FAKULTÄT FÜR !NFORMATIK

Design eines sicherheits-, zeit- und kostenkritischen Kommunikationsnetzwerkes mittels Lagrange Relaxierung und Spaltengenerierung

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom-Ingenieurin

im Rahmen des Studiums
Software Engineering & Internet Computing

eingereicht von

Nina Musil
Matrikelnummer 0325888

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Univ.Ass. Mag. Dipl.-Ing. Andreas Chwatal

Wien, 27. November 2008

(Unterschrift Verfasserin) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Nina Musil
Eslarngasse 15/68
1030 Wien

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Wien, 27. November 2008,

Abstract

Efficient network routing is becoming more and more important – examples are internet telephony or event-based message systems. The particular problem emerged from the industry cooperation SWIS in the context safety-critical communication of air traffic management. In this thesis the design of a network is optimized for design and protocol costs. The goal is to find a minimum cost network enabling the simultaneous routing of multiple messages. Capacity constraints for each edge, delay constraints for individual messages, as well as a global delay, and security constraints make the optimization difficult.

In this thesis several algorithmic approaches for solving this NP-hard problem are discussed. An Integer Linear Programming based approach using a multi-commodity flow formulation enables to solve small problem instances to provable optimality.

To simplify the problem constraints are brought into the objective function by attaching Lagrangean multipliers to them. This approach is called Lagrangean relaxation. This yields independent subproblems for each transport. In an iterative process the coefficients of the Lagrange multipliers are systematically adapted and lower bounds for the original problem are obtained. Heuristics can be used to derive valid solutions.

Based on an alternative formulation, which introduces a variable for each possible path, the problem is solved by Column Generation. Starting from a valid solution, further improving variables are iteratively added. These variables can be found by solving the same subproblem as for Lagrangean relaxation.

By the presented algorithms small instances can be solved within a few minutes to optimality. For larger instances good heuristic solutions can be found. Tight lower bounds obtained by Lagrangean relaxation enable to measure the quality of heuristic solutions. Based on extensive experimental tests, pros and cons of each approach are discussed in this thesis.

Zusammenfassung

Das effiziente Senden von Daten über Netzwerke gewinnt zunehmend mehr an Relevanz – von Internettelefonie bis hin zu Event-basierten Nachrichtensystemen. Die konkrete Aufgabenstellung entstand aus dem Industrieprojekt SWIS im Bereich der sicherheitskritischen Kommunikation bei der Flughafensicherung. In dieser Arbeit wird das Design eines Netzwerkes, in welchem mehrere Nachrichten gleichzeitig übermittelt werden sollen, hinsichtlich Erstellungs- und Übertragungskosten optimiert. Die Verbindungen sind durch Kapazitäten beschränkt, die Nachrichten müssen Zeitvorgaben hinsichtlich der Übermittlungsdauer und Vorgaben bezüglich des Übertragungsprotokolls einhalten.

In dieser Arbeit werden verschiedene algorithmische Ansätze zur Lösung dieses NP-schweren Problems erörtert. Ein Verfahren ganzzahliger linearer Programmierung basierend auf einer Mehrgüterflussformulierung ermöglicht das exakte Lösen kleiner Probleminstanzen.

Um das Problem zu vereinfachen werden bei der Lagrange Relaxierung schwierige Nebenbedingungen als Lagrange-Multiplikatoren in die Zielfunktion aufgenommen. Dadurch ergibt sich für jeden Transport ein unabhängiges Teilproblem. Aufgrund dessen werden durch iterative Verfahren die Koeffizienten der Lagrange-Multiplikatoren angepasst und somit untere Schranken für das Ausgangsproblem gefunden. Durch Heuristiken wird versucht daraus gültige Lösungen zu erzeugen.

Basierend auf einer alternativen Problemformulierung, bei welcher jedem möglichen Pfad eine Variable entspricht, wird das Problem durch Spaltengenerierung gelöst. Ausgehend von einer gültigen Lösung werden in einem iterativen Prozess nur jene Variablen hinzugefügt, welche die Lösung weiter verbessern können. Die Bestimmung dieser Variablen erfolgt aufgrund der Lösung des selben Subproblems wie bei der Lagrange Relaxierung.

Mit den vorgestellten Verfahren können kleine Instanzen in wenigen Minuten beweisbar optimal gelöst werden, für größere Instanzen können gute heuristische Lösungen gefunden werden. Die scharfen unteren Schranken der Lagrange Relaxierung ermöglichen zuverlässige Aussagen über die Qualität heuristischer Lösungen. Umfangreiche experimentelle Tests belegen die Vor- und Nachteile der vorgestellten Verfahren.

Danksagungen

Ich möchte mich sehr herzlich bei meinen Betreuern Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl und Univ.Ass. Mag. Dipl.-Ing. Andreas Chwatal, die mir stets unterstützend zur Seite standen, bedanken. Durch ihre kritischen und konstruktiven Rückmeldungen erhielt ich immer wieder neue Blickwinkel und interessante Denkanstöße.

Mein Dank gilt weiters den Mitarbeitern des Projekts SWIS, die durch ihre Vorarbeiten und Unterstützung die Grundlage für diese Diplomarbeit lieferten.

Außerdem möchte ich meiner Familie danken, die mir mein Studium ermöglichte und immer für mich da ist. Danke auch an meinen lieben Freund für seine moralische Unterstützung während der Erstellung dieser Arbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

Die vorliegende Diplomarbeit ist aus dem Projekt "Systemweites Informationsverteilungssytem" (SWIS), einer Forschungskooperation des Instituts für Softwaretechnik und Interaktive Systeme der Technischen Universität Wien mit dem Unternehmen Frequentis, hervorgegangen. Innerhalb des fakultätsinternen Forschungsschwerpunkts "Komplexe Systeme" sollte ein Teilbereich dieses Projekts, nämlich die Optimierung eines Kommunikationsnetzwerkes mit diversen Beschränkungen hinsichtlich Kosten, näher betrachtet werden.

Ausgehend von der Entstehung des Themas und der Beschreibung des Problemumfelds, wird zunächst das SWIS-Projekt und der Industriepartner Frequentis vorgestellt. Danach werden die Anforderungen an das zu entwickelnde System erläutert. Diese ergeben sich einerseits durch den Einsatz im sicherheitskritischen Bereich, andererseits durch den Wunsch nach einer möglichst schnellen und kostengünstigen Übertragung. Anschließend wird das Problem genauer beschrieben und weitere Anwendungsgebiete des erarbeiteten Modells aufgezeigt. Abschließend werden die algorithmischen Anforderungen dargelegt.

1.1 Projekt SWIS – System-Wide Information Sharing Network

Das Projekt SWIS [DL08] ist als Forschungsprojekt im Bereich des Air Traffic Managements in Zusammenarbeit mit Frequentis [Fre08] und Austro Control [Aus08] entstanden. Ziel war es *"eine wissenschaftlich robuste und praktikable Methode zur Einführung von Informationsverteilung zwischen Partnern, bestehenden IT-Systemen und Services auf einer heterogenen Infrastruktur flexibel, performant und zuverlässig"* [Tec07] zu ermöglichen. Neben

der Beschreibung der Daten, Funktionen und Leistungsmerkmale der Systeme auf semantischer Ebene sollten *"Algorithmen für die Ableitung einer 'intelligenten Verbindung' "* entworfen werden. Basierend auf den Forschungsergebnissen sollte eine Applikation entwickelt werden, welche prototypisch vier Beispielapplikationen integriert. Die Notwendigkeit der Zusammenarbeit von unterschiedlichsten autonomen Organisationen bei der Abwicklung des Flugverkehrs, dem Air Traffic Management (ATM), zieht eine hohe Komplexität nach sich. Diese ergibt sich, neben der hohen Anzahl von teilnehmenden Systemen, aus den gestellten hohen Anforderungen: Ein in diesem Bereich eingesetztes System muss sicher, zuverlässig und ausreichend schnell sein. Auch im Krisenfall muss ein deterministisches Verhalten gewährleistet sein.

Das Projekt selbst wurde zwar Ende 2007 abgeschlossen, jedoch besteht ein erhebliches Verbesserungspotential bezüglich der Optimierung. Bei dem entstandenen Prototypen wurden *Brute-Force*-Methoden und Heuristiken eingesetzt. Vor allem Brute-Force-Methoden (Enumerationsverfahren) stoßen bei größeren Instanzen an ihre Grenzen, da das Ausprobieren aller Möglichkeiten zeitlich nicht mehr durchführbar ist. Deshalb wurde im Projekt versucht heuristische Methoden einzusetzen, bei diesen kann allerdings keine Aussage über die Güte der Lösung gegeben werden.

Im Zuge dieser Diplomarbeit wurden verbesserte exakte, als auch heuristische Optimierungsmethoden entwickelt. Diese Methoden ermöglichen auf der einen Seite größere Instanzen exakt zu lösen, und auf der anderen Seite genauere Aussagen über die Qualität der heuristischen Lösungen zu treffen.

1.2 Industriepartner des Projektes

Frequentis AG (www.frequentis.com) ist ein international operierendes Unternehmen mit Sitz in Wien, das unter anderem in den Bereichen Air Traffic Management, Public Transport und Public Safety tätig ist.

Der ursprüngliche Tätigkeitsbereich von Frequentis umfasste vor allem Sprachvermittlungssysteme im Flugsicherungsbereich, die Fernsprechen und Funksprechen ermöglichten. Mittlerweile werden die Sprachvermittlungssysteme in diversen anderen Einsatzgebieten verwendet. Weltweit sind neben Flugsicherungssystemen auch diverse Einrichtungen des Küstenfunk und des öffentlichen Personennahverkehrs und Einsatzleitzentralen mit Frequentis-Systemen ausgestattet. Die Systeme ermöglichen das Zusammenfassen von vielen Sprachverbindungen wie Notruf, Telefon, Nebenstellenanlage, analoger und digitaler Funk, Mobiltelefon, Türsprechanlage etc., und bieten Dienste wie Sprachaufzeichnung und -archivierung an. Neben dem Kommunikationssystem können bei diesen Systemen weitere Informationen integriert werden.

Dies reicht von anderen flugplatzbezogenen Informationen wie Wetterdaten, Windgeschwindigkeit in der Flugsicherung, bis hin zur Integration *Grafischer Informationssysteme* (GIS) zur Visualisierung bei der Einsatzplanung in Blaulichtorganisationen.

Um die Größenordnung der von Frequentis entwickelten Systeme zu veranschaulichen, werden nachfolgend einige Beispielprojekte vorgestellt:

- **London Metropolitan Police**
  Ausstattung der London Metropolitan Police (besser bekannt als "Scotland Yard") mit einer integrierten Befehls- und Kommunikationsplattform.

- **NASA MOVE**
  Im Rahmen von NASA MOVE ("Mission Operations Voice Enhancement"), wird eine Neuausstattung der NASA-Leitzentrale vorgenommen.

- **Norwegisches Sicherheitsnetz Noednett**
  Beim Norwegisches Sicherheitsnetz Noednett werden alle Leitzentralen der norwegischen Polizei-, Feuerwehr- und Gesundheitsorganisationen mit neuen Systemen ausgestattet.

Diese angeführten Systeme stellen klarerweise nur ein kleiner Ausschnitt aus dem Tätigkeitsbereich von Frequentis dar.

1.3 Anforderungen an ein System im sicherheitskritischen Bereich

Wie die angeführten Projekte zeigen, liegen die Problemstellungen des Kooperationspatners im sicherheitskritischen Bereich. Der Einsatz eines Systems bei einem Notruf oder bei der Flugsicherung muss gewisse Standards an Sicherheit und Geschwindigkeit einhalten, um einen reibungslosen Ablauf gewährleisten zu können. Dies sind auch die Hauptanforderungen an das Informationsverteilungssystem auf dessen Grundlage diese Diplomarbeit aufbaut:

- sicherer Datenaustausch
- Erfüllung der Zeitvorgaben

Natürlich soll die Verteilung der Informationen darüber hinaus zu möglichst geringen Kosten möglich sein.

Das dem Informationsverteilungssystem zugrunde liegende Netzwerk wird als gegeben angenommen und ist durch eine konkrete technische Umsetzung des Anwendungsszenarios bestimmt. Jegliche Kommunikation in diesem Netzwerk erfolgt über den Austausch von *Nachrichten*. Bei diesen kann es sich auch um kontinuierliche Datenströme handeln. Beispiele für Nachrichten sind Messwerte von Sensoren, Audio- oder Videoübertragungen. Die zur Optimierung herangezogenen Nachrichten entsprechen einem möglichen maximalen Datenaufkommen in einer bestimmten Zeiteinheit. Alle diese Nachrichten müssen in einer spezifizierten Maximalzeit durch das Netzwerk geroutet werden. Da das Netzwerk folglich zu einem bestimmten Zeitpunkt betrachtet wird, findet die Zeitkomponente selbst keinen unmittelbaren Eingang in die Problemformulierung. Eine zeitgleiche Nutzung der Netzwerkressourcen durch verschiedene Nachrichten ist möglich, die unterschiedlichen Nachrichten müssen sich jedoch die Kapazitäten der Leitungen im Netzwerk teilen. Die jeweiligen Kapazitätsbeschränkungen sind hierbei einzuhalten. Der Transport der Nachrichten kann gewissen zusätzlichen Einschränkungen unterworfen sein. Diese beschränken den Transport der Nachrichten auf bestimmte Leitungen und/oder Transportwege. Im vorliegenden Fall werden hierfür zwei unterschiedliche Typen unterstützt. Nachrichten können einen sicheren Transport verlangen oder zeitkritisch sein:

- **sichere Nachrichten**
  Muss eine Nachricht sicher übertragen werden, so kann ein sicheres Protokoll (z.B. Hypertext Transfer Protocol Secure (HTTPS) siehe [Res00]) auf einer sicheren Leitung dafür sorgen. Nicht jede Leitung unterstützt sichere und unsichere Protokolle. Diese haben unterschiedliche Kosten beim Transport einer Nachricht.

- **zeitkritische Nachrichten**
  Manche Nachrichten sollen in einer bestimmten Zeit beim Empfänger ankommen, da sie sonst möglicherweise nutzlos sind. Jede Leitung verursacht eine gewisse Verzögerung (Delay) und auch je nach verwendetem Protokoll dauert der Transport unterschiedlich lange.

Die auftretenden Kosten sollen so gering wie möglich sein. Die Kosten entstehen einerseits durch die generelle Verwendung einer Verbindung durch irgendeine Nachricht. Neben diesen Basiskosten treten Protokollkosten pro Nachricht je nach verwendetem Protokoll auf. Die Vergabe der Kosten verdeutlicht den Aspekt des Netzwerk-Designs. Es soll ein möglichst kostengünstiges Netzwerk ermittelt werden, welches den Anforderungen entspricht.

1.4 Anwendungsgebiete

Das beschriebene System kann wie schon besprochen in Domänen wie zum Beispiel dem Air Traffic Management verwendet werden. Es gibt jedoch auch andere mögliche Anwendungsgebiete, die nachfolgend kurz erläutert werden.

Grundsätzlich ist der Kontext solcher Anwendungen bei sicherheitskritischen verteilten Anwendungen zu finden. Mögliche Einsatzgebiete sind:

- **Event-basierte Nachrichtensysteme**
  Bei Event-basierten Nachrichtensystemen müssen sich einzelne Komponenten gegenseitig über ihren aktuellen Zustand auf dem Laufenden halten [KM07]. Letztendlich ist dies nichts anderes als ein Netzwerk bei dem kontinuierliche Daten von sich ändernden Komponenten an andere gesendet werden. Je nach Einsatzgebiet kann es notwendig sein den Transport von bestimmten Nachrichten nur über festgelegte Leitungen zu erlauben. Dies kann aus Sicherheitsüberlegungen, aber auch auf Grund anderer "Quality of Service"-Kriterien, wie zum Beispiel Ausfallssicherheit, der Fall sein.

