer Quelle s und einer Senke t, mehrere Quellen $S = \{s_1, s_2, \ldots, s_k\}$ und mehrere Senken $T = \{t_1, t_2, \ldots, t_l\}$ besitzt. Sei f_s der Wert des Flusses von Quelle s aus, dann soll $\sum_{s \in S} |f_s|$ maximiert werden. Dieses Problem kann einfach auf die ursprüngliche Fragestellung reduziert werden, indem das Netzwerk um eine Superquelle s_0 und eine Supersenke t_0 erweitert wird. Diese Quelle wird nun mit allen Senken aus S verbunden, wobei die Kapazität einer Kante $(s_0, s_i), 1 \leq i \leq k$ der Summe der Kapazitäten aller Kanten ist, die s_i verlassen. Analog dazu wird mit der Supersenke verfahren.

Eine klassische Lösungsmethode haben Ford und Fulkerson [29] mit dem *Max-Flow-Min-Cut* Theorem entdeckt. Dieses Theorem besagt, dass der minimale Schnitt in einem Flussnetzwerk dem maximalen Fluss entspricht. Ein Schnitt (S, T) in einem Flussnetzwerk $G = (V, E)$ ist eine Partition der Knoten V in zwei Mengen S und T, sodass $s \in S$, $t \in T$ und $T = V - S$ ist. Die Kapazität eines Schnitts (S, T) ist $c(S, T)$. Ein minimaler Schnitt (S, T) ist ein Schnitt, dessen Kapazität minimal unter der Menge aller Schnitte ist. Nach dem Theorem gilt für einen minimalen Schnitt demnach $c(S, T) = |f| = f(S, T)$.

Der zu diesem Theorem gehörige Algorithmus basiert auf flussvergrößernden Pfaden und berechnet diese in einem Residualnetzwerk G_f. Ein solches Netzwerk besteht aus der gleichen Knotenmenge wie G, jedoch existieren nur noch Kanten, die anzeigen, zwischen welchen Knoten noch Fluss fließen kann. Die Kapazität einer Residualkante ist durch $c_f(u, v) = c(u, v) - f(u, v)$ gegeben. Daraus folgt, dass das Residualnetzwerk $G_f = (V, E_f)$ aus $G = (V, E)$ durch $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$ hergestellt werden kann. Der Algorithmus startet mit einem leeren Fluss und sucht nun in G_f iterativ nach Pfaden von s nach t auf denen Fluss fließen kann, ohne die Kapazitäten zu verletzen. Dieser Algorithmus hat eine Laufzeit von $O(E \cdot f^*)$ wobei f^* den maximalen Fluss bezeichnet. Edmonds und Karp [27] verbessern diese Laufzeit, indem sie anstatt einem beliebigen s-t Pfad einen kürzesten suchen. Dinic [26] hat diese Idee noch einmal verbessert, indem er das Prinzip von blockierenden Flüssen einführte. Die Laufzeit wird hierbei verbessert, indem in jeder Iteration ein Fluss berechnet wird, sodass über keinen Pfad zusätzlicher Fluss im Residualnetzwerk von s nach t geschickt werden kann.

Ein vollkommen neuer Ansatz wurde von Goldberg [35] entwickelt und wird als *push-relabel* Methode bezeichnet. Seine Funktionsweise lässt sich am Besten intuitiv anhand eines Wassernetzes erklären. Die obigen Algorithmen pumpen iterativ Wasser über einen Pfad durch das Netzwerk, bis dies nicht mehr möglich ist. Goldbergs Algorithmus folgt einer anderen Idee. Hierbei besitzen Knoten zusätlich ein beliebig großes Reservoir, in dem Wasser gespeichert werden kann. Darüber hinaus liegt jeder Knoten mit seinem Wasserspeicher auf einer Plattform, welche eine bestimmt Höhe besitzt. Bei so genannten*push*-Operationen

wird der Fluss, der in einem Knoten gespeichert ist immer nur zu Knoten weiter geleitet, die auf einer Plattform unterhalb des aktuellen Knotes liegen, also bergab.
Zu Beginn des Algorithmus besitzt die Quelle s eine Höhe von $|V|$ und die Senke t die Höhe 0. Beide Werte werden nicht verändert. Die Höhen aller übrigen Knoten sind initial 0 und werden während der Ausführung des Algorithmus erhöht. Im ersten Schritt schickt der Algorithmus so viel Wasser durch alle Kanten die Kanten, dass alle Kanten saturiert sind. Nun haben alle Knoten, die direkt mit s verbunden sind Wasser empfangen und speichern es. Von dort aus wird es schließlich weiter bergab geschickt.

Falls eine *push*-Operation an einem Knoten u versucht, Fluss bergab zu schicken, alle ausgehenden Kanten jedoch zu Knoten auf der gleichen oder einer höheren Ebene zeigen, so muss die Höhe von u erhöht werden. Diese Operation wird als *relabeling* bezeichnet. Die Plattform von u wird auf eine Ebene höher als die höchste von u erreichbare Ebene gesetzt. Anschließend kann die *push*-Operation wieder Fluss bergab leiten.

Schließlich kommt der Fluss, der die Senke erreichen kann auch dort an. Fluss, der aufgrund der Kapazitäten nicht zur Senke kommt muss das Netzwerk wieder verlassen, damit der Fluss zulässig ist. Dies geschieht, indem überschüssige Flusseinheiten wieder zurück zur Quelle geschickt werden.

2.3.2 Minimum Cost Flow

Bei der Betrachtung von *minimum cost flow* Problemen soll der maximale Fluss, der durch ein Netzwerk geschickt werden kann, auf Pfaden fließen, so dass die Kosten, die der gesamte Fluss erzeugt, minimal sind. Hierfür wird eine Abbildung benötigt, die jeder Kante einen Kostenfaktor zuweist, der angibt, wie teuer es ist, eine Flusseinheit über diese Kante zu schicken. Nach Ahuja [12] besteht das Problem aus einem gerichteten Graphen $G = (V, E)$ mit Kantenkosten c_{ij} und Kantenkapazitäten u_{ij} für jede Kante $(i, j) \in E$. Jedem Knoten $v \in V$ ist ein Bedarf $b(v)$ zugeordnet. Falls $b(v) < 0$ gilt, so stellt v eine Senke dar, die Bedarf $b(v)$ besitzt. Analog dazu ist v eine Quelle, die $b(v)$ Flusseinheiten verschicken kann, falls $b(v) > 0$ gilt. Das Problem kann nun wie folgt beschrieben werden:

$$\min \quad z(x) = \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{$MinCost$}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b(i) \quad \forall i \in V \tag{2.1}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in E \tag{2.2}$$

Es existieren im Wesentlichen drei verschiedene Ansätze um *minimum cost* Flüsse zu berechnen. Der erste besteht darin, Kreise mit negativen Kosten im Residualnetzwerk zu finden. Über solche Kreise wird dann Fluss augmentiert. Diese Prozedur wird so lange

wiederholt, bis es keine negativen Kreise mehr gibt. Solche Algorithmen heißen *cycle-canceling* Algorithmen.

So genannte *successive shortest path* Algorithmen dagegen augmentiert Fluss über einen geeignet definierten kürzesten Pfad zwischen einer Quelle und einer Senke. Dieses Verfahren wird so lange wiederholt, bis ein optimaler Fluss erreicht ist.

Die letzte Möglichkeit das obige Problem zu lösen, besteht ebenfalls in der Berechnung von kürzesten Pfaden. Bei diesen *primal-dual* Algorithmen wird der Fluss, je nach Algorithmus z. B. mit Hilfe einer *maximum flow* Prozedur, über mehrere kürzeste Pfade durch das Netzwerk geschickt. Beim *out-of-kilter* Algorithmus werden sogar zunächst Kapazitätsverletzungen zugelassen.

Minimum cost flow Probleme tauchen fast überall auf. Sei es in der Industrie, in der Kommunikation, Energie, Gesundheitswesen oder Logistik. Ein Beispiel für letzteres ist das so genannte *Transportation* Problem. Bei diesem Problem gibt es p Produktionsstandorte und q Lager mit bekannten Bedarfen. Ziel ist es, einen Fluss so zu bestimmen, dass alle Bedarfe erfüllt werden und die Kosten des gesamten Flusses minimal sind. Es ist sogar möglich, dieses Szenario auf mehrere Produktklassen zu erweitern.

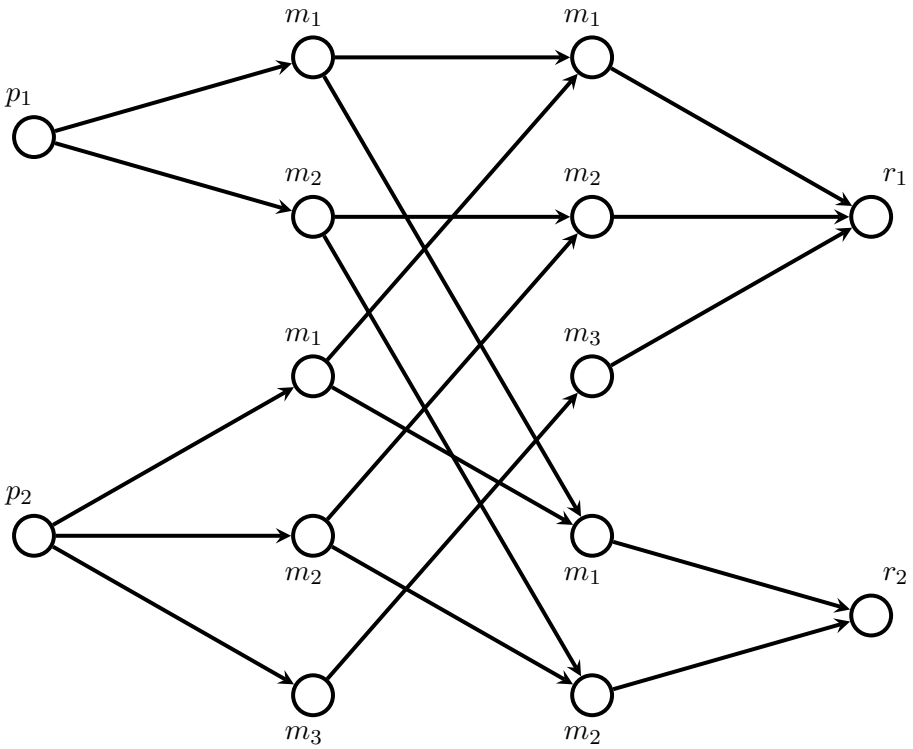

Abbildung 2.7: Transportation Probleminstanz mit Werken p_1 und p_2, Kunden r_1 und r_2 und Modellen m_1, m_2 und m_3.

Abbildung 2.7 veranschaulicht dies. Die Knoten p_1 und p_2 stellen Fabriken und die Knoten r_1 und r_2 Kunden dar. Die mittleren Knoten stellen die Bedarfe der Fabriken bzw. Kunden in Bezug auf die Modelle m_1, m_2 und m_3 dar. Dabei kann eine Transportkante zwischen diesen Modellknoten für einen beliebigen Transport stehen.

Sobald mehrere Güter betrachtet werden, die gemeinsam Ressourcen in Anspruch nehmen müssen, ist dieses Modell jedoch ungeeignet. Dies liegt daran, dass Flusseinheiten auf einer Kante nicht unterschieden werden können. Im obigen Beispiel lässt sich dies umgehen, da für jedes Modell m_i ein unabhängiger Pfad existiert. Möchte man jedoch Sendungen auf einer einzigen Kante bündeln, so dies endgültig nicht möglich. Ein generelles Problem dieser Algorithmenklasse ist die genau Zuordnung einer Sendung zu einer bestimmten Quelle und Senke. Für solche Probleme werden so genannte *multicommodity flow* Algorithmen eingesetzt.

2.3.3 Multicommodity Flow

Sowohl *maximum flow* als auch *minimum cost flow* Probleme betrachten immer genau eine Sendung, die von einer Quelle zu einer Senke geschickt werden soll. In der Praxis besteht aber oft die Notwendigkeit mehrere Sendungen spezifischen Quelle-Senke-Paaren zuzuordnen. Ein einfaches Beispiel hierfür ist die Kommunikation über das Telefon. Abstrakter gesehen braucht man solche Paare für die meisten Kommunikationsformen, auch z. B. für das Internet. Diesen Beispielen ist gemein, dass die Sendungen innerhalb des Netzwerkes sowohl gemeinsame Ressourcen nutzen, als auch unter Umständen individuelle Eigenschaften und Restriktionen besitzen können. Beim *multicommodity flow* Problem teilen sich einzelne Sendungen, auch *Commodities* genannt, gemeinsam Kantenkapazitäten. Dabei besitzt eine Commodity neben einer Quelle und einer Senke auch einen Bedarf, der von der Quelle zur Senken fließen muss. Ahuja [12] definiert dieses Problem wie folgt. Sei x_{ij}^k der Fluss von Commodity k auf der Kante (i, j). Darüber hinaus stellt x^k den Flussvektor dar, der angibt über welche Kante wie viel Fluss der Commodity k fließt. Der Kostenvektor c^k gibt für jede Kante an, welche Kosten durch eine Flusseinheit von Commodity k auf dieser Kante verursacht werden. Eine Beschreibung des *multicommodity flow* Problem lautet dann wie folgt:

$$\min \quad z(x) = \sum_{1 \le k \le K} c^k x^k \tag{P_{mc}}$$

$$\text{s.t.} \quad \sum_{1 \le k \le K} x_{ij}^k \le u_{ij} \quad \forall (i,j) \in E \tag{2.3}$$

$$\mathcal{N} x^k = b^k \quad \forall k = 1, 2, \ldots, K \tag{2.4}$$

$$0 \le x_{ij}^k \le u_{ij}^k \quad \forall (i,j) \in E, k = 1, 2, \ldots, K \tag{2.5}$$

Ungleichung 2.3 stellt die Gesamtkapazitätsbeschränkung einer Kante sicher. In Gleichung 2.4 wird die Verteilung der Bedarfe sichergestellt. Die Bedeutung der Funktion b ist analog zu der bei den *minimum cost flow* Problemen. Obwohl es für viele praktische Probleme nicht von Bedeutung ist, gibt es die Möglichkeit, für jede Commodity auf jeder Kante eine individuelle Kapazität anzugeben. Die Einhaltung dieser Restriktion wird durch Ungleichung 2.5 sichergestellt.

Algorithmisch sind viele Methoden zur Lösung des Problems entdeckt worden. Ein grundlegender Ansatz stellt die Lagrange Relaxierung dar. Dieses Lösungsverfahren gehört zur Familie der kosten-gerichteten Dekomposition und beschreibt einen allgemeinen Ansatz, der auch auf viele andere Probleme anwendbar ist.

Bei der Lagrange Relaxierung werden die Ungleichungen des zugrunde liegenden linearen Programms in „einfache“ und „schwierige“ Ungleichungen aufgeteilt. Das Programm wird anschließend so verändert, dass die „'schwierigen'“ Ungleichungen mit so genannten Lagrange Multiplikatoren gewichtet und der Zielfunktion hinzugefügt werden. Für das obige Programm P_{mc} würde dies bedeuten, dass Ungleichung 2.3 entfernt und in die Zielfunktion eingebaut wird. Diese sähe dann wie folgt aus:

$$L(w) = \min \sum_{1 \leq k \leq K} c^k x^k + \sum_{(i,j) \in E} w_{ij} (\sum_{1 \leq k \leq K} x_{ij}^k - u_{ij}) \tag{2.6}$$

$$= \min \sum_{1 \leq k \leq K} \sum_{(i,j) \in E} (c_{ij}^k + w_{ij}) x_{ij}^k - \sum_{(i,j) \in E} w_{ij} u_{ij} \tag{2.7}$$

Dabei stellen w_{ij} die Lagrange Multiplikatoren dar. Es ist zu beachten, dass der letzte lineare Term in Gleichung 2.7 für eine beliebige Belegung der w_{ij} konstant ist. Die Kosten für den Wert des Flusses auf einer Kante bestehen nun sowohl aus den ursprünglichen Kosten und den Lagrange Multiplikatoren. Da nun keine Ungleichung aus Variablen besteht, die mehrere Commodities betreffen, kann das Problem in k unabhängige *minimum cost flow* Probleme partitioniert werden.

Die Lösung eines solchen linearen Programms kann durch die *subgradient* Optimierung geschehen. Bei diesem Verfahren werden iterativ zunächst die k *minimum cost flow* Probleme für feste w_{ij} berechnet. Anschließend werden die w_{ij} neu berechnet. Dieses Verfahren wird so oft wiederholt, bis die optimale Lösung erreicht ist.

2.3.4 Multicommodity Concurrent Flow

Das *multicommodity concurrent flow* Problem ist ähnlich definiert, wie das *multicommodity flow* Problem. So besteht die Eingabe ebenfalls aus einem gerichteten Graphen $G = (V, E)$, einer Kapazitätsfunktion $u : E \rightarrow \mathcal{R}^+$ sowie k Commodities (s_j, t_j) mit Bedarfen d_j. Das Ziel ist nun jedoch nicht einen maximalen Fluss zu finden, sondern ein maximales λ, sodass von jeder Commodity j mindestens λd_j Flusseinheiten durch das Netzwerk geschickt

werden. Die *minimum cost flow* Variante wird analog zur *maximum flow* Variante definiert und erhält zusätzlich eine Kostenfunktion $c : E \to \mathbb{R}^+$. Ziel ist es nun, alle Bedarfe mit minimalen Gesamtkosten zu erfüllen.

Falls, durch entsprechende Auswahl der Kantenkapazitäten, sichergestellt wird, dass λ höchstens 1 werden kann, so ist dieses Problem äquivalent zum normalen *multicommodity flow* Problem.

Die momentan asymptotisch besten worst-case Laufzeiten für dieses Problem werden durch so genannte *interior point* Algorithmen erreicht. Ein anderer Bereich sind Approximationsalgorithmen. Shahrokhi and Matula [63] stellten ein Framework vor, das im Laufe der Zeit mehrfach erweitert und verbessert wurde.

Autor	*min cost concurrent flow*
Leighton et al. [48]	$\tilde{O}(\epsilon^{-2}k^2mn)$
Plotkin et al. [56]	$\tilde{O}(\epsilon^{-2}k^2m^2)$
Kamath et al. [43]	$\tilde{O}(\epsilon^{-6}kmn^2)$
Karger und Plotkin [45]	$\tilde{O}(\epsilon^{-3}kmn)$
Grigoriadis und Khachiyan [38]	$\tilde{O}(\epsilon^{-2}kmn)$
Garg und Könemann [32]	$\tilde{O}(\epsilon^{-2}(k+m)m + kmn)$
Fleischer [28]	$\tilde{O}(\epsilon^{-2}m(m+k))$
Karakostas [44]	$\tilde{O}(\epsilon^{-2}m^2)$

Tabelle 2.2: Komplexitätsvergleich deterministischer *minimum cost multicommodity flow* Probleme

Tabelle 2.2 spiegelt die erreichten Laufzeiten wider. Radzik [57] hat verschiedene Approximationslösungsansätze analysiert und Gemeinsamkeiten extrahiert. Dabei stellte er zwei unterschiedliche Ansätze bei der Berechnung von *multicommodity concurrent min-cost flow* Problemen fest. Plotkin, Shmoys and Tardos [56], Karger und Plotkin [45] sowie Grigoriadis und Khachiyan [38] augmentieren den Fluss aufgrund von *minimum cost flow* Berechnungen für einzelne Commodities. Dabei wird zunächst ein Fluss etabliert, der alle Bedarfe erfüllt, aber vermutlich die Kapazitätsgrenzen verletzt. Anschließend wird der Fluss sukzessive vermindert, bis eine $(1 + \epsilon)$ Approximation erreicht ist. Garg und Könemann [32] gehen einen umgekehrten Weg. Sie starten mit einem 0-Fluss und erhöhen sukzessiv den Fluss bis zum Erreichen einer $(1 + \epsilon)$ Approximation.

In dieser Diplomarbeit kommt der Algorithmus von Karakostas [44] zum Einsatz. Dies geschieht aufgrund mehrerer Gesichtspunkte. Zum einen war es Ziel dieser Arbeit einen deterministischen Algorithmus zu implementieren. Da das allgemeine *multicommodity flow* Problem NP-vollständig ist (siehe z. B. Ahuja [12]) ist, fiel die Wahl auf einen Approximationsalgorithmus. In der Logistik ist die Anzahl der Sendungen ziemlich hoch, sodass meist

$k > n$ gilt. Hinzu kommt, dass es sich bei der Transportnetzwerkstruktur meist um sehr dünne Graphen handelt und $m = O(n)$ gilt. Unter diesen Voraussetzungen ist der Algorithmus von Karakostas eine gute Wahl, da er nur logarithmisch von k abhängt und darüber hinaus bei großem k eine schnellere Laufzeit als die übrigen besitzt. Speziell der Algorithmus von Grigoriadis und Khachiyan [38] $\tilde{O}(\epsilon^{-2}kmn)$ ist in diesem Fall schlechter in Bezug auf die theoretische worst-case Laufzeit. Ein weiteres Plus des Algorithmus von Karakostas ist die einfache Erweiterbarkeit auf multiple Kostenfunktionen sowie eine einfache Analyse, die auf einen einfachen und damit schnellen Algorithmus hoffen lässt. Zwar existiert meines Wissens nach für diese Variante des Algorithmus noch keine praktische Anaylse der Laufzeit, jedoch gibt es seit der Veröffentlichung von Fleischer [28] Anwendungen, die zu einer Verbesserung in der Praxis beigetragen haben [13].

2.4 Chemie der Atmosphäre und ihre Umweltfolgen

Durch eine Vielzahl an Aktivitäten wie Verdunstungen, Vulkanismus und Aktivitäten von Tieren und Menschen gelangen unter anderem aus Ozeanen, Fabriken und Haushalten fortwährend große Mengen natürlicher und anthropogener *Quellgase* in die Atmosphäre. Dort verbleiben sie entsprechend ihrer Lebensdauer. Diese kann von einer Sekunde bei OH bis hin zu mehreren Millionen Jahren für N_2 reichen. Während ihrer Verweildauer werden sie durch Winde lokal oder global über die Erde verteilt und schließlich durch Sonnenbestrahlung bzw. andere Stoffe in der Atmosphäre umgewandelt, chemisch verändert oder mit dem Regen als *Senkengase* ausgewaschen.

Aufgrund dieser Kreisläufe können sich Stoffe in der Atmosphäre nicht dauerhaft ansammeln. Sollte es vorübergehend zu einem Ungleichgewicht der Stoffe in der Atmosphäre kommen, so würde dieses Verhältnis nach Abschaltung der Veränderung nach einiger Zeit wieder ein normales Maß erreichen. Trotzdem sollte eine Anreicherung der Atmosphäre mit giftigen Stoffen vermieden werden, da sie dort großen Schaden anrichten können. Beispiel hierfür sind der Ab- oder Aufbau von Ozon, die Bildung von chemischem Smog oder saurer Regen.

Im nächsten Abschnitt werden die wichtigen Schadstoffe definiert und zueinander in Beziehung gesetzt. Anschließend wird eine Übersicht über vorhandene Werkzeuge zum Vergleich von Transporten unter Umweltaspekten gegeben. Die nachfolgenden Informationen zu den Luftschadstoffen Kohlenstoffoxid, Schwefeldioxid und Schwefeldioxid sind aus Wiberg et al. [67].

2.4.1 Primärenergieverbrauch

Laut dem technischen Report zu EcoTransIT [40] setzt sich der Primärenergieverbrauch aus der Bereitstellungsenergie und der Endverbrauchsenergie zusammen. Erstere ist die Energie, welche benötigt wird, um die Energie aus Rohstoffen zu erzeugen und diese zu

transportieren. Letztere entsteht beim Endverbraucher, z. B. im Straßenverkehr am Fahrzeug. Der Primärenergieverbrauch ist eine wichtige Kenngröße, da Energie zum Großteil aus nicht regenerativen Rohstoffen entsteht. Der Energieverbrauch wird in Kilojoule angegeben. Oft erfolgt die Bestimmung des Energieverbrauchs über den Treibstoffverbrauch wobei die konkrete Umrechnung über den bekannten Energiegehalt eines Treibstoffs sowie die Effizienz der Energieumwandlung erfolgt.

2.4.2 Kohlenstoffoxid

2/3 des giftigen Kohlenstoffmonoxids CO entsteht durch die unvollständige Verbrennung von fossilen Brennstoffen. Dies geschieht überwiegend auf der fahrzeugreichen Nordhalbkugel. Aufgrund der Lebensdauer von ca. 2 Monaten verteilt sich CO auf der gesamten nördlichen Hemisphäre. Steigt, z. B. durch starken Verkehr, die Stickstoffoxid (NO) Konzentration, so wird bei einer Reaktion mit CO ein Ozon-Molekül gebildet. Sinkt die NO Konzentration, so wird wiederum ein Ozon-Molekül verbraucht.

Durch die Oxidation von CO entsteht Kohlenstoffdioxid CO_2. Dies macht an der gesamten Menge CO_2 in der Atmosphäre jedoch nur einen geringen Teil aus. Seit der Industrialisierung steigt die Menge an CO_2 kontinuierlich an. Zwar können durch die Photosynthese der Pflanzen, Auflösung im Meer oder Verwitterung von Urgestein die Mengen CO_2, die durch die Atmung von Tieren und Menschen und die Ausgasung der Meere entstehen, kompensiert werden. Durch die Verbrennung von fossilen Brennstoffen fügt der Mensch jedoch beständig zusätzliche Mengen von CO_2 der Atmosphäre hinzu und vermindert gleichzeitig deren Abbau durch die Rodung von Wäldern.

Kohlenstoffdioxid hat die Eigenschaft infrarotes Licht zu absorbieren und in Wärme umzuwandeln. Aus diesem Grund ist es entscheidend für den Wärmehaushalt der Erde. Durch eine gesteigerte Konzentration von CO_2 in der Atmosphäre entsteht der so genannte *Treibhauseffekt*.

Obwohl in dieser Diplomarbeit ausschließlich der CO_2-Ausstoß minimiert wird, so wäre es prinzipiell möglich, durch die Anpassung der Kostenfunktion durch einen bestimmten Faktor, auch die folgenden Schadstoffe zu minimieren bzw. sie auszuweisen.

2.4.3 Schwefeldioxid

Schwefeldioxid SO_2 entsteht durch die Oxidation schwefelhaltiger Verbindungen, hauptsächlich jedoch durch die Verbrennung fossiler Brennstoffe sowie durch Haushalte und KFZ-Motoren. Es kann sich, bedingt durch seine Lebensdauer von mehreren Tagen, regional in der Troposphäre verteilen und stellt eine Hauptkomponente des chemischen Smogs dar. Schwefeldioxid SO_2 oxidiert zu Schwefelsäure H_2SO_4, die zusammen mit anderen Säuren in geringen Mengen durch Regen ausgewaschen wird. Man spricht hierbei auch von saurem Regen. Die stetige Erhöhung des SO_2 und NO Ausstoßes hat zu einem Anstieg

der Säurekonzentration in Gewässern und Böden geführt. Damit einhergehende Konsequenzen sind z. B. der Rückgang von Seeplankton, die Schädigung von Amphibien- und Fischpopulationen, das Auswaschen einiger, für das Pflanzenwachstum notwendiger, Ionen sowie das Freisetzen von Schwermetallionen. Letztere „vergiften“ Böden, Gewässer und Seen. Aufgrund dieser massiven Schädigungen gilt SO_2 als Mitverursacher des seit einigen Jahren beobachteten Waldsterbens. Durch verschärfte Auflagen konnte der Ausstoß von Schwefeldioxid jedoch stark reduziert werden.

2.4.4 Stickstoffoxide

Distickstoffoxid N_2O entsteht durch Bodenbakterien und die Verbrennung von fossilen Brennstoffen. In der Stratosphäre reagiert es zu Stickstoffmonoxid NO, welches eine Lebensdauer von 2 bis 3 Jahren hat und sich deshalb global verteilen kann. Das in der Troposphäre angereicherte NO entsteht hauptsächlich durch die Verbrennung von KFZ-Motoren. Es besitzt nur eine Lebensdauer von einem Tag, verteilt sich somit nur lokal, und oxidiert schnell zu Stickstoffdioxid NO_2.

Alle Stickstoffoxide sind Komponenten des photochemischen Smogs, der sich während windarmer Schönwetterperioden in Ballungsgebieten aufgrund von starkem Verkehr bildet. NO_2 bewirkt dann die Bildung der typischen Atemgifte Ozon und Peroxyacylnitrat.

2.4.5 Feinstaub

Das Umweltbundesamt stellt umfangreiche Informationen zum Thema Feinstaub in [64] zur Verfügung. Demnach bezeichnet der Begriff Feinstaub (engl. Particulate Matter) Teilchen, die nicht direkt zu Boden sinken sondern eine gewisse Zeit in der Atmosphäre verbleiben. Man klassifiziert verschiedene Partikel anhand ihres aerodynamischen Durchmessers (siehe Tabelle 2.3).

Bezeichnung	Durchmesser
Gesamtschwebstaub (TSP)	$< 60\mu m$
PM_{10}	$< 10\mu m$
$PM_{2.5}$	$< 2.5\mu m$
ultrafeine Partikel	$< 0.1\mu m$

Tabelle 2.3: Klassifizierung von Feinstaub

Darüber hinaus wird zusätzlich zwischen primären und sekundären sowie zwischen natürlichen und anthropogenen Feinstäuben unterschieden. Erstere entstehen direkt an einer Quelle, z. B. einem Verbrennungsprozess während sekundäre aus Vorläufersubstanzen wie Schwefel- und Stickstoffoxiden, Ammoniak oder Kohlenwasserstoffen entstehen.

Feinstäube, die natürlich entstehen, werden z. B. von Vulkanen, Meeren oder Bodenerosionen ausgestoßen. Quellen anthropogenen Ursprungs sind dagegen der Verkehr, Braunkohlekraftwerke, sowie technisch veraltete Heizungsöfen. In Ballungsgebieten ist die Belastung durch den Verkehr, insbesondere durch den Straßengüterverkehr, besonders schwerwiegend. Hier entstehen sowohl durch den Motor, als auch durch Brems- und Reifenabrieb sowie Straßenstaub-Aufwirbelungen Feinstäube.

Das Gesundheitsrisiko, das mit der Einatmung von Feinstaubpartikeln verbunden ist, hängt vor allem von der Größe der Partikel und der damit verbundenen Tiefe, bis zu der sie in den Atemtrakt eindringen und verbleiben, ab. Große Partikel werden bereits im oberen Teil der Atemwege aufgehalten, während ultrafeine Partikel sogar in den Blutkreislauf gelangen können. Vor allem vor dem Hintergrund, dass sich an der Oberfläche von Feinstaubpartikeln Schwermetalle und krebserregende Stoffe ansiedeln können, sind kleinere Partikel, aufgrund ihrer größeren Oberfläche, gesundheitsgefährdender als größere.

Um die Belastung durch Feinstaub zu verringern, hat die EU Grenzwertrichtlinien erlassen. In Deutschland wird diese Regelung durch die Einführung von Umweltzonen umgesetzt.

2.4.6 Werkzeuge zur Ausweisung von Schadstoffemissionen

Aufbauend auf der Datengrundlage mehrerer Forschungsprojekte haben verschiedene Institutionen weitere Untersuchungen durchgeführt. Neben theoretischen Artikeln (siehe z. B. [50, 60, 59, 2]) sind auch eine Reihe von Werkzeugen erschienen, die es ermöglichen dieses Wissen einfach zu nutzen. Hervorzuheben ist hier das "Handbuch Emissionsfaktoren des Straßenverkehrs 2.1" [41]. Dieses Handbuch ist vielmehr eine Datenbank, die Emissionsfaktoren des Straßengüterverkehrs für verschiedene Schadstoffe enthält. Es wurde von der Firma INFRAS [8] im Auftrag der Umweltbundsämter Deutschlands, Österreichs und der Schweiz zusammengestellt. Hierbei handelt es sich um ein Programm, das es ermöglicht, in einer umfangreichen Datenbank nach Schadstoffemissionen verschiedener Straßenfahrzeuge zu suchen. Die Informationen können unter anderem nach Fahrzeugtyp, Art der Straße, Neigung der Straße, Geschwindigkeit und Emissionstyp gefiltert werden. Das Handbuch erschien zum ersten Mal im Jahr 1995, die aktuelle Version 2.1 wurde 2004 veröffentlicht. Für diese Arbeit wurde mir eine Kopie des Programms freundlicherweise zur Verfügung gestellt.

Eine Reihe von Nutzungsmöglichkeiten der Forschungsergebnisse ist in Form von Internetseiten vorhanden. Die bekanntesten sind "EcoPassenger" [4] und "EcoTransIT" [5]. Beide Portale werden wissenschaftlich vom Institut für Energie- und Umweltforschung Heidelberg GmbH (ifeu) [7] betreut. Bei "EcoPassenger" steht der Umwelteinfluss von Personenreisen im Vordergrund, während "EcoTransIT" dem Güterverkehr gewidmet ist. Beiden Plattformen ist gemein, dass sie Ziele in ganz Europa zur Auswahl anbieten, sowie eine

Unterscheidung der verschiedenen Verkehrsträger anbieten. "EcoTransIT" bietet darüber hinaus die Möglichkeit einen multimodalen Transport zu spezifizieren. Beide Seiten weisen nach Eingabe verschiedener Parameter die Umwelteinflüsse für eine Reise beziehungsweise einen Transport aus.

Nicht nur im Straßengüterverkehr wird verstärkt auf die Umwelt geachtet. So gibt es ein von der EU unterstütztes Projekt, mit dem Schüler die CO_2-Emissionen, welche auf ihrem Weg zur Schule entstehen, ermitteln und vergleichen können [62]. In einigen großen Städten gibt es die vom Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit initiierte Aktion "Für 0 CO_2 auf Kurzstrecken" [9], mit dem Ziel, unwirtschaftliche Autofahrten durch das Rad oder den ÖPNV zu ersetzen. Aber auch in der Wirtschaft werden die Begriffe Nachhaltigkeit und Umweltschutz thematisiert. Deutlichstes Anzeichen hier ist die Etablierung des Begriffs "Green Logistics". Verschiedene Unternehmen überprüfen ihren Energieverbrauch und versuchen ihn zu reduzieren. Dies hat neben Einsparungen auch einen positiven ökologischen Nutzen. Ein Beispiel hierfür ist das Unternehmen DHL. Durch den "GoGreen Service" [6] bietet DHL an, die transportbedingten CO_2-Emissionen mit Emissionsminderungsprojekten auszugleichen.

Kapitel 3

Modellierung

Dieses Kapitel beschreibt die Modellierung des in Kapitel 1 vorgestellten Transportproblems. Ziel der Modellierung ist es, dieses in ein Mehrgüterflussproblem umzuwandeln. Dafür ist es nötig, die Eigenschaften des Problems in einen Flussgraphen mit entsprechenden Kostenfunktionen zu kodieren. Im Folgenden werden diese Eigenschaften zunächst formalisiert und anschließend die Transformation in ein Flussproblem beschrieben.

3.1 Modell

Die Grundelemente des multimodalen Transports wurden bereits in Abschnitt 1.3 beschrieben.

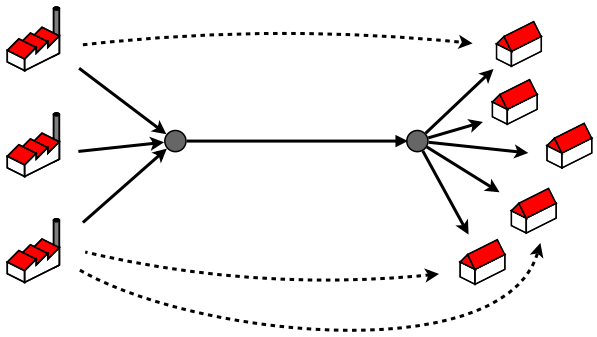

Abbildung 3.1: Elemente eines multimodalen Transports. Sendungen werden von den Fabriken entweder über GVZ (graue Punkte) oder direkt (gestrichelte Linien) transportiert.

Abbildung 3.1 stellt eine schematische Übersicht der Bestandteile dar. Diese werden nun detailiert beschrieben.

3.1.1 Zeithorizont

Um Zeiten im Modell einfach abbilden zu können, werden diskretisierte Zeitschritte verwendet. Die kleinste Zeiteinheit ist eine Minute. Dies ist ausreichend um reale Verhältnisse hinreichend genau abbilden zu können. Die Optimierung findet über einem Zeithorizont $\mathcal{T} \subset \mathbb{N}$ statt. Dieser beginnt bei 0 und endet bei $t_{\max}$. Ein Zeitpunkt $t \in \mathbb{T}$ gibt nun die Minuten an, die seit Beginn des Zeithorizonts verstrichen sind. Betrachtet man t als lineare Funktion der Form $t = a \cdot 1440 + b, a \in [0, 364], b \in [0, 1439]$, so kann man den aktuellen Tag durch $\lfloor t/1440 \rfloor$ und die seit Mitternacht dieses Tages verstrichenen Minuten durch (t mod 1440) leicht ermitteln.

3.1.2 Orte

Der Transport von Waren findet zwischen Orten statt. Die Menge aller Orte sei $\mathcal{L}$. Ein Ort $l \in \mathcal{L}$ ist definiert durch seine geographischen Koordinaten, also durch Längen- und Breitengrad. Um die Distanz zwischen zwei Orten zu ermitteln gibt es zwei Möglichkeiten: Eine einfache ist die Ermittlung einer euklidische Distanz unter Berücksichtigung der Erdkrümmung. Diese Entfernung kann allerdings beliebig schlecht sein. Beispiele hierfür wären zwei Orte, die an gegenüberliegenden Seiten einer Bucht lägen. Der direkte Weg würde quer durch das Wasser führen. Ungenau ist auch der Versuch die Fahrtzeit zwischen zwei Orten auf der Grundlage dieser Distanz zu ermitteln. Neben der Unkenntnis über die Existenz von Straßen, fehlt auch das Wissen über die Art der zur Verfügung stehenden Infrastruktur. Auf einer Autobahn z. B. ist die Fahrtzeit deutlich geringer als im Innenstadtbereich.

Eine wesentlich genauere Ermittlung dieser Daten wird durch den Einsatz von sogenannten Geo-Datenbanken erreicht. Je nach Umfang einer solchen Datenbank enthält sie Länder, Orte und Straßenkarten. So ist es möglich, hinreichend genaue Informationen über die Distanz, sowie unter Extraktion der benutzen Straßentypen, die Fahrtzeit zwischen zwei beliebigen Orten zu ermitteln. Um eine konkrete Verbindung zwischen zwei Orten zu berechnen, kommt ein kürzester-Wege-Algorithmus zum Einsatz.

Unabhängig von der Methode werden im Straßennetz Distanzen durch $dist(u, v) : \mathcal{L} \times \mathcal{L} \to \mathbb{R}^+$ und Fahrtzeiten durch $time(u, v) : \mathcal{L} \times \mathcal{L} \to \mathbb{R}^+$ abgebildet.

3.1.3 Sendungen

Zur Eingabe des Modlls gehört eine Anzahl an Sendungen, die z. B. von einem Produktionsstandort, zu einem Kunden geliefert werden müssen. Diese Sendung werden *Commodities* genannt und in der Menge $\mathcal{C}$ zusammengefasst. Hierbei gilt $k = |\mathcal{C}|$. Eine Commodity $c \in \mathcal{C}$ sagt aus, dass Waren der Menge $d_c \in \mathbb{R}^+$, auch Bedarf genannt, von einem Startort, einer Quelle, $l_c^o \in \mathcal{L}$ zu einem Zielort, seiner Senke, $l_c^d \in \mathcal{L}$ transportiert werden müssen. Alle Quellen werden in der Menge $\mathcal{O}$ und alle Senken in der Menge $\mathcal{D}$ zusammengefasst. Es ist möglich, für diesen Transport eine Abfahrtszeit $t_c^d \in \mathbb{T}$ am Startort, sowie eine Ankunftszeit

$t_c^a \in \mathbb{T}$ am Zielort festzulegen. Zur Berechnung der CO_2-Kostenfunktion (siehe Abschnitt 3.2.1) wird zusätzlich die Abgasnorm n_i des Motors des eingesetzten LKW benötigt.

3.1.4 Güterverkehrszentren

Ein Güterverkehrszentrum (GVZ) (siehe auch Abschnitt 2.1) $g \in \mathcal{G} \subseteq \mathcal{L}$ stellt eine Schnittstelle im multimodalen Verkehr dar. In manchen Fällen verfügt ein GVZ neben einem Gleisanschluss auch über einen Wasseranschluss. Auch verfügt ein GVZ oft über Möglichkeiten der Zwischenlagerung oder Weiterverarbeitung von Gütern. In diesem Modell beschränkt sich jedoch das Aufgabenspektrum auf die Be- und Entladung von LKW und Zügen. Für diese Vorgänge wird eine gewisse Zeit benötigt. An einem GVZ g beträgt die Zeit für das Entladen eines LKW t_g^{ll} und für das Beladen t_g^{ul}. Ein Zug wird in Zeit t_g^{lt} be- und in Zeit t_g^{ut} entladen.

Darüber hinaus besitzt ein GVZ auch einen Einzugsbereich. Unabhängig von der Kostenfunktion können zwei Radien g^{pre} und g^{post} um ein GVZ gelegt werden, die besagen, dass nur Orte innerhalb dieses Radius für einen Vor- bzw. Nachlauf in Frage kommen. Wie bereits in Abschnitt 2.1 erwähnt, gibt es gewisse Faustregeln, ab denen sich ein Vor- oder Nachlauf nicht mehr lohnt.

3.1.5 Fahrplan

Neben dem Verkehrsträger Straße wird auch der Gütertransport auf der Schiene betrachtet. Schienenverkehre können jedoch im Verleich zur Straße nicht beliebig fließen, sondern sind an Fahrpläne gebunden. Eine Zugrelation $r \in R$ kann als 5-Tupel $(g_r^d, g_r^a, t_r^d, t_r^a, c_r)$ angesehen und wie folgt interpretiert werden. Der Zug r verlässt zum Zeitpunkt $t_r^d \in \mathbb{T}$ das GVZ $g_r^d \in \mathcal{G}$ und erreicht $g_r^a \in \mathcal{G}$ zum Zeitpunkt $t_r^a \in \mathbb{T}$. Dabei hat der Zug eine maximale Kapazität von $c_r \in \mathbb{R}^+$ Tonnen.

3.2 Kostenfunktionen

Insgesamt werden die vier Kostenfunktionen, Zeit, Distanz, ökonomische Kosten und CO_2-Verbrauch modelliert, welche durch die Optimierung minimiert werden können. Allen Funktionen ist gleich, dass sie lineare Funktionen sind, die durch den Ursprung verlaufen. Dies begründet sich in der Tatsache, dass spätere Algorithmen jeder Kante einen Kostenwert zuordnen. Jede Flusseinheit, die über eine Kante fließt, verusacht dann Kosten, die diesem Wert entsprechen.

In dem verwendeten Modell muss jedoch darauf geachtet werden, dass Flusseinheiten auf unterschiedlichen Kanten unterschiedliche Repräsentationen besitzen. Auf einer Zugkante verkehrt zum Beispiel genau ein Zug während auf Straßenkanten beliebig viele LKW fahren können. Um dieses Problem zu umgehen, werden Transporte auf der Straße und

der Schiene normiert. Eine Flusseinheit wird als eine Tonne interpretiert. Diese Normierung bedeutet wiederum, dass die Kostenfunktionen auf diese Einheit angepasst werden müssen. Für LKW wird eine maximale Beladung von 26 Tonnen angenommen. Für einen Zug r_i entspricht das maximale Bruttogewicht des gesamten Zuges u_i. Ein Wagon hat eine maximale Zuladung von 61 Tonnen bei einem Leergewicht von 23 Tonnen. Abhängig vom Auslastungsgrad der Ladekapazität müssen die Kostenfaktoren der benutzen Kostenfunktion entsprechend skaliert werden.

3.2.1 Emissionsberechnung

Um Transporte unter dem Stichwort Green Logistics beurteilen zu können, ist es notwendig die einhergehenden Umwelteinflüsse beurteilen und vergleichen zu können. Hierfür eignen sich besonders der Primärenergieverbrauch sowie die Emission luftgetragener Komponenten. Zu beiden Kenngrößen steht eine große Datenbasis zur Verfügung, die es erlaubt die unterschiedlichen Verkehrsträger zu bewerten. Vor allem für Transporte auf der Straße wurden umfangreiche Untersuchungen durchgeführt. Ein Beispiel für eine solche Forschungsarbeit stellt das ARTEMIS Projekt [2] dar. Das in dieser Diplomarbeit untersuchte Modell soll zeigen, wie lohnend eine Verlagerung von Gütertransporten von der Straße auf die Schiene in Bezug auf Schadstoffemissionen ist. Dazu werden optimale Routen in Bezug auf die durch den Transport entstehenden CO_2-Emissionen berechnet. Da diese Emissionen direkt vom Primärenergieverbrauch abhängen, wird auch diese Kenngröße implizit betrachtet.

Im Folgenden wird beschrieben, wie CO_2-Emissionen berechnet werden und welche Parameter in die Berechnung mit einfließen. Als Berechnungsgrundlage dient zum einen das „Handbuch Emissionsfaktoren des Straßenverkehrs 2.1“[41], zum anderen das Onlineberechnungsprogamm ”EcoTransIT” [5] bzw. die dahinter stehenden theoretischen Grundlagen [40].

Die im Anschluss berechneten Emissionen umfassen sowohl die Schadstoffe, die bei der Energiegewinnung ausgestoßen werden, als auch diejenigen, die direkt am Verkehrsträger entstehen. Außerdem fließen folgende Parameter in die Berechnungen ein:

Distanz (D) Entspricht der zur überwindenden Distanz in Kilometern.

Beladungsfaktor (L) Gibt an, zu wie viel Prozent der Verkehrsträger beladen ist.

Leerfahrten (E) Gibt in Prozent an, wie viele Leerfahrten zusätzlich anfallen. Leerfahrten treten oft im Zugverkehr auf, da Wagons unbeladen wieder zum Ausgangsort zurückgebracht werden müssen.

Bruttozuggewicht (X) Gibt an, wie viele Tonnen ein Zug inklusive Ladung, Leerwagons und Zugmaschine wiegt.

LKW Emissionen

LKW stoßen an zwei Stellen Schadstoffe aus: bei der Energiegewinnung des Treibstoffs und am Fahrzeug selbst. Laut EcoTransIT [40] werden bei der Produktion von Treibstoff pro hergestelltem Kilogramm Dieselkraftstoff $0,47$ kg CO_2 freigesetzt.

Sowohl für die Bestimmung des Kraftstoffverbrauchs eines LKW als auch für Berechnung der Verkehrsträgeremissionen wird auf das Handbuch für Emissionsfaktoren [41] zurückgegriffen. Mittels der Filterfunktion werden die Verbrauchs- und CO_2-Werte für verschiedene EURO Abgasnormen ermittelt. Grundlage hierfür ist ein LKW mit einem Gesamtgewicht zwischen 34 und 40 Tonnen und einer maximalen Zuladung von 26 Tonnen. Das zugrunde liegende Straßenprofil ist der Durchschnitt über alle Autobahnen. Darin zusammengefasst sind, gewichtet nach ihrem tatsächlichen Aufkommen, verschiedene Geschwindigkeitsprofile und Längsneigungen. Bei Transporten auf der Straße macht die auf der Autobahn zurückgelegte Distanz den größten Anteil einer Fahrt aus. Aus diesem Grund werden alle anderen Straßenarten nicht betrachtet.

Die so ermittelten Werte für den Verbrauch eines leeren und voll beladenen 40 Tonnen LKW für verschiedene Euro-Normen werden in Tabelle 3.1 in kg/km zusammengefasst. Die während der Fahrt verursachten CO_2-Emissionen fasst Tabelle 3.2 zusammen. Hierbei wird sowohl der Verbrauch eines leeren sowie eines voll beladenen LKW in kg/km angegeben.

Euro-Norm	Beladung	
	0 %	100 %
EURO1	0,198211	0,314333
EURO2	0,187723	0,303542
EURO3	0,197352	0,312675
EURO4	0,19774	0,317135

Tabelle 3.1: Kraftstoffverbrauch eines LKW (34-40t, gewichteter Durchschnitt über alle Autobahnen) in kg/km

Sei $U^p_{\min}$ der jeweilige Verbrauch eines leeren und $U^p_{\max}$ der eines vollen LKW. Der durchschnittliche Verbrauch in kg pro km kann demnach durch $U^p_{\min} + (U^p_{\max} - U^p_{\min}) \cdot L$ bestimmt werden. Die Gesamtmenge CO_2 in Kilogramm, die benötigt wird um eine Distanz D zu überwinden, ist somit

$$P(L, D) = U^p_{\min} + (U^p_{\max} - U^p_{\min}) \cdot L \cdot D \cdot 0,47 \tag{3.1}$$

Bezeichne $U^f_{\min}$ bzw. $U^f_{\max}$ den minimalen bzw. maximalen CO_2-Emissionswert in Abhängigkeit der Euro-Norm des Motors. Die Berechnung der Emissionen am Fahrzeug in Kilogramm CO_2 je LKW erfolgt somit durch

$$F(L, D) = U^f_{\min} + (U^f_{\max} - U^f_{\min}) \cdot L \cdot D \tag{3.2}$$

Euro-Norm	Beladung	
	0 %	100 %
EURO1	0,62932	0,998006
EURO2	0,596019	0,963746
EURO3	0,626594	0,992744
EURO4	0,627823	1,006905

Tabelle 3.2: CO_2-Emissionen für LKW (34-40t, gewichteter Durchschnitt über alle Autobahnen) in kg/km

Der Schadstoffausstoß von Leerfahren E (in %) kann nun durch die Summe der vorherigen Formeln mit $L = 0$ bestimmt werden. Durch die Verbindung der beiden obigen Funktionen kann schließlich eine Berechnungsvorschrift angegeben werden, die eine Menge an Flusseinheiten in Tonnen auf die dafür ausgestoßenen CO_2-Emissionen in Tonnen abbildet:

$$U(L, E, D) = \frac{E \cdot (P(0, D) + F(0, D)) + P(L, D) + F(L, D)}{26t \cdot L \cdot 1000} \tag{3.3}$$

Die letzten Faktoren erklären sich zum einen durch die Umrechnung von Kilogramm in Tonnen. Zum anderen sollen die spezifischen CO_2-Emissionen berechnet werden. Dazu muss der berechnete Wert noch durch die Nettolast der Ladung geteilt werden.

Der so berechnete Wert entspricht nicht dem von EcoTransIT, sondern weicht im Durchschnitt um 5-10% ab. Eine mögliche Erklärung für diese Diskrepanz liegt in der Tatsache, dass sowohl der zugrunde liegende Treibstoffverbrauch, als auch die von EcoTransIT benutzte Berechnungsvorschrift nicht veröffentlicht wurde.

Güterzugemissionen

Für die Berechnung von Güterzügen werden prinzipiell die gleichen Parameter wie bei der LKW Berechnung benutzt. Der Parameter Beladung entspricht hier der Beladung eines Wagons. Ein zusätzlicher Parameter ist das Bruttogewicht des gesamten Zuges. Nach EcoTransIT [40] sind drei Größen üblich: 500t, 1000t und 1500t. Für die Berechnungen wird angenommen, dass einer dieser Werte, unabhängig von der tatsächlichen Beladung, für den gesamten Zug gilt. Für die Benutzung eines Komplettzuges ist diese Annahme realistisch. Falls nur einzelne Wagons genutzt werden, so wird angenommen, dass der Zug entsprechend anderweitig ausgelastet wird.

Anstatt einer Unterscheidung der verwendeten Motor-Normen werden Züge in strom- und dieselbetrieben eingeteilt. Bis auf einige Faktoren ist die CO_2-Berechnung von Zügen jedoch identisch zu der von LKW.

Zunächst wird der CO_2-Verbrauch eines Zuges in Abhängigkeit des Bruttogewichts ermittelt. Hierfür ist es wichtig, in welchem Land ein Zug eingesetzt wird. In Ländern

deren Höhenprofil sehr flach sind ist der Verbrauch geringer, als in Ländern mit vielen montanen Gebieten.

In Abhängigkeit des Bruttogewichts X eines Zuges geben C_e und C_d den Energieverbrauch eines Elektrozuges in Wattstunden pro Tonnenkilometer (Wh/tkm) bzw. eines Dieselzuges in Kilogramm Dieseltreibstoff pro Tonnenkilometer (kg/tkm) an. Für Deutschland gilt $C_e(X) = 675/(X^{-0,5} \cdot 1000)$ Wh/tkm und $C_d(X) = 153,07/(X^{-0,5} \cdot 1000)$ kg/tkm.

Die Bestimmung des Bruttoladungsgewichts in Tonnen der transportierten Güter, welche ein Nettogewicht von G Tonnen haben, schließt auch die Leerfahrten E (in %) mit ein und berechnet sich durch:

$$W(G, L, E) = G \cdot \left(\frac{L \cdot 61 + 23}{L \cdot 61} + E \cdot \frac{23}{L \cdot 61}\right) = G \cdot \left(1 + \frac{23 \cdot (1 + E)}{L \cdot 61}\right) \tag{3.4}$$

Bei dieser Berechnung wird davon ausgegangen, dass ein Wagon ein Leergewicht von 23t hat und maximal 61t zugeladen werden können. Das Bruttogewicht besteht also aus dem Gewicht des Wagons und der Ladung. Bei Leerfahrten ist nur der leere Wagon als Gewicht zu zählen.

Die Faktoren, die angeben, wie viel CO_2 ausgestoßen wird, hängen von der Antriebsart ab. Bei Elektrozügen werden während des Betriebes keine Emissionen ausgestoßen, jedoch bei der Produktion der elektrischen Energie. Diese kann auf unterschiedliche Weise hergestellt werden: Kernkraft, Kohlekraft, erneuerbare Energien, etc.. Je nach Land ist die vorhandene Mischung sehr unterschiedlich. So ist der CO_2-Ausstoß in der Schweiz und in Frankreich mit 0,005 bzw. 0,069 kg/kWh wesentlich geringer als in Deutschland mit 0,592 kg/kWh. Dies liegt daran, dass die elektrische Energie in der Schweiz zu 75% aus erneuerbaren Energien erzeugt wird und in Frankreich zu 86% aus Kernkraft. In Deutschland wird gut die Hälfte des Stroms durch Festbrennstoffe erzeugt. Bei dieselbetriebenen Güterzügen entstehen, analog zu LKW, Schadstoffe sowohl bei der Treibstoffproduktion als auch direkt am Verkehrsmittel. Analog zur Berechnung der LKW Emissionen fällt bei der Dieselkraftstoffproduktion 0,47kg CO_2 je produziertem Kilogramm Diesel an. Während des Betriebes wird zusätzlich 3,17kg CO_2 freigesetzt [40]. Kombiniert man diese Erkenntnisse, so entstehen folgende Formeln zur Bestimmung des CO_2-Ausstoßes in Tonnen in Abhängigkeit von transportierten Tonnen.

$$U(X, G, L, E) = C_e(X)W(G, L, E) \cdot 0,592kg/1000kg \tag{3.5}$$

bzw.

$$U(X, G, L, E) = C_d(X) \cdot W(G, L, E) \cdot (3,71 + 0,47)kg/1000kg \tag{3.6}$$

3.2.2 Tarife

Neben der Schadstoffberechnung werden auch die tatsächlichen Kosten für einen Transport auf der Straße oder Schiene berechnet. Da es sich hierbei ebenfalls um eine Funktion

handelt, die jeder Kante einen gewissen Betrag zuordnet, ist es prinzipiell möglich, diese Kostenfunktion für die Optimierung zu verwenden. Preistarife sind tendenziell regressiv, d. h. je mehr transportiert wird, desto geringer fallen die anteiligen Kosten je Ladeeinheit aus. Für die in dieser Arbeit untersuchten Algorithmen muss die Kostenfunktion jedoch eine lineare Funktion, also eine Ursprungsgerade, sein. Drückt man eine regressive Funktion durch eine lineare aus, indem man sie durch den Ursprung und den Punkt mit dem größten x-Wert legt, so liegt der Funktionswert dieser Funktion immer unter dem der regressiven.

LKW Tarif

Der folgende Tarif stellt einen realen Tarif eines Transportunternehmens dar. Es handelt sich dabei um einen regressiven Tarif mit einer komplizierten Berechnungsformel. Dieser Tarif wurde für die Verwendung im Algorithmus linearisiert und somit vereinfacht. Abbildung 3.2 beschreibt die originale Funktion, die bei gegebener Entfernung den Transport-

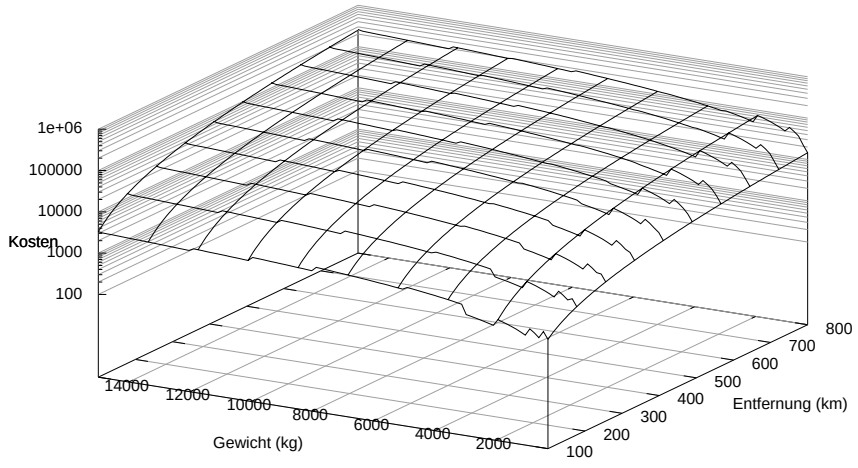

Abbildung 3.2: Kosten für den Transport per LKW.

preis in Euro in Abhängigkeit des zu transportierenden Gewichts beschreibt. Diese Fläche wurde zunächst in Bezug auf das Gewicht diskretisiert und anschließend durch eine lineare Funktion approximiert, um sie als Kostenfunktion im Algorithmus einsetzen zu können.

Güterzug Tarif

Der hier verwendete Tarif stellt die aktuelle Preisliste 2009 der DB Schenker Rail Deutschland AG dar [10]. Bei der Kostenbetrachtung werden Zusatzleistungen aller Art der Einfachheit halber nicht berücksichtigt. Die Struktur des Tarifs für 2-achsige Wagen mit einer

maximalen Zuladung von bis zu 75 Tonnen ist in Abbildung 3.3 abgebildet. Für eine Benutzung in einem linearen Modell müssen diese Funktionen entsprechend linearisiert werden. Dies geschieht, indem eine Gerade durch durch den ersten und letzten Funktionswert einer Funktion gelegt wird.

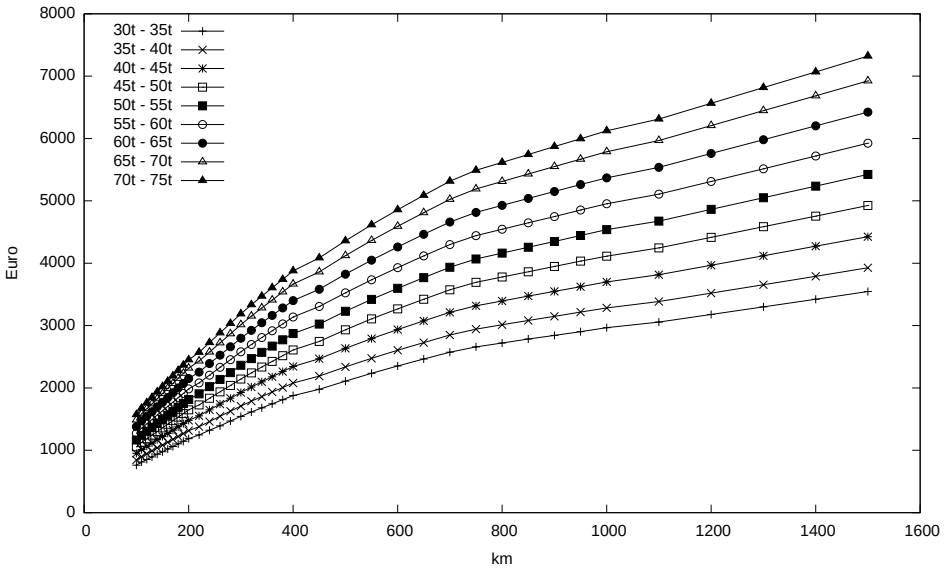

Abbildung 3.3: Kosten für den Transport eines unterschiedlich schwer beladenen, 2-achsigen Wagons über eine gewisse Distanz (nach [11]).

3.3 Netzwerkflussmodell

Um eine Instanz des oben beschriebenen Modells lösen zu können muss es zunächst in ein Netzwerkflussmodell überführt werden. Anschließend kann das Problem mittels eines geeigneten *minimum-cost multicommodity flow* Algorithmus berechnet werden.

Die Umwandlung geschieht in mehreren Schritten und ist ähnlich strukturiert, wie die Beschreibung der Elemente des obigen Modells. Das Ergebnis der Transformation ist ein gerichteter Graph $G = (V, E), n = |V|, m = |E|$ mit einer Kapazitätsfunktion $c : E \to \mathbb{R}^+$ sowie einer Kostenfunktion $b : E \to \mathbb{R}^+$. Weiterhin gibt es k Commodities, die jeweils eine Quelle s_i, eine Senke t_i und einen Bedarf d_i, $1 \leq i \leq k$ haben.

Mit dem nun entwickelten Netzwerkflussmodell ist es möglich, die reinen Transportkosten zu optimieren. Das heißt, alle Vorgänge in einer Quelle oder Senke, wie zum Beispiel Wartezeit, werden nicht betrachtet. Es ist jedoch möglich auch andere Fragestellungen abzubilden. Dazu sind lediglich simple Anpassungen nötig, die im letzten Abschnitt dieses Kapitels näher beschrieben werden.

3.3.1 Commodities

In einem ersten Schritt werden die Commodities in den Graphen integriert. Hierbei sind zwei Varianten zu unterscheiden. Für jeden Ort in der Menge der Start- und Zielorte $\mathcal{O} \cup \mathcal{D} = \{l_i^o, l_i^d | 1 \leq i \leq k\}$ wird eine Knotenschicht, ein sogenannter Layer, erstellt. Ein solcher Layer besitzt eine Menge von Knoten denen jeweils ein Zeitpunkt $t \in \mathbb{T} = [0; t_{max}]$ zugeordnet ist. Dieser Zeitpunkt repräsentiert eine Abfahrts- oder Ankunftszeit. Nach dem Einfügen aller Commodities enthält jeder Startort-Layer einen Knoten mit dem Zeitpunkt 0 und jeder Zielort-Layer einen Knoten mit dem Zeitpunkt t_{max}. Seien diese v^o respektive v^d für die Orte l^o und l^d. Diese Knoten sind gleichzeitig Superquellen bzw. Supersenken für alle Commodities, die an diesen Orten starten oder enden. Falls für eine Commodity c keine Abfahrts- oder Ankunftszeit festgesetzt wurde, so wird im Senken-Layer ein Knoten v_c erzeugt, der als Zeitattribut die Fahrtzeit zwischen l_c^o und l_c^d erhält. Anschließend wird eine Kante $e = (v^d, v^c)$ eingefügt. Diese Kante repräsentiert eine direkte Straßenverbindung. Für den Fall, dass eine Abfahrtszeit t_c^d gesetzt wurde, wird ein Knoten im Quell-Layer erzeugt, der die Abfahrts als Zeitattribut zugewiesen bekommt. Entsprechend wird ein Knoten im Senken-Layer erstellt, der als Zeitattribut die Summe aus Abfahrtszeit und Fahrtzeit $t_c^d + time(l_c^o, l_c^d)$ zugewiesen bekommt. Diese beiden Knoten werden verbunden und stellen die Straßenverbindung dar. In allen anderen Fällen wird wie im ersten Fall verfahren. Falls die Fahrtzeit größer ist, als die Differenz aus Ankunfts- und Abfahrtszeit, so wird das Modell als ungültig bezeichnet. Die Kapazität für diese Straßenverbindungskante entspricht der Summe der Bedarfe aller Commodities, die von Ort l^o aus starten und bei l^d enden. Damit bleibt gewährleistet, dass alle Güter prinzipiell per LKW transportiert werden könnten. Die Werte für die einzelnen Kostenfunktionen werden entsprechend gesetzt. Zu Beachten ist, dass Be- und Entladevorgänge in Quellen und Senken nicht modelliert werden.