- **Kontinuierliche Messergebnisse**
  In Netzwerken mit Messgeräten und Sensoren werden kontinuierlich Daten zur Verfügung gestellt, die an andere Komponenten des Netzwerks gesendet werden müssen. Solche Messwerte können zum Beispiel Durchflussgeschwindigkeitsmesser, Spannungsmesser und Temperatursensoren sein. Die Daten müssen innerhalb einer bestimmten Zeit beim Empfänger ankommen, da sie sonst nutzlos sind (z.B. Reaktion auf Sensordaten wird zu spät ausgelöst, neuere Daten liegen bereits vor etc.). Auch hier können bestimmte Leitungen für manche Transporte verboten sein, um Qualitätskriterien einzuhalten.

- **Video-on-demand, Internettelefonie**
  Die Live-Übertragung von Audio- und Videodaten hat einen zunehmend höheren Anteil an der Verwendung von Kommunikationsnetzwerken. Anwendungsgebiete wie Video-on-demand oder Internettelefonie müssen Übertragungsraten und maximale Verzögerungszeiten garantieren, um beim Empfänger keine sichtbaren bzw. hörbaren Pausen entstehen zu lassen. Modelle hierfür können durch das besprochene System simuliert werden. [CN98]

- **Transportplanung im öffentlichen Verkehr**
  Wie kann ein Verkehrsnetz für Fahrgäste von öffentlichen Verkehrsmitteln entwickelt (bzw. erweitert) werden, welches den Kundenwünschen

> entspricht? Dies könnte in einem einfachen Fall die Minimierung der Transportzeit sein, wobei bestimmte Fahrgäste (z.B. ältere Menschen, Personen mit Sehbehinderung, Fahrrädern, Rollstühlen, Kinderwägen, Einkaufswägen) nur bestimmte Verbindungen nutzen können. Jeder zu erwartende Transportwunsch entspricht einer Nachricht im Netzwerk. Ziel ist es die Kosten zu minimieren, während Kundenwünsche (z.B. Frequentierung, Streckenausbau, Fahrtdauer) zu beachten sind, um keine Fahrgäste an Konkurrenten – andere Anbieter oder alternative Verkehrsmittel – zu verlieren. [TZ06]

Generell können viele Probleme, die auf ein Modell von Mehrgütertransporten mit gewissen Einschränkungen zurückzuführen sind, ein Anwendungsgebiet für das erarbeitete Modell darstellen.

1.5 Algorithmische Anforderungen

Verfahren die auf die vollständige Enumeration aller Lösungsmöglichkeiten ("Brute Force") bauen, stoßen schon bei vergleichsweise kleinen Probleminstanzen an ihre Grenzen, da die Laufzeit im Allgemeinen exponentiell zur Instanzgröße steigt. Beispielsweise sind zwei Wochen Laufzeit für Instanzen, die dennoch für den Praxiseinsatz zu klein sind, keine Seltenheit. Die Laufzeit des Algorithmus ist zwar nicht vordergründig entscheidend, da kein Echtzeitrouting praktiziert werden soll, sondern das Netzwerkdesign im Vordergrund steht, muss aber praktikabel sein. Hierbei muss auch beachtet werden, dass das Erstellen eines Netzwerks ein iteratives Verfahren ist. Durch geänderte Spezifikation wird meistens mehrmaliges Berechnen des Netzwerkes notwendig. Demzufolge sind Lösungszeiten von wenige Minuten wünschenswert, bei großen Instanzen sind jedoch auch einige Tage akzeptabel.

Die gefundene Lösung muss nicht optimal sein, sollte jedoch auch nicht allzu weit von Optimum entfernt liegen. Dessen ungeachtet ist es wünschenswert auch effiziente exakte Verfahren zu entwickeln, um die Qualität eines heuristischen Lösungsverfahrens besser abschätzen zu können.

1.6 Ausblick

Zunächst wird in Kapitel 2 das Problem modelliert, dies schließt eine Formulierung als ILP in Abschnitt 2.1 mit ein. Anschließend wird in Kapitel 3 verwandte Literatur zum aufgeworfenen Problem aufgezeigt. Danach werden verschiedene algorithmische Ansätze zur Lösung des Problems vorgestellt. Dies ist einerseits die Lagrange Relaxierung in Kapitel 4 und andererseits die

Spaltengenerierung in Kapitel 5. In Kapitel 6 werden wichtige Aspekte der konkreten Implementierung der Algorithmen in C++ präsentiert. Darauffolgend werden in Kapitel 7 die verwendeten Testinstanzen beschrieben und die Ergebnisse der experimentellen Tests im Detail vorgestellt. Eine Zusammenfassung dieser Arbeit findet sich in Kapitel 8.

Kapitel 2

Modellierung des Problems

Formal kann dieses Problem auf einem ungerichteten Graphen $G = (V, E)$ beschrieben werden, wobei V die Knoten und E die Kanten (die Verbindungen) des Netzwerkes darstellen. Die Menge der Knoten enthält Quellen S, Senken T und Zwischenknoten M, wobei

$$V = S \ \cup \ T \ \cup \ M \, . \tag{2.0.1}$$

Aufbauend auf dem ungerichteten Graph G wird ein gerichteter Graph $G' = (V, A)$ definiert, der die gleiche Knotenmenge V enthält. Die Kantenmenge A enthält für jede ungerichtete Kante $(i, j) \in E$ zwei gerichtete Kanten: (i, j) und (j, i).

Jeder Kante $(i, j) \in E$ zwischen den Knoten i und j sind Kosten c_{ij} und ein Delay d_{ij} zugeordnet; die Kapazität einer Kante ist jeweils beschränkt durch eine Kapazität u_{ij}, wobei

$$c_{ij} \geq 0, \quad d_{ij} \geq 0, \quad u_{ij} \geq 0 \qquad \forall (i, j) \in E \, . \tag{2.0.2}$$

Kanten können sichere Übertragungen ermöglichen, das heißt sie bieten eine sichere Verbindung und ein sicheres Netzwerkprotokoll. Solche Kanten sind durch die Menge E^+ dargestellt. Dem gegenüber stehen unsichere Kanten E^-, die nur unsichere Übertragung erlauben. Über die Kanten E^* kann sowohl sicher als auch unsicher übertragen werden. Es gilt

$$E^+ \subset E, \quad E^- \subset E, \quad E^* \subset E, \quad E = E^+ \, \dot\cup \, E^- \, \dot\cup \, E^* \, , \tag{2.0.3}$$

wobei $\dot\cup$ die disjunkte Vereinigung symbolisiert.

Das Senden der Nachrichten wird durch m *Transporte* T dargestellt. Ein Transport mit Index $k \in \{1, \ldots, m\}$ beinhaltet $v_k \in \mathbb{R}$ Einheiten, die von einer bestimmten Quelle s_k zu einer bestimmten Senke t_k transportiert werden sollen. Ist eine Nachricht als *sicher* eingestuft, dann muss sie auf sicheren

Verbindungen mit sicheren Protokollen transportiert werden. Die Sicherheitseinstufung einer Nachricht wird mit

$$\sigma_k = \begin{cases} 1, & \text{sichere Nachricht} \\ 0, & \text{sonst} \end{cases} \tag{2.0.4}$$

beschrieben. Transporte können zeitkritisch sein, das bedeutet sie müssen mit einem maximalen Delay $\delta_k > 0$ bei der Senke t_k ankommen. Ist der Transport nicht zeitkritisch so ist $\delta_k = \infty$. Ein Transport k kann somit durch das Tupel $(s_k, t_k, v_k, \sigma_k, \delta_k)$ dargestellt werden.

Neben Kosten für die Übertragung fallen auch Protokollkosten für einen Transport k auf einer Kante (i, j) an. Diese werden mit

$$p_{ij}^k = \begin{cases} \min(p_{\text{sec}}, p_{\text{insec}}), & \text{wenn } \sigma_k = 0 \wedge (i,j) \in E^* \\ p_{\text{insec}}, & \text{wenn } (i,j) \in E^- \\ p_{\text{sec}}, & \text{sonst} \end{cases} \tag{2.0.5}$$

definiert, wobei p_{sec} die Kosten des sicheren Protokolls bezeichnet und p_{insec} die des unsicheren. Im Falle einer Verbindung die sichere und unsichere Verbindungen zulässt, und eines Transportes, der nicht sicher transportiert werden muss, wird das günstigere Protokoll verwendet. Neben den Kosten für ein Protokoll entsteht bei Verwendung eines Protokolls auch ein Delay, welcher mit

$$a_{ij}^k = \begin{cases} a_{\text{sec}}, & \text{wenn } p_{ij}^k = p_{\text{sec}} \\ a_{\text{insec}}, & \text{sonst} \end{cases} \tag{2.0.6}$$

dargestellt wird. Hierbei ist a_{sec} der Delay des sicheren und a_{insec} der des unsicheren Protokolls. Alle Nachrichten dürfen zusammen maximal eine bestimmte Verzögerung aufweisen, den maximalen Gesamtdelay d_{ges}. Der Delay aller Nachrichten berechnet sich aus der Summe der Delays aller Transporte.

Sei $A^k = \{(i,j) \in A \mid \sigma_k = 0 \vee (i,j) \notin E^-\}$ die Menge an Kanten die ein Transport k benutzen darf. Die Menge E^k sei analog für den ungerichteten Fall definiert. Ob eine Kante $(i, j) \in A$ für einen Transport k verwendet wird, stellt

$$f_{ij}^k = \begin{cases} 1, & \text{die Kante wird verwendet} \\ 0, & \text{sonst} \end{cases} \qquad \forall (i,j) \in A^k \tag{2.0.7}$$

dar. Wie schon beschrieben, dürfen dabei sichere Nachrichten nur über sichere Verbindungen transportiert werden. Ob eine Kante $(i, j) \in E$ von irgendeinem Transport verwendet wird, bestimmt

$$x_{ij} = \begin{cases} 1, & \text{die Kante wird von einem Transport verwendet} \\ 0, & \text{sonst.} \end{cases} \tag{2.0.8}$$

2.1 Ganzzahliges Lineares Programm

Eine Möglichkeit zur exakten Formulierung des Problems ist die ganzzahlige Lineare Programmierung. Durch Gleichungen werden mögliche Variablenbelegungen eingeschränkt. Die Optimierung erfolgt durch Maximieren beziehungsweise Minimieren der Zielfunktion.

2.1.1 Allgemeines

Eine Form der mathematischen Optimierung sind *Lineare Programme (LP)*, die in sehr vielfältiger Form in der Praxis auftreten. Eine Menge an Variablen ist durch eine Menge an mathematischen Gleichungen (Nebenbedingungen) verknüpft. Unter Beachtung dieser Nebenbedingungen muss eine diese Variablen enthaltende lineare Zielfunktion optimiert werden. Dies kann entweder ein Maximierungs- oder Minimierungsproblem sein. Die Linearität der Zielfunktion und der Nebenbedingungen gibt der Linearen Programmierung ihren Namen.

Die allgemeine Form für ein Lineares Programm ist

$$\begin{aligned} \min \quad & c^T x && \text{(2.1.1a)} \\ s.t. \quad & Ax < b && \text{(2.1.1b)} \\ & x \in \mathbb{R}^n && \text{(2.1.1c)} \end{aligned}$$

Die Zielfunktion (2.1.1a) soll unter den beiden Nebenbedingungen (2.1.1b) und (2.1.1c) minimiert werden. Wobei $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ und $A \in \mathbb{R}^{m \times n}$.

Ganzzahliges Lineares Programm. Wird bei einem Linearen Programm die Ganzzahligkeit von Variablen gefordert, so bezeichnet man es als ganzzahliges Lineares Programm, *Integer Linear Program (ILP)*.

Die allgemeine Form für ein ganzzahliges Lineares Programm ist durch

$$\begin{aligned} \min \quad & c^T x && \text{(2.1.2a)} \\ s.t. \quad & Ax < b && \text{(2.1.2b)} \\ & x \in \mathbb{Z}^n && \text{(2.1.2c)} \end{aligned}$$

gegeben. Im Unterschied zum vorigen LP (2.1.1) ist hier gefordert, dass x ganzzahlig sein muss. Die Forderung der Ganzzahligkeit von Variablen erschwert das Lösen des Problems erheblich. Das Lösen von ganzzahligen Linearen Programmen ist NP-schwierig.

2.2 Formulierung des Problems als Mehrgüterflussproblem

Das gegebene Problem als ganzzahliges Lineares Programm formuliert:

$$\min \quad \sum_{(i,j)\in E} \underbrace{c_{ij} \cdot x_{ij}}_{\text{Basiskosten}} + \sum_{k=1}^{m} \sum_{(i,j)\in E^k} \underbrace{f_{ij}^k \cdot p_{ij}^k + f_{ji}^k \cdot p_{ji}^k}_{\text{Protokollkosten}} \tag{2.2.1a}$$

$$\text{s.t.} \sum_{(i',i)\in A^k | i' \neq t_k} f_{i'i}^k - \sum_{(i,i'')\in A^k | i'' \neq s_k} f_{ii''}^k = 0 \qquad \forall i \in V \backslash \{s_k, t_k\},\ k = 1, \ldots, m \tag{2.2.1b}$$

$$\sum_{(s_k,j)\in A^k} f_{s_k j}^k = 1 \qquad \forall k = 1, \ldots, m \tag{2.2.1c}$$

$$\sum_{(i,t_k)\in A^k} f_{it_k}^k = 1 \qquad \forall k = 1, \ldots, m \tag{2.2.1d}$$

$$\sum_{k=1}^{m} f_{ij}^k \cdot \upsilon_k + f_{ji}^k \cdot \upsilon_k \leq u_{ij} \qquad \forall (i,j) \in E \tag{2.2.1e}$$

$$\sum_{(i,j)\in A^k} (d_{ij} + a_{ij}^k) \cdot f_{ij}^k \leq \delta_k \qquad \forall k = 1, \ldots, m \tag{2.2.1f}$$

$$\sum_{k=1}^{m} \sum_{(i,j)\in A} (d_{ij} + a_{ij}^k) \cdot f_{ij}^k \leq d_{\text{ges}} \tag{2.2.1g}$$

$$x_{ij} \geq f_{ij}^k + f_{ji}^k \qquad \forall k = 1, \ldots, m,\ (i,j) \in A^k \tag{2.2.1h}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{2.2.1i}$$

$$f_{ij}^k \in \{0,1\} \qquad \forall k = 1, \ldots, m,\ (i,j) \in A^k \tag{2.2.1j}$$

Die Flüsse der Nachrichten sollen so über das Netzwerk geleitet werden, dass die Kosten dabei minimiert werden (2.2.1a). Die Kosten entstehen einerseits durch Basiskosten, die einmal pro verwendeter Kante anfallen, andererseits treten Protokollkosten proportional zur Anzahl der darübergeleiteten Nachrichten auf. In jedem Knoten müssen die Flussbedingung aufrecht erhalten werden – Nachrichten die in einen Knoten einfließen müssen auch wieder

aus dem Knoten herausfließen (2.2.1b). Für jeden Transport muss ausgehend von der zugehörigen Quelle s_k eine Kante verwendet werden, damit die Nachricht aus der Quelle herausfließt (2.2.1c) und die Nachricht muss zur richtige Senke t_k gelangen (2.2.1d). Über eine Kante darf nur maximal eine bestimmte Anzahl an Daten fließen. Diese Kapazität u_{ij} jeder Kante wird durch die Ungleichungen (2.2.1e) berücksichtigt. Bei zeitkritischen Transporten muss der maximal erlaubte Delay δ_k einer Nachricht berücksichtigt werden. Die Ungleichungen (2.2.1f) gewährleistet, dass alle zeitkritischen Transporte rechtzeitig ankommen. Zudem darf der Gesamt-Maximaldelay, das heißt die Summe der Delays aller Transporte, nicht überschritten werden (2.2.1g). Um sicherzustellen, dass x_{ij} gesetzt ist, wenn irgendein Transport die Kante verwendet bzw. nicht gesetzt ist, wenn sie von keinem Transport verwendet wird, muss die Ungleichungen (2.2.1h) erfüllt sein. Da die Flüsse nicht gesplittet werden dürfen, sind die Flussvariablen (2.0.7) in Folge dessen $\in \{0, 1\}$. Daher handelt es sich um ein ganzzahliges Lineares Programm.

2.3 Algorithmische Ansätze

Nach der Modellierung des Problems werden im folgenden Kapitel 3 verwandte und vorangegangene Arbeiten diskutiert. Anschließend werden neben einer exakten Lösung, die wie beschrieben durch das ganzzahlige Lineare Programm berechenbar ist, weitere schnellere Ansätze vorgestellt. Diese sollen in kürzerer Zeit möglichst gute Schranken für das Problem finden. Vor allem für größere Instanzdaten sind solche Methoden geeignet. In dieser Arbeit wird eine ähnliche Herangehensweise wie in [GPSS08] verwendet. Das Ziel der Autoren ist einen minimalen Spannbaum zu generieren, wobei jeder Weg einen bestimmten maximalen Delay einhalten muss. Die Autoren wählen Lagrange Relaxierung und Spaltengenerierung, um Schranken für das Problem zu erhalten. Diese beiden Verfahren werden auch in dieser Arbeit eingesetzt. Einerseits kann man mittels der Lagrange Relaxierung eine untere Schranke für das Problem finden. Andererseits ermöglicht es die Spaltengenerierung relativ schnell eine optimale Lösung für das LP-relaxierte Problem zu finden, woraus einfach eine obere Schranke erzeugt werden kann. Für die Spaltengenerierung ist allerdings eine Startlösung notwendig. In den nachfolgenden Kapiteln werden diese Ansätze genauer erklärt. Zunächst wird die Lagrange Relaxierung in Kapitel 4 und danach die Spaltengenerierung in Kapitel 5 erörtert.

Kapitel 3

Verwandte und vorangegangene Arbeiten

Beim vorliegenden Problem handelt es sich um ein Mehrgüterflussproblem (*Multicommodity Flow Problem*) [AMOR95, HK95]. Mehrere Transporte werden über ein Netzwerk geleitet und teilen sich dabei Ressourcen – in diesem Fall Kapazitäten der Verbindungen (*Kanten*). Die Kosten sind dabei zu minimieren (*Minimum Cost Multicommodity Flow*).

Betrachtet man die auftretenden Kosten – diese setzen sich aus so genannten Basiskosten, welche pro Verwendung einer Kante einmal verrechnet werden und den Protokollkosten, welche für jede Benutzung anfallen zusammen – näher, so entspricht das Problem einem Netzwerkdesignproblem. Bei solch einem Problem fallen Kosten einerseits für das Verwenden – meist Erstellung oder Bau einer Verbindung (Kante) – und für jede Benutzung an. Diese Problemstellung mit einer Kostenfunktion, die sich aus fixen und variablen Kosten zusammensetzt, wird als *Network Design Problem* in der Literatur beschrieben. Stehen für die zu transportierenden Güter gemeinsam nur bestimmte Kantenkapazitäten zur Verfügung, so handelt es sich um ein "Multicommodity Capacitated Network Design Problem" [GCF98]. Dieses Problem ist NP-vollständig [KLHG07, JLK78]. Kleeman et al. beschreiben in [KLHG07] einen evolutionären Algorithmus, der ein Multicommodity Capacitated Network Design Problem mit mehreren Zielfunktionen optimiert. Das vorliegende Problem unterscheidet sich von den oben genannten Standardproblemen durch zusätzlichen Constraints: der maximale Gesamtdelay, zeitkritische und sicherheitskritische Transporte.

3.1 (Delay) Constrained Least Cost Problem

Beschränkungen hinsichtlich der Transportzeit werden beim *Delay Constrained Least Costs (DCLC)* Problem berücksichtigt, welches neben Kosten den Delay als zweite beschränkte Größe aufweist [JV06, RS00, JLX05]. Es ist das Problem den kostengünstigsten Weg in einem Netzwerk zu finden, während Delay-Beschränkungen beachtet werden müssen. Im Gegensatz zu dem in dieser Arbeit betrachteten Problem, wird beim DCLC-Problem nur ein Pfad gesucht. Es kann daher nur ein Transport über das Netzwerk transportiert werden. Das DCLC-Problem ist NP-vollständig [WC96, GJ90].

Das DCLC-Problem kann als Sonderfall des in dieser Arbeit betrachteten Problems mit nur einem Transport und keiner Unterscheidung zwischen sicheren und unsicheren Verbindungen gesehen werden. Eine etwas allgemeinere Darstellung, nicht nur über das Delay Constraint Least Cost Problem, sondern allgemein über (Multi-)Constrained Path Selection bietet [KMKK02]. Generell ist das Einsatzgebiet solcher Probleme das Quality of Service Routing. Hierbei ist die Aufgabe, Daten über ein Netzwerk zu leiten, wobei der gefundene Pfad gewisse Qualitätskriterien einhalten muss. Solche Kriterien können zum Beispiel Geschwindigkeit der Nachricht, die Sicherheit der verwendeten Protokolle und Leitungen, die Ausfallssicherheit der Route sein.

Weitere Bezeichnungen für das beschriebene Constrained Shortest Path Problem und die etwas allgemeinere Formulierung mit mehreren Beschränkungen sind unter anderem: *Resource Constrained Shortest Paths* [MZ00], *Weight-Constrained Shortest Path Problem* [DB03] und *Restricted Shortest Path Problem* [KMKK02].

In [Kli06] gibt Klincewicz unter anderem einen Überblick über Publikationen, die Delay als Qualitätskriterium für das Routen beschreiben. Mögliche Problemlösungen stellen das Einbeziehen des Delays in die Kostenfunktion: *"Various authors have taken delay into account by incorporating a cost of delay into the objective function"* [Kli06] oder das Relaxieren durch Lagrange Relaxierung dar.

3.2 Flow Routing with Multi-Hop and Real-Time Constraints

Ausgehend von der Problemstellung, kontinuierliche Messergebnisse über ein Netzwerk zu routen, beschäftigen sich Trdlicka, Hanzalek und Johansson in [THJ07] mit einer ähnlichen Aufgabe wie in der vorliegenden Arbeit. In einem Netzwerk, dem ein gerichteter Graph zugrunde liegt, sind Leitungs-

kapazitäten und Knotenkapazitäten zu beachten. Darüber hinaus muss jeder Datenfluss einen maximalen Delay einhalten. In der angesprochenen Publikation wird dies mit einer bestimmten Anzahl an Hops gleichgesetzt. In einer Erweiterung (auf welche die Arbeit hinweist) kann auch jede Verbindung je nach Transport unterschiedliche Delays verursachen. Umgesetzt wird dieser "real-time constraint" durch Layer, die pro Einheit eingezogen werden. Dadurch steigt die Anzahl dieser Layer proportional mit der Größe des maximal erlaubten Delays. Diese Problemstellung erlaubt darüber hinaus mehrere Quellen und Senken für einen Transport(multi-source, multi-sink). Ein weiterer Unterschied dieser Formulierung zu dem in dieser Arbeit behandelten Problem ist das Splitten von Nachrichten. Es ist erlaubt, einen Transport über mehrere Routen zum Ziel (bzw. zu den Zielen) zu routen (multi-path). In Anlehnung daran schreiben die Autoren, wenn die Daten nicht geteilt werden können, ist das Problem NP-schwer.

> *"The case that the data flow cannot be fragmented (e.g. the data transferred in packet, which cannot be fragmented), the multicommodity network flow problem is NP-hard even for two communication demands."*[THJ07]

Das in der vorliegenden Arbeit besprochene Problem ist ein Mehrgüterflussproblem und hat zusätzlich die Schwierigkeit, dass Transporte nicht geteilt werden dürfen. In diesem Fall ist das Problem selbst bei zwei Transporten schon NP-schwer. Demnach ist das in dieser Arbeit behandelte Problem NP-schwer.

3.3 Rooted Distance-Constrained Minimum Spanning Tree Problem

Das Problem des Findens eines Spannbaumes mit minimalen Kosten, unter der Berücksichtigung zusätzlicher Delaybeschränkungen, wird als *Rooted Distance-Constrained Minimum Spanning Tree Problem (RDMSTP)* bezeichnet. Auf dem zugrunde liegenden Graphen sind neben Kosten auch Delays für jede Kante angeben. Die Einschränkung ist ein maximales Zeitlimit für die Pfade von der Wurzel zu jedem einzelnen Blatt.

In [GPSS08] beschreiben Gouveia et al. dieses Problem und geben Lösungswege an. Das Problem ist NP-schwer, da schon der Sonderfall, das Hop-Constrained Minimum Spanning Tree Problem, NP-schwer ist. Bei diesem Problem ist der Delay jeder Kante gleich eins. Das RDMSTP ist insofern dem in dieser Arbeit besprochenen Problem ähnlich, als ebenfalls beschränkte

kostengünstige Pfade zu suchen sind. Der Unterschied liegt darin, dass beim RDMSTP alle Pfade von einem Punkt ausgehen, um einen Spanning Tree zu erhalten.

3.4 Projekt SWIS

Bei dem schon in Kapitel 1.1 erwähnten Projekt SWIS – System-Wide Information Sharing Network, welches der Ausgangspunkt dieser Diplomarbeit bildet, stand der in dieser Arbeit fokussierte Optimierungsteil nicht im Mittelpunkt. In diesem Bereich kamen zuerst Enumerationsverfahren zum Einsatz. Mittels eines Brute Force-Ansatzes wurde durch Ausprobieren aller Möglichkeiten versucht die optimale Lösung zu finden. Die Laufzeit eines derartigen Ansatzes ist exponentiell bezüglich der Größe der Eingabeinstanzen. Da diese Herangehensweise selbst bei sehr kleinen Instanzen sehr lange Lösungszeiten nach sich zieht, beziehungsweise diese gar nicht lösbar sind, wurde in weiterer Folge ein heuristischer Ansatz für das Problem implementiert.

Der evolutionärer Algorithmus **P**rototype **O**ptimisation with **E**volved I**M**provement **S**teps (POEMS) von Jiri Kubalik und Jan Faigl [KF06] wurde an das Problem angepasst. Solch eine Metaheuristik kann auch sehr gute Ergebnisse generieren, es kann jedoch keine Aussage über die Lösungsgüte getroffen werden. Leider ist ein unmittelbarer Vergleich mit diesen Ergebnissen nicht möglich, da die Problemstellung leicht geändert wurde. In der Umsetzung für das SWIS Projekt war es möglich Kanten wenn nötig doppelt zu verwenden. Kapazitätsgrenzen konnten wenn nötig mehrfach überschritten werden. Außerdem dürfte der Gesamtdelay anders berechnet worden sein. Dieser wurde überdies nicht als Beschränkung sondern als zweite Zielfunktion aufgefasst. Vergleiche sind dennoch geplant, Jiri Kubalik wird eine Anpassung seines Algorithmus vornehmen.

Der größte uns zur Verfügung gestellte und auch im Einsatz gewesenen Testfall hat 30 Knoten, 63 Verbindungen und 13 Transporte. Die Projektteilnehmer gaben an, dass die Optimierung ihrer Testinstanzen sehr lange Laufzeiten nach sich zog. In diesem Sinne kann in diesem Bereich noch ein großes Verbesserungspotential festgestellt werden – einerseits in Hinsicht auf die Laufzeit, andererseits in Bezug auf die Lösungsgüte, welche mit den im Projekt verwendeten Methoden nicht angegeben werden kann. Diese Diplomarbeit soll vor allem in diesen Aspekten Verbesserungen bringen.

Kapitel 4

Lagrange Relaxierung

Der Lösungsansatz mittels ganzzahligem Linearen Programm stößt vor allem bei größeren Instanzen schnell an seine Grenzen. Da die Laufzeit ab bestimmten Problemengrößen nicht mehr tragbar ist, kann ein alternativer Ansatz weiterhelfen.

Eine Möglichkeit ist, das Problem zu vereinfachen – zu relaxieren – und dann für dieses Problem Lösungen zu suchen. Durch die Vereinfachung ist die Lösungsmenge größer. Infolge dessen ergibt sich eine Lösung, die besser ist als die Optimallösung für das Ausgangsproblem (oder dieser entspricht). Bei Minimierungsproblemen bedeutet besser eine untere Schranke für das ursprüngliche Problem. Ziel ist es eine möglichst gute, das heißt große, untere Schranke für das Problem zu finden. Diese unteren Schranken sind meistens keine gültigen Lösungen für das anfängliche Problem, können die Optimallösung allerdings eingrenzen. Außerdem ermöglicht oft eine einfache Anpassung dieser Resultate gute heuristische Lösungen.

Zwei sehr häufig verwendete Relaxierungsmethoden sind LP-Relaxierung und Lagrange Relaxierung. Bei der LP-Relaxierung wird die Ganzzahligkeitsbedingung für ein (gemischt-) ganzzahliges Lineares Programm fallengelassen. Dieses relaxierte Problem ist dann einfacher zu lösen. Ist das Problem so groß, dass schon das Lösen der LP-Relaxierung zu lange dauert, ist die Lagrange Relaxierung oftmals eine gute Alternative. Praktische Erfahrungen haben gezeigt, dass die Lagrange Relaxierung sehr gute untere Schranken in vertretbarer Zeit produziert [Bea93].

4.1 Allgemeines

Wie schon beschrieben ist die Grundidee der Relaxierung die Vereinfachung des Problems, um in einem Minimierungsproblem eine möglichst gute untere

Schranke (andernfalls eine möglichst gute obere Schranke) für das Problem zu erhalten. Das Prinzip der Lagrange Relaxierung ist es, Bedingungen wegzulassen und stattdessen Bestrafungsfaktoren in die Zielfunktion aufzunehmen. Eine Einführung zur Lagrange Relaxierung gibt Beasley [Bea93].

Beasley beschreibt das allgemeine Prinzip der Lagrange Relaxierung ausgehend von folgendem Linearen Programm:

$$\begin{array}{lll} \min & cx & A, B \in \mathbb{R}^{n \times m} \\ \text{s.t.} & Ax \geq b & c, d, b \in \mathbb{R}^{n,m} \\ & Bx \geq d & \\ & x \in (0,1) & \end{array} \tag{4.1.1}$$

Es sollen im Folgenden die Ungleichungen $Ax \geq b$ relaxiert werden. Zuerst wird der Term $\lambda(b - Ax)$ zur Zielfunktion hinzugefügt. Es ist klar, dass durch dieses Hinzufügen der Zielfunktionswert verkleinert wird, da der Term nichtnegativ ist ($\lambda \geq 0$ und $(b - Ax) \leq 0$). Hierbei wird der Bestrafungsvektor λ als Lagrange Multiplikator bezeichnet. Es ergibt sich somit folgendes LP:

$$\begin{array}{ll} \min & cx + \lambda(b - Ax) \\ \text{s.t.} & Ax \geq b \\ & Bx \geq d \\ & x \in (0,1) \end{array} \tag{4.1.2}$$

Weiters kann das Weglassen von Bedingungen in einem Minimierungsproblem den Zielfunktionswert nur verkleinern. Daher ist die folgende reduzierte Form (4.1.3) sicher eine untere Schranke für das ursprüngliche Problem:

$$\begin{array}{ll} \min & cx + \lambda(b - Ax) \\ \text{s.t.} & Bx \geq d \\ & x \in (0,1) \end{array} \tag{4.1.3}$$

Das resultierende Programm (4.1.4) wird als *Lagrangean lower bound program (LLBP)* bezeichnet, da es für alle $\lambda \geq 0$ eine untere Schranke bietet. Eine weitere Umformung von 4.1.3 ergibt:

$$\begin{array}{ll} \min & (c - \lambda A)x + \lambda b \\ \text{s.t.} & Bx \geq d \\ & x \in (0,1) \end{array} \tag{4.1.4}$$

Wir suchen jedoch die beste untere Schranke, daher ist das Ziel Lagrange Multiplikatoren zu finden, welche eine maximale untere Schranke ergeben.

$$\max_{\lambda \geq 0} \left\{ \begin{array}{ll} \max & cx + \lambda(b - Ax) \\ \text{s.t.} & Bx \geq d \\ & x \in (0,1) \end{array} \right\} \tag{4.1.5}$$

Dies wird als *Lagrange-duales Programm* bezeichnet. Ein Standardverfahren, um diese Maximierung zu erreichen, ist das *Subgradientenverfahren*. Es ist ein iteratives Verfahren, welches die Lagrange-Multiplikatoren fortlaufend anpasst und sich so der maximalen unteren Schranke immer mehr nähert.