3.3.2 Zugverbindungen

Die GVZ werden ebenfalls unter Benutzung von Knotenschichten modelliert. Jedoch müssen hier für eine Zugverbindung ebenfalls die Be- und Entladevorgänge beachtet werden. Die folgende, beispielhafte Konstruktion wird in Abbildung 3.4 veranschaulicht. Zunächst wird für eine Verbindung $r \in \mathcal{R}$ in der Schicht des GVZ $p = g_r^d$ ein Knoten v_p erzeugt, dessen Zeitpunkt der Differenz aus Abfahrts- und Beladezeit entspricht. Von diesem wird eine Kante $e_p = (v_p, v_d)$ erzeugt, deren Endknoten der tatsächliche Abfahrtsknoten ist. Mit dieser Kante wird der Umschlag zwischen Straßen und Schiene repräsentiert und besitzt eine Kapazität ist c_{max}. Die Kostenfunktionen werden entsprechend der Beladungszeit t_p^{lt} zugewiesen. Analog werden Knoten beim Ziel-GVZ $q = g_r^a$ erzeugt. Hier kommt der Zug am Knoten v_a zum Zeitpunkt t_r^a an und die Beladungszeit für die Handlingskante $e_q = (v_a, v_q)$ beträgt t_q^{ut}. Die tatsächliche Zugverbindungskante ist $e_r = (v_d, v_a)$. Die Kapazität dieser Kante wird auf $c(e_r) = c_r$, die Kosten für e_r entsprechend gesetzt.

Die Verbindung der GVZ mit den Quellen und Senken geschieht bei jeder Betrachtung einer Verbindung. Hierbei werden jedoch nur Quellen und Senken betrachtet, die innerhalb der Radien g^{pre} und g^{post} liegen. Diese Verbindungen bilden somit den Vor- und Nachlauf ab. Es wird nun exemplarisch das Vorgehen zur Erzeugung des Vorlaufs beschrieben. Die Einbindung des Nachlaufs geschieht analog dazu.
Betrachtet wird eine Zugverbindung r, die bereits eingefügt wurde. Der korrespondierende Knoten im Layer des GVZ $p = g_r^d$ sei v_p. Es wird ein zusätzlicher Knoten v_h erzeugt, der die Ankunft des LKW am GVZ p repräsentiert. Der Entladevorgang des LKW wird durch die Kante $e_h = (v_h, v_p)$ dargestellt. Diese Kante besitzt die Kapazität c_{max} und die Kostenwerte werden in Abhängigkeit von t_p^{ul} gebildet. In der Knotenschicht der Quelle l wird der Abfahrtsknoten v_l erzeugt. Dessen Zeitattribut entspricht der Abfahrtszeit des LKW, also der Differenz $t_r^d - t_p^{lt} - t_p^{ul} - time(l, r_g^d)$. Mit anderen Worten wird die Abfahrtzeit des Zuges um die Summe aus der Fahrtzeit des LKW von der Quelle zum GVZ, der Entladezeit des LKW und der Beladezeit des Zuges vermindert. An einer Quelle werden keine Handlingskanten eingefügt, da angenommen wird, dass diese Zeit intern verrechnet wird. Schließlich wird die Quelle mit dem GVZ über die Kante $e = (v_l, v_h)$ verbunden.

Mit dieser Konstruktion ist sichergestellt, dass eine Zugverbindung von Quellen aus erreichbar bzw. ein anschließender LKW Transport zu einer Senke möglich ist. In jedem Layer kommt jedes Zeitattribut nur genau einmal vor. Eventuell doppelt erstellte Knoten werden verschmolzen.

3.3.3 Postprocessing

Im letzten Schritt werden alle Knoten innerhalb der Knotenschichten miteinander verbunden. Hierzu werden die Knoten innerhalb einer Schicht aufsteigend nach ihrem Zeitattribut sortiert und in dieser Reihenfolge verbunden. Diese Kanten simulieren Wartezeit. Jede dieser Kanten besitzt eine Kapazität c_{max}. Für Kostenfunktionen entspricht die simulierte Wartezeit an einem GVZ der Differenz zweier aufeinander folgenden Knoten in einer Schicht. Bei Quellen und Senken ist die Wartezeit 0, da hier der späteste Abfahrts- und der früheste Ankunftszeitpunkt gesucht wird.

3.3.4 Komplexität

In Abhängigkeit der Anzahl der Quellen, Senken und GVZ kann die Größe des entstehenden Netzwerks angegeben werden. Die Anzahl der Kanten, die direkten Straßenverbindungen entsprechen, ist $O(|\mathcal{C}|)$. Jede Commodity erzeugt zu Beginn je einen Knoten im Quellen- und Senken-Layer, also insgesamt $O(|\mathcal{O}| + |\mathcal{D}|)$ viele. Jede Verbindung erzeugt maximal vier Handlingskanten und -knoten, je zwei Paare für die Zugverbindung und je eins für den Vor- und Nachlauf. Zusätzlich kommen die eigentliche Zugkante und für jede Quelle eine Vor- und für jede Senke eine Nachlaufkante hinzu. In allen beteiligten Layern kommen

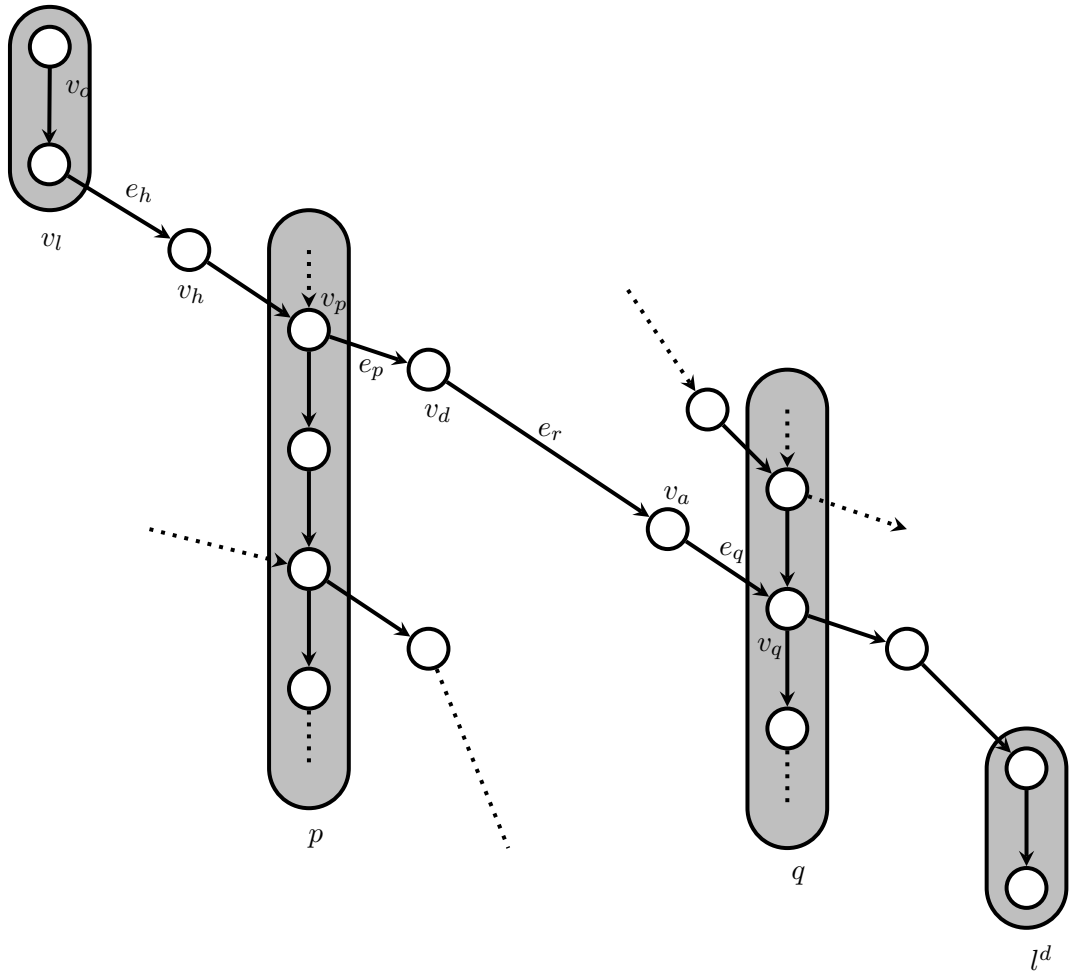

Abbildung 3.4: Konstruktion einer Zugverbindung zwischen p und g mit Vor- und Nachlauf. Gestrichelte Linien deuten weitere Elemente an. Alle restliche Kanten sind Transferkanten.

noch einmal je eine Transferkante und ein Transferknoten hinzu. Zusammengefasst gilt $n = O(|\mathcal{O}| + |\mathcal{D}| + |\mathcal{R}| \cdot (4 + 2 + |O| + |D|))$ und $m = O(|\mathcal{C}| + |\mathcal{R}| \cdot (4 + 1 + 2 \cdot (|O| + |D|)))$.

3.3.5 Varianten

Durch einfache Modifikationen des oben beschriebenen Netzwerks ist es möglich, weitere Fragestellungen beantworten zu können.

Zeithorizont

Es mag für bestimmte Fragestellungen ausreichend sein, wenn Sendungen innerhalb eines definierten Zeithorizonts ausgeliefert werden. In diesem Fall ist die Betrachtung der reinen Fahrtzeit vernachlässigbar und Wartezeiten in Quellen und Senken werden in Kauf genommen. Um diese Aussagen abzubilden, müssen bei der Konstruktion des Graphen einige

Anpassungen vorgenommen werden. So werden für eine Verbindung keine LKW Kanten mehr zu den Quellen erstellt. Die Verbindung von einer Quelle zu einem GVZ erfolgt nun unabhängig von der Verbindung. Sofern eine Quelle l^o im Vorlaufradius p^{pre} eines GVZ p ist, wird die Superquelle v_o durch eine Straßenkante über einen Handlingsknoten mit p verbunden. Die Wartezeit, die unter Umständen an der Quelle auftreten würde, wird somit in das jeweilige GVZ übertragen. Es werden weiterhin direkte Kanten zu den Senken erstellt.

Zeitfenster

Eine allgemeinere Form des obigen Punktes stellen Zeitfenster dar. Sollen Zeitfenster bei Quellen, GVZ oder Senken modelliert werden, so geschieht dies, indem bei der abschließenden Sortierung der Knoten innerhalb eines Layers Knoten, deren Zeitattribute nicht im Zeitfenster liegen, einfach gelöscht werden. Bei der Behandlung von Senken muss insbesondere darauf geachtet werden, dass die Supersenken entsprechend der Zeitfenster berücksichtigt werden. Entsprechendes gilt auf für Quellen. Liegt der Abfahrtszeitpunkt außerhalb eines Zeitfensters, so wird das Modell ungültig.

Kapitel 4

Algorithmus

Zur Lösung des im vorherigen Kapitel erarbeiteten Modells wird ein *multicommodity flow* Algorithmus eingesetzt. In Abschnitt 2.3.3 wurde bereits in das Thema eingeführt und die unterschiedlichen Herangehensweisen vorgestellt. In diesem Kapitel wird nun der in dieser Arbeit zum Einsatz kommende Algorithmus detailliert beschrieben. Dabei wird zunächst das allgemeine Framework zur Lösung des *maximum concurrent multicommodity* Problems von Garg und Könemann [32] vorgestellt. Anschließend werden die nötigen Veränderungen des Algorithmus beschrieben, um ein *maximum cost-bounded concurrent flow* Problem lösen zu können. Im Zuge dessen werden die Erweiterungen von Karakostas [44] eingeführt. Am Ende des Kapitels werden weitere Techniken und insbesondere verschiedene *single source shortest path* Algorithmen vorgestellt, welche die Laufzeit des Algorithmus noch einmal verbessern.

4.1 Grundalgorithmus

Zunächst wird der Algorithmus in seiner Grundform ausgeführt. Es handelt sich dabei um den *maximum concurrent flow* Algorithmus von Garg und Könemann [32]. Dieser Algorithmus ist ein $FPTAS$ (siehe Abschnitt 2.2), also ein Approximationsalgorithmus, dessen Lösung nicht weiter als Faktor $(1+\epsilon)$ vom Optimum entfernt ist und dessen Laufzeit sowohl polynomiell in der Eingabe, als auch in $1/\epsilon$ ist.

Dieser Rahmenalgorithmus lässt sich einfach in ein *maximum cost-bounded concurrent flow* Problem überführen und wurde zunächst von Fleischer [28] und schließlich von Karakostas [44] weiter verbessert. Letzterer wird im Rahmen dieser Arbeit implementiert.

4.1.1 Lineares Programm

Das allgemeine *maximum concurrent flow* Problem ist wie folgt definiert. Gegeben sei ein Graph $G = (V, E)$ mit einer Kapazitätsfunktion $c : E \to \mathbb{R}^+$ sowie k Commodities. Eine Commodity i besteht aus einer Quelle s_i, einer Senke t_i und einem Bedarf d_i. Ziel des

Algorithmus ist es, das größte λ zu finden, sodass ein Mehrgüterfluss existiert, der λd_j Flusseinheiten von Commodity j durch das Netzwerk schickt. Anders formuliert soll der kleinste Flusswert einer Commodity über alle Commodities maximiert werden.

Dieses Problem kann in einer Pfad-basierten Formulierung als lineares Programm P_{mcf} dargestellt werden. Dabei bezeichnet $\mathcal{P}$ die Menge aller Pfade. $\mathcal{P}_e$ und $\mathcal{P}_j$ schränken die Pfade auf diejenigen ein, die eine Kante $e \in E$ enthalten oder von Commodity j benutzt werden.

$$
\begin{aligned}
\max \quad & \lambda && (P_{mcf})\\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}_e} x(p) \leq c(e) \quad \forall e \in E \\
& \sum_{p \in \mathcal{P}_j} x(p) \geq \lambda d_j \quad \forall 1 \leq j \leq k \\
& x \geq 0
\end{aligned}
$$

Das duale Problem D_{mcf} besitzt für jede Kante $e \in E$ eine Variable $l(e)$ und für jede Commodity j eine Variable z_j.

$$
\begin{aligned}
\min \quad & D(l,\phi) \stackrel{\text{def}}{=} \sum_{e \in E} c(e) l(e) && (D_{mcf})\\
\text{s.t.} \quad & \sum_{e \in p} l(e) \geq z_j \quad \forall 1 \leq j \leq k, \forall p \in \mathcal{P}_j && (4.1)\\
& \sum_{j=1}^{k} d_j z_j \geq 1 && (4.2)\\
& l \geq 0 && (4.3)\\
& z \geq 0 && (4.4)
\end{aligned}
$$

Eine Variable $l(e), e \in E$ wird im Folgenden als *Länge* der Kante e bezeichnet. Für eine Funktion $l : E \to \mathbb{R}^+$ kann z_j als die Länge des kürzesten Pfades von s_j nach t_j unter der Kostenfunktion $l(e)$ angesehen werden, wobei diese Länge als $dist_j(l)$ bezeichnet wird. Sei $\alpha(l) \stackrel{\text{def}}{=} \sum_j d_j \cdot dist_j(l)$. Das duale Problem kann demnach auch als Zuordnungsproblem betrachtet werden, bei dem eine Belegung der Variablen $l(e)$ gesucht ist, sodass $D(l)/\alpha(l)$ minimiert wird. Die Äquivalenz zur Zielfunktion $D(l,\phi)$ wird deutlich, wenn man beachtet, dass α der Ungleichung (4.2) entspricht. Zusammen mit der Ungleichung (4.1) folgt, dass alle z_j minimiert werden und α als kleinsten Wert 1 annehmen kann. Daraus ergibt sich die obige, alternative Zielfunktion $D(l)/\alpha(l)$, dessen Minimum β sei.

Das oben vorgestellte lineare Programm P_{mcf} lässt sich grundsätzlich in polynomieller Zeit lösen, da die Anzahl der Ungleichungen polynomiell ist. Die Zahl der Ungleichungen

wächst jedoch sehr schnell, sodass eine effiziente Lösung des entstehenden Gleichungssystems bei großen Instanzen nicht effizient möglich ist.

4.1.2 Approximationsalgorithmus

Der Approximationsalgorithmus, der D_{mcf} löst, ist in Phasen, Iterationen und Schritte unterteilt. Dabei durchläuft der Algorithmus eine Anzahl von Phasen, jede Phase ist in Iterationen unterteilt und jede Iteration besteht aus einer Menge von Schritten. Zu Beginn des Algorithmus werden alle Längen-Variablen $l(e)$ auf $\delta/c(e)$ gesetzt, wobei δ im Folgenden näher spezifiziert wird. In der j-ten Iteration der i-ten Phase werden in einer Folge von Schritten alle d_j Flusseinheiten von Commodity j durch das Netzwerk geschickt. Dies geschieht jeweils auf einem kürzesten Pfad bezüglich der Längen $l(e)$. Sollte es aufgrund der Kapazitäten dieses Pfads nicht möglich sein, den gesamten Bedarf zu erfüllen, so wird ein Fluss, welcher der größten Kapazität auf diesem Pfad entspricht, durch das Netzwerk geschickt. Die j-te Iteration endet, wenn der komplette Bedarf d_j geroutet worden ist. An diesem Punkt werden die Längen $l(e)$ geeignet verändert. Algorithmus 4.1 fasst die Vorgehensweise zusammen.

1: Initialisierung $\forall e : l(e) = \delta/c(e), x \equiv 0$
2: **while** $D(l) < 1$ **do**
3: **for** $j = 1$ to k **do**
4: $d'_j \leftarrow d_j$
5: **while** $D(l) < 1$ and $d'_j > 0$ **do**
6: $P \leftarrow$ kürzester Pfad in $\mathcal{P}_j$ in Bezug auf l
7: $c \leftarrow \min\{d'_j, \min_{e \in P} c(e)\}$
8: $d'_j \leftarrow d'_j - u$
9: $x(P) \leftarrow x(P) + u$
10: $\forall e \in P : l(e) \leftarrow l(e)(1 + \epsilon \frac{c}{c(e)})$
11: **end while**
12: **end for**
13: **end while**

Algorithmus 4.1: Maximum Concurrent Multicommodity Flow Algorithmus für $\beta \geq 1$ (Garg et al. [32])

4.1.3 Analyse

In der j-ten Iteration der i-ten Phase sei $l_{i,j}^{s-1}$ die Länge des kürzesten Weges von s_j nach t_j vor dem s-ten Schritt und sei $P_{i,j}^{s}$ ein Pfad dieser Länge. Der noch nicht verschickte Restbedarf sei $d_{i,j}^{s-1}$. In Schritt s werden dann $f_{i,j}^{s} = \min\{c, d_{i,j}^{s-1}\}$ Flusseinheiten, wobei c die Pfadkapazität darstellt, über diesen Pfad geschickt. Nachdem der gesamte Bedarf d_j

in p Schritten versandt wurde, wird $l_{i,j}(e) = l_{i,j-1}(e)(1 + \epsilon f_{i,j}/c(e))$ gesetzt und der Wert der Zielfunktion kann wie folgt abgeschätzt werden:

$$D(l_{i,j}^p) = \sum_{e \in E} l_{i,j}(e)c(e) \tag{4.5}$$

$$\leq D(l_{i,j}^0) + \epsilon \sum_{e \in E} l_{i,j} f_{i,j}(e) \tag{4.6}$$

$$\leq D(l_{i,j}^0) + \epsilon \cdot d_j dist_j(l_{i,j}^p) \tag{4.7}$$

Dabei sind die Werte der Variablen $l(e)$ zu Beginn der $(i+1)$-ten Phase identisch zu denen am Ende der i-ten Phase: $l_{i+1,0} = l_{i,k}$.
Sei die Längenfunktion am Ende einer Phase, bestehend aus k Iterationen, abgekürzt durch $l_{i,k} = i$. Aufgrund der Tatsache, dass die Werte der Variablen $l(e)$ im Verlauf des Algorithmus monoton steigen, also $dist_j(l_{i,j-1}) \leq dist_j(l_{i,k})$, kann D weiter abgeschätzt werden:

$$D(i) = D(l_{i,k}) \leq D(l_{i,0}) + \epsilon \sum_{j=1}^{k} d_j dist_j(l_{i,k}) \tag{4.8}$$

$$\leq D(i-1) + \epsilon\alpha(i) \tag{4.9}$$

Aufgrund der Tatsache, dass $D(i)/\alpha(i) \geq \beta$ ist ergibt sich

$$D(i) \leq D(i-1)\left(1 - \frac{\epsilon}{\beta}\right)^{-1} \tag{4.10}$$

Durch die initiale Belegung der $l(e)$ mit $\delta/c(e)$ ergibt sich $D(0) = \sum_{e \in E} l(e)c(e) = m\delta$. Zusammen mit Gleichung (4.10) ergibt sich für $i \geq 1$

$$D(i) \leq m\delta\left(1 - \frac{\epsilon}{\beta}\right)^{-i} \tag{4.11}$$

$$= \frac{m\delta}{1 - \frac{\epsilon}{\beta}}\left(\frac{\beta - \epsilon}{\beta}\right)^{-(i-1)} \tag{4.12}$$

$$= \frac{m\delta}{1 - \frac{\epsilon}{\beta}}\left(\frac{\beta}{\beta - \epsilon}\right)^{i-1} \tag{4.13}$$

$$= \frac{m\delta}{1 - \frac{\epsilon}{\beta}}\left(1 + \frac{\epsilon}{\beta - \epsilon}\right)^{i-1} \tag{4.14}$$

$$\leq \frac{m\delta}{1 - \frac{\epsilon}{\beta}}\left(e^{\frac{\epsilon}{\beta - \epsilon}}\right)^{i-1} \tag{4.15}$$

$$= \frac{m\delta}{1 - \frac{\epsilon}{\beta}} e^{\frac{\epsilon(i-1)}{\beta - \epsilon}} \tag{4.16}$$

$$\leq \frac{m\delta}{1 - \epsilon} e^{\frac{\epsilon(i-1)}{\beta(1-\epsilon)}} \tag{4.17}$$

Ungleichung (4.15) folgt aus der bekannten Abschätzung der Exponentialfunktion $e^x \geq 1 + x$ und Ungleichung (4.17) ergibt sich aus der Annahme, dass $\beta \geq 1$.
Der Algorithmus terminiert, sobald $D(t) \geq 1$. Daraus ergibt sich

$$1 \leq D(t) \leq \frac{m\delta}{1-\epsilon} e^{\frac{\epsilon(i-1)}{\beta(1-\epsilon)}} \tag{4.18}$$

Dies wiederum impliziert

$$\frac{\beta}{t-1} \leq \frac{\epsilon}{(1-\epsilon)\ln\frac{1-\epsilon}{m\delta}} \tag{4.19}$$

Zur Abschätzung der Güte muss der Wert der primalen Lösung λ abgeschätzt werden. Hierfür ist es möglich eine untere Schranke anzugegeben.

4.1.1 Satz. $\lambda > \frac{t-1}{\log_{1+\epsilon} 1/\delta}$

Beweis. In den ersten $t-1$ Phasen wurden $(t-1)d_j$ Flusseinheiten für jede Commodity j durch das Netzwerk geschickt. Allerdings verletzt der so erzeugte Fluss wahrscheinlich die Kapazitätsbeschränkungen.

Jedes Mal, wenn Fluss über eine Kante e fließt, welcher der Kapazität der Kante $c(e)$ entspricht, wird $l(e)$ um einen Faktor von mindestens $(1+\epsilon)$ erhöht. Sei diese Anzahl x. Zu Beginn des Algorithmus ist $l(e) = \delta/c(e)$ und nach $t-1$ Phasen ist $l_{t-1,k} < 1/c(e)$. Letzteres begründet sich daraus, dass $D(t) \geq 1$ und somit $D(t-1) < 1$. Der gesamte Fluss f_e durch e in den ersten $t-1$ Phasen kann also wie folgt abgeschätzt werden:

$$\frac{1}{c(e)} > \frac{\delta}{c(e)}(1+\epsilon)^{\frac{x}{c(e)}} \tag{4.20}$$

$$\frac{1}{\delta} > (1+\epsilon)^{\frac{x}{c(e)}} \tag{4.21}$$

$$\log_{1+\epsilon}\frac{1}{\delta} > \frac{x}{c(e)} \tag{4.22}$$

$$c(e)\log_{1+\epsilon}\frac{1}{\delta} > x \tag{4.23}$$

Skaliert man nun den gesamten Fluss mit $\log_{1+\epsilon} 1/\delta$, so erhält man einen gültigen primalen Fluss und der Satz ist damit bewiesen. □

4.1.2 Theorem. *Algorithmus 4.1 berechnet eine* $(1+\epsilon)$*-Approximation für das maximum concurrent flow Problem für* $\beta \geq 1$.

Beweis. Um die Güte beweisen zu können, muss das Verhältnis γ von primaler und dualer Lösung abgeschätzt werden. Nach Satz 4.1.3 hat die primale Lösung einen Wert von $\frac{t-1}{\log_{1+\epsilon} 1/\delta}$ und nach der Definition zu Beginn des Kapitels ist der Wert der dualen Lösung β. Unter Zuhilfenahme von Gleichung (4.19) ergibt sich

$$\gamma < \frac{\epsilon \log_{1+\epsilon} 1/\delta}{(1-\epsilon)\ln\frac{1-\epsilon}{m\delta}} = \frac{\epsilon}{(1-\epsilon)\ln(1+\epsilon)} \frac{\ln 1/\delta}{\ln\frac{1-\epsilon}{m\delta}} \tag{4.24}$$

Setzt man $\delta = (\frac{m}{1-\epsilon})^{-\frac{1}{\epsilon}}$, so fällt der letzte Bruch zu $(1-\epsilon)^{-1}$ zusammen:

$$\frac{\ln \frac{1}{\delta}}{\ln \frac{1-\epsilon}{m\delta}} = (1-\epsilon)^{-1} \tag{4.25}$$

$$(\epsilon - 1)\ln\delta = \ln\frac{1-\epsilon}{m\delta} \tag{4.26}$$

$$\delta^{\epsilon-1} = \frac{1-\epsilon}{m\delta} \tag{4.27}$$

$$\delta = \Big(\frac{m}{1-\epsilon}\Big)^{-1/\epsilon} \tag{4.28}$$

und man erhält

$$\gamma \leq \frac{\epsilon}{(1-\epsilon)^2\ln(1+\epsilon)} \leq \frac{\epsilon}{(1-\epsilon)^2(\epsilon-\epsilon^2/2)} \leq (1-\epsilon)^{-3} \tag{4.29}$$

Bleibt noch ϵ so zu belegen, dass $(1-\epsilon)^{-3}$ dem gewünschten Approximationsfaktor $1+\omega$ entspricht. Setzt man beide Terme gleich, so erhält man

$$\epsilon = \frac{\sqrt[3]{-\omega^2 - 2\omega - 1}}{1+\omega} + 1 \tag{4.30}$$

□

4.1.3 Lemma. *Algorithmus 4.1 ist ein FPTAS.*

Beweis. Um die Aussage der Güte von Theorem 4.1.2 auch für $\beta < 1$ zu zeigen, wird eine Skalierungsmethode nach Garg und Könemann [32] angewendet. Nach Anwendung des schwachen Dualitätssatzes gilt:

$$1 \leq \gamma \leq \frac{\beta}{t-1}\log_{1+\epsilon}\frac{1}{\delta} \tag{4.31}$$

Daraus folgt, dass für die Anzahl der Phasen $t < 1+\beta\log_{1+\epsilon} 1/\delta$ und somit $t = \lceil \frac{\beta}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil$ gilt.

Die Laufzeit hängt demnach von β ab und kann wie folgt verringert werden. Sei ξ_j der maximale Fluss, der durch das Netzwerk geschickt werden kann, wenn ausschließlich Commodity j betrachtet wird. Sei $\xi = \min_j\{\xi_j/d_j\}$. ξ ist somit eine obere Schranke für die Lösung, da im besten Fall alle Commodities vollständig geroutet werden können. Eine untere Schranke stellt der Fluss dar, der von jeder Commodity nur $1/k$ durch das Netzwerk schickt. Daraus folgt, dass $\xi/k \leq \beta \leq \xi$. Multipliziert man nun die Bedarfe aller Commodities mit ξ/k so gilt $\beta \geq 1$.

Für diese Abschätzung müssen k *maximum flow* Berechnungen durchgeführt werden. Fleischer [28] hat dies auf kürzeste Wege Berechnungen übertragen. Danach ist es ausreichend, wenn eine m-Approximation der ξ berechnet wird. Daraus folgt, dass, falls $\hat{\xi}_j \geq \frac{1}{m}\xi_j$, die obere Grenze für β auf km anwächst. Da alle Flusspfade für eine Commodity j in höchstens m Pfade aufgesplittet werden können, stellt ein Pfad zwischen s_j und t_j mit maximaler Kapazität die gewünschte m-Approximation dar. Durch einen modifizierten kürzesten

Wege Algorithmus lassen sich die Pfade aller ξ_j in Zeit $O(\min\{n,k\}m\log n)$ berechnen. Die Modifikation bezieht sich darauf, dass an einem Knoten die bisher größte vorkommende Kapazität als Label gespeichert wird.

Zur weiteren Laufzeitverbesserung schlagen Garg und Könemann [32] folgendes Verfahren vor. Der Algorithmus wird gestartet. Terminiert er nicht nach $T = 2\lceil\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil = O(\epsilon^{-2}\log m)$ Phasen, so gilt $\beta \geq 2$. Die Bedarfe aller Commodities können daher verdoppelt werden, sodass anschließend wieder $\beta \geq 1$ gilt. Dieses Verfahren wird so oft wiederholt, bis der Algorithmus schließlich stoppt. Da β alle T Phasen halbiert wird, beträgt die gesamte Anzahl an Phasen $T\log km$.

Dieser Wert kann noch weiter gesenkt werden. Hierzu wird eine Idee von Plotkin et al. [56] aufgegriffen. Zunächst wird mit dem obigen Verfahren eine 2-Approximation berechnet, deren Lösung $\tilde{\beta}$ ist. Somit gilt $\beta \leq \tilde{\beta} \leq 2\beta$. Multipliziert man nun alle Bedarfe mit $\tilde{\beta}/2$, so benötigt man nur noch T zusätzliche Phasen, um eine ϵ-Approximation zu erhalten, da nun $1 \leq \beta \leq 2$ gilt. Die Anzahl der Phasen für die 2-Approximation ist somit $O(\log m\log km)$.

Zusammengefasst ergibt sich also folgende Gesamtlaufzeit: Die Anzahl an Iterationen ist k. Insgesamt werden $T + O(\log m\log km)$ Phasen benötigt. Daraus ergibt sich eine Gesamtanzahl an Iterationen von $O(k\log m(\log km + \epsilon^{-2}))$. Die Länge $l(e)$ einer Kante e wird in jedem Schritt einer Iteration, außer dem letzten, um mindestens Faktor $(1+\epsilon)$ erhöht. Analog zu Satz 4.1.3 ist die Anzahl dieser Schritte höchstens $m\log_{1+\epsilon}\frac{1+\epsilon}{\delta} = O(\epsilon^{-2}m\log m)$. Zusammen ergibt sich eine Gesamtanzahl der Schritte von $O(k\log m(\log km+\epsilon^{-2}) + \epsilon^{-2}m\log m) = \tilde{O}(\epsilon^{-2}(k+m))$[1]. □

4.1.4 Theorem. *Der Algorithmus 4.1 berechnet eine $(1+\epsilon)$ Approximation des Maximum Concurrent Flow Problem in $O(\omega^{-2}(k\log km+m)\log m\cdot T_{sp})$ wobei T_{sp} der Zeit entspricht, die benötigt wird um einen kürzesten Weg zwischen zwei Knoten in einem Graphen zu berechnen.*

4.2 Verbesserungen von Karakostas

Karakostas [44] hat den obigen Ansatz weiter verbessert, indem er bei der Abschätzung der Approximationsgüte genauer gerechnet und vor allem die Abhängigkeit der Laufzeit in Bezug auf k reduziert hat. Dabei präsentiert er zwei Versionen seines Algorithmus: eine implizite und eine explizite. Beide Varianten stellen ein FPTAS für das *maximum concurrent multicommodity flow* Problem dar. Im impliziten Fall wird jedoch darauf verzichtet, die Pfade, auf der Fluss einer bestimmten Commodity fließt, zu speichern. Dies legitimiert die Zusammenfassung von Commodities, die in Quelle und Senke übereinstimmen. Daraus folgt, dass $k = O(\min\{n^2, m\})$ gilt und von einer Quelle aus nur noch $O(n)$ Commodities ausgehen können. Diese Bündelung resultiert in einem Algorithmus, der bis auf

[1] $\tilde{O}$ entspricht O, vernachlässigt jedoch logarithmische Faktoren

logarithmische Faktoren, nicht mehr von k abhängt. Prinzipiell besteht die Möglichkeit, die entstehenden Pfade auch in der impliziten Variante auszuweisen. Allerdings müssten dafür alle kürzesten Wege-Bäume, die in jedem Schritt entstehen, gespeichert werden. Für große Graphen stellt dies jedoch eine enorme Datenmenge dar. In der expliziten Variante wird jede Commodity einzeln durch das Netzwerk geschickt. Dies ermöglicht es, die Wege einer Commodity einfach zu protokollieren. Der Preis hierfür ist allerdings eine schlechtere Laufzeit, die der von Fleischer [28] entspricht.