Die Lösungen die man beim relaxierten Programm erhält sind untere Schranken und stellen daher nicht notwendigerweise gültige Lösungen für das ursprüngliche Problem dar. Um eine solche aus einer unteren Schranke zu erhalten, muss die Lösung im Allgemeinen repariert werden. Dafür wird eine *Lagrange-Heuristik* verwendet. Derartige erzeugte Lösungen sind oftmals gute heuristische Lösungen für das ursprüngliche Problem.

Eine Lösung ist optimal, wenn die Lösung für das Ausgangsproblem gültig ist und der Zielfunktionswert gleich jenem des LLBP ist. Bei dem Linearen Programm (4.1.1) wäre eine optimale Lösung gefunden, wenn die Lösung $\bar{x}$ gültig ist und

$$c\bar{x} = c\bar{x} + \lambda(b - A\bar{x}) \tag{4.1.6}$$

erfüllt. Anders ausgedrückt: die Lösung ist dann optimal, wenn sie gültig ist und $\lambda \cdot (b - A\bar{x}) = 0$ gilt.

In diesem Beispiel haben wir die erste Bedingung relaxiert, doch wie soll in einem konkreten Beispiel die zu relaxierende Bedingung gewählt werden? Dazu muss man sich den Grund für das Relaxieren in Erinnerung rufen. Wir relaxieren, um ein leichter lösbares Problem zu erhalten. Deshalb sollten jene Bedingungen weggelassen werden, die das Problem schwer lösbar machen. Vor allem in Hinblick auf iterative Verfahren, die dieses Problem sehr oft lösen müssen, sollte eine schnelle Möglichkeit zur Lösung bestehen. Kann ein Problem durch Relaxierung als mehrere unabhängige Teilprobleme betrachtet werden, so wird dies als *Lagrange Dekomposition* bezeichnet. Die einzelnen Subprobleme werden gelöst, um untere Schranken für das Ausgangsproblem zu erhalten.

4.1.1 Subgradientenverfahren

Das Subgradientenverfahren ist ein iteratives Verfahren, welches ausgehend von gesetzten Lagrange Multiplikatoren systematisch neue generiert. Einen Überblick über das Subgradientenverfahren wird in Algorithmus 4.1.1 gegeben. Neben Start-Multiplikatoren $\lambda_i \geq 0$ werden π, $0 < \pi \leq 2$, und eine obere Schranke Z_{UB} für das Verfahren benötigt. Folgend sind die einzelnen Schritte des Verfahrens näher erklärt:

Algorithmus 4.1.1 Überblick Subgradientenverfahren

1: Setze obere Schranke Z_{UB}
2: Setze untere Schranke Z_{LB}
3: Initialisiere λ_i
4: $\pi = 2$
5: **repeat**
6: Löse Subprobleme
7: Berechne die Gradienten G_i
8: Berechne die Schrittgröße

$$T = \frac{\pi \cdot (Z_{UB} - Z_{LB})}{\sum_i G_i^2}$$

9: Aktualisiere die Lagrange-Multiplikatoren

$$\lambda_i = \max(0, \lambda_i + T \cdot G_i)$$

10: **until** Abbruchbedingung erfüllt

1. **Lösen des LLBP**
   Zunächst muss eine untere Schranke Z_{LB} mit den Werten X_j durch Einsetzen der Lagrange Multiplikatoren λ_i berechnet werden. Danach werden die Schritte 2-4 iterativ ausgeführt.

2. **Ermitteln der Subgradienten** G_i
   Aus der berechneten Lösung werden die Subgradienten

$$G_i = b_i - \sum_{j=1}^{n} a_{ij} \cdot X_j \quad i = 1, \ldots, m \tag{4.1.7}$$

   berechnet.

3. **Bestimmen der skalaren Schrittgröße** T
   Durch den Subgradienten und die obere und untere Schranke wird die Schrittgröße

$$T = \frac{\pi \cdot (Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} G_i^2} \tag{4.1.8}$$

   definiert.

4. **Aktualisieren der Lagrange Multiplikatoren** λ_i
   Als letzter Schritt werden die Lagrange Multiplikatoren

$$\lambda_i = \max(0, \lambda_i + T \cdot G_i) \quad i = 1, \ldots, m \tag{4.1.9}$$

durch die berechneten Werte aktualisiert.

Wann soll nun dieser Vorgang gestoppt werden? Hierfür gibt es zwei Möglichkeiten. Entweder wird die Anzahl der Iterationen begrenzt oder die Abbruchbedingung wird über den Wert von π bestimmt, dessen Wert immer wieder reduziert wird wenn sich die Schranke eine definierte Anzahl an Durchläufen nicht mehr verbessert hat.

4.1.2 Volume Algorithmus

Eine Alternative zum Subgradientenverfahren ist der Volume Algorithmus von Barahona und Anbil [BA00]. Auch der Volume Algorithmus ist ein iteratives Verfahren, welches ausgehend von ersten Lagrange Multiplikatoren systematisch neue generiert. Der Volume Algorithmus findet oft schneller bessere Resultate als das Subgradientenverfahren [HS06].

In Algorithmus 4.1.2 wird ein Überblick über die Funktionsweise des Volume Algorithmus gegeben. Als wesentlicher Unterschied zum Subgradientenverfahren werden beim Volume Algorithmus durch x^0 die Ergebnisse vergangener Iterationen auch im aktuellen Durchlauf herangezogen. Im Folgenden wird die Berechnung von α und π genauer betrachtet. Der Wert für α kann für einige Iterationen auf einen festen Wert gesetzt und dann verkleinert werden. Barahona und Anbil schlagen in [BA00] darüber hinaus eine andere Möglichkeit vor. Sei α_{opt} definiert als

$$\alpha_{opt} = \min_{\alpha} \|\alpha \cdot G_i^t + (1-\alpha) \cdot G_i^0\| \,. \tag{4.1.10}$$

Ist $\alpha_{opt} < 0$ so wird $\alpha = \alpha_{max}/10$ gesetzt, sonst $\alpha = \min\{\alpha_{opt}, \alpha_{max}\}$. Als guten Startwert geben die Autoren $\alpha_{max} = 0{,}1$ an, der dann gegen Ende hin verkleinert wird. Jedem Durchlauf wird ein Iterationstyp zugeordnet: Rot, Gelb oder Grün.

- ***Rot***: die aktuelle Iteration hat keine Verbesserung gebracht ($Z_{LB}^t \leq L^0$)
- ***Gelb***: die aktuelle Iteration hat eine Verbesserung gebracht ($Z_{LB}^t > Z_{LB}^0$) und $d < 0$, wobei
$$d = G_i^t(b - Ax^t) \,. \tag{4.1.11}$$
- ***Grün***: sonst

Bei jeder grünen Iteration wird f mit 1,1 multipliziert, nach einer Sequenz von 20 aufeinander folgenden roten Iterationen wird f mit 0,66 multipliziert. Es wird vorgeschlagen dann die Iterationen zu stoppen, wenn $\|G_i\|$ und $\|c \cdot x^0 - Z_{LB}^0\|$ unter einem bestimmten Wert sind.

Algorithmus 4.1.2 Überblick Volume Algorithmus

1: Setze obere Schranke Z_{UB}
2: Setze untere Schranke Z^0_{LB}
3: Setze Zählvariable $t = 0$
4: Initialisiere λ^0_i
5: Löse das LLBP mit λ^0_i, erhalte eine Lösung x^0 mit dem Wert Z^0_{LB}
6: **repeat**
7: Berechne die Gradienten G_i mit x^0
8: Berechne die Schrittgröße

$$T = \frac{\pi \cdot (Z_{UB} - Z^0_{LB})}{\sum_i G_i^2}$$

9: Aktualisiere die Lagrange-Multiplikatoren

$$\lambda^t_i = \max(0, \lambda^0_i + T \cdot G_i)$$

10: Löse das LLBP mit λ^t_i, erhalte die Lösung x^t mit dem Wert Z^t_{LB}
11: Aktualisiere

$$x^0 = \alpha \cdot x^t + (1 - \alpha) \cdot x^0$$

12: **if** bessere untere Schranke gefunden **then**
13: Aktualisiere λ^0_i und Z^0_{LB}
14: **end if**
15: $t = t + 1$
16: **until** Abbruchbedingung erfüllt

4.2 Anwendung auf das Problem

Im Folgenden soll nun die besprochene Lagrange Relaxierung auf unser Transportproblem angewendet werden. Zunächst muss hierfür entschieden werden, welche Bedingungen relaxiert werden sollen.

Da einzelne nicht miteinander gekoppelte Transporte einfacher über ein Netzwerk geroutet werden können, liegt es nahe diese Beschränkungen zu relaxieren. Somit stellt sich das Problem nicht mehr als Mehrgüterproblem dar, sondern als viele einzelne Gütertransporte auf dem gleichen Netzwerk. Die Kapazitätsbeschränkungen werden in die Zielfunktion verschoben.

Für die Lagrange Relaxierung wird das Problem ungerichtet betrachtet. Es wird

$$b_{ij}^k = f_{ij}^k + f_{ji}^k \tag{4.2.1}$$

eingeführt, um auszudrücken, dass eine Kante (i, j) von einem Transport k verwendet wird. Da die Protokollkosten symmetrisch sind, wird aus dem Linearen Programm (2.2.1) durch Relaxierung:

$$\min \sum_{(i,j)\in E} \left(c_{ij} \cdot x_{ij} \; + \sum_{k=1}^{m} b_{ij}^k \cdot p_{ij}^k \right) \tag{4.2.2a}$$

$$+ \sum_{(i,j)\in E} \lambda_{ij} \cdot \left(\sum_{k=1}^{m} \left(b_{ij}^k \cdot \upsilon_k \right) - u_{ij} \right) \tag{4.2.2b}$$

$$+ \lambda' \left(\sum_{k=1}^{m} \sum_{(i,j)\in E^k} \left((d_{ij} + a_{ij}^k) \cdot b_{ij}^k \right) - d_{\text{ges}} \right) \tag{4.2.2c}$$

$$+ \sum_{k=1}^{m} \sum_{(i,j)\in E} \lambda''_{k,ij} \left(b_{ij}^k - x_{ij} \right) \tag{4.2.2d}$$

$$\text{s.t.} \quad b_{ij}^k \in C^k \tag{4.2.2e}$$

$$\sum_{(i,j)\in E^k} (d_{ij} + a_{ij}^k) \cdot b_{ij}^k \leq \min(\delta_k, d_{\text{ges}}) \qquad \forall k = 1, \ldots, m \tag{4.2.2f}$$

$$b_{ij}^k \cdot \upsilon_k \leq u_{ij} \qquad \forall k = 1, \ldots, m, \; (i, j) \in E^k \tag{4.2.2g}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E \tag{4.2.2h}$$

$$b_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in E \tag{4.2.2i}$$

Neben den Kosten (4.2.2a) (Basis- und Protokollkosten) werden die Kapazitätsbeschränkungen (4.2.2b), der Gesamtdelay (4.2.2c) und die Kopplungsbedingungen zwischen den Entscheidungsvariablen x und den Flussvariablen b_{ij}^k (4.2.2d), in die Zielfunktion aufgenommen. Dadurch erfolgt eine Entkoppelung der einzelnen Transporte (Lagrange Dekomposition). Diese relaxierten Bedingungen können jedoch als abgeschwächte Bedingungen erhalten bleiben, um eine bessere Schranke zu erreichen. Einerseits können schwächere Kapazitätsbeschränkungen der Kanten eingeführt werden. Die maximale Kapazität darf, wie in der ursprünglichen, auch in der relaxierten Formulierung von allen Transporten zusammen nicht überschritten werden. Genauso darf kein einzelner Transport gegen diese Beschränkung verstoßen (4.2.2g). Andererseits gilt das gleiche für die Gesamtdelaybeschränkung; der Gesamtdelay darf von jedem einzelnen Transport maximal so groß wie d_{ges} beziehungsweise bei zeitkritischen Transporten außerdem wie der jeweilige maximale Delay (4.2.2f). In der Beschränkung (4.2.2e) stellt C^k die Menge aller gültigen Pfade von der Quelle s_k zur Senke t_k dar. Aus (4.2.2) geht hervor, dass ein valider Pfad für den jeden Transport gesucht ist unter Einhaltung der Delay- und Kapazitätsbeschränkungen. Durch Umformung der Zielfunktion lassen sich die auftretenden Kosten einer Kante ablesen:

$$\min \quad -\lambda' \cdot d_{\text{ges}} + \sum_{(i,j)\in E} \Big((c_{ij} \cdot x_{ij} - \lambda_{ij} \cdot u_{ij}) \tag{4.2.3}$$

$$+ \sum_{k=1}^{m} \left(b_{ij}^k \cdot \left(p_{ij}^k + \lambda_{ij} \cdot \upsilon_k + \lambda' \cdot (d_{ij} + a_{ij}^k) + \lambda''_{k,ij} \right) - \lambda''_{k,ij} \cdot x_{ij} \right) \Big) . \tag{4.2.4}$$

Für jede durch Transport k verwendete Kante (i,j) (in diesem Fall gilt $b_{ij}^k = 1$) fallen demnach Kosten in der Höhe von

$$\hat{c}_{ij}^k = p_{ij}^k + \lambda_{ij} \cdot \upsilon_k + \lambda'(d_{ij} + a_{ij}^k) + \lambda''_{k,ij} . \tag{4.2.5}$$

an. Die Werte von x_{ij} können beliebig gewählt werden. Da es sich um ein Minimierungsproblem handelt wird versucht die Terme mit x_{ij} in der Zielfunktion möglichst klein zu halten. Der Faktor von x_{ij} in der Minimierungsformel ist $c_{ij} - \lambda''_{k,ij}$. Die Kosten sind demnach dann möglichst klein, wenn

$$x_{ij} = \begin{cases} 0, & \text{wenn } c_{ij} - \sum_{k=1}^{m} \lambda''_{k,ij} > 0, \\ 1, & \text{sonst.} \end{cases} \tag{4.2.6}$$

gesetzt wird.

4.2.1 Subproblem: Constrained Shortest Path (CSP)

Das Problem ist somit durch Lagrange Relaxierung in m Shortest Path Probleme zerfallen. Die Nachrichten müssen auf dem ursprünglichen Netzwerk mit veränderten Kosten unabhängig voneinander transportiert werden. Bei Transport k treten für die Kante (i, j) die zuvor besprochenen Kosten $\hat{c}_{ij}^k$ auf.

Die Schwierigkeit bei diesem Shortest Path Problem – im Gegensatz zu einem herkömmlichen Shortest Path Problem – sind die zusätzlichen Beschränkungen (siehe Gleichungen 4.2.2f , 4.2.2g) hinsichtlich Delay und Kapazitäten. Das Einhalten der Kapazitätsbeschränkungen kann durch ein einfaches Preprocessing garantiert werden. Durch das Löschen aller nichtverwendbaren Kanten aus der betrachteten Kantenmenge wird diese Beschränkung berücksichtigt. Dies sind jene Kanten, die kleinere Kapazitäten haben als die vom jeweiligen Transport benötigten. Nach diesem Preprocessing muss für jede verbleibende Kante (i, j) für den aktuell betrachteten Transport k gelten

$$u_{ij} \leq \upsilon_k \, . \tag{4.2.7}$$

Improved Preprocessing

Dumitrescu und Boland beschreiben in [DB03] eine Preprocessing-Methode und einen anschließenden Labeling Algorithmus für dieses Problem. Das Prinzip ihres Preprocessing-Algorithmus ist, kürzeste Wege hinsichtlich Kosten und Delay (beziehungsweise anderer Restriktionen) getrennt von der Quelle und Senke aus zu berechnen. Auf Grund dieser Ergebnisse werden danach Knoten und/oder Kanten, die nicht in Frage kommen, gelöscht. In einem iterativen Vorgang wird so das Netzwerk kleiner und das Problem dadurch leichter lösbar. Wird ein kostengünstigster Pfad gefunden, der die Delaybeschränkungen erfüllt, so ist das Problem gelöst. Anderenfalls wird versucht das Netzwerk so weit wie möglich zu vereinfachen. Ist ein weiteres Vereinfachen nicht mehr möglich (das bedeutet es trat im letzten Durchgang keine Veränderung mehr auf) ist das Preprocessing beendet. Darauf aufbauend kommt ein Labeling-Algorithmus zum Einsatz, welcher einen kürzesten Pfad mit den Delaybeschränkungen im Netzwerk findet.

Wie schon angesprochen kann durch vorheriges Löschen der Kanten mit zu kleinen Kapazitäten dieser Algorithmus auf das vorliegende, aus der Lagrange Relaxierung gewonnene, Teilproblem angewandt werden. Im Anhang A ist der Pseudocode des Algorithmus verfügbar. Hier wird ein Überblick über die Funktionsweise gegeben werden (Algorithmus 4.2.1). In diesem Algorithmus gibt *delay*(Pfad) den Delay des gesamten Pfades an.

Algorithmus 4.2.1 Preprocessing Algorithmus – Überblick

1: Sei s die Quelle, t die Senke und $d_{\max}$ der maximal erlaubte Delay des gesuchten Pfades
2: **repeat**
3: Sei C_{si} der kostengünstigste Pfad von der Quelle s zum Knoten i
4: **if** existiert kein Weg C_{st} **then**
5: **return** kein Weg gefunden
6: **end if**
7: **if** delay$(C_{st}) < d_{\max}$ **then**
8: **return** Weg gefunden
9: **end if**
10: Sei C_{it} der kostengünstigste Pfad von der Senke t zu Knoten i
11: Sei D_{si} der hinsichtlich Delay kürzeste Pfad von der Quelle s zu Knoten i
12: **if** existiert kein Weg mit delay$(D_{st}) < d_{\max}$ **then**
13: **return** kein Weg gefunden
14: **end if**
15: Sei D_{it} der hinsichtlich Delay kürzeste Pfad von der Senke t zu Knoten i
16: **for all** $i \in V$ **do**
17: **if** delay(D_{si}) + delay$(D_{it}) > d_{\max}$ **then**
18: Lösche Knoten i und inzidente Kanten
19: **end if**
20: **end for**
21: **for all** $(i,j) \in E$ **do**
22: **if** delay$(D_{si}) + d_{ij} + D_{jt} > d_{\max}$ **then**
23: Lösche Kante (i,j)
24: **end if**
25: **end for**
26: **until** sich nichts ändert

Tabelle 4.1: Netzwerkdaten für das Fallbeispiel

Kante	Start	Ziel	Kapazitäten	Kosten	Delay	Sicherheit
0	0	1	14	6	4	un-/sicher
1	0	4	8	3	2	sicher
2	0	7	15	4	4	sicher
3	1	2	12	1	3	sicher
4	1	5	3	2	1	sicher
5	1	4	9	2	1	sicher
6	4	5	11	1	9	sicher
7	4	8	17	12	9	unsicher
8	4	7	1	2	3	unsicher
9	7	8	13	1	3	sicher
10	2	3	10	3	3	sicher
11	2	5	8	2	1	sicher
12	5	3	9	2	1	sicher
13	5	9	9	5	2	unsicher
14	3	6	11	3	1	sicher
15	9	6	6	1	2	sicher

Tabelle 4.2: Transportdaten für das Fallbeispiel

Transport	Start	Ziel	Kapazität	Delay	Sicherheit
0	0	6	5	8	sicher

Beispiel Preprocessing. Zur Veranschaulichung der Funktionsweise des Algorithmus soll nun ein kleines Beispiel dienen. Ausgegangen wird von dem Subproblem, dass ein Pfad für einen Transport gefunden werden soll. Eine genaue Auflistung der in diesem Beispiel verwendeten Werte befindet sich in den Tabellen 4.1 und 4.2.

In unserem Beispiel soll ein Transport sicher von der Quelle im Knoten 0 zur Senke im Knoten 6 (markierte Knoten im Netzwerk) transportiert werden. Der Transport hat eine Kapazität von 5 und es ist ein maximaler Delay von 8 erlaubt. Abbildung 4.2.1 zeigt den Ausgangsgraphen für das Beispiel. Die Kantenbeschriftungen zeigen die Kapazität der Kanten an. Die grauen Linien zeigen an, dass die Kapazität der Kante nicht für den Transport ausreicht. Gestrichelte Kanten sind unsichere Verbindungen.

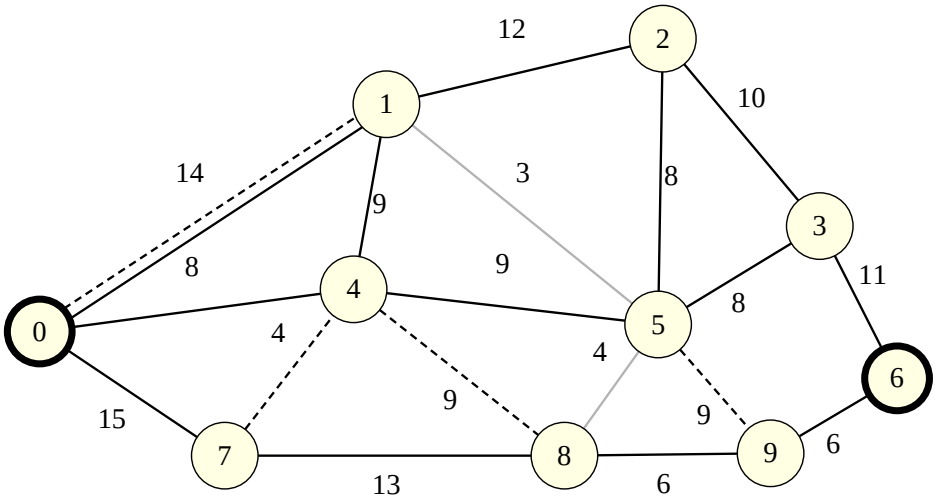

Abbildung 4.2.1: Ausgangsgraph (mit Kantenkapazitäten)

Soll nun – wie hier im Beispiel – ein Transport sicher über das Netzwerk transportiert werden, so sind unsichere Verbindungen nicht relevant. Das Preprocessing eliminiert zudem Kanten, die ungenügende Kapazitäten aufweisen. Als nächstes müssen die kostengünstigsten Pfade ausgehend vom Startknoten berechnet werden (Abbildung 4.2.2). Gerichtete Kanten zeigen den kürzesten Pfad zum jeweiligen Knoten an. Die Kantenbeschriftungen zeigen die Kosten für die jeweilige Kante an und die Knotenbeschriftungen entsprechen den Kosten des kürzesten Pfades zu diesem Knoten. Der kostengünstigste Pfad ist demnach von Knoten 0 über die Knoten 7, 8 und 9 zu Knoten 6. Dieser Pfad ist zur Verdeutlichung stärker hervorgehoben. Gepunktete Linien zeigen Kanten an, welche nicht für die kürzesten Pfade verwendet wurden.

Wird kein Pfad von der Quelle zur Senke gefunden, so ist entweder die Unlösbarkeit des Problems gegeben oder es wurde schon in einem früheren Durchlauf ein gültiger Pfad gefunden. Würde dieser kürzeste Pfad einen Delay kleiner dem maximalen Delay aufweisen, so wäre das Problem gelöst und der kostengünstigste delay-beschränkte Pfad gefunden. Dies ist im Beispiel nicht der Fall. Daher werden in Folge die kürzesten Pfade hinsichtlich Delay von der Quelle aus berechnet. (Abbildung 4.2.3) Die gerichteten Kanten zeigen, wie schon zuvor bei den kürzesten Pfaden hinsichtlich Kosten, den kürzesten Weg an. Die Kantenbeschriftungen entsprechen dem Delay einer Kante. Knotenbeschriftungen geben den bestmöglichen Delay eines Pfades von der Quelle zu dem jeweiligen Knoten an. Aus diesen Berechnungen kann wiederum abgelesen werden, wenn es keinen Pfad gibt, der den Delaybedin-

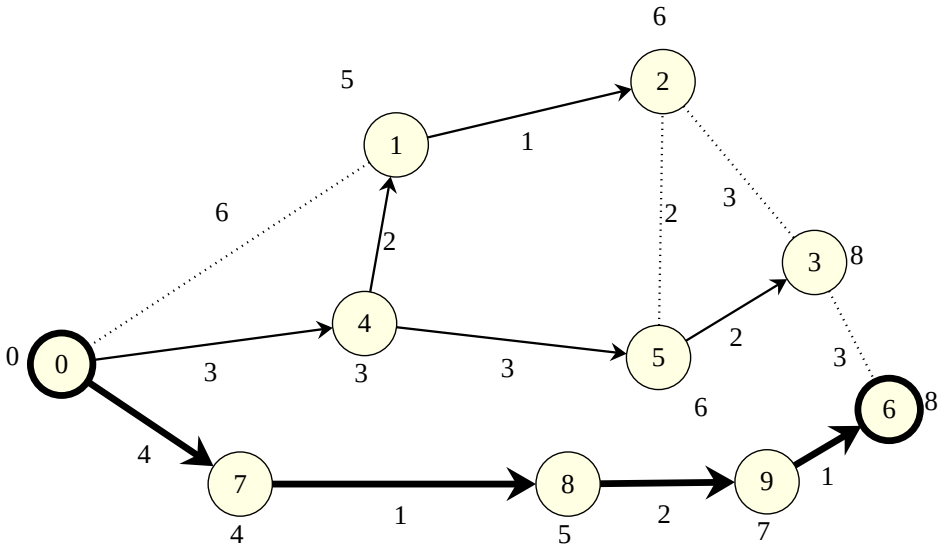

Abbildung 4.2.2: Kostengünstigste Pfade

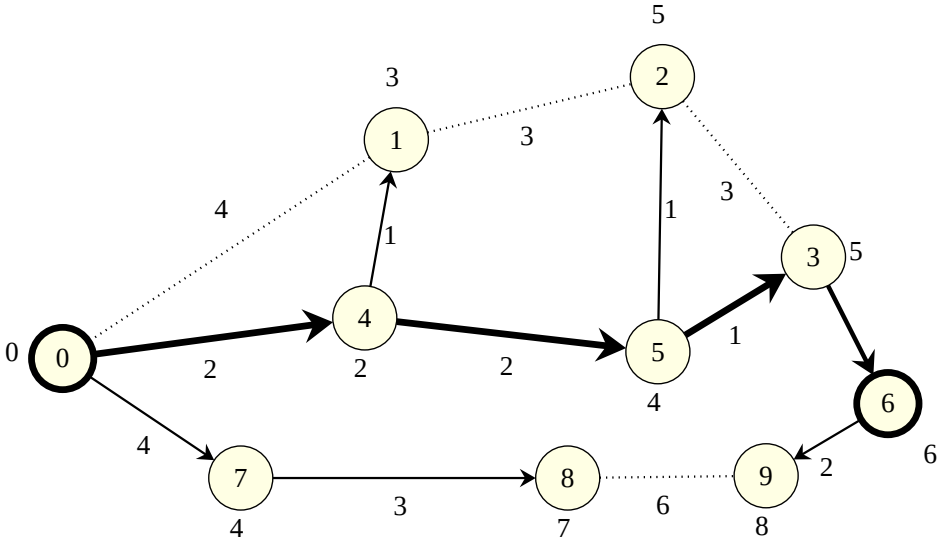

Abbildung 4.2.3: Kürzeste Pfade hinsichtlich Delay

gungen entspricht. Das Problem wäre diesfalls ebenso unlösbar. Kann hingegen durch die Berechnung des kürzesten Pfades hinsichtlich Delay ein Weg für den Transport von der Quelle zur Senke gefunden werden, welcher die Delaykriterien erfüllt, so ist eine gültige Lösung für das Problem gefunden – möglicherweise eine neue obere Schranke.

Zusätzlich zu den kürzesten Pfaden von der Quelle aus werden auch die kürzesten Pfade von der Senke aus berechnet – jeweils für Kosten und De-

lay. Dadurch ist es möglich Knoten und Kanten, die sicher nicht Teil eines Lösungspfades sind, zu eliminieren. Hierfür wird der kürzeste Weg zu einem Knoten von der Quelle und der Senke aus berechnet. Verletzt ein Pfad den erlaubten Delay oder ist er über der unteren Schranke, so kann der Knoten (und seine Kanten) gelöscht werden (Abbildung 4.2.4). Gleiches gilt für Kanten.

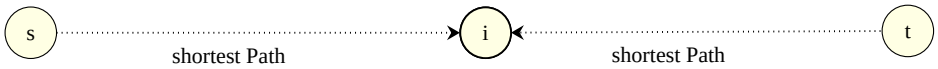

Abbildung 4.2.4: Bestimmung, ob Knoten gelöscht werden sollen

Es wird der kürzeste Pfad zu den Knoten dieser Kante von der Quelle und der Senke aus berechnet. Verletzt der Pfad, der sich aus den kürzesten Pfaden zu den Knoten und der Kante zusammensetzt, den erlaubten Delay oder ist er über der unteren Schranke, so kann die Kante gelöscht werden (Abbildung 4.2.5). In unserem Beispiel würden die Knoten 7, 8 und 9 gelöscht werden,

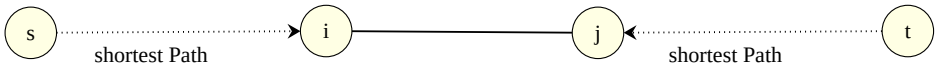

Abbildung 4.2.5: Bestimmung, ob Kante gelöscht werden soll

da kein Pfad über diese Knoten einen den maximalen Gesamtdelay einhalten könnte. Weiters fallen die Kanten (0,1) und (2,3) weg. Das Ergebnis ist aus Abbildung 4.2.6 ersichtlich. Diese Vorgangsweise wird so lange wiederholt bis

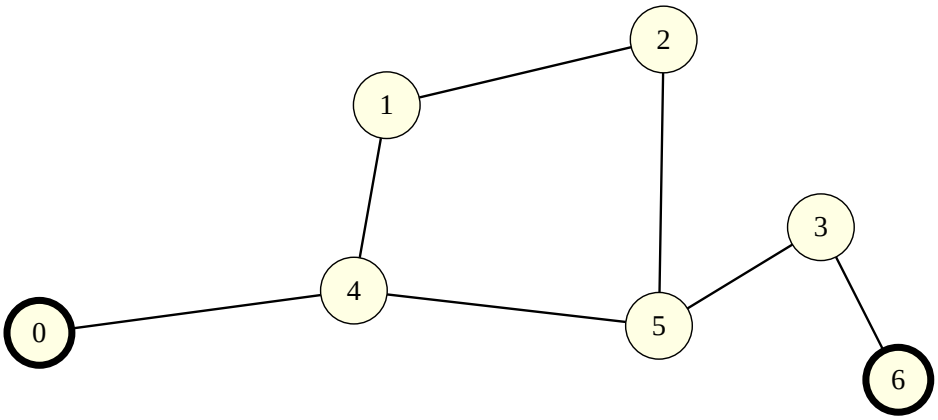

Abbildung 4.2.6: Graph nach einem Preprocessing-Durchlauf

sich im Netzwerk nichts mehr ändert. Somit wird oft schon der gesamte Pfad gefunden oder das Netzwerk zumindest reduziert. Ein darauf aufbauender Algorithmus kann somit einfacher zu einem Ergebnis kommen.