4.2.1 Implizite Variante

Analyse

Die Analyse dieser Variante ist nahezu identisch zu der des Rahmenalgorithmuses in Abschnitt 4.1.3. Eine Ausnahme ist der Beweis zu Satz 4.1.3. Hier wird argumentiert, dass am Ende des Algorithmus $l_t < (1+\epsilon)/u(e)$ gilt. Dies begründet sich darin, dass in jedem Schritt der letzten Phase die Länge $l(e)$ einer Kante um mindestens einen Faktor von $(1+\epsilon)$ erhöht wird und der Algorithmus abbricht, sobald $D(l_{t-1}) \geq 1$. Daraus ergibt sich ein Skalierungsfaktor von $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ und es folgt durch die weiteren Berechnungen, dass $\delta = 1/(1+\epsilon)^{(1-\epsilon)/\epsilon} \cdot ((1-\epsilon)/m)^{(1/\epsilon)}$ gilt.

Die entscheidendere Verbesserung ist die Eliminierung des Faktor k. Dies wird erreicht, indem Commodities bezüglich Quellen gruppiert werden. Dies ermöglicht es in einem Schritt alle Commodities mit gleicher Quelle gleichzeitig durch das Netzwerk zu schicken. Hierfür muss allerdings Algorithmus 4.1 angepasst werden. Das Ergebnis ist Algorithmus 4.2.

Zu beachten ist, dass in einem Schritt alle Commodities einer Quelle betrachtet werden, deren Restbedarf größer 0 ist. σ stellt sicher, dass nur soviel Fluss durch das Netzwerk geschickt wird, wie es die Kapazitäten zulassen. Für das Update der Längen $l(e)$ muss nun der summierte Fluss aller, in diesem Schritt verschickten, Bedarfe berücksichtigt werden.

Der Korrektheitsbeweis ändert sich nicht, da auch dieser Algorithmus in jedem Schritt, außer dem letzten, die Länge $l(e)$ einer Kante e um mindestens Faktor $(1+\epsilon)$ erhöht.

Laufzeit

Bei der Berechnung der Laufzeit wirkt sich die Bündelung der Commodities entscheidend aus. Karakostas schätzt die Laufzeit zur Berechnung der ξ nach der Methode von Fleischer [28] ab. Leider hat sich bei Karakostas ein Fehler in der Analyse eingeschlichen. Bei anschließender Berechnung des Flusses mittels des oben vorgestellten Verfahrens über eine 2-Approximation, entspricht die Anzahl der Phasen zur Berechnung der 2-Approximation nicht $O(\log k \log m)$, sondern $O(\log m(\log k \log km) = O(\log^2 m)$. Daraus ergeben sich zwar andere Werte in der Analyse mit O Abschätzung, jedoch ändert sich das um logarithmische Faktoren verminderte Ergebnis $\tilde{O}$ nicht. Die Anzahl aller Phasen beträgt also

Eingabe: Graph $G = (V, E)$, Kapazitäten $u(e)$, j Commodities (s_j, t_j) mit Bedarfen d_j, ϵ
Ausgabe: Primale Lösung x, λ

1: Initialisierung $\forall e : l(e) = \delta/u(e), x \equiv 0$
2: **while** $D(l) < 1$ **do**
3: **for** $i = 1$ to $|\mathcal{S}|$ **do**
4: $d'(c_q) := d(c_q), q = 1, \ldots, r$
5: **while** $D(l) < 1$ and $d'(c_q) > 0$ for some q **do**
6: $P_{c_q} \leftarrow$ kürzester Pfad in $\mathcal{P}_{c_q}$ in Bezug auf l, $d'(c_q) > 0$
7: $f_{c_q} \leftarrow d'(c_q)$, $d'(c_q) > 0$
8: $\sigma \leftarrow \max\{1, \max_{e \in P_{c_1} \cup \ldots \cup P_{c_r}} \{\frac{\sum_{c_q : e \in P_{c_q}} f_{c_q}}{u(e)}\}\}$
9: $\left.\begin{array}{l} f_{c_q} \leftarrow f_{c_q}/\sigma \\ d'(c_q) = d'(c_q) - f_{c_q} \\ x(P_{c_q}) \leftarrow x(P_{c_q}) + f_{c_q} \end{array}\right\}$ $d'(c_q) > 0$
10: $l(e) \leftarrow l(e)(1 + \epsilon \cdot \frac{\sum_{c_q : e \in P_{c_q}} f_{c_q}}{u(e)})$, $\forall e \in P_{c_1} \cup \ldots \cup P_{c_r}$
11: **end while**/* Ende eines Schritte */
12: **end for**/* Ende einer Iteration */
13: **end while**/* Ende einer Phase */
14: $x(P) = x(P)/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$, $\forall P$
15: $\lambda = \min_i \frac{\sum_{P \in \mathcal{P}_i} x(P)}{d_i}$

Algorithmus 4.2: Maximum Concurrent Flow Algorithmus (implizit) für $\beta \geq 1$ (Karakostas [44])

$O(\log^2 m + \epsilon^{-2} \log m)$. Die Anzahl der Iterationen ist demnach $O(\min\{n, k\} \log m(\log m + \epsilon^{-2})) = \tilde{O}(\epsilon^{-2}(m))$.

4.2.1 Lemma. *Jeder Schritt umfasst einen Aufruf eines kürzesten Wege Algorithmus, dessen Zeit $O(n \log n + m)$ beträgt. Alle weiteren Operationen benötigen Zeit $O(m)$, sodass die Gesamtlaufzeit eines Schritts mit $O(m + n \log n)$ angegeben werden kann.*

Beweis. Der Algorithmus von Dijkstra unter Verwendung von Fibonacci-Heaps berechnet das kürzeste Wege Problem und besitzt die angegebene Laufzeit (siehe z. B. Cormen et al. [22]).

Alle Berechnungen, die in einem Schritt ausgeführt werden, benötigen Zeit $O(k) = O(m)$. Einzige Ausnahme ist die Berechnung von σ in Zeile 8 des Algorithmus 4.2, die in Zeit $O(n)$ erfolgen kann. Um dies zu zeigen, wird die Berechnung

$$F(e) = \sum_{c_q : e \in P_{c_q}} f_{c_q}, \forall e \in P_{c_1} \cup \ldots \cup P_{c_r}$$

näher untersucht. Sei P_{c_q} der kürzeste Pfad zur Senke von Commodity c_q. Grundlage für diese Berechnung stellt ein kürzester-Wege-Baum der aktuellen Quelle dar. O. B. d. A.

stellen die Blätter des Baums die Senken dar. Andernfalls werden künstliche Blätter eingefügt. Für jede Commodity c_q soll der Fluss f_{c_q} durch das Netzwerk, also den Baum, geschickt werden. Dazu wird wie folgt vorgegangen: Zunächst wird jedem Blatt der jeweilige Flusswert f_{c_q} zugewiesen. Nun wird ein Knoten v gesucht, sodass alle Kinder von v einen Flusswert besitzen. Der Flusswert von v wird dann auf die Summe der Flusswerte der Kinder gesetzt. Dieses Vorgehen wird solange fortgeführt, bis alle Knoten einen Wert zugewiesen bekommen haben. Eine Kante $e = (v, w)$ besitzt dann den Flusswert von Knoten w. Diese Kantenwerte können bereits bei der Ausführung der Prozedur zugewiesen werden.

Die Laufzeit beträgt unter Verwendung einer Queue in Zeit $O(n)$. Am Ende stehen die Werte aller $F(e)$ zur Verfügung und können anschließend ebenfalls zum Update der Längenvariablen $l(e)$ benutzt werden. □

Zusammenfassend beträgt die Anzahl der Schritte also $O(\min\{k, n\} \log m(\log mk + \epsilon^{-2}) + \epsilon^{-2} m \log m) = \tilde{O}(m\epsilon^{-2})$. Alle Berechnungen, die für einen Schritt benötigt werden, sind in Zeit $O(m)$ möglich. Alle ξ_j können in $O(\min\{k, n\} m \log m) = \tilde{O}(m)$ berechnet werden woraus sich eine Gesamtlaufzeit von $\tilde{O}(\epsilon^{-2} m^2)$ ergibt.

4.2.2 Explizite Variante

Die explizite Variante speichert, im Gegensatz zur impliziten, zur Laufzeit die Pfade, auf denen Fluss einer Commodity fließt. Bezeichne im Folgenden $x_e(q)$ den Fluss auf der Kante e, der zu Commodity q gehört.

Es ist möglich Algorithmus 4.2 so abzuwandeln, dass er alle $x_e(q), \forall q$ zur Laufzeit abspeichert. Für die anschließende Analyse ist festzuhalten, dass, solange das Grundprinzip der Analyse beibehalten wird, sich die Korrektheit nicht ändert. Dieses Prinzip besteht darin, in einem Schritt so viel Fluss durch das Netzwerk zu schicken, dass entweder eine Kante saturiert wird, oder alle Bedarfe der aktuell betrachteten Commodities erfüllt werden. Kanten werden saturiert, indem in einer Iteration so lange Bedarfe von Commodities verschickt werden, bis dies für eine Commodity nicht mehr möglich ist. Die Kosten für dieses Verfahren werden wie folgt umgelegt. Die Pfade, die entstehen, wenn der komplette Bedarf einer Commodity geroutet werden kann, werden der Commodity selbst angerechnet. Pfade, die zu Commodities gehören, deren Bedarf nicht vollständig verschickt werden konnte, werden der saturierten Kante zugeschrieben. Daraus folgt, dass eine Iteration j mit genau der gleichen Anzahl an Pfaden belastet wird, wie die Commodities, die s_j als Quelle besitzen. Eine Phase hingegen wird mit den Kosten der saturierten Kanten belastet, also für genau k Pfade. Darüber hinaus wird eine saturierte Kante für genau einen Pfad belastet. Algorithmus 4.3 stellt dieses Vorgehen dar.

Eingabe: Graph $G = (V, E)$, Kapazitäten $u(e)$, j Commodities (s_j, t_j) mit Bedarfen d_j, ϵ
Ausgabe: Primale Lösung x, λ

1: Initialisierung $\forall e : l(e) = \delta/u(e), x \equiv 0$
2: **while** $D(l) < 1$ **do**
3: **for** $i = 1$ to $|\mathcal{S}|$ **do**
4: $d'(c_q) := d(c_q), q = 1, \ldots, r$
5: **while** $D(l) < 1$ and $d'(c_q) > 0$ for some q **do**
6: $P_{c_q} \leftarrow$ kürzester Pfad in $\mathcal{P}_{c_q}$ in Bezug auf l, $d'(c_q) > 0$
7: $u'(e) \leftarrow u(e), \forall e \in P_{c_1} \cup \ldots \cup P_{c_r}$
8: **for** $q = 1$ to r **do**
9: $c \leftarrow \min_{e \in P_{c_q}} u'(e)$
10: **if** $d'(c_q) \leq c$ **then**
11: $\left.\begin{array}{l} x_e(c_q) \leftarrow x_e(c_q) + d'(c_q) \\ u'(e) \leftarrow u'(e) - d'(c_q) \end{array}\right\} \quad \forall e \in P_{c_q}$
12: $f_{c_q} \leftarrow d'(c_q)$
13: $d'(c_q) \leftarrow 0$
14: **else**
15: $x_e(c_q) \leftarrow x_e(c_q) + c, \forall e \in P_{c_q}$
16: $f_{c_q} \leftarrow c$
17: $d'(c_q) \leftarrow d'(c_q) - c$
18: **break**
19: **end if**
20: **end for**
21: $l(e) \leftarrow l(e)(1 + \epsilon \cdot \frac{\sum_{c_q : e \in P_{c_q}} f_{c_q}}{u(e)}), \forall e \in P_{c_1} \cup \ldots \cup P_{c_r}$
22: **end while**/* Ende eines Schritte */
23: **end for**/* Ende einer Iteration */
24: **end while**/* Ende einer Phase */
25: $x_e(q) = x_e(q)/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}, \forall e \in E, q = 1, \ldots, k$
26: $\lambda = \min_i \frac{flow(s_i, t_i)}{d(i)}$

Algorithmus 4.3: Maximum Concurrent Flow Algorithmus (explizit) für $\beta \geq 1$ (Karakostas [44])

Korrektheit

Der einzige Unterschied zur impliziten Variante besteht in der Art, wie der Fluss durch das Netzwerk geschickt wird. Die Korrektheit ändert sich daher nicht und vollzieht sich analog zur Analyse in 4.2.1.

Laufzeit

Da sich die Modifikationen lediglich auf die Verteilung des Flusses beschränken, ändern sich die Abschätzungen, die in der Analyse 4.2.1 der impliziten Version bezüglich der Anzahl der Phasen und Iterationen gemacht wurden, nicht. Die Anzahl der Phasen beträgt somit $O(\log^2 m + \epsilon^{-2} \log m)$ und die der Iterationen $\tilde{O}(\epsilon^{-2}(m))$.

Einzig die Anzahl der Schritte ändert sich. Wie zuvor enden alle Schritte einer Iteration, außer der letzte, mit der Saturierung einer Kante. Die Abschätzung für die Anzahl der saturierten Kanten ist identisch zu der aus Satz 4.1.3, also $\tilde{O}(\epsilon^{-2}m)$. Wie bereits erwähnt, werden in einem Schritt solange komplette Bedarfe von Commodities durch das Netzwerk geschickt, bis dies für eine bestimmte Commodity nicht mehr möglich ist und eine Kante saturiert wird oder alle Bedarfe erfüllt wurden. Oben wurde bereits erwähnt, wie die Kosten dieser Operation auf verschiedene Elemente verteilt werden. Die Kosten für eine Pfadaktualisierung betragen immer $O(n)$, da ein Pfad höchstens aus $n-1$ Kanten bestehen kann. Eine Commodity, deren Bedarf vollständig verschickt werden konnte, ohne dabei eine Kante zu saturieren, wird in einer Phase genau einmal betrachtet. Die Gesamtanzahl solcher Aktualisierungen beträgt somit höchstens $O((\text{Anzahl Phasen}) \cdot k)$. Die Zahl der Aktualisierungen, die entstehen, falls ein Bedarf nicht komplett verschickt werden kann, entspricht höchstens der Anzahl der saturierten Kanten. Hiervon existieren nach Satz 4.1.3 $O(\epsilon^{-2}m \log m)$ viele. Zusammengefasst treten $O(\text{Anzahl Phasen} \cdot k + \epsilon^{-2}m \log m) = \tilde{O}(\epsilon^{-2}(m + k))$ viele Pfadaktualisierungen auf, die eine Laufzeit von $\tilde{O}(\epsilon^{-2}(m + k)n)$ besitzen, da ein Pfad aus höchstens n Kanten bestehen kann.

Die restlichen Operationen können wie in Abschnitt 4.2.2 abgeschätzt werden. Für alle Operationen, außer den Pfadaktualisierungen, gilt eine Laufzeit von $\tilde{O}(\epsilon^{-2}m^2)$. Daraus ergibt sich eine Gesamtlaufzeit von $\tilde{O}(\epsilon^{-2}(m^2 + kn))$.

4.2.3 Maximum Cost-Bounded Concurrent Flow

Der vorgestellte *maximum concurrent flow* Algorithmus kann leicht zu einem *maximum cost-bounded concurrent flow* Algorithmus erweitert werden. Auch diese Variante wurde von Garg und Könemann [32] vorgestellt und von Fleischer [28] und Karakostas [44] sukzessive erweitert. Hierbei wird die ursprüngliche Problembeschreibung um eine Kostenfunktion $b : E \to \mathbb{R}^+$ sowie einen Budgetwert B ergänzt. Ziel ist es, λ unter der Bedingung, dass die Kosten des gesamten Flusses nicht größer als B werden, zu maximieren. Wenn für dieses Problem ein FPTAS existiert, so existiert ebenfalls eines für das *minimum cost concurrent flow* Problem, indem mittels binärer Suche das kleinste Budget B gesucht wird, unter dem ein maximaler Fluss durch das Netzwerk fließen kann.

Um ein *maximum cost-bounded concurrent flow* Modell zu erhalten, wird das primale lineare Programm P_{mcf} um folgende Ungleichungen erweitert.

$$\sum_{p \in \mathcal{P}} b(p)x(p) \leq B$$

Das zugehörige duale lineare Programm D_{mcf} erhält entsprechend zusätzlich eine Variable ϕ für die obige Budget-Ungleichung und sieht nun wie folgt aus.

$$\begin{aligned} \min \quad & D(l,\phi) \stackrel{\text{def}}{=} \sum_{e \in E} c(e)l(e) + \phi B && (D_{mcmcf}) \\ \text{s.t.} \quad & \sum_{e \in p} (l(e) + b(e)\phi) \geq z_j \quad \forall 1 \leq j \leq k, \forall p \in \mathcal{P}_j \\ & \sum_{j=1}^{k} d_j z_j \geq 1 \\ & l \geq 0 \\ & z \geq 0 \end{aligned}$$

Die Algorithmen 4.1 und 4.2 können nun einfach an die veränderte Struktur angepasst werden. Während der Initialisierung wird $\phi = \delta/B$ gesetzt. Anstelle der Längenfunktion $l(e)$ wird nun $l(e)+b(e)\cdot\phi$ betrachtet. In einem Schritt wird der durch das Netzwerk geleitete Fluss nun so skaliert, dass er weder Kapazitätsgrenzen verletzt, noch die Gesamtkosten B überschreitet. Am Ende eines Schrittes wird neben der Längenfunktion auch ϕ aktualisiert: $\phi = \phi \cdot (1 + \frac{\text{cost of flow}}{B})$.

Der Korrektheitsbeweis vollzieht sich, unter Verwendung der neuen Längenfunktion $l(e) + b \cdot \phi$, analog zur Analyse des *maximum concurrent flow* Algorithmus. Lediglich der Laufzeitbeweis ist anzupassen. Hier muss gewährleistet werden, dass die Kosten des gesamten Flusses B nicht übersteigen. In jedem Schritt einer Iteration, außer im letzten, wird entweder eine Variable $l(e)$ oder ϕ um mindestens einen Faktor von $(1 + \epsilon)$ erhöht. Analog zu Satz 4.1.3 kann sich nun auch ϕ höchstens $B \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ Mal um Faktor $(1 + \epsilon)$ erhöhen. Die Anzahl an Schritten, in denen einer der beiden Werte, B oder $l(e)$, um diesen Faktor steigt, ist demnach $(m+1)\lceil \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \rceil$. Damit ist ebenfalls gezeigt, dass die Kosten des skalierten primalen Flusses höchstens B betragen.

Eine weitere Änderung ist in der Abschätzung der ξ nötig. Es reicht nicht mehr aus, einen Pfad mit maximaler Kapazität für jede Commodity zu berechnen. Stattdessen muss ein Pfad maximaler Kapazität berechnet werden, dessen Kosten B nicht überschreiten. Auch hierfür kann der Ansatz der Berechnung einer m-Approximation der ξ von Fleischer [28] verwendet werden. Dafür reicht es aus, mit binärer Suche unter allen m Kapazitäten diejenige Kapazität zu finden, die einen Pfad mit maximalem Fluss zulässt, dessen Kosten nicht größer als B sind. Hierfür kann ein modifizierter Dijkstra Algorithmus verwendet werden. Sei b_c die aktuelle Kapazitätsgrenze, sodass nur Pfade gefunden werden, dessen kleinste Kapazität mindestens b_c ist. Der Dijkstra Algorithmus ist nun derart modifiziert,

dass er Kanten, deren Kapazität geringer als b_c ist, ignoriert. Wird schließlich ein Pfad gefunden, so muss überprüft werden, ob die Kosten für diesen Pfad B übersteigen. Falls ja, wird b_c vermindert. Falls kein Pfad mit einer Kapazität von mindestens b_c existiert, so wird b_c erhöht und die Prozedur startet erneut. Die Laufzeit dieser Prozedur entspricht $O(\min\{k, n\} \log m \cdot m \log m) = O(m^2 \log^2 m) = \tilde{O}(m^2)$.

4.2.2 Theorem. *Algorithmus 4.2 stellt, zusammen mit den besprochenen Erweiterungen, ein FPTAS für das minimum concurrent flow Problem mit einer Laufzeit von $\tilde{O}(\epsilon^{-2} m^2 \log M)$, wobei M eine obere Schranke für die Kosten des Flusses darstellt.*

4.3 Eigenschaften des Algorithmus

4.3.1 Große Zahlen

Bei der Implementierung und Ausführung des Algorithmus werden die verwendeten und berechneten Werte bei großen Graphen und kleinem Epsilon sehr groß. Die Größe ist ab einem gewissen Grad problematisch, da die primitiven Datentypen in Programmiersprachen nur eine begrenzte Größe haben. Bei Java wird der Typ `double` durch IEEE 754 festgelegt und deckt das Intervall $[2^5 3 \cdot 2^{(1023-53+1)}, -2^{(-1022-53+1)}] = [4.94065645841246544 \cdot 10^{-324}, 1.7976931348623157 \cdot 10^{308}]$ ab [37].

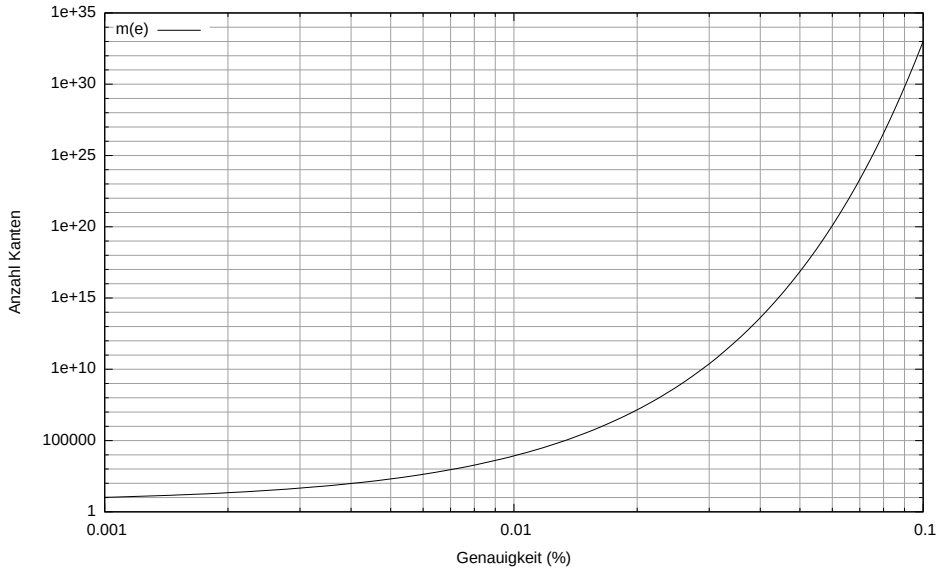

Abbildung 4.1: Obere Grenze für die Anzahl der Kanten in Abhängigkeit der gewünschten Genauigkeit. Die Achsen sind logarithmisch skaliert.

Ab einer bestimmten Graphgröße oder Genauigkeit können somit keine Berechnungen mehr durchgeführt werden, da die entstehenden Werte außerhalb des darstellbaren Bereichs

liegen. Dies ist vor allem bei δ ausschlaggebend, da sowohl die Längen $l(e)$ als auch ϕ von δ abhängen. Außerdem kommt δ im Skalierungsterm von Satz 4.1.3 vor.

Bezeichne im Folgenden $r = 4.94065645841246544 \cdot 10^{-324}$ die kleinste darstellbare Zahl. Um eine Ausführung des Algorithmus gewährleisten zu können muss $\delta \geq r$ gelten. Löst man diese Ungleichung nach m auf, so erhält man

$$\delta \geq r \tag{4.32}$$

$$\frac{1}{(1+\epsilon)^{\frac{1-\epsilon}{\epsilon}}}\left(\frac{1-\epsilon}{m}\right)^{\frac{1}{\epsilon}} > r \tag{4.33}$$

$$\left(\frac{1-\epsilon}{m}\right)^{\frac{1}{\epsilon}} \geq r(1+\epsilon)^{\frac{1-\epsilon}{\epsilon}} \tag{4.34}$$

$$\frac{1-\epsilon}{m} \geq r^{\epsilon}(1+\epsilon)^{1-\epsilon} \tag{4.35}$$

$$m \leq r^{-\epsilon}(1-\epsilon)(1+\epsilon)^{\epsilon-1} \tag{4.36}$$

Somit gibt die Funktion $m(\epsilon) = r^{-\epsilon}(1-\epsilon)(1+\epsilon)^{\epsilon-1}$ die maximale Anzahl an Kanten in Abhängigkeit der Genauigkeit ϵ an, für die gültige Berechnungen möglich sind. Abbildung 4.1 stellt diese Funktion dar. Aus ihr lässt sich erkennen, dass für eine Genauigkeit von 1% nur ca. 10000 Kanten vorhanden sein dürfen. In der Implementierung des Algorithmuses im Rahmen dieser Arbeit wurde anstatt r die Zahl 10^{-300} als Grenzwert verwendet.

Fleischer [28] hat herausgefunden, dass eine Skalierung von δ unter gleichzeitiger Anpassung des Zielfunktionswerts, ab dem der Algorithmus stoppt, möglich ist. Dieser Ansatz wurde in [28] zwar nur für das *maximum flow* Problem genannt, lässt sich jedoch einfach auf das *maximum cost-bounded concurrent flow* Problem überführen. Danach kann δ durch ein beliebiges δ' ersetzt werden. Mit dieser Änderung verändert sich automatisch das Abbruchkriterium. Zuvor stoppte der Algoritmus, falls $D(l,\phi) \geq 1$ ist. Nun stoppt er, falls $D(l,\phi) \geq \frac{\delta'}{\delta}$. Dies folgt direkt aus Abschätzung 4.18.

Auch diese Methode stößt bei genügend großen Zahlen an die Grenzen der Zahldarstellung, erlaubt es jedoch, die oben erwähten Einschränkungen einfach zu erweitern.

4.3.2 Budgetraum M

Ein wichtiger Faktor der Laufzeit ist der zu durchsuchende Budgetraum M. Um die Laufzeit gering zu halten, ist es notwendig diesen so weit wie möglich einzuschränken. Um dies zu erreichen wurde folgende Heuristik angewendet. Eine untere Schranke für die Gesamtkosten erhält man, indem für jede Commodity j ein Pfad von s_j nach t_j mit minimalen Kosten b_j unter der Funktion b gesucht wird. Hierbei werden Kapazitäten vernachlässigt. Die untere Schranke ist dann $\sum_j b_j$ und die Berechnung benötigt Zeit $O(\min\{k,n\}m \log m)$. Eine triviale obere Schranke ist $O(m \cdot b_{\max} \cdot \sum_{1 \leq j \leq k} d_j)$, wobei $b_{\max}$ den höchsten Kostenwert darstellt. Mit dieser Schranke wird der Algorithmus zu Beginn einmal gestartet. Die tat-

sächlichen Kosten des so berechneten Flusses stellen dann eine neue obere Schranke für die weiteren Berechnungen dar.

Um den maximalen Fluss zu bestimmen wird der Algorithmus zunächst mit der oberen Schranke für B ausgeführt. Dieser Flusswert entspricht demjenigen, der durch das Netzwerk geschickt werden kann. Die binäre Suche versucht nun diesen Flusswert beizubehalten und dabei die Kosten zu minimieren. Eine typische Entwicklung des Flusswerts zeigt Abbildung 4.2.

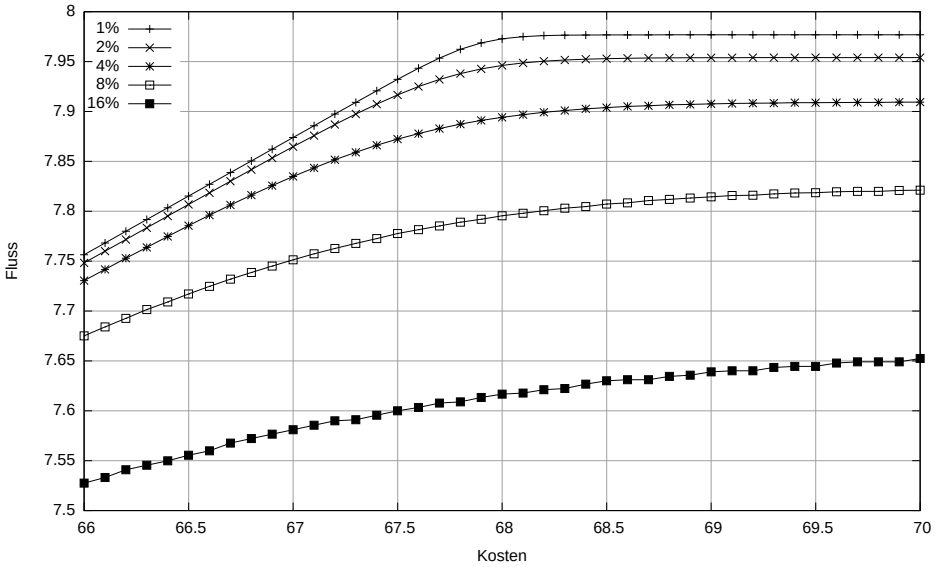

Abbildung 4.2: Flusswert in Abhängigkeit von B für verschiedene ϵ in einem Beispielgraphen. Das optimale Budget ist $B_{\text{opt}} = 68$ bei einem Flusswert von 8.

Ein Budget wird als optimal betrachtet, wenn es um Faktor 0.001 vom maximalen Funktionswert entfernt ist. Dieser Faktor ist die übliche Fehlerabschätzung bei Rechnungen mit großen Zahlen.

4.3.3 Multikriterielle Optimierung

Eine wichtige Eigenschaft für den praktischen Einsatz des Algorithmus ist die einfache Erweiterbarkeit der Kostenfunktion. Anstatt einer dualen Kostenvariable ϕ und eines Kostenvektors c können leicht h verschiedene Kostenfunktionen, implementiert werden. Das duale lineare Programm D_{mcf} sähe für multiple Kostenfunktionen so aus:

$$
\begin{aligned}
\min \quad & D(l,\phi) \stackrel{\text{def}}{=} \sum_{e\in E} c(e)l(e) + \sum_{1\le j\le h} \phi_j B_j && (D_{mcmcmcf})\\
\text{s.t.} \quad & \sum_{e\in p}(l(e) + \sum_{1\le j\le h} b_j(e)\phi_j) \ge z_j \quad \forall 1 \le j \le k, \forall p \in \mathcal{P}_j \\
& \sum_{j=1}^{k} d_j z_j \ge 1 \\
& l \ge 0 \\
& z \ge 0
\end{aligned}
$$

4.3.4 Fraktionalität

Karakostas Algorithmus berechnet faktionale Lösungen und splittet die Bedarfe einer Commodities unter Umständen auf, d. h. der Fluss fließt über unterschiedliche Pfade zur jeweiligen Senke. Dabei stellen fraktionale Lösungen generell kein Problem dar. Falls es sich bei der Eingabe um ganzzahlige Werte, z. B. bei den Bedarfen, handelt, so können diese durch eine abschließende Rundung wieder hergestellt werden. Um den Fehler hierbei möglichst gering zu halten muss das ϵ entsprechend klein gewählt werden. Handelt es sich dagegen um fraktionale Zahlen in der Eingabe, so ist dies unter Umstängen gewünscht, da es sich z. B. um Schüttgüter handelt.