Labeling-Algorithmus

Einen auf dem zuvor besprochenen Verfahren aufbauenden Algorithmus beschreiben Dumitrescu und Boland in der selben Publikation [DB03] als Modified Label-Setting Algorithm (MLSA). Dieser versucht aufbauend auf den bisherigen Ergebnissen den beschränkten kürzesten Pfad zu finden. Hierfür kommt ein Labeling-Algorithmus zum Einsatz. Ein Label ist in diesem Fall ein Tupel aus Kosten und Delay, welches einen Pfad darstellt. Jedem Knoten kann eine Menge von Labels zugeordnet sein, welche die besten Pfade zu diesem Knoten im Sinne von Delay oder Kosten darstellen. Ähnlich wie beim Algorithmus von Dijkstra werden ausgehend vom Startknoten die Labels berechnet. Wird ein neuer Pfad bis zu einem Knoten gefunden, muss das neu entstandene Label betrachtet werden. Dominiert eines von zwei Labeln in der Labelmenge des Knotens – das heißt Delay und Kosten sind besser – wird nur das bessere behalten, andernfalls müssen beide Labels weitergeführt werden. Ein Label kann ebenso entfernt werden, wenn es den maximal erlaubten Delay überschreitet. Die Laufzeit des MLSA ist $\mathcal{O}(|A| \cdot d_{\max})$, sie ist demnach von der Anzahl der Kanten und dem maximal für diesen Transport erlaubten Gesamtdelay abhängig. Angewandt auf den Beispielgraphen sieht das Resultat des Algorithmus wie in Abbildung 4.2.7 aus.

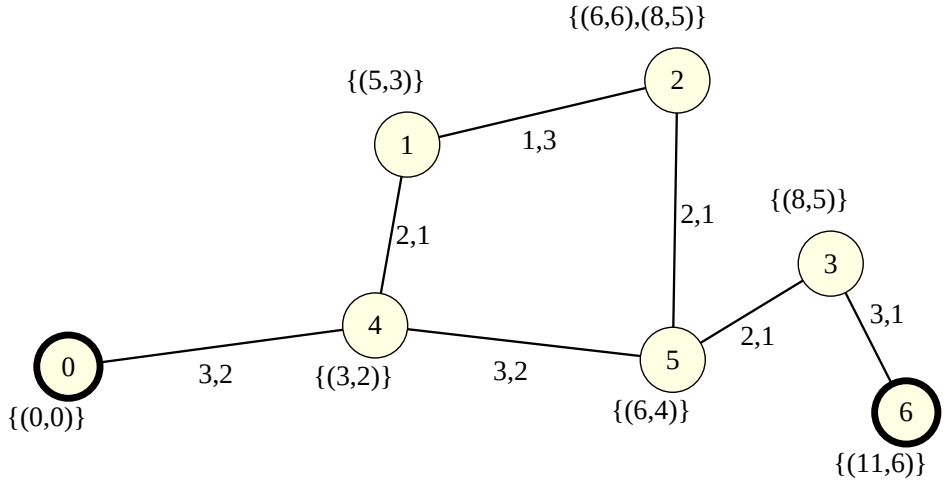

Abbildung 4.2.7: Graph nach Anwendung des Label-Algorithmus

Wurden alle Labels berechnet, so kann aus den Labels im Endknoten das Beste – das heißt kostengünstigste – Label gewählt werden. Der zugehörige Pfad entspricht dem kürzesten delaybeschränkten Pfad. Während des Labelings fallen Label mit zu hohem Delay weg.

4.3 Lagrange-Heuristik

Eine Lagrange-Heuristik versucht aus der Lösung des LLBP eine gültige Lösung zu erzeugen. Die Lösungen des LLBP kann in zweierlei Hinsicht die Bedingungen des Problems verletzten. Zum einen können die Kapazitätsbeschränkungen der Kanten verletzt sein, zum anderen kann der maximale Gesamtdelay überschritten sein.

Im Folgenden wird eine einfache Methode beschrieben, die versucht aus dem Ergebnis des LLBP eine gültige Lösung für das ursprüngliche Problem zu finden. In Algorithmus 4.3.1 wird diese vorgestellt. Ausgangspunkt ist die

Algorithmus 4.3.1 Reparieren einer LLBP-Lösung

```
1: T = ∅ //zu routende Transporte
2: for all Transporte k do
3:    Füge Kapazitäten für den gefundenen Pfad für Transport k hinzu
4:    if Kapazitäten einer Verbindung überschritten then
5:       Lösche hinzugefügte Kapazitäten dieses Transportes
6:       Füge Transport k zu T hinzu
7:    end if
8: end for
9: for all Transporte in T do
10:   Finde einen neuen delay-beschränkten Pfad im Netzwerk mit den verbleibenden Kapazitäten
11: end for
12: if Gesamtdelay OK then
13:   return Gültige Lösung
14: end if
```

Lösung des LLBP, die einen Pfad pro Transport angibt. Zuerst wird versucht nacheinander die Pfade der Transporte hinzuzufügen. An jeder Kante des Pfades werden somit Kapazitäten beansprucht – die verwendete Kapazität der Kante wird jeweils erhöht. Wird durch das Hinzufügen des Pfades eines Transportes eine Kantenkapazität überschritten, so wird dieser Transport später behandelt. In diesem ersten Teil wird für alle Transporte versucht den Pfad der Lösung des LLBP hinzuzufügen.

Im zweiten Teil müssen nun für die Transporte, welche nicht hinzugefügt werden konnten, neue Pfade gefunden werden. Dies sind delay-beschränkte kürzeste Pfade auf dem Netzwerk mit den Restkapazitäten. Konnte im zweiten Teil für jeden Transport ein gültiger Pfad gefunden werden, so ist eine neue gültige Lösung entstanden, sofern der maximal erlaubte Gesamtdelay nicht überschritten wird.

Kapitel 5

Spaltengenerierung

Die Idee der Spaltengenerierung stammt aus dem Simplexverfahren. Bei diesem kommen immer nur Variablen mit negativen reduzierten Kosten (bei einem Minimierungsproblem) in die Basis, um die Lösung zu verbessern [LD05]. Bei der Spaltengenerierung wird mit einer kleinen Menge an Variablen, die eine gültige Startlösung bilden, begonnen. Zu dieser beschränkten Anzahl an Variablen werden iterativ verbessernde Variablen hinzugefügt. Dies sind jene Variablen, die so genannte *reduzierte Kosten* aufweisen.

Durch diese, gegenüber dem Simplexalgorithmus andere Herangehensweise an das Problem, wird erreicht, dass nicht alle Variablen betrachtet werden müssen. Dieser Vorteil kann bei sehr vielen potentiellen Variablen genutzt werden. Spaltengenerierung ist daher eine Möglichkeit, Lineare Programme mit sehr vielen Variablen schneller zu lösen.

5.1 Allgemeines

Spaltengenerierung wurde erstmals von Gilmore und Gomory [GG61] eingesetzt. Ausgangspunkt bei der Spaltengenerierung ist ein zu lösendes Lineares Programm, welches im Folgenden als Master Problem (MP) bezeichnet wird. Direktes Lösen des MP mittels Simplexverfahren würde erfordern, in der (großen) Menge aller Variablen in jeder Iteration eine Nicht-Basisvariable zu finden, welche die Zielfunktion verbessert. Spaltengenerierung umgeht dieses Problem indem diese Variablen erst bei Bedarf hinzugefügt werden. Genauer gesagt wird das Master Problem nicht auf der ursprünglichen Menge an Variablen betrachtet, sondern als so genanntes *Restricted Master Problem (RMP)* auf einer sehr beschränkten Variablenmenge. Es wird mit einer beschränkten Anzahl an Variablen (Spalten) gestartet, die eine gültige Lösung bilden. Jede hinzugefügte Variable entspricht einer Spalte in der

Koeffizientenmatrix, wodurch sich der Name des Verfahrens erklärt. Für dieses RMP wird zuerst die LP-Relaxierung berechnet, danach wird nach einer Variable gesucht, welche negative (im Falle eines Minimierungsproblems) reduzierte Kosten aufweist. Solch eine Variable erweitert den Lösungsraum und kann dadurch die Zielfunktion weiter verbessern. Das Subproblem des Findens der Variable mit den minimalen (bei einem Minimierungsproblem) reduzierten Kosten wird als *Pricing-Problem* bezeichnet. Die gefundene Variable wird dann in die Variablenmenge aufgenommen. Iterativ werden immer neue Variablen aufgenommen, bis keine verbessernden Variablen mehr gefunden werden können. Lösungen des RMPs bilden obere Schranken (im Falle eines Minimierungsproblems) für das Ausgangsproblem. Durch diese Herangehensweise an das Problem ist es möglich, aus einer nicht explizit bekannten Menge an möglichen Lösungen, nur jene auszuwählen, die tatsächlich gebraucht werden. Auf diese Weise können Lineare Programme mit sehr vielen Variablen, die sonst nicht (oder nur sehr schwer) gelöst werden könnten, behandelt werden.

Sei folgende allgemeine Form des Lineares Programm das Master Problem(MP):

$$\begin{aligned} \min \quad & cx && \text{(5.1.1a)} \\ s.t. \quad & Ax \geq b && \text{(5.1.1b)} \\ & c, x \in \mathbb{R}^n && \text{(5.1.1c)} \\ & b \in \mathbb{R}^m && \text{(5.1.1d)} \\ & A \in \mathbb{R}^{m \times n} && \text{(5.1.1e)} \end{aligned}$$

In jeder Iteration des Simplexverfahrens wird eine Nicht-Basisvariable gesucht, welche die Zielfunktion verbessert. Sei λ_j, $\forall j = 1, \ldots, n$ die duale Variable zu (5.1.1b), so wird eine Variable gesucht die

$$\arg\min \left\{ \bar{c}_j = c_j - \sum_{i=1}^{m} \lambda a_{ij} \mid j = 1, \ldots, n \right\}. \tag{5.1.2}$$

Da die Anzahl n der möglichen Variablen sehr groß ist, wird von einer beschränkten Variablenmenge ausgegangen. Solange es Variablen mit reduzierten Kosten

$$\bar{c}_j \leq 0 \tag{5.1.3}$$

gibt, werden diese zur Variablenmenge hinzugefügt. Gibt es keine Variable mit negativen reduzierten Kosten, kann keine Variable die Lösung verbessern. Eine umfassende Einführung in das Gebiet der Spaltengenerierung findet sich zum Beispiel in [LD05].

5.2 Alternative ILP-Formulierung

Anders als in den vorigen Kapiteln, kann das vorliegende Netzwerkproblem auch durch eine andere Darstellung beschrieben werden. Hatten wir bisher die Formulierung, ob eine Kante für einen Transport verwendet wird, so wird in diesem Abschnitt eine Beschreibung durch Pfade eingesetzt.

Ein Pfad für einen Transport k von seiner Quelle s_k zu seiner Senke t_k wird durch φ_k beschrieben. Dies ist ein möglicher Pfad für den Transport k, wobei $\varphi_k \in P_k$. Die Menge aller möglichen Pfade für einen Transport k wird durch P_k angegeben. Die Menge P_k für einen Transport k setzt sich aus allen Pfaden φ_k zusammen, die von seiner Quelle s_k zu seiner Senke t_k gehen, mindestens v_k Einheiten transportiert können (5.2.2) und dies maximal innerhalb des maximalen Gesamtdelays d_{ges} schaffen. Ist der Transport zudem zeitkritisch muss dessen maximaler Delay eingehalten werden. Sichere Transporte dürfen nur auf Kanten mit sicheren Protokollen transportiert werden. Für jeden Pfad müssen die Flussbedingungen (2.2.1b, 2.2.1c, 2.2.1d) gelten. Die Zugehörigkeit einer Kante (i, j) zu einem Pfad φ_k, gibt

$$\delta_{ij}^{\varphi_k} = \begin{cases} 1, & \text{wenn } (i,j) \in \varphi_k \\ 0, & \text{sonst} \end{cases} \qquad \begin{matrix} \forall k = 1, \ldots, m, \\ \forall (i,j) \in E^k \end{matrix} \tag{5.2.1}$$

an. Das Einhalten der Kapazitätsbeschränkungen der Kanten wird durch

$$\delta_{ij}^{\varphi_k} \cdot v_k \leq u_{ij} \quad \forall (i,j) \in E^k \tag{5.2.2}$$

erreicht. Der entstehende Gesamtdelay eines Pfades ist durch

$$a'_{\varphi_k} = \sum_{(i,j) \in E^k} \delta_{ij}^{\varphi_k} \cdot \left(a_{ij}^k + d_{ij} \right) \tag{5.2.3}$$

gegeben. Die Protokollkosten eines Pfades sind durch

$$c'_{\varphi_k} = \sum_{(i,j) \in E^k} \delta_{ij}^{\varphi_k} \cdot p_{ij}^k \tag{5.2.4}$$

gegeben. Schließlich wird durch

$$w_{\varphi_k} = \begin{cases} 1, & \text{Pfad gewählt} \\ 0, & \text{sonst} \end{cases} \tag{5.2.5}$$

festgelegt, ob ein Pfad gewählt wird.

Unter Verwendung dieser Variablen wird nun eine alternative Formulierung für das Problem angegeben. Das Ziel ist es, die Kosten, die sich aus den Protokollkosten für die gewählten Pfade und den Basiskosten zusammensetzen, zu minimieren (5.2.6a).

$$\min \sum_{k=1}^{m} \sum_{\varphi_k \in P_k} c'_{\varphi_k} \cdot w_{\varphi_k} + \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} \tag{5.2.6a}$$

$$\text{s.t.} \sum_{\varphi_k \in P_k} w_{\varphi_k} \geq 1 \qquad \forall k = 1, \ldots, m \tag{5.2.6b}$$

$$\sum_{\varphi_k \in P_k} \delta_{ij}^{\varphi_k} \cdot w_{\varphi_k} \leq x_{ij} \qquad \forall k = 1, \ldots, m,\ (i,j) \in E \tag{5.2.6c}$$

$$\sum_{k=1}^{m} \sum_{\varphi_k \in P_k} a'_{\varphi_k} \cdot w_{\varphi_k} \leq d_{\text{ges}} \tag{5.2.6d}$$

$$\sum_{k=1}^{m} \sum_{\varphi_k \in P_k} w_{\varphi_k} \cdot \delta_{ij}^{\varphi_k} \cdot v_k \leq u_{ij} \qquad \forall (i,j) \in E \tag{5.2.6e}$$

$$a'_{\varphi_k} \leq \min(\delta_k, d_{\text{ges}}) \quad \forall \varphi_k \in P_k,\ \forall k = 1, \ldots, m \tag{5.2.6f}$$

$$w_{\varphi_k} \in \{0, 1\} \qquad \forall \varphi_k \in P_k \tag{5.2.6g}$$

Für jeden Transport k muss mindestens ein Pfad w_{φ_k} aus der Menge der für diesen Transport möglichen Pfade P_k gewählt werden (5.2.6b). Die Kopplungsvariable x_{ij} drückt aus, ob eine Kante (i, j) von einem Transport benützt wird. Sie wird durch (5.2.6c) definiert. Der Gesamtdelay ist wieder beschränkt durch d_{ges}; diese Bedingung wird durch (5.2.6d) sichergestellt. Für jede Kante müssen die Kantenkapazitäten u_{ij} beachtet werden (5.2.6e). Der Delay eines Pfades wird durch den maximalen Gesamtdelay und dem jeweiligen maximalen Delay (5.2.6f) für jeden Transport beschränkt.

5.3 Pricing Problem

Die Anzahl der möglichen Pfade in der ILP-Formulierung 5.2.6 ist zu groß, um das Problem direkt effizient zu lösen. Eine Möglichkeit, die Enumeration aller Pfade zu umgehen, ist die in Abschnitt 5.1 besprochene Spaltengenerierung. Durch das Starten mit einer kleineren, reduzierten Menge $P_k^R \in P_k$, ist

es nicht notwendig alle möglichen Pfade aufzuzählen. Diese reduzierte Startmenge enthält im Idealfall eine gültige Lösung, welche durch eine Heuristik (siehe Abschnitt 5.4) erzeugt wird. Das Problem, welches auf dieser reduzierten Pfadmenge gelöst wird, ist das Restricted Master Problem (RMP).

Zunächst muss eine neue Variable gefunden werden, welche die Lösung verbessert. Sei μ_k, $\forall k = 1, \ldots, m$, die duale Variable zur Bedingung (5.2.6b), die fordert, dass mindestens ein Pfad pro Transport existiert und $\pi_{e,k}$, $\forall e \in E$, $\forall k = 1, \ldots, m$, die duale Variable für (5.2.6c). Weiters sei η die duale Variable für die Bedingung (5.2.6d), welche den Gesamtdelay beschränkt und ρ_e, $\forall e \in E$, die duale Variable der Kapazitätsbeschränkungen der Kanten (5.2.6e). Durch Lösung des reduzierten Masterproblems erhalten wir konkrete Werte für die dualen Variablen μ_k, $\pi_{e,k}$, η und ρ_e. Die reduzierten Kosten ergeben sich dann für jeden Pfad zu

$$\bar{c}_{\varphi_k} = c'_{\varphi_k} - \mu_k + \sum_{e \in \varphi_k} \pi_{e,k} + a'_{\varphi_k} \cdot \eta + \sum_{e \in \varphi_k} \rho_e \upsilon_k \,. \tag{5.3.1}$$

Das Pricing Problem ist das suchen eines Pfades mit negativen reduzierten Kosten. Durch das Hinzufügen der entsprechenden Variablen kann anschließend eine bessere Lösung für das RMP gefunden werden. Es ist also ein Pfad φ_k mit $\bar{c}_{\varphi_k} < 0$ gesucht. Dies entspricht einem kürzesten Pfad $\forall \varphi_k \in P_k$ auf dem Graphen mit Kantenkosten

$$\hat{c}^k_{ij} = p^k_{ij} + \pi_{e,k} + \eta \cdot (a_{ij} + d_{ij}) + \rho_e \cdot \upsilon_k \,. \tag{5.3.2}$$

Diese Kosten ergeben sich aus den reduzierten Kosten für einen Pfad eines Transportes $\bar{c}_{\varphi_k}$. Bei der Wahl des Pfades ist der maximale Gesamtdelay und bei zeitkritischen Transporten zusätzlich der spezifische maximale Delay zu beachten. Weiters sollen nur Kanten für den Pfad gewählt werden deren Kantenkapazitäten ausreichen. Die gefundenen Pfade werden als Variable zum RMP hinzugefügt und dieses wird erneut gelöst. Sind keine Pfade mit negativen reduzierten Kosten vorhanden, so kann keine weitere Variable hinzugefügt werden, welche die Zielfunktion weiter verbessern kann. In Analogie zur Lagrange Relaxierung (Kapitel 4) werden derartige Pfade anhand des in [DB03] vorgestellten Algorithmus berechnet. Zu beachten ist hierbei, dass dieser Algorithmus im Falle von negativen Kostenzyklen nicht gewährleistet, dass ein elementarer Pfad gefunden wird, d.h. ein Pfad in welchem Kanten jeweils nur einmal vorkommen. Da jedoch nicht zu erwarten ist, dass die Kantenkosten $\hat{c}^k_{ij}$ oftmals negative Kostenzyklen bewirken, wird zugunsten der besseren Performance kein Algorithmus verwendet der auch diesfalls elementare Pfade ermittelt. Eine genauere diesbezügliche Analyse sprengt jedoch den Rahmen dieser Diplomarbeit.

5.4 Primale Heuristik: Generieren einer gültigen Lösung

Für die Spaltengenerierung wird eine gültige Startlösung benötigt, um davon ausgehend eine bessere Lösung zu generieren. Das Erhalten (irgend-)einer gültigen Lösung kann für das Problem jedoch auch in anderer Hinsicht von Bedeutung sein. Einerseits ist es bei sehr großen Instanzen wichtig überhaupt eine Lösung zu erhalten, wenn mit anderen Verfahren in absehbarer Zeit keine gefunden wird. Andererseits ist eine gültige Lösung eine obere Schranke für das Problem und kann daher bei Verfahren, die in erster Linie untere Schranken generieren, die Lösungsqualität verdeutlichen. In diesem Abschnitt wird eine Möglichkeit aufgezeigt, um eine gültige Lösung für das Problem zu generieren. Einen Überblick des angewandten Verfahrens gibt Algorithmus 5.4.1. In diesem wird in erster Linie versucht für jeden Transport einen Pfad zu finden, der genügend Kapazitäten für den jeweiligen Transport bietet. Bei diesem Vorgang werden die Kapazitäten des Netzwerks um die Größe des Transports auf dem gefundenen Pfad reduziert. Der nächste Transport wird dann über dieses reduzierte Netzwerk geroutet. Findet sich für einen Transport kein Weg im verbleibenden Netzwerk, so wird versucht für die schon durch das Netzwerk gerouteten Transporte andere Pfade zu finden. Hierbei werden jene Transporte ausgewählt, die den Weg des aktuellen Transportes blockieren. Nach dem erfolgreichen Routen aller Transporte ist auf eine mögliche Überschreitung des maximal erlaubten Gesamtdelays hin zu prüfen. Ist dieser überschritten, wird angestrebt für jeden Transport einen zeitlich kürzeren Pfad als den aktuell zugewiesenen zu finden. Dies wird so lange wiederholt bis der aktuelle Delay unterhalb des maximalen Delays liegt oder während der letzten Iteration keine Verbesserung des Delays mehr erwirkt werden konnte.

Da das Finden einer gültigen Lösung selbst schon ein schweres Problem darstellen kann, garantiert Algorithmus 5.4.1 nicht, dass eine derartige Lösung gefunden wird. Ohne gefundener gültiger Lösung hat die Spaltengenerierung keine Ausgangslösung. Für diesen Fall werden während des Generierens der gültigen Lösung alle gefundenen Pfade gespeichert. Es ist möglich, dass es mit diesen Pfaden eine nicht-ganzzahlige Lösung gibt. Ist dies der Fall kann die Spaltengenerierung ohne gültige ganzzahlige Startlösung angewandt werden.

Algorithmus 5.4.1 Primale Heuristik: Generieren einer validen Lösung

1: Seien u_{ij} die Kapazitäten des Netzwerks
2: Seien $\tilde{u}_{ij}$ die restlichen Kapazitäten des Netzwerks; $\tilde{u}_{ij} = u_{ij}\ \forall (i,j) \in E$
3: **for all** Transporte k **do**
4: Setze $U = \emptyset$
5: Finde den kürzesten delay-beschränkten Pfad für Transport k im Netzwerk ohne Kanten U mit den Kapazitäten $\tilde{u}_{ij}$
6: **if** Pfad für Transport k gefunden **then**
7: Vermindere die Kapazitäten $\tilde{u}_{ij}$ an diesem Pfad um die Größe des Transports
8: **else** // Versuche die Pfade anderer Transporte zu verändern um einen Pfad für Transport k zu finden
9: Finde einen kürzesten delay-beschränkten Pfad für Transport k im Netzwerk mit Kapazitäten u_{ij}
10: **if** kein Pfad gefunden **then** // Kein Pfad existiert für Transport k
11: **return** keine Lösung gefunden
12: **else**
13: **for all** Kanten (i,j) dieses Pfades, die nicht von Transport k benutzt werden können, da die Kapazitäten nicht ausreichen **do**
14: **for all** Transporte die Kante (i,j) benutzen **do**
15: Finde einen Pfad im Netzwerk mit den Kapazitäten $\tilde{u}_{ij}$ aktualisiere die Kapazitäten $\tilde{u}_{ij}$
16: **if** genügend Kapazität $\tilde{u}_{ij}$ für Transport k **then**
17: Setze mit nächste Kante in Zeile 13 fort
18: **end if**
19: **end for**
20: **end for**
21: **end if**
22: **if** nicht genügend Platz auf diesem Pfad geschaffen werden kann **then**
23: Füge alle Kanten die den Transport nicht ermöglichten zu U hinzu
24: Wiederhole ab Zeile 9
25: **else**
26: Pfad für Transport k gefunden; aktualisiere Kapazitäten $\tilde{u}_{ij}$
27: **end if**
28: **end if**
29: **end for**
30: **if** Gesamtdelay der aktuellen Lösung zu groß **then**
31: **repeat**
32: **for all** Transporte k **do**

33: Finde einen Pfad mit geringerem Delay für Transport k unter Beachtung der verwendeten Kapazitäten
34: **end for**
35: **until** Gesamtdelay OK oder keine Verbesserung gefunden
36: **if** Gesamtdelay nicht OK **then**
37: **return** keine Lösung gefunden
38: **end if**
39: **end if**
40: **return** die aktuelle Lösung

Kapitel 6

Implementierung

In diesem Kapitel werden Details zur Implementierung der Algorithmen beschrieben. Zuerst wird das Format der Eingabedaten spezifiziert. Im Anschluss werden die Programmstruktur und wichtige Aspekte der Implementierung der Algorithmen erläutert. Im Programm wurden alle besprochenen Verfahren – ganzzahliges Lineares Programm, Lagrange Relaxierung (mit Subgradientenverfahren und Volume Algorithmus), Spaltengenerierung – umgesetzt.

6.1 Aufbau der Testinstanzen

Eingabedaten werden durch zwei Dateien an das Programm übergeben. Eine beinhaltet die Informationen bezüglich des Netzwerks, die andere enthält die Transportdaten.

In der Netzwerkdatei werden die Knoten und Verbindungen zwischen diesen beschrieben. Darüber hinaus sind für jede Kante Daten gespeichert, wie zum Beispiel Informationen bezüglich Kosten, dem durch sie hervorgerufenen Delay und ihre Kapazität. Weiters sind in dieser Datei die auf der jeweiligen Verbindung möglichen Protokolle näher spezifiziert.

Die Transportdatei enthält Informationen die festlegen welche Kapazitäten von einem bestimmten Start- zu einem definierten Endpunkt transportiert werden sollen. Manche Transporte müssen innerhalb einer bestimmten Zeit an ihrem Ziel ankommen. Dieser maximale Delay ist ebenso angegeben wie eine Sicherheitseinstufung, die der jeweilige Transport erfüllen muss.

Nachfolgend wird der Aufbau der beiden Inputdateien genauer beschrieben.

6.1.1 Netzwerkdatei

Die Netzwerkdatei setzt sich aus drei Teilen zusammen. Zuerst werden die Knoten beschrieben – jeweils durch eine fortlaufende Nummer (#) und einen Namen (name). Danach werden die Protokolle beschrieben. Diese haben eine Nummer (#), einen Namen (name), der sie identifiziert, Kosten (cost) und einen Delay (delay) zugeordnet. Außerdem können sie entweder (secure) sicher oder unsicher sein, dies ist durch *true* für sicher und *false* für unsicher angegeben. Danach folgt die Beschreibung der Verbindungen zwischen den Knoten. Eine Nummer identifiziert die Verbindung, die von einem angegebenen Startknoten (start) zu einem Endknoten (end) führt. Weiters sind die Linkkosten (cost), der Delay (delay) und die Kapazität der Kante (cap) angegeben. Das Protokoll (protocol) wird durch den Namen des Protokolls angegeben. Jede Kante hat außerdem einen Namen (name).

Bietet eine Verbindungen verschiedene Protokolle an, so wird dies durch zwei Zeilen dargestellt. Diese enthalten jeweils gleiche Werte für id, start, end, costs, delay, cap und name, jedoch unterschiedliche Protokolle.

Die Kanten in den Testdaten sind ungerichtet, Nachrichten können in beide Richtungen transportiert werden. Ein Beispiel für den Aufbau einer Netzwerkdatei zeigt Abbildung 6.1.1 .

6.1.2 Transportdatei

Die Transportdatei beschreibt die zu routenden Transporte. Für jeden Transport wird neben einer Identifikationsnummer (#), dem Start- (start) und Zielknoten (end), der Größe des Transports (size) und einem Namen (name), ein maximaler Delay angegeben. Ist dieser 0, so ist der jeweilige Transport nicht zeitkritisch, andernfalls muss er in der angegebenen Zeit sein Ziel erreichen. Weiters wird für jeden Transport die benötigte Sicherheitseinstellung (secure) angegeben. Der Transport muss bei *true* mit einem sicheren Protokoll transportiert werden, andernfalls ist das Protokoll nicht relevant. Jeder Transport kann nur genau einen Pfad wählen, da die Nachrichten eine nicht teilbare Einheit darstellen. Abbildung 6.1.2 zeigt eine mögliche Transportdatei.

6.2 Programmstruktur SWIS

Das Programm zur Lösung des beschriebenen Problems wurde vollständig in C++ implementiert. Der verwendete Compiler ist G++ 4.1.2 Compiler unter Debian 4.0 (x86_64). Zur Lösung des ganzzahligen Linearen Programms wurde Ilog CPLEX [ILO08], Version 11.0 verwendet.

```
# 50 nodes

#       name
0  Node_0
1  Node_1
2  Node_2

...

# 2 protocols

#  name  cost  delay  secure
0  TCP  1  1  false
1  HTTPS  2  2  true


# 800 links

#  start  end  cost  delay  cap  protocol  name
0  0    10  4  2  10  HTTPS  Link_0
0  0    10  4  2  10  TCP  Link_0
1  4    15  2  1  100  TCP  Link_1
2  2    16  4  1  10  TCP  Link_2

...
```

Abbildung 6.1.1: Beispiel einer Netzwerkdatei

```
#  start  end  size  delay secure name
0  4  10  100  0  true  Transp_0
1  7  0  10  20  false  Transp_1
```

Abbildung 6.1.2: Beispiel einer Transportdatei

Über ein Kommandozeilenprogramm können Testdaten als Input angegeben und Lösungmethoden ausgewählt werden. Je nach Methode wird versucht eine gültige Lösungen beziehungsweise obere und/oder untere Schranken zu generieren. Durch ein Zeitlimit kann die Ausführungsdauer eingeschränkt werden.

In Abbildung 6.2.1 zeigt einen Überblick der Struktur des Programmes. Ausgehend von SWIS_Main, welches das Kommandozeilenprogramm startet, werden zuerst die Daten aus der Netzwerkdatei und der Transportdatei ausgelesen. Dies geschieht in der für die Ein- und Ausgabe zuständigen Klasse SWIS_IO. Ab diesem Zeitpunkt sind durch ein Singleton-Objekt (siehe [GHJV95]) sämtliche Informationen zum Netzwerk und zu den Transporten über SWIS_Data aufrufbar. Infolge können die einzelnen Algorithmen – ILP mit CPLEX, Spaltengenerierung, eine valide Lösung und Lagrange Relaxierung mit Subgradientenverfahren und Volume Algorithmus – gestartet werden. Wurde eine Lösung gefunden, so ist sie als SWIS_Solution verfügbar.