Auch die Aufspaltung des Flusses einer Commodity ist unter Umständen erwünscht, da nur so eine effiziente Bündelung der Transporte möglich ist.

4.3.5 Lineare Kostenfunktionen

Eine weitere Eigenschaft des Algorithmus ist die Behandlung von linearen Kostenfunktionen. Angenommen, es existieren zwei Pfade, die von einer Commodity genutzt werden können, deren Kosten identisch und für das spezielle Quelle-Senke-Paar dieser Commodity minimal sind. Dann würde der Algorithmus den Fluss dieser Commodity, sofern die Kanten der beiden Pfade nicht mit Flusseinheiten anderer Commodities belastet werden, gleichmäßig auf beide Pfade verteilen. Dieser Fall könnte dann eintreten, wenn z. B. zwei identische Transportrouten an zwei aufeinander folgenden Tagen existieren. Als Abhilfe könnte eine multikriterielle Optimierung dienen, die als zusätzliche Kostenfunktion die Transportzeit besitzt zusammen mit der Einführung von Wartezeit an der Quelle. Somit würde eine Verbindung mit kürzerer Transportzeit bevorzugt gewählt werden.

4.4 Kürzeste Wege

Neben den bereits vorgestellten Skalierungsansätzen für β sind eine Reihe von weiteren, laufzeitverbessernden Maßnahmen denkbar. Eine Möglichkeit, die immer besteht, ist die

Befreiung der Eingabe von Redundanz. Bei der Modellierung des Flussgraphs wird bereits stets darauf geachtet, keine unnötigen Kanten oder Knoten zu erzeugen. Bei dem oben vorgestellten Algorithmus existieren im Prinzip zwei Ansatzpunkte, um weitere Verbesserungen zu erzielen. Der erste liegt in der theoretischen Analyse. Die Schrittlänge ϵ, mit der die Variablen $l(e)$ und ϕ in jedem Schritt erhöht werden, ist fix. Bei Untersuchungen von verwandten *multicommodity minimum cost* Algorithmen haben Goldberg et al. [36] herausgefunden, dass eine dynamische Anpassung dieses Parameter während der Ausführung eine wesentliche Laufzeitverbesserung mit sich bringt. Eine solche Verbesserung, sofern sie für den hier verwendeten Algorithmus überhaupt möglich ist, liegt jedoch außerhalb des Betrachtungsraumes dieser Arbeit.

Die zweite Möglichkeit einer Verbesserung besteht in der Auswahl eines geeigneten kürzesten Wege Algorithmus. In jedem Schritt des Algorithmus muss ein Baum, der die kürzesten Wege zu allen Knoten enthält, berechnet werden. Dies unterstreicht die Wichtigkeit der sorgfältigen Auswahl dieser Komponente.

Im Folgenden wird zunächst das *single source shortest path (SSSP)* Problem definiert. Anschließend werden verschiedene Ansätze zur Lösung vorgestellt und ihre Eignung in Bezug auf die Integration in den obigen Algorithmus überprüft. Abschließend werden drei Algorithmen näher vorgestellt, die für einen Einsatz im *multicommodity* Algorithmus in Frage kommen.

4.4.1 Definition

Die Eingabe eines kürzesten Wege Problems besteht aus einem Graphen $G = (V, E), n = |V|, m = |E|$, einem Quellknoten $s \in V$ und einer Längenfunktion $l : V \times V \rightarrow \mathbb{R}^+ \cup \infty$. Prinzipiell ist es möglich auch negative Kanten zuzulassen. Jedoch muss ein SSSP-Algorithmus dann Kreise mit negativen Kanten erkennen können. Diese Eigenschaft besitzen allerdings nicht alle Algorithmen. Da im benutzten Modell keine negativen Kantengewichte vorkommen, hat diese Einschränkung jedoch keine weiteren Auswirkungen. Zur Laufzeit erhalten die Knoten ein Label $d(v)$, auch Potential genannt, das die Distanz zwischen der Quelle s und dem Knoten v angibt. Terminiert der Algorithmus, so bezeichnet $d(v)$ die kürzeste Distanz von der Quelle s zum Knoten v. Darüber hinaus wird der interne Status eines Knoten zur Laufzeit durch $S(v) \in \{unlabled, scanned, labled\}$ festgehalten. Zur Speicherung der kürzesten Pfade wird eine Vorgänger-Relation π über die gesamte Laufzeit des Algorithmus vorgehalten. Sei $e = (v, w)$ eine Kante auf einem kürzesten Pfad, so ist $\pi(w) = v$. Dadurch ist es möglich von einem Knoten u rückwärts auf einem kürzesten Pfad zur Quelle zu gelangen. Vereinigt man diese Pfade für alle Knoten, so entsteht ein kürzester-Wege-Baum. Es handelt sich dabei um einen Spannbaum, in dem jeder Pfad von der Wurzel aus ein kürzester Pfad ist.

Sei die Potentialfunktion d gegeben, so gibt $l_d : V \times V \to \mathbb{R}^+$ die reduzierten Kosten einer Kante mit $l_d(v, w) = l(v, w) + d(v) - d(w)$ an. Eine Kante (u, v) heißt *zulässig*, falls $l_d(u, v) \leq 0$. In diesem Fall kann das Potential verbessert werden, indem w über die Kante (v, w) erreicht wird.

4.4.2 Dynamic Single Source Shortest Path

In der dynamischen Variante existiert bereits ein kürzester-Wege-Baum. Zusätzlich dazu besteht die Eingabe aus einer Menge von Änderungen von Kantengewichten. Ziel ist es, diese Änderungen (Updates) so zu verarbeiten, dass der kürzeste Wege Baum korrekt bleibt, ohne ihn nach jedem Update neu zu berechnen. Für eine Übersicht über Algorithmen dieser Art siehe z. B. [58, 53, 30].

Ein Update bezieht sich gewöhnlich auf eine Kante und beschreibt eine der folgenden Aktionen:

- Einfügen einer neuen Kante
- Löschen einer Kante
- Erhöhung der Länge einer Kante
- Reduzierung der Länge einer Kante

Einfügen und Löschen einer Kante kann als Spezialfall der beiden anderen Varianten angesehen werden. Die Löschung einer Kante entspricht dem Setzen der Länge auf Unendlich und das Einfügen der Reduzierung eines Längenwertes von Unendlich auf einen endlichen Wert. Beherrscht ein Algorithmus beide Updateformen, so wird er *fully dynamic*, ansonsten *semi-dynamic* genannt.

Dieser Ansatz scheint gut mit dem *maximum concurrent flow* Algorithmus vereinbar, da sich hier in jedem Schritt Kantengewichte ändern. Zu beachten ist, dass während der Ausführung des Algorithmus Bäume von verschiedenen Quellen aus berechnet werden müssen. Eine Kante kommt wahrscheinlich in mehreren Bäumen vor. So muss zum einen nachgehalten werden, welche Kante Teil welchen Baumes ist und welche Kantenlänge sich in welchem Maße ändert. Außerdem muss für jede Quelle ein kürzester-Wege-Baum vorgehalten werden.

Für den *maximum cost-bounded concurrent flow* Algorithmus ist diese Verfahrensweise jedoch nicht anwendbar. Dies begründet sich durch die Tatsache, dass sich in jedem Schritt des Algorithmus neben der Länge $l(e)$ auch die Kostenvariable ϕ ändert. Da die in einem kürzeste Wege Algorithmus verwendete Längenfunktion die Form $l(e) + b(e) \cdot \phi$ hat, ändern sich die Distanzwerte aller Kanten in jedem Schritt. Obwohl sich *dynamic shortest path* Algorithmen in Experimenten von Bauer und Wagner [17], bei denen sich nur wenige Distanzen ändern, performanter als eine Neuberechnung erwiesen haben, verschwindet dieser

Vorteil, falls sich alle Kantengewichte gleichzeitig verändern. Beim *maximum concurrent flow* Problem könnte dieser Ansatz jedoch zu Laufzeitverbesserungen verhelfen. Tendenziell sind diese Algorithmen schneller, je weniger Kanten sich ändern.

4.4.3 Static Single Source Shortest Path

Bei den statischen kürzesten Wege Algorithmen gibt es eine ganze Reihe von Algorithmen, die eine beachtliche Laufzeitverbesserung im Bereich der Optimierung von Straßennetzen, im Vergleich zu einer normalen Dijkstra Implementierung, erzielen. Einen Überblick über die verschiedenen Ansätze sowie ihre Performance geben z. B. Delling et al. [24] oder Willhalm und Wagner [68]. Der Geschwindigkeitsgewinn beruht vor allem auf geographischen Eigenschaften der Eingabe, sowie auf Preprocessing. Letzteres impliziert, dass sich die Kantengewichte über die Zeit nicht ändern. Da dies im obigen Algorithmus jedoch der Fall ist, kann auch diese Klasse von Algorithmen nicht angewendet werden.

Im Allgemeinen verwenden viele SSSP Algorithmen die von Gallo [31] eingeführte Grundstruktur. In der Initialisierung werden $d(v) = \infty, \pi(v) = nil, S(v) = unlabled \quad \forall v \in V \setminus \{s\}$ gesetzt. Für s gilt $d(s) = 0$ und $S(s) = labled$. Zu Beginn wird s einer Menge Q hinzugefügt. Der Algorithmus arbeitet in Phasen und terminiert, falls Q leer ist. In einer Phase wird ein Knoten v aus Q entfernt und für alle Knoten $w : (v, w) \in E$ wird überprüft, ob $l_d(v, w) < 0$ gilt. Bei dieser Überprüfung wird $S(w) = scanned$ gesetzt. Falls die reduzierten Kosten negativ sind, so wird $S(w) = labled$ gesetzt, w zu Q hinzugefügt und $d(w) = l(v, w) + d(v)$ gesetzt. Eine solche Überprüfung wird auch *Relaxierung* oder *Scanning* genannt. Bekommt ein Knoten eine neue Distanz zugeordnet so wurde er *relaxiert* oder *gescannt*. Der Algorithmus terminiert, falls die Optimalitätsbedingung $d(u) + l(u, v) \leq d(v), \forall (u, v) \in E$ gilt. Algorithmus 4.5 fasst diese Vorgehensweise zusammen.

Eingabe: Graph $G = (V, E)$, Quelle $s \in V$, Kantengewichte $l(e), e \in E$, Menge Q
Ausgabe: Kürzester-Wege-Baum mit Quelle s

```
1: init(V)
2: Q ← ∅
3: S(s) ← labled
4: Q ← s
5:
6: while Q ≠ ∅ do
7:     u ← Q
8:     scan(u)
9: end while
```

Algorithmus 4.4: Labeling Methode nach Gallo [31]

Eingabe: Knotenmenge V, Potentialfunktion d, Nachfolgerelation π

1: **for all** $v \in V$ **do**
2: $d(v) = \infty$
3: $\pi(v) \leftarrow nil$
4: **end for**
5: $d(s) = 0$

Algorithmus 4.5: Methode $init(V, d, \pi)$

Eingabe: Knoten u

1: **for** $(u, v) \in E$ **do**
2: **if** $d(v) > d(u) + l(u, v)$ **then**
3: $d(v) = d(u) + l(u, v)$
4: $S(v) \leftarrow labled$
5: $\pi(v) \leftarrow u$
6: $Q \leftarrow v$
7: $S(u) \leftarrow scanned$
8: **end if**
9: **end for**

Algorithmus 4.6: Methode $scan(u)$

Die verschiedenen existierenden *single source shortest path* Algorithmen unterscheiden sich darin, in welcher Form die Knoten aus der Menge Q entnommen werden. Hier unterscheidet man zwei generelle Ansätze. Bei der *label-setting* Methode wird ein Knoten, dessen Potential minimal über alle Knoten in Q ist, extrahiert. Es lässt sich leicht zeigen, dass das Potential eines solchen Knotens minimal ist und nur ein Mal während der gesamten Laufzeit aus Q extrahiert wird. Der berühmteste Vertreter dieses Paradigmas ist der Dijkstra Algorithmus [25]. Ein Vorteil dieser Methode wird deutlich, wenn kürzeste Wege nur zu einer bestimmten Menge von Knoten gesucht werden. In diesem Fall kann der Algorithmus abgebrochen werden, sobald alle gewünschten Knoten aus Q extrahiert wurden. Um diese spezielle Eigenschaft gewährleisten zu können, muss die Menge Q eine spezielle Datenstruktur sein, die eine nach Potentialwerten sortierte Liste der Knoten vorhält.

Algorithmen, die dieser Vorgehensweise nicht folgen, werden unter dem Begriff *label-correcting* zusammengefasst. Hierbei wird auf eine spezielle Datenstruktur von Q aus Performancegründen verzichtet. Dafür wird in Kauf genommen, dass ein Knoten mehrfach aus Q extrahiert werden kann.

In der Vergangenheit wurden einige umfangreiche Untersuchungen zur Effizienz verschiedener SSSP Algorithmen auf unterschiedlichen Graphklassen durchgeführt. Die wohl umfangreichste wurde von Cherkassky et al. [21] unternommen. Weitere Studien auf diesem Feld stammen von Zhan und Noon [71] und Glover et al. [33]. Basierend auf diesen Ergeb-

nissen wurden drei Algorithmen ausgewählt, die in Algorithmus 4.2 zum Einsatz kommen können. Im Folgenden werden alle drei Algorithmen kurz vorgestellt. Vergleichende Tests werden in Kapitel 5 ausgeführt.

4.4.4 Eigenschaften von kürzesten Pfaden

Um die Korrektheitsbeweise der folgenden Algorithmen kurz zu halten, werden eine Reihe von Lemmata vorgestellt. Diese werden hier nicht bewiesen. Dazu wird zum Beispiel auf [22] verwiesen. Darüber hinaus werden die Korrektheits- und Laufzeitbeweise teilweise nicht ausgeführt, sondern auf andere Quellen verwiesen. Diese Entscheidung beruht auf dem abweichenden Fokus dieser Arbeit.

Sei $\delta(u, v)$ die Länge des kürzesten Pfades von u nach v. Sollte es keinen Pfad zwischen u und v geben, so ist $\delta(u, v) = \infty$.

4.4.1 Lemma (Dreiecksungleichung). *Für eine Kante $(u, v) \in E$ gilt $\delta(s, u)+d(u, v) \geq \delta(s, v)$*

4.4.2 Lemma (Obere Schranken Eigenschaft). *Für alle Knoten gilt stets, dass $d(v) \geq \delta(s, v)$. Falls für einen Knoten $d(v) = \delta(s, v)$ gilt, so wird $d(v)$ nicht mehr verändert.*

4.4.3 Lemma (Kein Pfad Eigenschaft). *Wenn es keinen Pfad zwischen s und v gibt, so gilt stets $d(v) = \delta(s, v) = \infty$.*

4.4.4 Lemma (Konvergenz Eigenschaft). *Sei $s \rightsquigarrow u \rightarrow v$ ein kürzester Pfad in G. Falls $d(u) = \delta(s, u)$ vor der Relaxierung der Kante (u, v) gilt, so gilt $d(v) = \delta(s, v)$ anschließend und ändert sich nicht mehr.*

4.4.5 Lemma (Pfad-Relaxierungseigenschaft). *Sei $p = (v_0, v_1, \ldots, v_k)$ ein kürzester Pfad von $s = v_0$ nach v_k und seien die Kanten von p in der Reihenfolge $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ relaxiert worden. Dann gilt $d(v_k) = \delta(s, v_k)$. Diese Eigenschaft gilt auch, falls andere Kanten relaxiert werden.*

4.4.6 Lemma (Vorgänger-Teilgraph Eigenschaft). *Gelte $d(v) = \delta(s, v)$ für alle $v \in V$, so ist der Vorgänger Graph G_π ein kürzester-Wege-Baum mit Wurzel s.*

4.4.5 Dijkstra

Der Dijkstra Algorithmus ist wohl der berühmteste *single source shortest path* Algorithmus und implementiert die *label-setting* Methode. Hierfür benötigt er eine Datenstruktur Q, welche die Operationen *insert*, *extractMin* und *decreaseKey* unterstützt. Innerhalb dieser Struktur werden Paare der Form $(v, d(v)), v \in V$ abgelegt. Dabei wird $d(v)$ auch Key genannt. Bei *insert* Operationen wird ein Konten zu Q hinzufügt. Die Operation *extractMin*

gibt unter allen in Q vorhandenen Knoten denjenigen zurück, dessen Key minimal über dieser Menge ist. Die *decreaseKey* Operation bekommt als Parameter einen Knoten v und einen neuen Key $d(v)$ der kleiner ist, als der bereits in Q gespeicherte. Neben der Menge Q hält der Algorithmus eine Menge S vor, in der sich alle Knoten befinden, dessen kürzeste Distanz zu s bereits ermittelt wurde.

Algorithmus 4.7 beschreibt den Algorithmus in Pseudocode.

Eingabe: Graph $G = (V, E)$, Quelle $s \in V$, Kantengewichte $l(e), e \in E$, Menge Q
Ausgabe: Kürzester-Wege-Baum mit Quelle s

```
1: for all v ∈ V do
2:    d(v) = ∞
3:    insert(v, d(v))
4:    π(v) ← nil
5: end for
6: d(s) = 0
7: insert(s, d(s))
8:
9: while Q ≠ ∅ do
10:   u ← extractMin(Q)
11:   S ← u
12:   for (u, v) ∈ E do
13:      if d(v) > d(u) + l(u, v) then
14:         d(v) = d(u) + l(u, v)
15:         decreaseKey(v, d(v))
16:         π(v) ← u
17:         Q ← v
18:      end if
19:   end for
20: end while
```

Algorithmus 4.7: Dijkstra Algorithmus

Korrektheit

Der Korrektheitsbeweis ist z. B. in [22] zu finden.

Laufzeit

Die Laufzeit des Algorithmus lässt sich einfach bestimmen. Zu Beginn wird für jeden Knoten eine *insert* Operation ausgeführt. Daraus folgt, dass es ebenso viele $extractMin$ Operationen gibt. Da jeder Knoten genau einmal zu S hinzugefügt wird, werden alle zu

einem Knoten v adjazenten Kanten genau einmal betrachtet. Dies ergibt eine $decreaseKey$ Operation pro Kante. Die Gesamtlaufzeit ist somit $O(n \cdot (O(insert) + O(extractMin)) + m \cdot O(decreaseKey))$. Die Laufzeit hängt also wesentlich von der verwendeten Datenstruktur ab. Heaps unterstützen alle drei Operationen und sind somit für den Einsatz geeignet. Tabelle 4.1 gibt einen Überblick über verschiedene Heap-Implementierungen zusammen mit den jeweiligen Laufzeiten für die obigen Operationen und der resultierenden Gesamtlaufzeit für den Dijkstra Algorithmus.

Variante	*insert*	*extractMin*	*decreaseKey*	Laufzeit Dijkstra
k-ary	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n + m \log n)$
Binomial	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n + m \log n)$
Fibonacci	$O^*(1)$	$O^*(\log n)$	$O^*(1)$	$O^*(n \log n + m)$

Tabelle 4.1: Übersicht über Performance von verschiedenen Heap-Implementierungen. O^* bezeichnet die amortisierte Laufzeit

Für eine genaue Analyse der unterschiedlichen Heaps und der Korrektheit des Dijkstra Algorithmus siehe z. B. Cormen et al. [22]. Die unter Verwendung von Fibonacci-Heaps erzielte theoretische worst-case Laufzeit ist die bisher beste gefundene für allgemeine *single source shortest path* Algorithmen.

4.4.6 Graph Grow

Pape [55] erweiterte den Grundalgorithmus 4.5 indem er Q als eine Queue mit zwei offenen Enden implementierte. Eine solche Queue wird auch *Deque* genannt und erlaubt das Einfügen an beiden Enden. Genauer gesagt besteht die Idee darin, Knoten, die nicht in Q sind, in zwei Mengen aufzuteilen. Knoten, die noch nie in Q waren gehören zur einen und alle Knoten, die mindestens einmal in Q waren, zur anderen Menge.

Knoten der ersten Gruppe werden am Ende von Q eingefügt, während die anderen am Anfang von Q eingefügt werden. Falls nun ein Knoten $v \notin Q, S(v) = labled$ wieder in Q eingefügt wird, so hat sich $d(v)$ verbessert. Sei T der aktuelle kürzeste Wege Baum, dann existiert ein Unterbaum $T(v)$, der v als Knoten hat. Der verbesserte Potentialwert von v muss nun in $T(v)$ propagiert werden. Damit dies möglichst schnell geschieht, wird v am Anfang von Q eingefügt. Möglich wird dieses Prinzip durch die Einführung des Attributs $P(v)$, mit dem zwischen bereits besuchten und noch nie besuchten Knoten unterschieden werden kann.

Die von Pape vorgeschlagene Implementierung von Q besteht aus einer Queue Q_1 und einem Stack Q_2. Siehe hierzu Abbildung 4.3 oben. Noch nie betrachtete Knoten werden in Q_1 eingefügt, während bereits zuvor gescannte Knoten auf Q_2 gelegt werden. Die Entnahme

der Knoten geschieht in der Regel von Q_2. Für den Fall, dass Q_2 keine Elemente enthält, werden Knoten aus Q_1 entnommen.

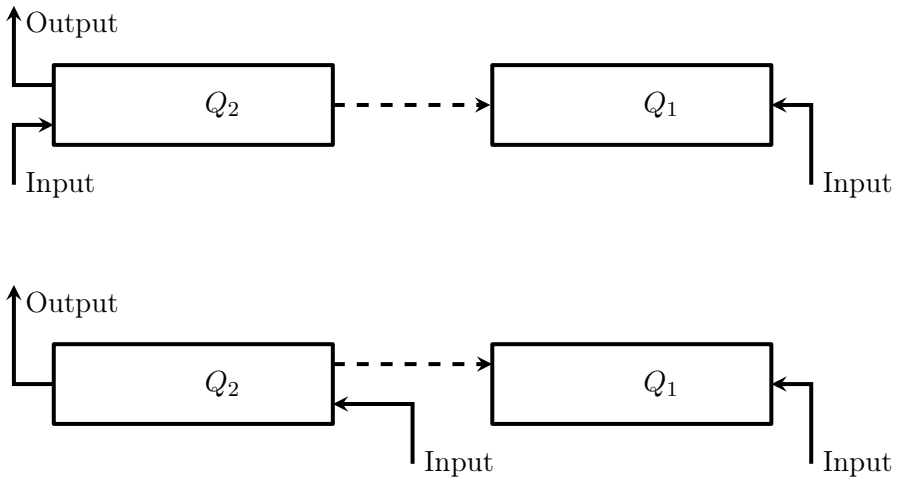

Abbildung 4.3: Darstellung der Struktur von Q. Oben: Q_1 als Queue, Q_2 als Stack. Unten: Q_1 und Q_2 als Queues. Gestrichelte Kanten geben die Entnahmereihenfolge an.

Das gesamte Verfahren ist in Algorithmus 4.8 zusammengefasst. Dabei gibt $R(v) \in \{0,1\}$ an, ob Knoten v gerade in Q (1) ist oder nicht (0). $P(v) \in \{0,1\}$ drückt aus, ob ein Knoten bereits einmal gescannt wurde (1) oder nicht (0).

Korrektheit

Die Korrektheit folgt direkt aus der Korrektheit der *labeling* Methode.

Laufzeit

Die worst-case Laufzeit des Pape Algorithmus beträgt $O(n2^n)$. Dies liegt an der Benutzung des Stacks. Abbildung 4.3 unten veranschaulicht diese Struktur. Die Variante von Pallottino [54], die anstelle des Stacks eine zweite Queue benutzt, besitzt eine Laufzeit von $O(mn^2)$. Der Algorithmus, der diese Struktur benutzt wird im Folgenden TWO_Q genannt. Kershenbaum [46] zeigt eine allgemeine Methode, mit der eine worst-case Instanz für den ersten Fall erzeugt werden kann.

Der Name *graph grow* Algorithmus leitet sich aus der speziellen Arbeitsweise des Algorithmus ab. Sei W die Menge der Knoten, für die $S(v) = scanned, v \in W$ gilt. Werden solche Knoten erneut gescannt, werden sie in Q_2 eingefügt. Falls diese Menge leer ist, bedeutet dies, dass der von den Knoten in W induzierte Teilbaum ein kürzester Teilbaum ist. Grund hierfür ist die Tatsache, dass keiner der Knoten aus W in Q_2 ist und somit die Optimalitätsbedingung $d(v) = d(u) + l(u,v)$ für alle Knoten aus W gilt. Falls dies der Fall ist, so wird ein Knoten w aus Q_1 der Menge W hinzugefügt. Falls Q_2 das nächste Mal

Eingabe: Graph $G = (V, E)$, Quelle $s \in V$, Kantengewichte $l(e), e \in E$, Queue Q_1, Stack Q_2, Attribute $S(v)$

Ausgabe: Kürzester-Wege-Baum mit Quelle s

```
1: for all v ∈ V do
2:    d(v) = ∞
3:    R(v) = 0
4:    P(v) = 0
5:    π(v) ← nil
6: end for
7: Q1 ← s
8: Q2 ← ∅
9: R(s) = 1
10:
11: while Q1 ≠ ∅ OR Q2 ≠ ∅ do
12:    if Q2 = ∅ then
13:       u ← Q1
14:    else
15:       u ← Q2
16:    end if
17:    R(u) = 0
18:    P(u) = 1
19:    for (u, v) ∈ E do
20:       if d(v) > d(u) + l(u, v) then
21:          d(v) = d(u) + l(u, v)
22:          π(v) ← u
23:          if P(v) = 0 then
24:             Q2 ← v // wurde noch nie gescannt
25:             R(v) = 1
26:          else
27:             Q1 ← v // wurde mindestens einmal gescannt
28:             R(v) = 1
29:          end if
30:       end if
31:    end for
32: end while
```

Algorithmus 4.8: Pape Algorithmus

leer wird, so wurde der kürzeste Teilbaum $W \cup \{w\}$ berechnet. Der kürzeste Wege Baum wächst also im Laufe des Algorithmus.

4.4.7 Graph Ordering

Sei von den Knoten u und v bereits ein Scan ausgeführt worden und gelte $l_d(u, v) < 0$. Dann ist es auf jeden Fall besser, zunächst einen weiteren Scan von u aus zu tätigen, da hierdurch der Potentialwert von v verbessert wird. Dieser Idee folgt der *graph ordering* Algorithmus von Goldberg und Radzik [34].

Der Algorithmus verwaltet zwei Mengen A und B. Ein Knoten v kann zu jeder Zeit nur in genau einer der beiden Mengen sein. Zu Beginn ist A leer und B enthält ausschließlich den Quellknoten. In einer Iteration wird aus der Menge B die Menge A berechnet. Anschließend werden alle Elemente aus B entfernt und alle Knoten in A gescannt. Ein Knoten, dessen Distanzwert hierbei verbessert werden konnte, wird in B eingefügt. Eine Iteration endet, falls A leer ist. Der Algorithmus terminiert, falls B zu Beginn einer Iteration leer ist.

Die Berechnung der Menge A geschieht durch folgende Schritte. Zunächst werden alle Knoten $v \in B$, für die $l_d(v, w) \geq 0, (v, w) \in E$ gilt, als gescannt markiert und aus B entfernt. Sei A nun die Menge an Knoten, die über Kanten $(u, v) \in E, u \in B$ mit $l_d(u, v) < 0$ erreicht werden können. Diese Knoten bekommen das Attribut *labled* zugewiesen. Abschließend werden alle Knoten in A topologisch sortiert.

Da vereinbart wurde, dass negative Kreise nicht auftreten, kann die topologische Sortierung mittels eines einfachen *Depth First Search* (DFS) Algorithmus, bei dem alle Rückwärtskanten ignoriert werden, erfolgen. Eine Pseudocode-Version des Algorithmus GOR zeigt Algorithmus 4.9. In Zeile 17 wird die topologische Sortierung mittels DFS durchgeführt. Dabei werden Knoten, die in der aktuellen Iteration bereits besucht wurden, ignoriert. Die verbleibenden durch DFS gefundenen Knoten werden in topologischer Reihenfolge auf den Stack A gelegt.

Korrektheit

Wenn für einen Knoten v mit $S(v) = \textit{scanned}$ $S(v)$ auf *labled* geändert wird, so verändert dies nicht die Korrektheit der *labeling* Methode. Dies bezieht sich auf den Schritt, in dem A gefüllt wird. Alle Knoten, die in A aufgenommen werden, haben jedoch einen endlichen Potentialwert und sind somit alle entweder *labled* oder *scanned*. Durch die Annahme, dass G azyklisch ist, ist die anschließende topologische Sortierung wohl definiert.

Laufzeit

Der Beweis ist dem des Bellman-Ford Algorithmus (siehe z. B. Cormen et al. [22]) sehr ähnlich, sodass der Algorithmus Zeit $O(nm)$ benötigt.

Eingabe:

Eingabe: Graph $G = (V, E)$, Quelle $s \in V$, Kantengewichte $l(e), e \in E$, Stack A, Stack B, Attribute $S(v)$, $visited(v) \in \{true, false\}$,

Ausgabe: Kürzester-Wege-Baum mit Quelle s

```
1: init(V, d, π)
2: A ← ∅
3: B ← s
4: repeat
5:    for all v ∈ V do
6:       visited(v) =false
7:    end for
8:    while B ≠ ∅ do
9:       u ← B
10:      if visited(u) =false then
11:         scan(u)
12:      end if
13:   end while
14:   if ∀v : l_d(u, v) ≥ 0 then
15:      S(u) = scanned
16:   else
17:      A ← DFS(v, visited)
18:   end if
19:   while A ≠ ∅ do
20:      u ← A
21:      scan(u)
22:   end while
23: until B = ∅
```

Algorithmus 4.9: GOR Algorithmus [34]

4.4.8 Directed Acyclic Graphs

Gerichtete azyklische Graphen (DAG) stellen eine spezielle Graphklasse dar. Da in einem DAG nach Definition keine negativen Kreise existieren, sind alle kürzesten Wege wohldefiniert.

Ein auf diese Struktur optimierter kürzester-Wege-Algorithmus berechnet zunächst eine topologische Sortierung der Knoten des Graphen G. Danach werden die Knoten in der Reihenfolge ihrer Sortierung relaxiert. Eine Pseudocode-Darstellung dieses Verfahrens zeigt Algorithmus 4.10.

Eingabe: Graph $G = (V, E)$, Quelle $s \in V$, Kantengewichte $l(e), e \in E$
Ausgabe: Kürzester-Wege-Baum mit Quelle s

1: $topSort(G)$
2: $init(V, d, \pi)$
3: **for all** $v \in V$ in topologischer Ordnung **do**
4: $\quad scan(v)$
5: **end for**

Algorithmus 4.10: DAG Algorithmus nach [22]

Korrektheit

Beweis. Falls es keinen Pfad von einem Knoten v zur Quelle gibt, so gilt nach Lemma 4.4.3 $d(v) = \delta(s, v) = \infty$. Angenommen, v ist von s aus erreichbar, so sei $p = (s = v_0, v_1, \ldots, v_k = v)$ ein kürzester Pfad in G. Da alle Kanten topologisch sortiert sind, werden sie in der Reihenfolge $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ relaxiert. Nach Lemma 4.4.5 gilt dann für alle Knoten v_i auf diesem Pfad $d(v_i) = \delta(s, v_i)$. Somit gilt für alle Knoten am Ende des Algorithmus $d(v) = \delta(s, v)$ und nach Lemma 4.4.6 ist G_π ein kürzester-Wege-Baum. □

Laufzeit

Die topologische Sortierung des Graphen kann mittels eines DFS Aufrufs ermittelt werden. Dies geschieht in Zeit $O(n + m)$. Der Aufruf von $init(V, d, S)$ benötigt Zeit $O(|V|)$. In der Schleife in Zeile 3 wird für jeden Knoten jede Kante genau einmal betrachtet. Die *scan* Operation wird somit genau m Mal aufgerufen. Da ein *scan* Aufruf Zeit $O(1)$ benötigt, beträgt die Gesamtlaufzeit $O(n + m)$.

4.4.9 Zusammenfassung

Die drei Algorithmen Dijkstra, GOR und TWO_Q wurden aufgrund der Resultate von Cherkassky et al. [21] ausgewählt. Dabei erwies sich GOR über alle getesteten Netzwerke als sehr robust. Mit Dijkstra und TWO_Q sind zwei Kandidaten ausgesucht worden, die auf einigen Graphklassen am besten abgeschnitten haben. Zusätzlich zu diesen Algorithmen wurde auch ein *single source shortest path* Algorithmus (DAG) speziell für gerichtete, azyklische Graphen implementiert. Da die spezielle Struktur des Modells aus Kapitel 3 eine zeitliche Abfolge ausdrückt, handelt es sich um einen solchen gerichteten, azyklischen Graphen. Dabei ist zusätzlich zu beachten, dass die topologische Sortierung sich im Verlauf des *multicommodity flow* Algorithmuses nicht verändert und somit nur einmalig zu Beginn berechnet werden muss. Anschließend müssen für eine Bestimmung der kürzesten Wege lediglich die Knoten entsprechend ihrer topologischen Sortierung gescannt werden.

Kapitel 5

Evaluation

Im Rahmen dieser Diplomarbeit werden mehrere Tests durchgeführt. Zum einen wird das Laufzeitverhalten der impliziten Version des *maximum cost-bounded multicommodity* Algorithmus 4.2 von Karakostas [44] analysiert. Obwohl die theoretische Laufzeit viele ähnliche Algorithmen übertrifft, so ist meines Wissens nach das tatsächliche Verhalten des Algorithmus in der Praxis bisher noch nicht untersucht worden. Die Tests konzentrieren sich auf die implizite Variante, da diese auch für die anschließenden Fallbeispiele in Bezug auf das in dieser Diplomarbeit entwickelte Modell verwendet wurde. Der Vollständigkeit halber wurden jedoch auch die explizite Variante sowie die ursprüngliche Version von Fleischer [28] getestet.

Um den Algorithmus von Karakostas mit anderen adäquat vergleichen zu können, müssen zunächst die zu untersuchenden Graphinstanzen mit verschiedenen kürzesten Wege Algorithmen analysiert werden, um herauszufinden, welcher dieser *single source shortest path* Algorithmen am besten für eine bestimmte Graphklasse geeignet ist. Darüber hinaus wurden vergleichende Tests zum PPRN Framework von Castro und Nabona [19] durchgeführt.

Abschließend wird das in Kapitel 3 entwickelte Modell untersucht. Dabei werden mehrere Instanzen vorgestellt und analysiert. Im Vordergrund steht die Frage, wie viele Güter innerhalb eines definierten Zeitrahmens von der Straße auf die Schiene verlagert werden können und welche Konsequenzen dies für den CO_2-Ausstoß hat.

Die Ergebnisse werden in Form von Diagrammen dargestellt. Die zugrunde liegenden Tabellen befinden sich entweder im Anhang A oder aus Platzgründen auf der beigefügen CD.

5.1 Testumgebung und verwendete Datenstrukturen

Alle Tests wurden auf einem Rechner mit einer Core 2 Quad CPU (2,39 GHz) mit 3,24 GB RAM durchgeführt. Als Betriebssystem kam Windows XP Prof. mit Service Pack 2

zum Einsatz und die Laufzeitumgebung war SUN JDK in der Version 1.6.0_07. Lediglich die Testdurchläufe zum Bündelungsverhalten, siehe Abschnitt 5.3.3, wurden auf einem MacBook mit Core 2 Duo (1,83 GHz) mit 2,5 GB RAM ausgeführt. Auf diesem System kam MacOS X 10.4.11 und Java $1.6_b etazum Einsatz$.

Die implementierte Graphdatenstruktur ist wie folgt aufgebaut. Jede Kante und jeder Knoten besitzt eine eindeutige ID. Darüber hinaus besitzt jede Kante eine Referenz auf die adjazenten Knoten. Alle IDs werde so vergeben, dass sie fortlaufend generiert werden. Die Mengen der Knoten und Kanten sind in Arrays gespeichert, sodass die Position in einem Array der ID eines Knoten bzw. einer Kante entspricht. Weiterhin werden zwei Listen für jeden Knoten vorgehalten, in denen alle aus- bzw. eingehenden Kanten gespeichert sind. Die Attribute eines Knotens oder einer Kante sind ebenfalls in Arrays gespeichert, so dass mittels einer ID direkt auf den jeweiligen Wert zugegriffen werden kann.

Zur Speicherung der Ergebnisse wurden die XStream Bibliotheken [1] benutzt. Sie erlauben eine einfache Serialisierung von Klassen und somit einen einfachen Export von Datenstrukturen, in denen die Ergebnisse gespeichert sind.

5.2 Graphinstanzen

Die Graphen, die für die Tests des Laufzeitverhaltens der *multicommodity* Algorithmen zum Einsatz kommen, werden von den beiden Generatoren GRIDGEN von Lee und Orlin [47] und GENRMF von Badics [15] erzeugt. Darüber hinaus werden ebenfalls Transportnetzwerkgraphen, die auf dem Modell aus Kapitel 3 basieren, untersucht. Die erstgenannten Generatoren erzeugen verschiedenartige Gitternetzstrukturen und wurden in der Literatur sehr häufig für vergleichende Analysen von Flussalgorithmen verwendet, siehe Goldberg et al. [36] oder Radzik [57].

5.2.1 GRIDGEN

Die Eingabe für den GRIDGEN Generator besteht aus der Anzahl an Knoten n, einem durchschnittlichen Knotengrad d, der Breite des entstehenden Netzwerks w sowie ein Kapazitäts- bzw. Kosteninterval $[c_l, c_u]$ und $[b_l, b_u]$. Der Generator erzeugt nun ein Gitter, das aus $\lceil n/w \rceil \cdot w$ Knoten besteht. Diese Zahl kann aufgrund der Rundung leicht von n abweichen. Anschließend werden wahlweise uni- oder bidirektionale Kanten zwischen benachbarten Knoten erzeugt. Wird d nicht erreicht, so werden Kanten zwischen uniform zufällig ausgewählten Knotenpaaren dem Graphen hinzugefügt. Zum Schluss werden Kantenkapazitäten und -kosten zufällig auf die Kanten verteilt. Das Ergebnis dieses Generators ist ein Gitternetz, welches einem Straßennetz ähnelt.

5.2.2 GENRMF

Der GENRMF Generator benötigt für die Erstellung eines Netzwerkes eine Gittergröße a, eine Tiefe b sowie ein Kapazitäts- bzw. Kostenintervall $[c_l, c_u]$ und $[b_l, b_u]$. Das generierte Netzwerk besteht aus b Gittern der Größe $a \times a$, also aus insgesamt $a \cdot a \cdot b$ Knoten. In jedem Gitter ist ein Knoten, analog zum GRIDGEN Generator, mit allen benachbarten Knoten über bidirektionale Kanten verbunden. Jeder Knoten eines Gitters ist außerdem mit genau einem Knoten des Nachbargitter, dessen Tiefe genau um 1 größer ist, verbunden. Die jeweiligen Knotenpaare werden zufällig ermittelt. Die Kapazitäten der Kanten, die zwei Gitter verbinden, werden uniform zufällig aus dem gegebenen Intervall $[c_l, c_u]$ gewählt, während die Kanten innerhalb eines Gitters eine wesentlich größere Kapazität zugewiesen bekommen. Die Kantenkosten werden über alle Kanten uniform zufällig aus dem gegebenen Intervall vergeben.

Für beide Graphklassen werden Quellen und Senken uniform zufällig über das Netzwerk verteilt. Bei GENRMF-Graphen muss hierbei darauf geachtet werden, dass eine Quelle niemals auf einem Gitter mit einer höheren Tiefe liegt als die Senke, da die Kanten, welche Gitter verbinden, immer nur zu Gittern mit einer größeren Tiefe zeigen.

5.2.3 Multimodales Transportnetzwerk

Als Testinstanzen wurden zufällige Transportnetzwerke anhand verschiedener Parameter erstellt. Die zufällig ausgewählten Orte für Quellen, Senken und GVZ entsprechen tatsächlichen Orten in Deutschland. Ein Quellort stammt aus der Menge der 15 größten Städte Deutschlands. Die Senken wurden zufällig aus einer Menge von 4000 deutschen Städten entnommen. Die GVZ-Standorte und die Zugverbindungen zwischen ihnen wurden bereits in Kapitel 2.1 vorgestellt.

Die Erzeugung der Commodities erfolgt uniform zufällig zwischen den Mengen Quellen und Senken. Der Bedarf einer Commodity wurde uniform zufällig aus dem Intervall $[0 : 26]$ Tonnen gewählt. Die Zuglänge bzw. das Bruttogewicht des gesamten Zuges, welches für die Berechnung der CO_2-Emissionen benötigt wird, wurde für jeden Zug auf 1000 Tonnen festgesetzt. Dies entspricht der mittleren Zuglänge, die von EcoTransIT [5] verwendet wird. Die Auslastung eines LKW wurde auf 58% und der Anteil der Leerfahrten auf 17% gesetzt. Bei einem Zug betragen diese Werte 58% bzw. 50%. Auch dies entspricht den Durchschnittswerten, die von EcoTransIT vorgeschlagen werden. Darüber hinaus ist die Motorisierung bei allen Commodities konform zur Euro4-Norm und es werden ausschließlich Elektrozüge eingesetzt.

5.3 Untersuchung der *multicommodity* Algorithmen

Im Folgenden werden die *maximum cost-bounded multicommodity flow* Algorithmen von Karakostas [44] und Fleischer [28] getestet. Dabei liegt der Fokus auf der impliziten Variante von Karakostas (siehe Algorithmus 4.2). Das Verhalten der Algorithmen wurde anhand der beiden oben vorgestellten Netzwerkgraphen untersucht. Dabei wurden die Bedarfe der erzeugten Commodities so skaliert, dass sie in einem Netzwerk mit 60% Kantenkapazitäten weiterhin gültig sind. Dieses Setting wurde ebenfalls von Goldberg et al. [36] verwendet. Das Verhalten der Algorithmen wurde in Bezug auf die Approximations-Genauigkeit ϵ und der Anzahl der Commodities k untersucht. Für die Untersuchung des Verhaltens der Algorithmen ist es ausreichend das *maximum cost-bounded multicommodity flow* Problem für einen bestimmten Budgetwert zu berechnen. Eine Lösung des *minimum cost multicommodity flow* Problems ist hierfür nicht erforderlich, da letzteres nur eine wiederholte Ausführung des erst genannten Problems ist.

5.3.1 Kürzeste Wege Algorithmen

Wie bereits in Kapitel 4.4 angesprochen, ist die Kernkomponente des Frameworks von Garg und Könemann [32] ein kürzester-Wege-Algorithmus. Bevor man einen Algorithmus dieses Frameworks auf einer Graphklasse testet, muss man prüfen, welcher *single source shortest path* Algorithmus die beste Laufzeit für diese Graphklasse besitzt. Es wurden sowohl GENRMF, GRIDGEN als auch Transportnetzwerkinstanzen mit den Algorithmen Dijkstra, GOR, TWO_Q und DAG aus Kapitel 4.4 getestet, letzterer jedoch nur auf Transportnetzwerken, da er nur auf gerichteten, azyklischen Graphen arbeiten kann. Auf jeder Instanz wurden pro Algorithmus 100 *one-to-all* Durchläufe mit uniform zufällig ausgewählten Startknoten durchgeführt. Die Ergebnisse sind in den Abbildungen 5.1, 5.2 und 5.3 zusammengefasst. Der abgebildete Wert entspricht der Summe der aller Durchgänge auf einer Instanz in Millisekunden.

Bei fast allen Instanzen lässt sich eine eindeutige Rangfolge erkennen. TWO_Q ist schneller als GOR und GOR ist schneller als Dijkstra. Bei GENRMF Instanzen in Abbildung 5.1 ist diese Reihenfolge deutlich ausgeprägt. Anders verhält es sich in Abbildung 5.2 bei GRIDGEN Instanzen. Hier dominiert TWO_Q meistens, ist jedoch manchmal genauso schnell oder sogar langsamer als GOR. Ähnlich sieht es bei den Transportnetzwerken aus, siehe Abbildung 5.3. Hier sind die Algorithmen DAG und TWO_Q fast ebenbürtig, auch wenn DAG meist etwas schneller ist.

Basierend auf diesen Erkenntnissen wird für die folgenden Tests TWO_Q für die Graphinstanzen der Generatoren GENRMF und GRIDGEN und DAG für die Transportnetzwerke verwendet. Beide Algorithmen haben nicht nur sehr gute Laufzeiten, ihre Konstruktion kommt auch mit sehr einfachen Datenstrukturen aus, die den Verwaltungsaufwand für diese Algorithmen stark senkt.

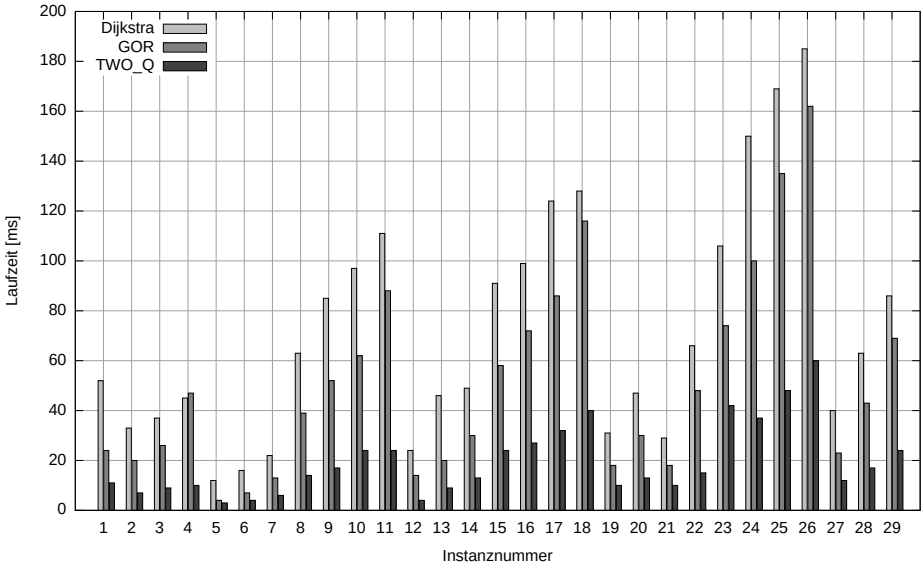

Abbildung 5.1: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen GENRMF Instanzen mit 100 Durchgängen pro Instanz.

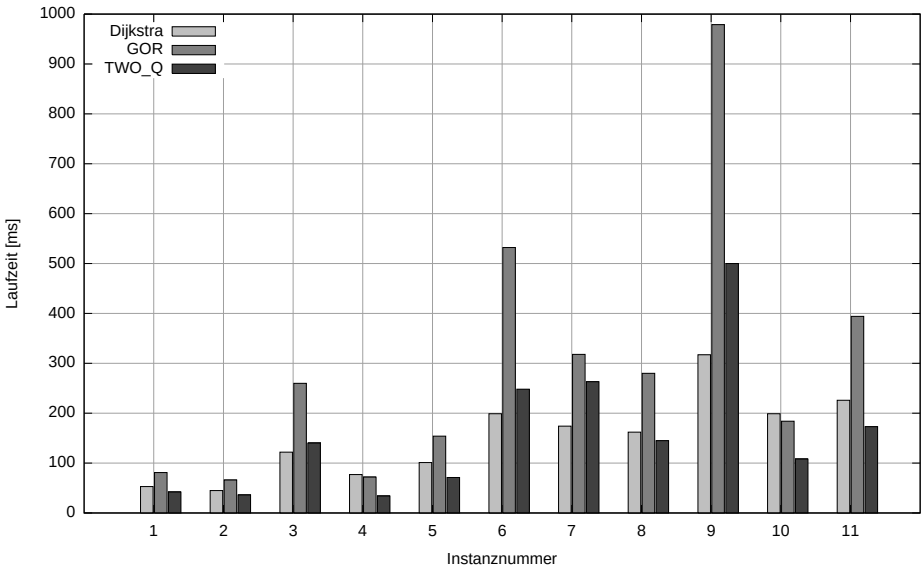

Abbildung 5.2: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen GRIDGEN Instanzen mit 100 Durchgängen pro Instanz.

5.3.2 Impliziter Algorithmus von Karakostas

Die Abbildungen 5.4 und 5.5 zeigen den Verlauf von verschiedenen GRIDGEN bzw.GENRMF Graphinstanzen in Abhängigkeit von ϵ. Die getesteten Werte von ϵ entstammen der Menge $\{0.99, 0.64, 0.32, 0.16, 0.08, 0.04, 0.02\}$. Test für $\epsilon = 0.01$ wurden aufgrund von Zeitbegren-

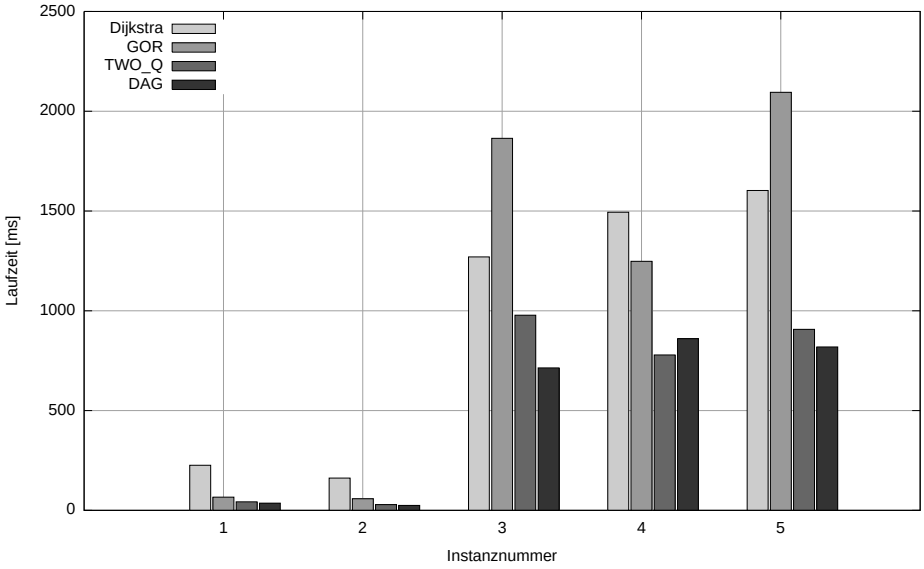

Abbildung 5.3: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen Transportnetzwerk-Instanzen mit 100 Durchgängen pro Instanz.

zungen abgebrochen. Zum Vergleich: die kleinste Instanz benötigte für $\epsilon = 0.02$ eine Zeit von gut 6 Stunden, während die größte Instanz knapp 21 Stunden benötigte. Als Budgetwert wurde, analog zur in Abschnitt 4.3 vorgestellten Heuristik, der Wert einer unteren Schranke genommen. Die Anzahl der Commodities k wurde auf 100 festgesetzt.

Die Kodierung der Instanzen in den Abbildungen 5.4 und 5.5 folgt dem folgenden Schema. Für GRIDGEN Graphen bedeutet das Schema `grid-n-d-u`, dass es sich um eine Instanz mit n Knoten, durchschnittlichem Knotengrad d und einem Kapazitäts- und Kostenintervall von $[1; u]$ handelt.
Für Graphinstanzen, die durch den GENRMF erzeugt werden, besagt das Benennungsmuster `rmf-a-b-u` folgendes: die jeweilige Instanz besitzt eine Gittergröße von $a \times a$, eine Tiefe von b und Kapazitäts- und Kostenintervalle von $[1 : u]$.

Aufgrund der vorliegenden Daten wurde die tatsächliche Funktionsverlauf von ϵ mittels dem Programm Gnuplot von Williams und Kelley [69] ermittelt. Hierbei wird die Methode der kleinsten Quadrate verwendet. Das Ergebnis dieser Berechnung lieferte sowohl für die GRIDGEN als auch für die für die GENRMF Instanzen ein Abhängigkeit von $O(\epsilon^{-1.95})$. Damit liegt der praktisch ermittelte Wert sehr nahe an der theoretischen Schranke von $O(\epsilon^{-2})$, siehe Abschnitt 4.2.1.

Bei der Untersuchung der Abhängigkeit der Laufzeit von der Anzahl der Commodities k wurden für k Werte aus der Menge $\{25, 50, 75, 100, 150, 200, 400, 800\}$ überprüft. Die durchgeführten Messreihen sind in Abbildung 5.6 und 5.7 dargestellt. Alle Graphinstan-

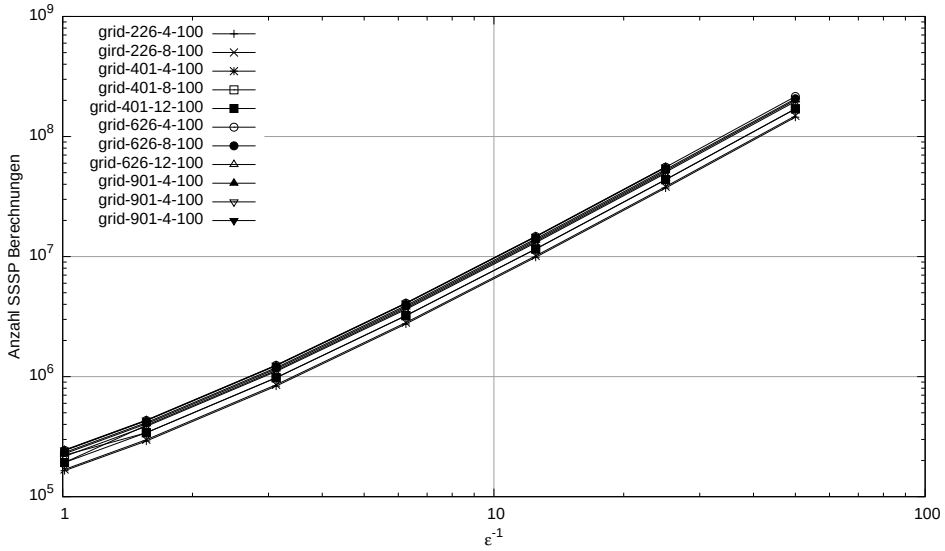

Abbildung 5.4: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Genauigkeit ϵ für GRIDGEN Instanzen. Anzahl der Commodities $k = 100$. Beide Achsen sind logarithmisch skaliert.

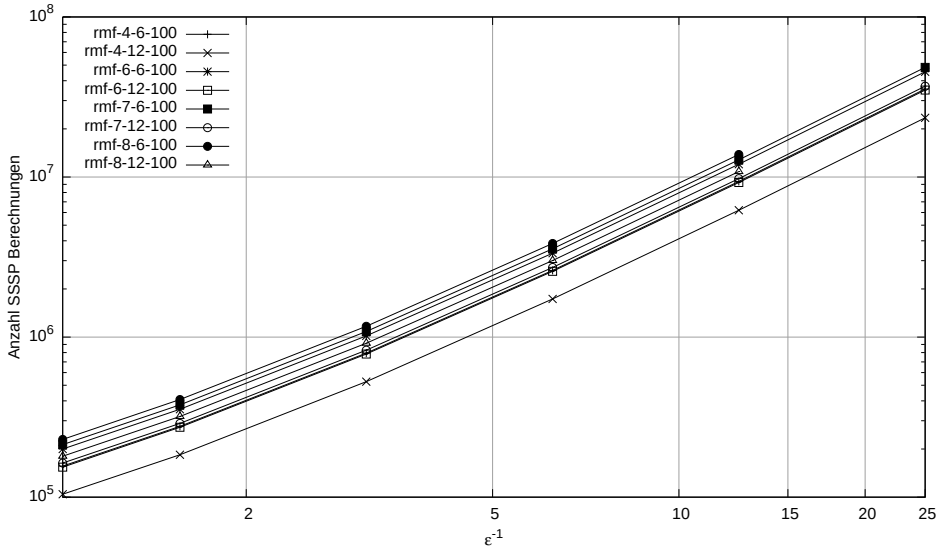

Abbildung 5.5: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Genauigkeit ϵ für GENRMF Instanzen. Anzahl der Commodities $k = 100$. Beide Achsen sind logarithmisch skaliert.

zen wurde mit einer Genauigkeit von 8% und mit einem Budgetwert, der dem Optimum entspricht, gelöst. Weiterhin wurden alle Bedarfe wie im obigen Test skaliert.

Die dargestellten Verläufe 5.6 und 5.7 scheinen für kleine k logarithmisch zu sein. Für größere k scheint die Abhängigkeit sogar weiter zu sinken. Mittels Gnuplot [69] wurde versucht eine logarithmische Funktionsvorschrift zu interpolieren. Der Fehler, der beim Test auf eine logarithmische Funktion auftritt liegt bei ca. 14%. Da Tests auf lineare oder exponentielle Funktionen einen größeren Fehler aufweisen, erscheint eine logarithmische Abhängigkeit der Laufzeit in Bezug auf k wahrscheinlich.

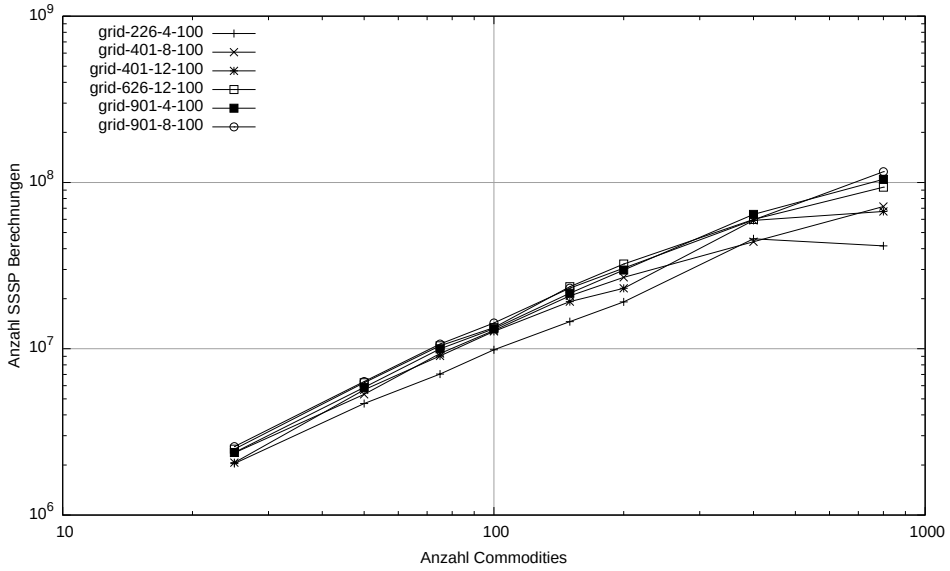

Abbildung 5.6: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Commodities k für GRIDGEN Instanzen. Genauigkeit $\epsilon = 0.08$. Beide Achsen sind logarithmisch skaliert.

Der letzte Test wurde in Bezug auf die Kanten durchgeführt. Hierfür wurden GRIDGEN und GENRMF Instanzen verschiedener Größe mit $k = 100$ Commodities generiert und mit einer Genauigkeit von $\epsilon = 0.08$ in Bezug auf den optimalen Kostenwert gelöst. Das Ergebnis ist in Abbildung 5.8 dargestellt. Die beiden Punktwolken haben eine lineare bis logarithmische Form. Die GENRMF zeigen einen deutlichen linearen Verlauf, während dieser bei den GRIDGEN Instanzen linear bis leicht logarithmisch verläuft. Insgesamt entspricht dies der theoretischen Abschätzung der Anzahl der Schritte von $O(m)$.

Die durchgeführten Tests bestätigen die theoretischen Laufzeitfaktoren sowohl in Bezug auf die Anzahl der Commodities k als auch in Bezug auf die gewählte Genauigkeit ϵ sowie die Anzahl der Kanten m.

5.3.3 Bündelungseigeschaften

Der Grund für die theoretisch bessere worst-case Laufzeit des impliziten Algorithmus von Karakostas [44] gegenüber der Version von Fleischer [28] besteht in der Bündelung von

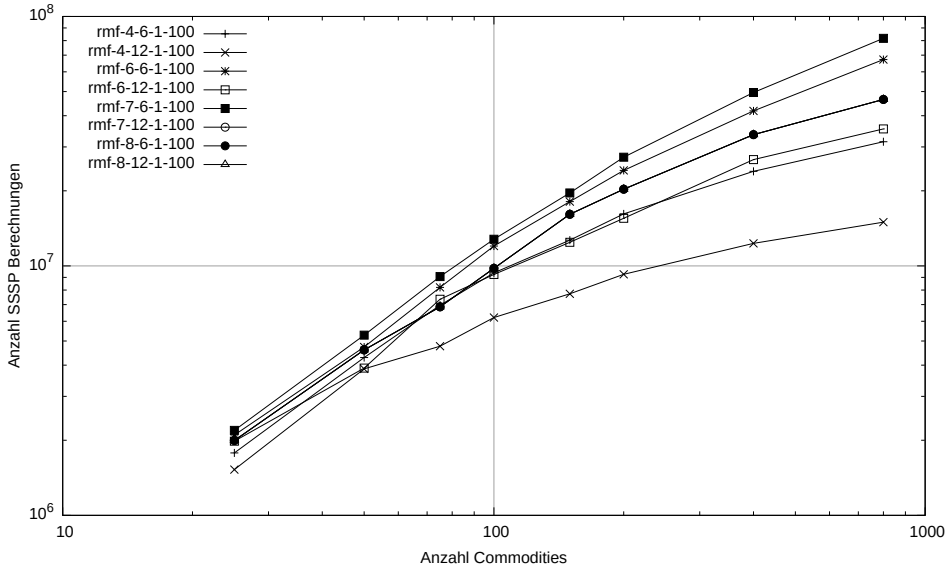

Abbildung 5.7: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Commodities k für GENRMF Instanzen. Genauigkeit $\epsilon = 0.08$. Beide Achsen sind logarithmisch skaliert.

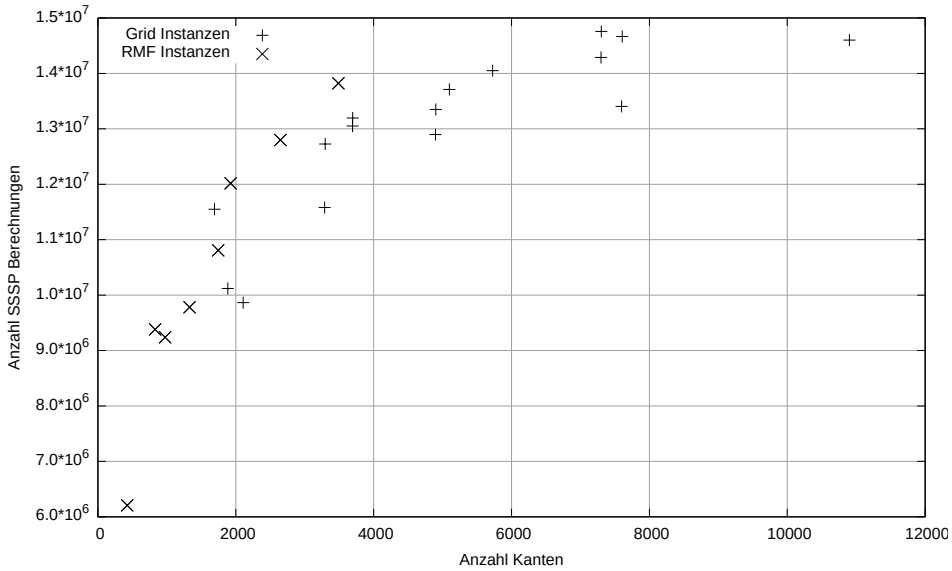

Abbildung 5.8: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Kanten m. Genauigkeit $\epsilon = 0.08$. Y-Achse logarithmisch skaliert.

Commodities mit gleicher Quelle. So können im besten Fall in einem Schritt alle Commodities gleicher Quelle gleichzeitig betrachtet werden. Im Folgenden wird untersucht, in wie weit sich dies auf die tatsächliche Laufzeit eines *maximum cost-bounded multicommodity flow* Algorithmus auswirkt. Die so getesteten Algorithmen umfassen die Version von Flei-

scher sowie den expliziten und impliziten Algorithmus von Karakostas. Die Bedarfe der Commodities wurden uniform zufällig aus dem Intervall $[1, 25]$ gewählt. Zusätzlich wurde für die erste Testrunde der PPRN-Algorithmus von Castro und Nabona [19] auf den Instanzen getestet. Eine Interpretation der Laufzeit des PPRN-Algorithmus findet in Abschnitt 5.3.5 statt.

In einer ersten Testrunde wurden drei verschiedene GENRMF Instanzen generiert. Für jede Instanz wurden 100 Commodities generiert, die paarweise verschiedene Quellen besitzen.

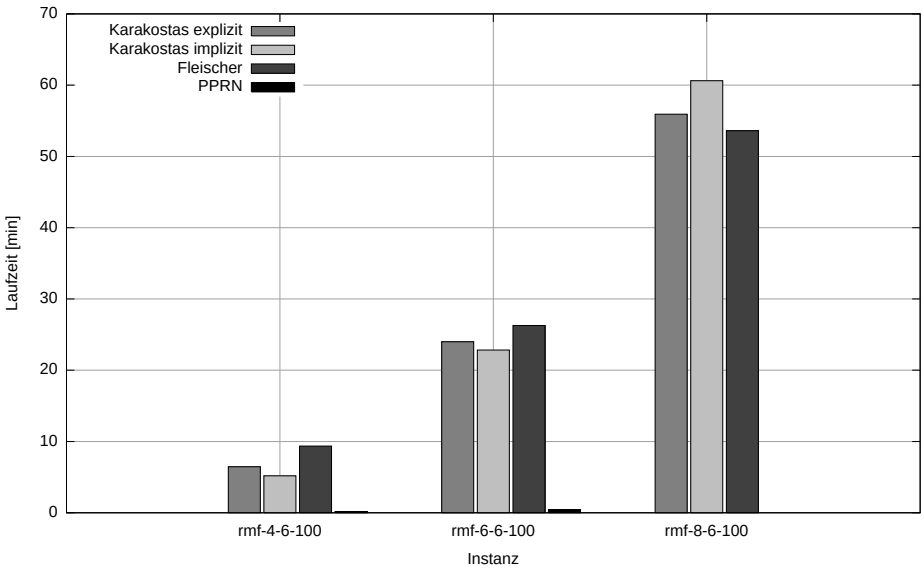

Abbildung 5.9: Laufzeit von vier Algorithmen für fünf GENRMF Instanzen mit 100 paarweise verschiedenen Commodities.

Abbildung 5.9 stellt die Laufzeit der Algorithmen in Abhängigkeit der Anzahl der Commodities dar. Alle Algorithmen verfahren bei paarweise verschiedenen Commodities exakt gleich. Dies liegt daran, dass alle Algorithmen das gleiche Vorgehensprinzip besitzen und keine Bündelung von Commodities mit gleicher Quelle möglich ist. Die trotzdem auftretenden Schwankungen der Laufzeiten können unter Umständen an einer unterschiedlichen Streuung der Commodities liegen.

In einer zweiten Testrunde wurden Commodities gebündelt. Dazu wurde eine bestimmte Anzahl an Quellen generiert und die Commodities gleichmäßig diesen Quellen zugeordnet. Es wurden zwei Quellmengen generiert: 4 Quellen mit je 25 Commodities und 10 Quellen mit je 10 Commodities.

Abbildungen 5.10 und 5.11 zeigen die Resultate. Wie man aus den beiden Abbildungen gut erkennen kann, senkt der Bündelungseffekt die Laufzeit der Algorithmen von Karako-

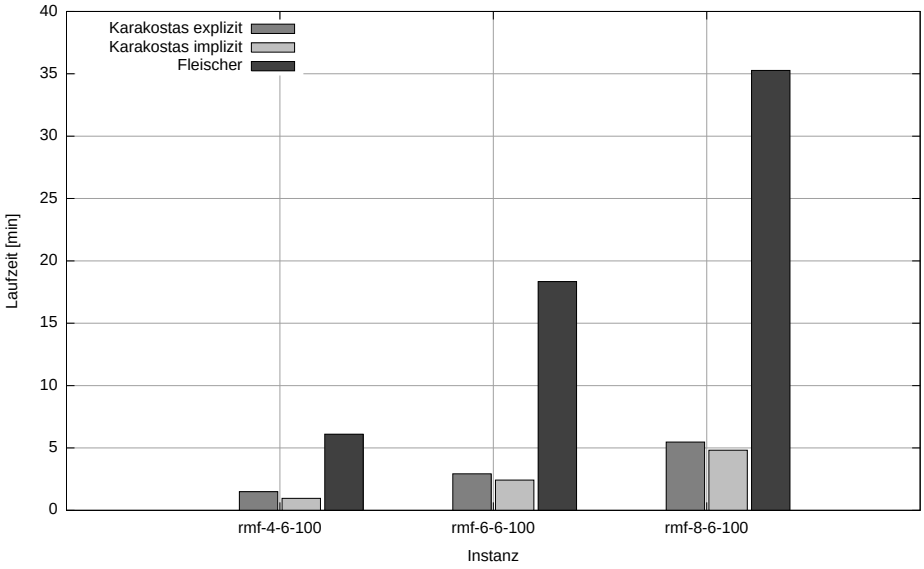

Abbildung 5.10: Laufzeit von drei Algorithmen für fünf GENRMF Instanzen mit 10 Quellen à 10 Commodities.

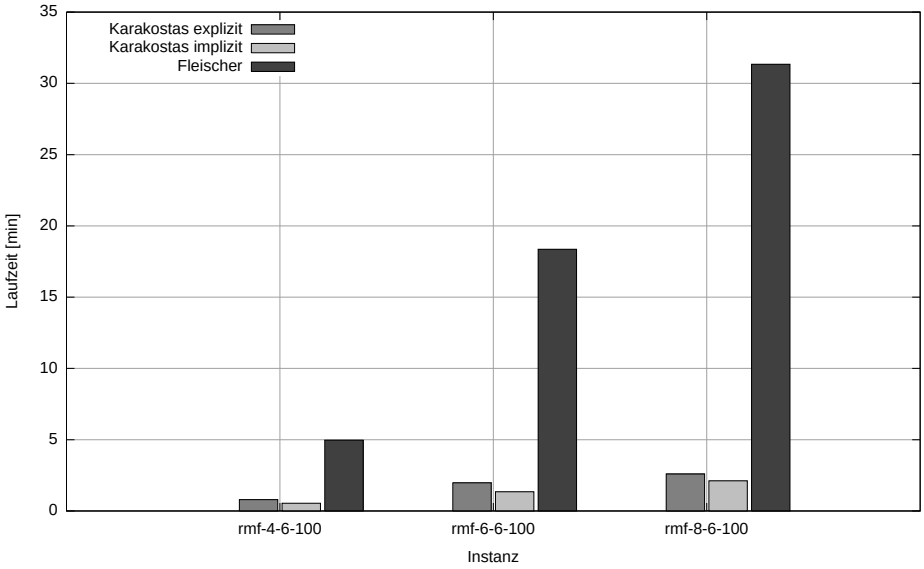

Abbildung 5.11: Laufzeit von drei Algorithmen für fünf GENRMF Instanzen mit 4 Quellen à 25 Commodities.

stas drastisch. Dieses Verhalten ist ideal für die Anwendung auf Transportnetzwerkinstanzen, da es dort üblicherweise wenige Quellen aber viele Senken gibt.

5.3.4 Logistische Instanzen

Die Aufgabe dieser Diplomarbeit besteht darin, ein Modell zu entwickeln, mit dem untersucht werden kann, inwieweit es möglich ist, in einem multimodalen Transportnetzwerk CO_2-Emissionen einzusparen. Hierzu wurde ein entsprechendes Modell in Kapitel 3 entwickelt. In diesem Abschnitt wird das Laufzeitverhalten von Karakostas implizitem Algorithmus in Bezug auf dieses Modell untersucht. Die Kenngrößen sind dabei die Anzahl an Quellen s, die Anzahl an Senken t, die Anzahl an Commodities k, die Vor- und Nachlaufradien, die in den generierten Instanzen identisch zueinander sind, r sowie der betrachtete Zeitraum t. Der letzte Parameter ist in soweit von großer Bedeutung, als das hierdurch die Menge der möglichen Zugverbindungen maßgeblich beeinflusst wird. Während bei einem Betrachtungszeitraum von einem Tag nur 20 Zugverbindungen möglich sind, so sind es bei zwei Tagen bereits 525. Während die Ergebnisse dieser Tests in Abschnitt 5.4 analysiert und interpretiert werden, diskutiert dieser Abschnitt das Laufzeitverhalten der generierten Transportnetzwerkinstanzen mit dem impliziten Algorithmus von Karakostas.

Die folgenden Abbildungen 5.12, 5.13 und 5.14 zeigen die Verläufe des Algorithmus in Abhängigkeit der Anzahl der Commodities für $s = 5, 10$ und 15 Quellen für einen Tag, also 20 Zugverbindungen. Dabei wurde das Transportnetzwerk in der ursprünglichen Version erstellt, d. h. für jede Zugverbindung existieren Kanten im Vorlauf zu allen Quellen und Kanten im Nachlauf zu allen GVZ und Senken. Der Radius r beträgt 350 km. Die Größe der Graphen steigt mit wachsender Anzahl Commodities linear und reicht von 700 Knoten und 1050 Kanten für die kleinste bis zu 1940 Knoten und 6340 Kanten für die größte Instanz.

Man kann anhand der 3 Diagramme 5.12, 5.13 und 5.14 erkennen, dass sich die Anzahl der Schritte bei kleinen Instanzen quadratisch entwickelt, während sich der Verlauf der Funktionswerte bei steigender Instanzgröße immer stärker abflacht.

5.3.5 Vergleich zu anderen Algorithmen

In diesem Abschnitt wird die implizite Version des *minimum cost multicommodity flow* Algorithmus von Karakostas [44] mit dem PPRN Algorithmus von Castro und Nabona [19] verglichen. Dieser Algorithmus löst *multicommodity* Probleme durch eine spezielle primale Partitionierung der Eingabeinstanzen. Dabei wird das Problem aufgeteilt in einen Teil für die Betrachtung der einzelnen Commodities und einen Teil, der sich mit den Kantenkapazitäten, die von vielen Commodities benutzt werden, beschäftigt. In einer ersten Phase werden die einzelnen Commodityprobleme durch ein Netzwerk-Simplex-Verfahren gelöst. Anschließend wird ein lineares Programm in Form einer Matrix gelöst, um den Fluss aller Commodities so über das Netz zu verteilen, dass keine Kantenkapazitäten verletzt werden und die Kosten minimal sind.

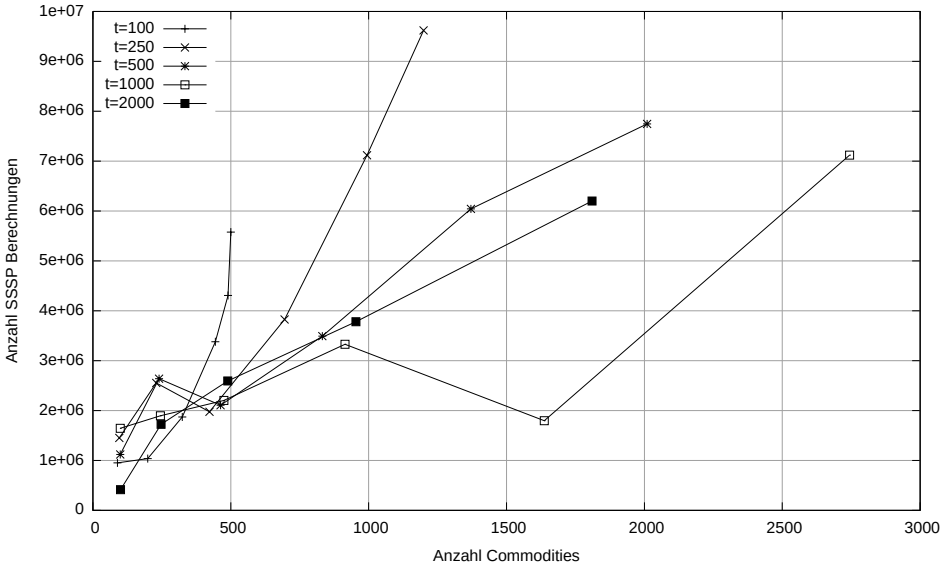

Abbildung 5.12: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Commodities k für unterschiedliche Anzahl an Senken. Anzahl Quellen $s = 5$, Genauigkeit $\epsilon = 0.08$, Radius $r = 350km$.

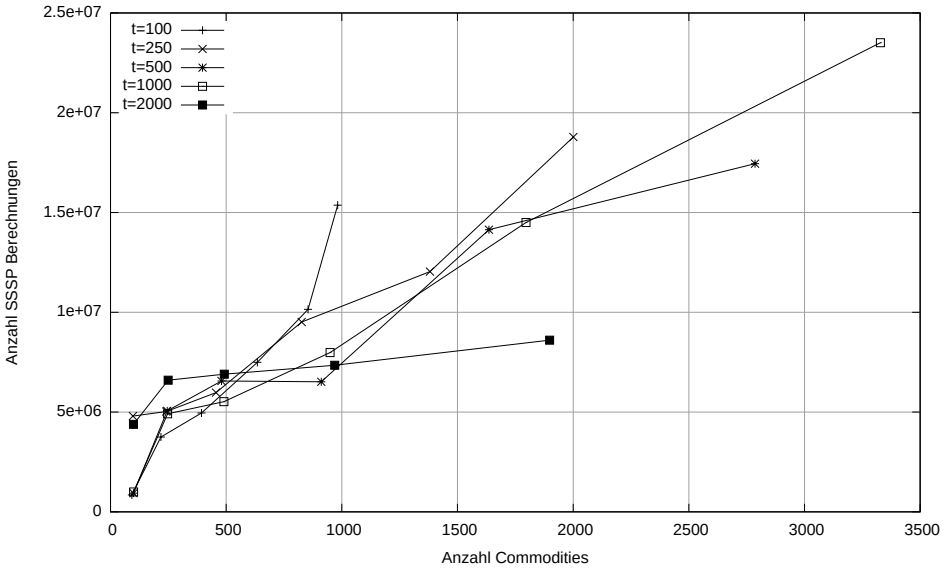

Abbildung 5.13: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Commodities k für unterschiedliche Anzahl an Senken. Anzahl Quellen $s = 10$, Genauigkeit $\epsilon = 0.08$, Radius $r = 350km$.

Goldberg et al. [36] implementierten den Algorithmus von Karger und Plotkin [45] unter Verwendung zahlreicher Optimierungen und verglichen ihn mit PPRN. Dabei stellte

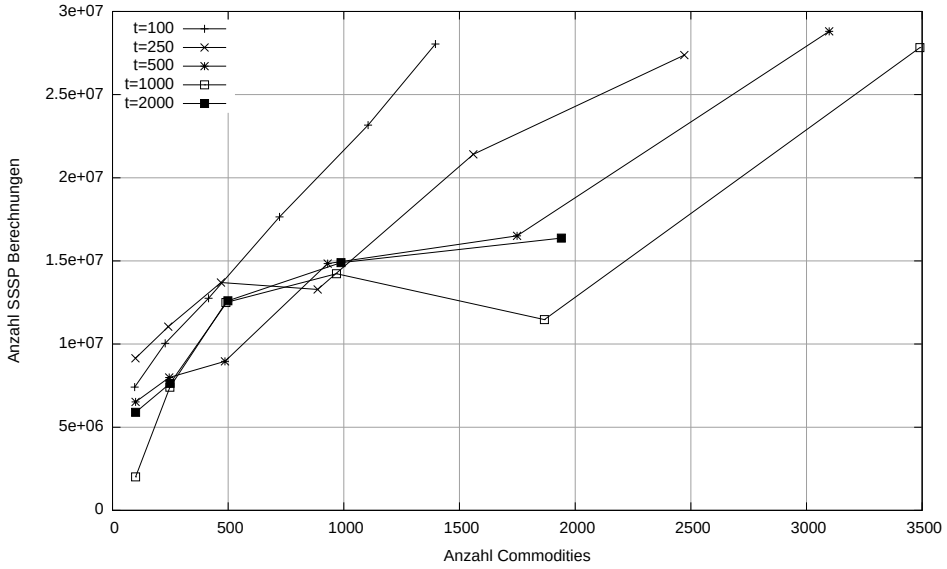

Abbildung 5.14: Anzahl der *single source shortest path* Berechnungen in Abhängigkeit der Anzahl der Commodities k für unterschiedliche Anzahl an Senken. Anzahl Quellen $s = 15$, Genauigkeit $\epsilon = 0.08$, Radius $r = 350km$.

sich heraus, dass PPRN eine deutlich schlechtere Laufzeit als der von Goldberg et al. implementierte Algorithmus.

Für die in dieser Diplomarbeit getesteten Instanzen wurden ebenfalls mit PPRN verglichen, siehe Abbildung 5.9. Die Laufzeiten der beiden Algorithmen unterscheiden sich jedoch deutlich. Während PPRN alle Instanzen binnen wenigen Sekunden bis Minuten optimal löste, so brauchte die in dieser Arbeit implementierte implizite Version von Karakostas oft deutlich länger um relativ kleine *maximum cost-bounded flow* Probleme mit einer Genauigkeit von 8% zu lösen. Hierbei sind die Iterationen, die nötig sind, um einen minimalen Budgetwert zu bestimmen, noch nicht mit berücksichtigt.

Betrachtet man die Verbesserungen, die Goldberg et al. in [36] für einen ähnlichen Algorithmus angewendet haben, genauer an, so fällt auf, dass die größte Verbesserung durch die dynamische Anpassung der Schrittweite erzielt wurde. Hierbei konnten Verbesserungen um einen Faktor von bis zu 190 in der Laufzeit erreicht werden. Dies bestätigt die Vermutungen von Goldberg et al [36] und Radzik [57], dass eine statische Schrittweite ungünstig für eine schnelle praktische Laufzeit ist.

5.4 Logistische Analyse

Dieser Abschnitt beschäftigt sich mit der Analyse der Ergebnisse von Tests der Algorithmen auf den Transportnetzwerk-Instanzen aus Kapitel 3. Die zugrunde liegende Fragestellung

ist, inwieweit der Ausstoß von CO_2-vermindert werden kann, indem Transporte von der Straße auf die Schiene verlagert werden.

Hierfür wurden vier Szenarien, die auf den allgemeinen Eigenschaften aus Abschnitt 5.2.3 basieren, untersucht. Szenario S^1_{350} betrachtet einen Tag und einen Vor- und Nachlaufradius von 350km. Die Szenarios S^2_{350}, C^2_{150} und S^2_{100} hingegen untersuchen einen Zeitraum von 2 Tagen und einen Vor- und Nachlaufradius von 350km bzw. 150km und 100km. Ein großer Unterschied zwischen dem Betrachtungszeitraum von einem und zwei Tagen liegt in Anzahl der möglichen Bahnverbindungen. Diese betragen im ersten Fall 20 während im zweiten 525 Verbindungen laut Fahrplan möglich sind. Dies wirkt sich stark auf die Größe des erzeugten Netzwerks aus. Aus diesem Grund wurden bei den Transportnetzwerken, die einen Zeithorizont von zwei Tagen betrachtet, vereinfachte Annahmen zugrunde gelegt. Instanzen aus S^1_{350} wurden so generiert, wie es im Modell in Kapitel 3 ausgeführt ist. Für die übrigen drei Szenarien wurden die Varianten aus Abschnitt 3.3.5 generiert. Dies bedeutet, dass es ausreichend ist zu gewährleisten, dass der Transport einer Sendung innerhalb des definierten Zeithorizonts möglich ist. Die Wartezeit an einer Quelle wird nun nicht explizit betrachtet. Stattdessen existiert von jeder Quelle eine Kante zu jeder Senke und jedem GVZ innerhalb des entsprechenden Radius. Außerdem wurde die Möglichkeit der Nutzung mehrerer Zugverbindungen für einen Transport eliminiert. Hierfür wurden, im Gegensatz zum originalen Modell, keine Kanten von einem GZV zu einem anderen generiert. Diese Veränderungen in der Netzwerkstruktur führen zu einer deutlichen Verkleinerung der erzeugten Transportnetzwerke. Diese Veränderung ist nötig, damit die Berechnung in annehmbarer Zeit durchgeführt werden können. Dennoch mussten vor allem bei den Szenarien, die zwei Tage untersuchen, aufgrund der benötigten Laufzeit Abstriche in der Anzahl der untersuchten Instanzen gemacht werden.

Innerhalb der vier Szenarien wurden Kombinationen aus $s \in \{5, 10, 15\}$ Quellen, $t \in \{100, 250, 500, 1000, 2000\}$ Senken und $k \in \{100, 250, 500, 1000, 2000\}$ getestet. Für Szenario S^1_{350} wurde alle Kombinationen aus s, t und k untersucht. Für die zwei-tägigen Szenarien wurden nur ausgewählte Instanzen getestet. Dies lag vor allem an der Laufzeit der Berechnung als auch an der Instanzgröße, die bei der Speicherung der Ergebnisse durch die XStream Bibliothek zu Speicherfehlern führte. Die verwendete Kostenfunktion ist bei allen Instanzen die CO_2-Funktion. Die Analyse vergleicht die resultierenden Transportwege für den berechneten kombinierten Verkehr mit den Wegen, den Direktfahrten per LKW für jede Sendung entsprechen würden. Hierbei werden neben den CO_2-Emissionen auch die Distanz sowie die ökonomischen Kosten ausgewiesen.

Im Folgenden werden Charakteristika dieser Ergebnisse interpretiert und analysiert.

Abbildung 5.15 zeigt den Anteil der Transporte, die auf die Schiene verlagert werden können, in Abhängigkeit der k Commodities für alle Szenarien. Nach der Anzahl der Quellen und Senken wurde nicht unterschieden. An dieser Graphik lassen sich mehrere Eigenschaften des Transportnetzwerks ablesen. Zum einen fällt der Zusammenhang zwi-

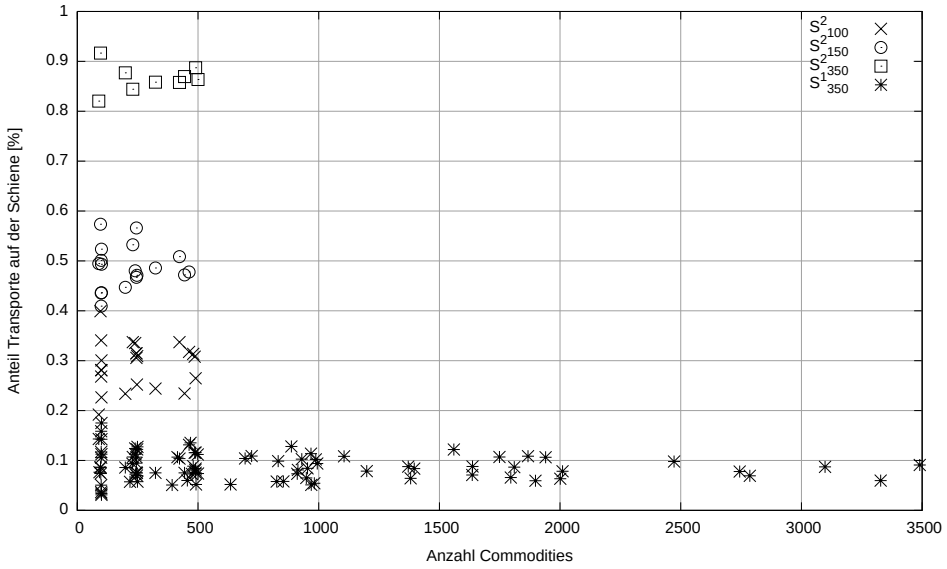

Abbildung 5.15: Anteil der Transporte, die mittels kombiniertem Verkehr transportiert werden an der Gesamttransportmenge für mehrere Szenarien.

schen Vor- und Nachlaufradius und dem Anteil der Schienentransporte auf. Je größer der Radius, desto mehr Transporte können durch den kombinierten Verkehr mit dem Zug realisiert werden. Da sich der Betrachtungsraum für Transporte in dieser Diplomarbeit auf Deutschland beschränkt, kann mit einem Radius von 350 km im Vor- und Nachlauf ein großer Teil des Landes abgedeckt werden. Tabelle 5.1 zeigt für jedes der drei Szenarien den minimalen und maximalen Anteil des kombiniertem Verkehrs sowie das arithmetische Mittel und die Standardabweichung. Auffallend ist die große Streuung der Messpunkte bei kleinem k. Dies lässt sich damit begründen, dass bei einer hohen Anzahl an Commodities viele Senken gleichmäßig über Deutschland verteilt liegen. Dadurch ist die Belieferung durch Züge wahrscheinlich für viele Commodities möglich. Ist k klein, so ist vermutlich die Anzahl der Senken innerhalb eines GVZ wesentlich geringer bzw. es existiert kein GVZ in Reichweite einer Senke. Für den Fall das dieser Effekt bei einer Instanz stark ausgeprägt ist sinkt der Anteil der kombinierten Verkehre während er bei einer starken Clusterung der Senken um ein GVZ ein großer Teil der Waren per Zug transportiert werden können. Die Wahrscheinlichkeit einer solchen unregelmäßigen Verteilung ist bei kleinem k größer als bei einer großen Anzahl an Commodities. Ebenfalls eine Folge aus dieser Überlegung ist die Eigenschaft, dass sich der Anteil des kombinierten Verkehrs mit steigender Anzahl der Commodities immer stärker an das arithmetische Mittel annähert, da die Verteilung bei steigendem k gleichmäßiger wird.

Ein weiterer Faktor, der stark damit zusammenhängt, wie viele Güter über die Schiene transportiert werden, ist der betrachtete Zeithorizont. Während bei der Betrachtung eines

Tages nur 20 Zugverbindungen möglich sind, so existieren 525 Verbindungen, die innerhalb eines Zeitraums von zwei Tagen operieren. Dies wirkt sich entscheidend auf die Anzahl der Transporte, die kombiniert erfolgen können, aus. Auch dies lässt sich aus Abbildung 5.15 erkennen.

Szenario	Minimum	Maximum	arithm. Mittel	Standardabweichung
S^1_{350}	0,03	0,18	0,09	0,03
S^2_{100}	0,19	0,4	0,29	0,05
S^2_{150}	0,41	0,57	0,49	0,04
S^2_{350}	0,82	0,92	0,87	0,03

Tabelle 5.1: Übersicht der Anteile des kombinierten Verkehrs am Gesamttransportaufkommen.

Eine wichtige Zahl, die aus den durchgeführten Tests entnommen werden kann, ist das Einsparungspotential an CO_2-Emissionen durch eine Verlagerung der Transporte auf die Schiene.

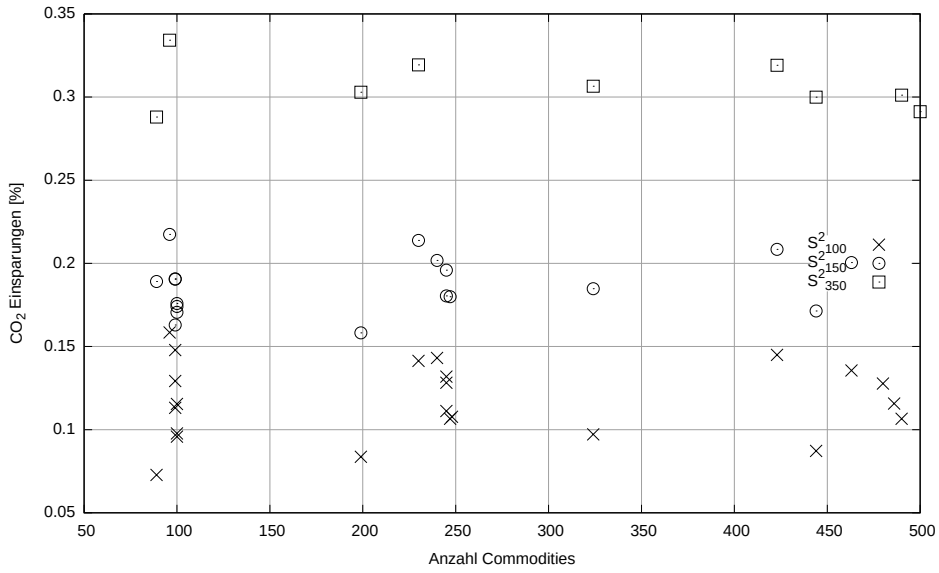

Abbildung 5.16: Einsparung von CO_2-Emissionen im Vergleich zum LKW Direktverkehr in Abhängigkeit der Anzahl an Commodities.

Abbildung 5.16 zeigt die prozentuale Einsparung von CO_2-Emissionen in Abhängigkeit der Anzahl der Commodities für die Instanzen der Szenarien $S^2_i, i \in \{100, 150, 350\}$ gegenüber LKW Direkttransporten. S^1_{350} ist aufgrund von nur 20 Zugverbindungen nicht aussagekräftig genug. Es ist deutlich zu erkennen, dass für eine realistischen Vor- und Nachlaufradius von 150km Einsparungen von rund 20% möglich sind. Allgemein ist zu erkennen, dass die Einsparungen wachsen, je größer der Einzugsradius der GVZ gewählt

wurde. Denn nur wenn der Radius um ein GVZ groß ist, ist auch die Wahrscheinlichkeit höher, dass eine Quelle oder Senke in diesem Einzugsbereich zu finden ist.

Schließlich sei noch auf die Verhältnisse zwischen den gemessenen Faktoren eingegangen. Neben dem Anteil des kombinierten Verkehrs und der eingesparten CO_2-Emissionen wurden auch gemessen, um wieviel Prozent die zurückgelegte Transportdistanz und die ökonomischen Kosten im Vergleich zu Direktfahrten gestiegen sind. Hierbei ist besonders hervorzuheben, dass die Distanz nur in geringem Maße in Abhängigkeit des Anteils des kombinierten Verkehrs steigt, siehe Abbildung 5.18. Dies liegt daran, dass auch LKW Verbindungen optimiert werden und so unnötig lange Wege vermieden werden. Außerdem liegen viele GVZ in direkter Nachbarschaft vieler Quellen, sodass ein Transport über ein GVZ nur zu einer geringen Distanzvergrößerung beiträgt. Da alle Kostenfunktionen linear sind ist es nicht überraschend, wenn sich dies in den ausgewerteten Testdurchläufen widerspiegelt. Abbildungen 5.17, 5.18 und 5.19 zeigen, dass die Parameter in Bezug auf den Anteil des kombinierten Verkehrs nur in einem relativ kleinen Bereich schwanken.

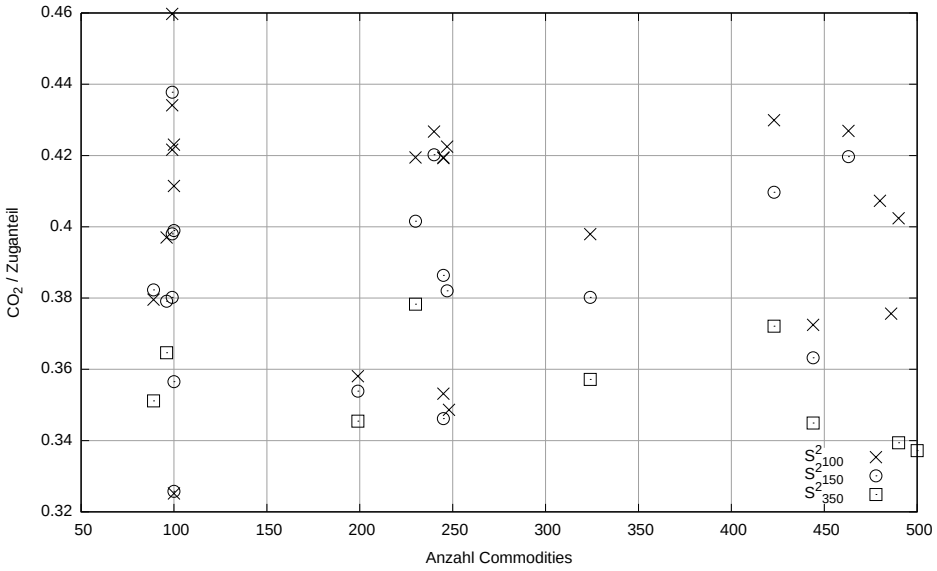

Abbildung 5.17: Verhältnis von CO_2-Einsparungen zum Anteil des kombinierten Verkehrs am Gesamtverkehr in Abhängigkeit der Anzahl an Commodities.

Als letzten Punkt wird die tatsächliche Laufzeit analysiert. Im Zuge dessen wird auch ein Blick auf die Größe der entstehenden Netzwerke geworfen. Abbildung 5.20 zeigt die Anzahl der Kanten m in Abhängigkeit der Commodities k.

Zu beachten ist hierbei, dass Szenario S^2_{350} auf dem Originalmodell aus Kapitel 3 basiert. Dies bedeutet, dass Wartezeiten an der Quelle eliminiert werden, indem für Zugverbindung eine Vorlaufkante zu allen im Radius befindlichen Quellen eingefügt wurde. Hieraus ergibt sich die relativ gesehen große Anzahl an Kanten für Instanzen dieses Sze-

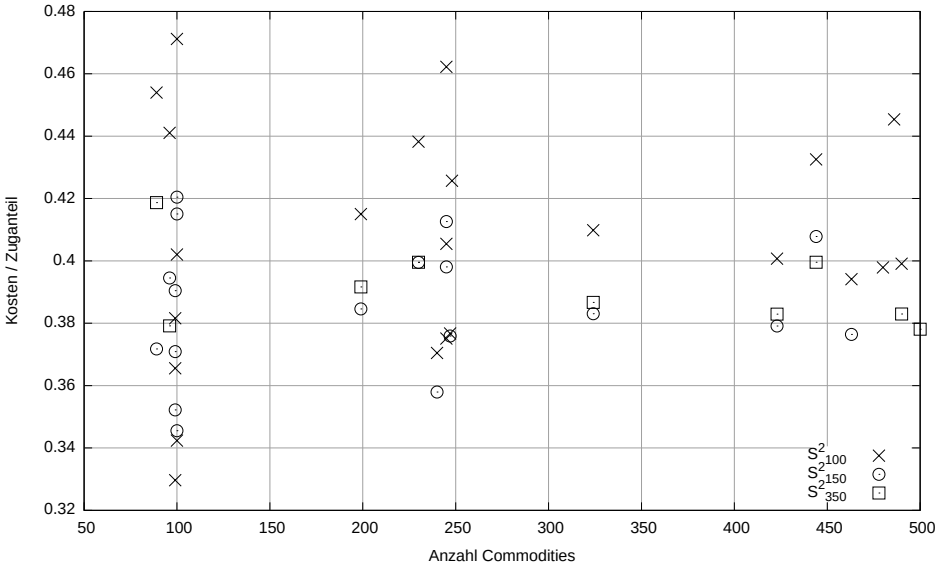

Abbildung 5.18: Verhältnis von Mehrfahrten zum Anteil des kombinierten Verkehrs am Gesamtverkehr in Abhängigkeit der Anzahl an Commodities.

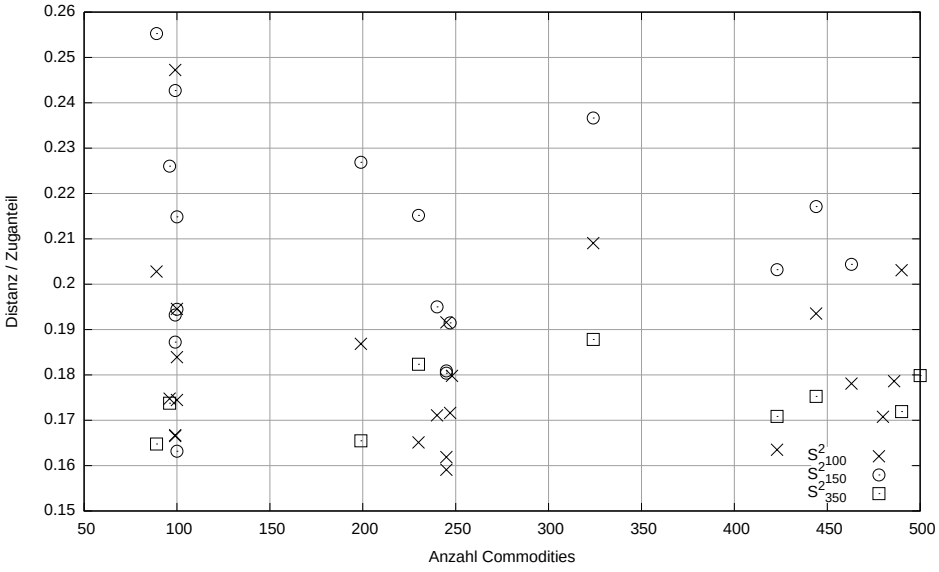

Abbildung 5.19: Verhältnis von Zusatzkosten zum Anteil des kombinierten Verkehrs am Gesamtverkehr in Abhängigkeit der Anzahl an Commodities

narios. Bei S^2_{100} und S^2_{150} wurden Modelle verwendet, die Transporte nur innerhalb eines Zeithorizonts sicherstellen und nur einen Hauptlauf zulassen. Aus diesem Grund sind die Instanzen dieser Szenarien deutlich kleiner. Die unterschiedlichen Größen für ein festes

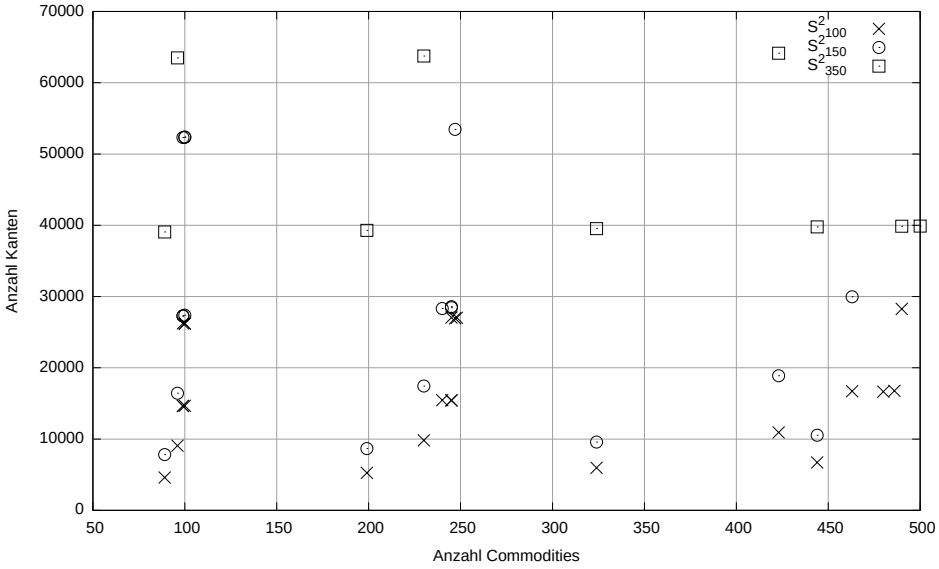

Abbildung 5.20: Größe der Transportnetzwerke anhand der Anzahl der Kanten in Abhängigkeit der Anzahl von Commodities.

Szenario und ein festes k lässt sich durch eine erhöhte Anzahl an Quellen und Senken erklären.

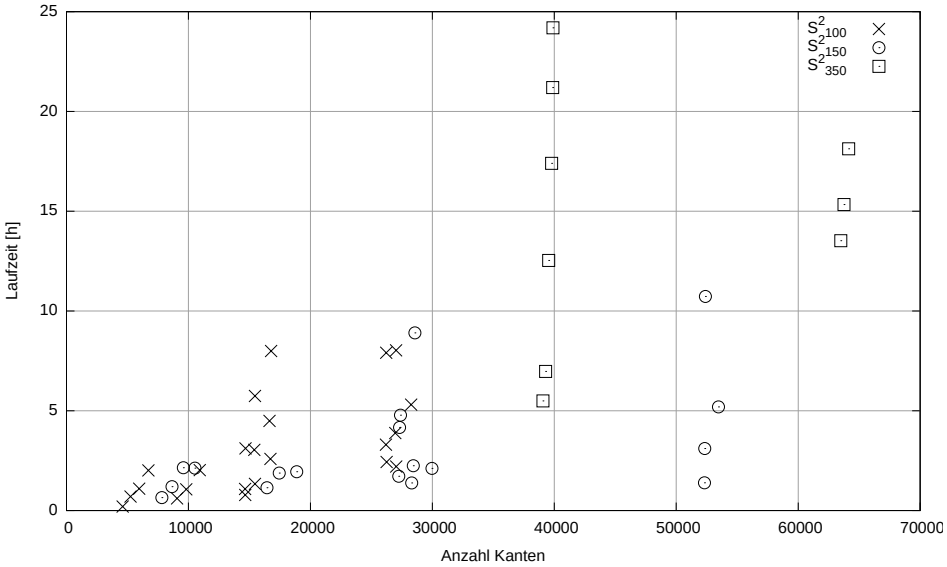

Abbildung 5.21: Laufzeit der Berechnung von *multicommodity min-cost flow* Instanzen der Transportnetzwerke in Abhängigkeit der Anzahl der Kanten.

Abbildung 5.21 zeigt die Laufzeit in Stunden für eine Instanz mit m Kanten. Hierbei ist auffällig, dass sich, bis auf S^2_{350}, die Laufzeiten der Instanzen der anderen beiden Szenarien

ungefähr im gleichen Bereich liegen. Allerdings wird auch deutlich, dass die Laufzeit mit steigender Kantenanzahl und vor allem steigender Komplexität, siehe S^2_{350}, stark ansteigen kann. Zum Teil wurden Berechnungen für diese Tests parallel durchgeführt, sodass diese Laufzeiten unter Umständen nicht denen entsprechen, die auf lastfreien Rechnern erzielt werden könnten.

Kapitel 6

Zusammenfassung / Ausblick

6.1 Zusammenfassung der Ergebnisse

In dieser Diplomarbeit wurde ein Ansatz entwickelt, der es sowohl erlaubt Logistik- und insbesondere Transportnetzwerke auf ihr Nachhaltigkeitspotential hin zu untersuchen als auch sie danach zu optimieren. Die Optimierung versucht in einem multimodalen Netzwerk Gütertransporte so zu bündeln, dass im Hauptlauf Züge anstelle von LKW eingesetzt werden können. Da Züge Massengut tauglich und wesentlich umweltfreundlicher im Bezug auf die beim Transport entstehende Schadstoffe sind als LKW, können zum Teil große Einsparungen im Bereich des CO_2-Ausstoßes erzielt werden. In Kapitel 5 wurde herausgefunden, dass durch diese Verlagerung von der Straße auf die Schiene unter Umständen CO_2-Einsparungen von bis zu einem drittel gegenüber reinen LKW-Direktverkehren erzielt werden können.

Um diese Optimierung berechnen zu können wurde in Kapitel 3 ein Modell entwickelt, dass es erlaubt ein multimodales Netzwerk, das die Straßen- und Schieneninfrastruktur vereint, abzubilden. Besondere Schwierigkeit bei dieser Synthese ist die Kombination aus immer einsetzbaren LKW-Routen und an einen Fahrplan gebundenen Zugverbindungen. Ein besonderes Merkmal der entstehenden Netzwerke ist die Tatsache, dass das entstehende Netzwerk ein Flussnetzwerk darstellt und es somit prinzipiell möglich ist dieses als Eingabe in die große Auswahl an Flussalgorithmen zu benutzen. Neben Ankunfts- und Abfahrtszeiten unterstützt das Modell ebenfalls Zeitfenster und modelliert explizit transportbegleitende Vorgänge wie das Be- oder Entladen.

Für die Berechnung der CO_2-Emissionen von LKW und Zügen existieren zwar eine Reihe von Messwerten und Tabellen, eine genau Berechnungsvorschrift zur Ermittlung des tatsächlichen Schadstoffausstoßes sucht man jedoch vergeblich. In dieser Arbeit wurde konkrete Formeln erarbeitet, die umfangreiche Berechnungsmöglichkeiten bieten. So werden für LKW verschiedene Motor-Normen, sowie der Grad der Auslastung eines LKW und der Anteil der Leerfahren berücksichtigt. Neben Auslastungsgrad und Leerfahren wird

bei der CO_2 Berechnung von Güterzügen zusätzlich berücksichtigt, ob es sich um einen Elektro- oder Dieselzug handelt. Weiterhin wurden aktuelle und konkrete Emissions- und Verbrauchswerte für Güterzüge und LKW ermittelt und verwendet. Bei den Berechnung werden sowohl die Emissionen betrachtet, die am Verkehrsmittel selbst entstehen, als auch die bei der Kraftstoffproduktion ausgestoßen werden. Darüber hinaus wird auch das zugrunde liegende Berechnungsmodell vorgestellt. So ist es möglich, durch den Austausch veralteter Emissionswerte durch aktuellere, die Berechnung auch in Zukunft korrekt durchführen zu können.

Um möglichst realistische Daten, insbesondere für die Zugverbindungen, zu erhalten, wurden aktuelle Verbindungsdaten aus dem Güterfahrplan der DB Railion AG [3] extrahiert. Weiterhin wurden exemplarisch 29 GVZ als Schnittstellen zwischen der Straße und Schiene ausgewählt. Das Straßennetzwerk sowie die Position der einzelnen Standorte wurde aus einer Geo-Datenbank extrahiert und stellen so tatsächliche Orte sowie realistische Entfernungen und Fahrtzeiten dar.

Das so entstehende Flussnetzwerk wurde mit einem deterministischen *multicommodity minimum cost flow* Algorithmus gelöst, der wiederum auf einem *maximum cost-bounded multicommodity flow* Algorithmus basiert. Bei beiden Algorithmen handelt es sich um $(1 - \epsilon)$-Approximationsalgorithmen, die ein FPTAS darstellen. Der verwendete Algorithmus von Karkostas [44] in der impliziten Variante basiert auf dem Framework von Garg und Könemann [32] und bietet ideale Eigenschaften für das oben beschriebene Transportnetzwerk. Die theoretische worst-case Laufzeit ist nur logarithmisch abhängig von der Anzahl der Commodities und unterstützt besonders Sendungsmengen, die nur aus wenigen verschiedenen Quellen bestehen. Ein weiterer Vorteil ist die mögliche Verwendung von mehreren, unterschiedlichen Kostenfunktionen.

Für den verwendeten Algorithmus existieren zwar theoretische Laufzeiten, jedoch wurden diese, nach bestem Wissen des Autors dieser Diplomarbeit, noch nicht durch praktische Analysen untersucht. In dieser Arbeit wurde das theoretische Laufzeitverhalten durch praktische Tests untersucht und bestätigt. Das gute Verhalten bei gebündelten Commodities im Gegensatz zu einem anderen Algorithmus des gleichen Frameworks, konnte ebenfalls bestätigt werden.

Der Vergleich des verwendeten Algorithmus mit dem PPRN Algorithmus von Castro und Nabona [19] offenbart jedoch eine sehr schlechte Performance, die schon auf kleinen Instanzen deutlich wird. Wie schon in Abschnitt 5.3.5 erwähnt ist vermutlich die statische Schrittweite der Hauptgrund für die großen Laufzeiten. Aber auch die großen Netzwerke, die durch obiges Modell entstehen tragen zu langen Laufzeiten bei.

6.2 Ausblick

Für das in dieser Diplomarbeit entwickelte Modell für multimodale Logistiknetzwerke sind eine Reihe von Erweiterungen denkbar. So wäre eine Erweiterung auf Schiff- und Flugverkehre denkbar. Auch für diese Verkehrsträger sind Emissionswerte vorhanden, sodass auch hier Nachhaltigkeitsanalysen und -optimierungen möglich sind. Eine genauere Untersuchung bzw. Abschätzungen in Bezug auf die Prozesse im Bereich der Be- und Entladung ist ebenfalls eine Ausbaupunkt. Hierbei ist insbesondere ein Augenmerk auf die benötigte Zeit und die damit verbundenen Kosten zu legen. Es wäre auch denkbar eine längere Pufferung von Waren an einem GVZ zu modellieren. Dies würde jedoch nur Sinn machen, wenn man einen längeren Zeithorizont betrachtet.

Ein großer Nachteil in Bezug auf die Kostenstruktur ist die Einschränkung auf lineare Funktionen. Während die Funktionen für CO_2-Emissionen linear sind, ist es jedoch nicht möglich regressive Tarife einzubauen und somit Kostenvorteile beim Transport vieler Güter ausnutzen zu können. Für diese Beschränkung ist der verwendete Algorithmus verantwortlich. In diesem Zusammenhang ist auch eine genauere Untersuchung der Kostenstrukturen für die getätigten Transporte von Interesse. Die in dieser Arbeit genutzten Kostenfunktionen berücksichtigen zwar einen realen LKW-Tarif sowie die aktuellen Kosten für einen Waggon. Jedoch ist es unwahrscheinlich, das dies die tatsächlichen Kosten widerspiegelt. So bleiben z. B. Kosten für den Umschlag oder eventuelle Rabatte und spezielle Angebote unberücksichtigt.

Ein weiterer Ansatzpunkt für eine weiterführende Arbeit ist die Verknüpfung der multimodalen Transporte mit anderen logistischen Problemen. Vorstellbar ist die Einbindung von Tourenplanungsalgorithmen im Nachlauf bei der Betrachtung von Distributionsnetzwerken. Ebenfalls denkbar wäre eine integrierte Kombination von multimodaler Transportnetzwerkoptimierung mit einer Standortoptimierung.

Anhang A

Testergebnisse

A.1 SSSP Ergebnisse

Ergebnisse des Vergleichs verschiedener *single source shortest path* Algorithmus aus Abschnitt 5.3.1.

Instanz Nummer	Knoten	Kanten	Dijkstra	GOR	TWO_Q
1	2035	227	53	81	42
2	1809	227	45	66	36
3	4813	402	122	260	140
4	1605	402	77	72	34
5	3209	402	101	154	71
6	7513	627	199	532	248
7	5635	627	174	318	263
8	5009	627	162	280	145
9	10813	902	317	979	500
10	3605	902	199	184	108
11	7209	902	226	394	173

Tabelle A.1: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen GRIDGEN Instanzen mit 100 Durchgängen pro Instanz.

Instanz Nummer	Knoten	Kanten	Dijkstra	GOR	TWO_Q
1	625	161	52	24	11
2	753	193	33	20	7
3	881	225	37	26	9
4	1009	257	45	47	10
5	241	65	12	4	3
6	369	97	16	7	4
7	497	129	22	13	6
8	1525	361	63	39	14
9	1837	443	85	52	17
10	2149	505	97	62	24
11	2461	577	111	88	24
12	589	145	24	14	4
13	901	217	46	20	9
14	1213	289	49	30	13
15	2122	491	91	58	24
16	2556	589	99	72	27
17	2990	687	124	86	32
18	3424	785	128	116	40
19	820	197	31	18	10
20	1254	295	47	30	13
21	901	217	29	18	10
22	1688	393	66	48	15
23	2817	641	106	74	42
24	3393	769	150	100	37
25	3969	897	169	135	48
26	4545	1025	185	162	60
27	1089	257	40	23	12
28	1665	385	63	43	17
29	2241	513	86	69	24

Tabelle A.2: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen GENRMF Instanzen mit 100 Durchgängen pro Instanz.

A.2 Bündelungsergebnisse

Ergebnisse des Vergleichs verschiedener Instanzen mit unterschiedlicher Bündelungsdichte der Commodities aus Abschnitt 5.3.3.

Instanz Nummer	Knoten	Kanten	Dijkstra	GOR	TWO_Q	DAG
1	4347	2439	226	66	43	36
2	3652	2190	162	58	29	25
3	39799	11676	1270	1864	978	714
4	52346	14933	1494	1248	779	860
5	52384	14969	1603	2095	907	819

Tabelle A.3: Gesamtlaufzeiten in Millisekunden für verschiedene kürzeste Wege Algorithmen auf unterschiedlichen Transportnetzwerk-Instanzen mit 100 Durchgängen pro Instanz.

Instanz	Knoten	Kanten	Karakostas im.	Karakostas ex.	Fleischer	PPRN
rmf-4-6-1-100	100	372	311	388	561	10
rmf-6-6-1-100	220	904	1370	1440	1577	24
rmf-8-6-1-100	436	1840	3637	3356	3216	70

Tabelle A.4: Gesamtlaufzeiten in Millisekunden für verschiedene GENRMF Instanzen mit 100 Quellen à 1 Commodities. Genauigkeit 8%.

Instanz	Knoten	Kanten	Karakostas im.	Karakostas ex.	Fleischer
rmf-4-6-1-100	100	372	58	90	366
rmf-6-6-1-100	220	904	146	176	1100
rmf-8-6-1-100	436	1840	290	328	2116

Tabelle A.5: Gesamtlaufzeiten in Millisekunden für verschiedene GENRMF Instanzen mit 10 Quellen à 10 Commodities. Genauigkeit 8%.

Instanz	Knoten	Kanten	Karakostas im.	Karakostas ex.	Fleischer
rmf-4-6-1-100	100	372	33	48	298
rmf-6-6-1-100	220	904	81	119	1102
rmf-8-6-1-100	436	1840	127	156	1880

Tabelle A.6: Gesamtlaufzeiten in Sekunden für verschiedene GENRMF Instanzen mit 4 Quellen à 25 Commodities. Genauigkeit 8%.

A.3 CD

Auf der beigefügten CD befinden sich alle Quelltexte, die im Rahmen dieser Diplomarbeit geschrieben wurden, sowie Ergebnistabellen für alle durchgeführten Tests. Die Datei `readme.txt` im Root-Verzeichnis der CD enthält eine kurze Beschreibung der Verzeichnisstruktur.

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] *XStream*. http://xstream.codehaus.org/.

[2] *ARTEMIS: Assessment and reliability of transport emission models and inventory systems: Final Report*. Technischer Bericht, TRL Limited, 2007.

[3] *DB Schenker Rail Deutschland AG - Güterfahrplan*. http://www.rail.dbschenker.de/site/railion/de/e__rail/gueterfahrplan/gueterfahrplan.html, 2009.

[4] *EcoPassenger*. http://www.ecopassenger.org/, 2009.

[5] *EcoTransIT*. http://www.ecotransit.org/, 2009.

[6] *GoGreen Service*. http://www.dhl.de/de/ueber-uns/nachhaltigkeit/gogreen-service.html, 2009.

[7] *IFEU: Institut für Energie- und Umweltforschung Heidelberg GmbH*. http://www.ifeu.de/, 2009.

[8] *INFRAS Forschung und Beratung*. http://www.infras.ch/d/index.php, 2009.

[9] *Kopf an*. http://www.kopf-an.de/, 2009.

[10] *Preise und Konditionen, Allgemeine Bestimmungen für Gütertransportleistungen mit Allgemeiner Preisliste / Gültig ab 1. Januar 2009*. Technischer Bericht, DB Schenker Rail Deutschland AG, 2009.

[11] *Preise und Konditionen der DB Schenker Rail Deutschland AG*. DB Schenker Rail Deutschland AG, 2009.

[12] Ahuja, R. K., T. L. Magnanti und J. B. Orlin: *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Upper Saddle River, New Jersey, 1993.

[13] Albrecht, C.: *Global Routing by New Approximation Algorithms for Multicommodity Flow*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20(5):622–632, 2001.

[14] Arnold, D., H. Isermann, A. Kuhn, H. Tempelmeier und K. Furmans: *Handbuch Logistik*. Springer, 3. Auflage, 2008.

[15] Badics, T.: Genrmf. `ftp://dimacs.rutgers.edu/pub/netflow/generators/network/genrmf/`, 1991.

[16] Barnhart, C. und H. D. Ratliff: *Modeling Intermodal Routing*. Journal of Buisness Logistics, 14(1):205–223, 1993.

[17] Bauer, R. und D. Wagner: *Batch Dynamic Single-Source Shortest-Path Algorithms: An Experimental Study*. In: *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, Band 5526 der Reihe *Lecture Notes in Computer Science*. Springer, 2009.

[18] Boardman, B. S., E. M. Malstrom, D. P. Butler und M. H. Cole: *Computer assisted routing of intermodal shipments*. Computers Industrial Engineering, 33(1-2):311–314, 1997.

[19] Castro, J. und N. Nabona: *An implementation of linear and nonlinear multicommodity network flows*. European Journal of Operational Research, 92(1):37–53, 1996.

[20] Chang, T.: *Best routes selection in international intermodal networks*. Computers Operations Research, 25:2877–2891, 2008.

[21] Cherkassky, B. V., A. V. Goldberg und T. Radzik: *Shortest paths algorithms: Theory and experimental evaluation*. Mathematical Programming, 73(2):129–174, 1996.

[22] Cormen, T. H., C. E. Leiserson, R. L. Rivest und C. Stein: *Introduction to Algorithms*. MIT Press, 2. Auflage, 2001.

[23] Crainic, T. G. und G. Laporte: *Planning models for freight transportation*. European Journal of Operational Research, 97:409–438, 1997.

[24] Delling, D., P. Sanders, D. Schultes und D. Wagner: *Engineering Route Planning Algorithms*. In: *Algorithmics of Large and Complex Networks*, Band 5515 der Reihe *Lecture Notes in Computer Science*, Seiten 117–139. Springer, 2009.

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

[26] Dinic, E. A.: *Algorithm for aolution of a probelm of maximum flow in a network with power estimation*. Soviet Mathematics Doklady, 11(5):1277–1280, 1970.

[27] Edmonds, J. und R. M. Karp: *Theoretical improvements in the algorithmic efficiency for network flow problems.* Journal of the Association for Computing Machinery, 19(2):248–264, 1972.

[28] Fleischer, L.K.: *Approximating Fractional Multicommodity Flow Independent of the Number of Commodities.* SIAM Journal on Discrete Mathematics, 13:505, 2000.

[29] Ford, L. R. und D. R. Fulkerson: *Flows in Networks.* Princeton University Press, 1962.

[30] Frigionia, D., A. Marchetti-Spaccamela und U. Nanni: *Fully Dynamic Algorithms for Maintaining Shortest Paths Trees.* Journal of Algorithms, 34(2):251–281, 2000.

[31] Gallo, G. und S. Pallottino: *Shortest path algorithms.* Annals of Operations Research, 13(1):1–79, 1988.

[32] Garg, N. und J. Könemann: *Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems.* In: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Washington, DC, USA, 1998.

[33] Glover, F., R. Glover und D. Klingman: *Computational study of an improved shortest path algorithm.* Networks, 14(1):25–36, 1983.

[34] Goldberg, A. V. und T. Radzik: *A heuristic improvement of the Bellman-Ford algorithm.* Applied Mathematics Letters, 6(3):3–6, 1993.

[35] Goldberg, A. V. und R. E. Tarjan: *A new approach to the maximum flow problem.* Journal of the ACM, 35(4):921–940, 1988.

[36] Goldberg, A.V., J.D. Oldham, S.A. Plotkin und C. Stein: *An Implementation of a Combinatorial Approximation Algorithm for Minimum-Cost Multicommodity Flow.* In: *Proceedings of the 6th International IPCO Conference on Integer Programming and Combinatorial Optimization*, Seiten 338–352, 1998.

[37] Gosling, J., B. Joy, G. Steele und G. Bracha: *The Java^{TM}Language Specification.* Addison Wesley, 3. Auflage, 2005.

[38] Grigoriadis, M. D. und L. G. Khachiyan: *Approximate minimum-cost multicommodity flows in $O(e^{(} - 2)KNM)time. MathematicalProgramming, 75 : 447 - -482, 1996.$*

[39] Gudehus, T.: *Logistik 2: Netzwerke, Systeme und Lieferketten.* Springer, 3. Auflage, 2006.

[40] IFEU - Institut für Energie- und Umweltforschung Heidelberg GmbH: *EcoTransIT: Ecological Transport Information Tool: Environmental Methodology and Data*, 2008.

[41] INFRAS: *Handbuch Emissionsfaktoren des Strassenverkehrs, Version 2.1.*

[42] Jünemann, R. und T. Schmidt: *Materialflußsysteme. Systemtechnische Grundlagen.* Springer, 2. Auflage, 1999.

[43] Kamath, A., O. Palmon und S. Plotkin: *Fast Approximation Algorithm for Minimum Cost Multicommodity Flow.* In: *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, Seiten 493–501, 1995.

[44] Karakostas, G.: *Faster approximation schemes for fractional multicommodity flow problems.* In: *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, Seiten 166–173. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2002.

[45] Karger, D. und S. Plotkin: *Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows.* In: *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, Seiten 18–25, 1995.

[46] Kershenbaum, A.: *A note on finding shortest path trees.* Networks, 11(4):399–400, 1981.

[47] Lee, Y. und J. Orlin: Gridgen. `ftp://dimacs.rutgers.edu/pub/netflow/generators/network/gridgen/`, 1991.

[48] Leighton, T., F. Makedon, S. Plotkin, C. Stein, E. Tardos und S. Tragoudas: *Fast Approximation Algorithms for Multicommodity Flow Problems.* Journal of Computer and System Sciences, 50(2):228–243, 1995.

[49] Macharis, C. und Y. M. Bontekoning: *Opportunities for OR in intermodal freight transport resarch: A review.* European Journal of Operational Research, 153(2):400–416, 2004.

[50] McKinnon, A.: *CO2 Emissions from Freight Transport in the UK.* Logistics Research Centre, Heriot-Watt University, Edinburgh, 2008.

[51] Min, H.: *International Intermodal Choices via Chance-Constrained Goal Programming.* Transportation Research - Part A: General, 25(6):351–362, 1991.

[52] Moccia, L., J.-F. Cordeau, G. Laporte, S. Røpke und M. P. Valentini: *Modeling and solving a multimodal routing problem with timetables and time windows.* submitted to Networks, 2008.

[53] Narváez, P., K.-Y. Siu und H.-Y. Tzeng: *New dynamic algorithms for shortest path tree computation.* IEEE/ACM Transactions on Networking (TON), 8(6):734–746, 2000.

[54] Pallottino, S.: *Shortest-path methods: Complexity, interrelations and new propositions.* Networks, 14(2):257–267, 1983.

[55] Pape, U.: *Implementation and efficiency of Moore-algorithms for the shortest route problem.* Mathematical Programming, 7(1):212–222, 1974.

[56] Plotkin, S., D. Shmoys und É. Tardos: *Fast approximation algorithms for fractional packing and convering problems.* Mathematic of Operations Research, 20:279–301, 1995.

[57] Radzik, T.: *Experimental study of a solution method for multicommodity flow problems.* In: *Proc. of the 2. Workshop on Algorithm Engineering and Experiments*, Seiten 79–102, 2000.

[58] Ramalingam, G. und T. Reps: *On the computational complexity of dynamic graph problems.* Theoretical Computer Science, 158(1-2):233–277, 1996.

[59] Samaras, Z.: *European Database of Vehicle Stock for the Calculation and Forecast of Pollutant and Greenhouse Gases Emissions with TREMOVE and COPERT.* Technischer Bericht, http://lat.eng.auth.gr/copert, 2008.

[60] Sandvik, E. T.: *Environmental impacts of intermodal freight transport.* Technischer Bericht, Møreforsking Molde AS, Molde University College, 2005.

[61] Sbihi, A. und R. W. Eglese: *Combinatorial optimization and Green Logistics.* 4OR: A Quarterly Journal of Operations Research, 5(2):99–116, 2007.

[62] School's Science Education, University of Bergen Centre of: *CO2connect.* `http://co2nnect.org/`, 2009.

[63] Shahrokhi, F. und D. W. Matula: *The maximum concurrent flow problem.* Journal of the ACM, 37(2):318–334, 1990.

[64] Umweltbundesamt: *Feinstaubbelastung in Deutschland*, 2009.

[65] Verkehr, Bau und Stadtentwicklung Bundesministerium für: *Verkehr in Zahlen 2008/09.* DVV Media Group, 2008.

[66] Wegener, I.: *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen.* Springer, 2003.

[67] Wiberg, N., E. Wiberg und A. F. Holleman: *Lehrbuch der Anorganischen Chemie*. Gruyter, 102. Auflage, 2007.

[68] Willhalm, T. und D. Wagner: *Speed-Up Techniques for Shortest-Path Computations*. In: *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS'07)*, Seiten 23–26, 2007.

[69] Williams, T. und C. Kelley: *Gnuplot 4*. `http://www.gnuplot.info/`, 2009.

[70] Yamada, T., B. F. Russ, J. Castro und E. Taniguchi: *Designing Multimodal Freight Transport Networks: A Heuristic Approach and Applications*. Transport Science, 43(2):129–143, 2009.

[71] Zhan, F. B. und C. E. Noon: *A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths*. Journal of Geographic Information and Decision Analysis, 4(2):1–11, 2002.

[72] Ziliaskopoulos, A. und W. Wardell: *An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays*. European Journal of Operational Research, 125(3):486–502, 2000.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 3. Dezember 2009

Matthias Woste