Alle Algorithmen bis auf die Lösung durch ein ganzzahliges Lineares Programm (ILP) greifen zur Lösung auf das Subproblem *Constrained Shortest Path* – realisiert durch SWIS_CSP – zurück. Für die Lösung des ILP und bei der Spaltengenerierung wird CPLEX eingesetzt. Näheres dazu wird bei den einzelnen Algorithmen in den nächsten Abschnitten besprochen.

Die Aufrufparameter des Programms SWIS_Main für den Kommandozeilenaufruf sind in Tabelle 6.1 zusammengefasst. Mit diesen kann der Lauf des Programms bestimmt werden. Ein Aufruf erfolgt durch *./swis [Optionen] NetzwerkDatei TransportDatei*.

6.2.1 Ein- und Ausgabe – Input/Output

Die Klasse SWIS_IO ermöglicht einerseits das Einlesen von Informationen über das Netzwerk und die Transporte. Andererseits kann nach einer erfolgreichen Lösung des Problems die gefundene Lösung mit dieser Klasse ausgegeben werden.

Es können Datenfiles mit dem in Kapitel 6.1 beschrieben Format eingelesen werden. Die zwei Dateien – die Netzwerkdatei und die Transportdatei – werden hierfür als Parameter übergeben. Es ist möglich anzugeben, ob es zeitkritische Transporte gibt. Ist dies nicht der Fall, so ist im Datenfile der Transporte die Spalte *delay* nicht vorhanden. Dies ist vor allem aus Kompatibilitätsüberlegungen zu schon vorhandenen Testfiles sinnvoll. In diesen älteren Files ist die Spalte delay im Transportfile nicht vorhanden.

Als zusätzliche Option kann die Anzahl der eingelesenen Transporte angegeben werden. So werden zum Beispiel nur die ersten 100 Transporte aus einem File genommen.

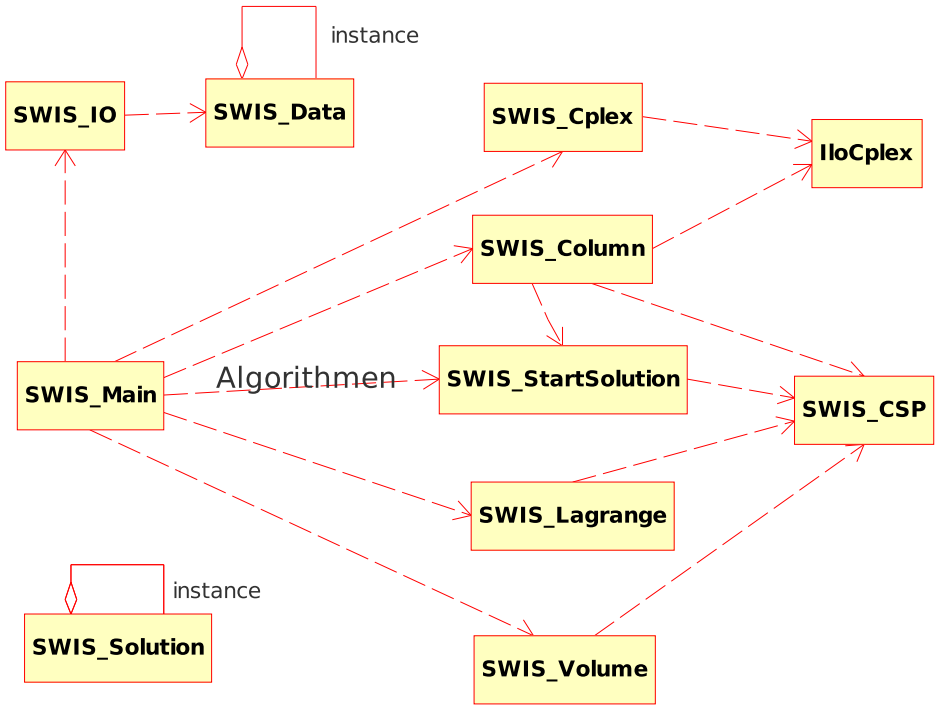

Abbildung 6.2.1: Überblick über die Programmstruktur

Die in den Dateien enthaltenen Informationen werden in SWIS_Data abgelegt und sind ab diesem Zeitpunkt für alle Klassen verfügbar. Wie schon angeführt ist eine weitere Aufgabe dieser Klasse die einheitliche Ausgabe der Lösung.

6.2.2 Datenhaltung

Um von überall im Programm auf die gesamten Daten zugreifen zu können, sind alle Informationen zum Netzwerk, den Transporten und einer etwaigen Lösung zentral gespeichert. Zum einen sind Informationen zur Problemstellung – Netzwerk und Transporte – durch SWIS_Data dargestellt. Zum anderen wird eine Lösung des Problems durch SWIS_Solution verfügbar. Der Zugriff auf beide Klassen erfolgt über ein Singleton-Objekt.

SWIS_Data stellt Informationen zum Netzwerk und den Transporten bereit. So sind beispielsweise Eckdaten des Netzwerks, wie die Anzahl der Knoten und Links, aber auch zur Struktur des Netzwerkes – Nachfolgeknoten und Eigenschaften der Verbindungen (z.B.: Kapazitäten und Linkdelay) verfügbar. Ähnliche Informationen stehen auch über Transporte zur Verfügung, wie zum Beispiel die Anzahl oder die Eigenschaften der Trans-

Tabelle 6.1: Aufrufoptionen für das Programm

Option	**Parameter**	**Erklärung**
-h		gibt eine Hilfe zu den Optionen aus
-s	SEC	maximale Dauer zur Lösung jedes Algorithmus in Sekunden (0 = kein Zeitlimit; Default = 7200s = 2h)
-u	NUM	Anzahl der Transporte, die eingelesen (Default = alle Transporte der Datei)
-t		zeitkritische Transporte kommen vor
-d	DELAY	maximaler Gesamtdelay (Default = 200)
-g		generiert eine Lösung mittels Primaler Heuristik
-i		löst das Problem als ILP mit CPLEX
-l		löst das Problem mit Lagrange Relaxierung (Subgradientenverfahren)
-v		löst das Problem mit Lagrange Relaxierung (Volume Algorithmus)
-c		löst das Problem mit Spaltengenerierung

porte (z.B.: Größe und maximaler Delay).

Während SWIS_Data die Ausgangsdaten des Problems enthält, ist über SWIS_Solution die Lösung des Problems abrufbereit. Es ist für jede Verbindung abrufbar, ob sie von einem bestimmten Transport verwendet wird.

6.2.3 Algorithmen

Um das Problem zu lösen, können aus dem Hauptprogramm SWIS_Main die einzelnen Algorithmen aufgerufen werden. Die Lösung durch ein ganzzahliges Lineares Programm erfolgt durch SWIS_Cplex, die Spaltengenerierung ist in SWIS_Column realisiert. Hierfür wird als Startlösung irgendeine gültige Lösung benötigt, solch eine gültige Lösung kann durch SWIS_StartSolution generiert werden. Diese kann auch abseits der Spaltengenerierung verwendet werden, um schnell irgendeine Lösung zu erhalten. Die Lagrange Relaxierung wird einerseits mit dem Subgradientenverfahren in SWIS_Lagrange umgesetzt, andererseits mit dem Volume Algorithmus in SWIS_Volume. Durch Kombination der angegebenen Einstellung (Konfiguration, Knoten, Kanten und Transporte) enthält jedes Testset 126 Testfälle. Um aussagekräftige Resultate zu erhalten wurde mit drei Testsets getestet.

Außer bei der Lösung als ganzzahliges Lineares Programm benötigen al-

le Algorithmen das Finden eines delay-beschränkten kürzesten Pfades beim jeweiligen Lösungsweg. Dieses Finden eines delay-beschränkten kürzesten Pfades wurde nach dem in Kapitel 4.2.1 beschriebenen Algorithmus von Dumitrescu und Boland implementiert. Es ist in SWIS_CSP implementiert, wobei ein vorhergehendes Preprocessing Kanten eliminiert, die von dem jeweiligen Transport aufgrund seiner Größe nicht verwendet werden können.

6.3 Ganzzahliges Lineares Programm

Das ganzzahlige Lineare Programm wird mit Hilfe von Ilog CPLEX [ILO08], einem kommerziellen (I)LP Solver, gelöst. Der Name CPLEX leitet sich aus der Verbindung der Programmiersprache C und dem Simplexverfahren ab. Heute bietet CPLEX allerdings auch Interfaces für C++, C#, Java, Visual Basic und FORTRAN an.

Das ganzzahlige Lineare Programm wird mit CPLEX durch ein Modell repräsentiert, dem Bedingungen und eine Zielfunktion hinzugefügt werden. Danach wird die Methode *solve()* aufgerufen. Der CPLEX-Solver versucht dann das Modell zu lösen. Soll der Algorithmus nur eine bestimmte Anzahl an Sekunden zur Lösung des Problems verwenden, so wird der Parameter für das Zeitlimit (Time Limit) IloCplex::TiLim auf die maximal zugelassene Anzahl an Sekunden gesetzt.

Kapitel 7

Ergebnisse

Dieses Kapitel fasst die Ergebnisse der Tests zusammen. Im ersten Abschnitt werden die vorliegenden Testdaten aus dem Projekt SWIS genauer betrachtet und mit den implementierten Verfahren gelöst. Da diese Testfälle nur sehr kleine Instanzgrößen (maximal 30 Knoten und 11 Transporte) beinhalten, wurden größere Testdaten (bis hin zu 1000 Knoten und 1000 Transporten) notwendig. Im zweiten Abschnitt wird diese Testdatengenerierung näher beschrieben. Danach werden die Ergebnisse für jeden einzelnen Algorithmus analysiert. Abschließend werden detaillierte Vergleiche zwischen den einzelnen Verfahren angestellt.

7.1 Testdaten des Projekts SWIS

Aus dem Projekt SWIS wurden für diese Arbeit drei Testfälle zur Verfügung gestellt: Semantic_scenario, Scenario7 und Complex_scenario. Einen Überblick über die Eckdaten der Testfälle gibt Tabelle 7.1.

Leider wiesen zwei dieser Testfälle Probleme auf. So sollen laut Testfall einige Nachrichten sicher zwischen bestimmten Sendern und Empfängern transportiert werden, obwohl keine sicher Leitung zwischen diesen besteht.

Tabelle 7.1: Überblick über die Größe der Testfälle aus dem Projekt SWIS

Testfall	Knoten	Kanten	Transporte
Semantic	11	12	7
Scenario7	14	25	11
Complex	30	63	11

Deshalb werden in den Testfällen Scenario7 und Complex_scenario jeweils zwei Kanten zusätzlich als sicher markiert. Dadurch ist jeder Transport hinsichtlich Sicherheit möglich.

Desweiteren sind die Testfälle aufgrund einer etwas anderen Problemstellung nicht direkt lösbar, da das mehrfache Verbuchen von Kanten im Projekt SWIS zugelassen wurde. Das bedeutet, es ist erlaubt die Kapazitäten der Kanten, sollten sie nicht ausreichen, um ein Vielfaches zu erhöhen. Dies entspricht einem Netzwerkes, welches in mehreren Zeiteinheiten betrachten wird. (Die Kapazitäten werden dementsprechend verdoppelt, verdreifacht usw.). Dieses mehrfache Verbuchen ist in dem erarbeiteten Modell dieser Diplomarbeit nicht vorgesehen, weshalb die Testdaten dahingehend angepasst wurden.

7.1.1 Ergebnisse der SWIS-Testdaten

Diese relativ kleinen Netzwerke mit wenigen Transporte können ohne Probleme in weniger als 2 Sekunden exakt mit CPLEX gelöst werden. Tabelle 7.2 zeigt diese Ergebnisse. Für die hier betrachteten Testläufe wurde ein maxima-

Tabelle 7.2: Ergebnisse der SWIS-Testfälle beim Direkten Lösen mit CPLEX

Szenarioname	Optimum	Zeit [s]
Semantic	34	1
Scenario7	35	0
Complex	111	0

ler Gesamtdelay von 10000 angenommen. Experimente mit unterschiedlichen maximalen Gesamtdelays haben nur sehr gering veränderte Resultate gezeigt, weshalb diese Daten hier keinen Eingang finden.

Die Lagrange Relaxierung mit Subgradientenverfahren ist bei kleinen Instanzen nur bedingt erfolgreich. Die unteren Schranken sind nicht sonderlich befriedigend. Das beste Ergebnis konnte bei Scenario7 mit 39,41% unter dem Optimum erreicht werden. Im Gegensatz dazu sind die oberen Schranken (gültige Lösungen) relativ gut – zwischen 5 und 6 % über dem Optimum. Die Resultate des Subgradientenverfahrens zeigt Tabelle 7.3.

Bessere Schranken als das Subgradientenverfahren liefert die Lagrange Relaxierung mit Volume Algorithmus. Die unteren Schranken sind deutlich näher beim Optimum – mit nur ungefähr 13 bis hin zu 5% unter dem Optimum. Bei den oberen Schranken konnte in zwei der drei Fälle sogar das

Tabelle 7.3: Ergebnisse der SWIS-Testfälle bei der Lagrange Relaxierung mit Subgradientenverfahren

Szenarioname	untere S.	obere S.	Zeit [s]	% unter Opt.	% über Opt.
Semantic	2,69936	36	0	92,06	5,88
Scenario7	21,2063	37	1	39,41	5,71
Complex	34,1963	117	39	69,19	5,41

Optimum erreicht werden. Eine Übersicht über die Ergebnisse für dieses Verfahren bietet Tabelle 7.4.

Tabelle 7.4: Ergebnisse der SWIS-Testfälle bei der Lagrange Relaxierung mit Volume Algorithmus

Szenarioname	untere S.	obere S.	Zeit [s]	% unter Opt.	% über Opt.
Semantic	30,0483	36	3	11,62	5,88
Scenario7	33,3891	35	4	4,60	0
Complex	97,519	111	17	12,15	0

Mit der Spaltengenerierung konnten durchwegs gute Ergebnisse erzielt werden (siehe Tabelle 7.5). Zwei der drei Testfälle wurden optimal gelöst. Die unteren Schranken – diese entstehen durch das LP-relaxierte Lösen des Problems – sind mit 12 bis 4 % unter dem Optimum ganz akzeptabel. Leider wurde bei einem der Testfälle keine gültige Lösung gefunden, da aus den in der LP-relaxierten Lösung vorkommenden Pfaden keine gültige Lösung konstruiert werden kann. Bei dieser Art der Umsetzung – Konstruktion der Lösung nur aus den in der "LP-relaxierten Pfaden" – besteht diese Möglichkeit. Wie die durchgeführten Experimente zeigten, kommt dieser Fall sehr selten vor. Eine Möglichkeit wäre nicht nur die in der LP-relaxierten Lösung vorkommenden Pfade zu betrachten, sondern alle gefundenen Pfade. Dies wäre insbesonderes bei großen Instanzen nicht mehr praktikabel, weshalb in dieser Diplomarbeit keine weiteren Schritte in diese Richtung unternommen werden.

Als letztes Verfahren wurde die Primale Heuristik angewandt. Sie konnte bei allen drei Testfällen gültige Lösungen finden, welche mit 3 – 26% über dem Optimum durchaus zufriedenstellend sind. Die Ergebnisse der Primalen

Tabelle 7.5: Ergebnisse der SWIS-Testfälle bei der Spaltengenerierung

Szenarioname	untere S.	obere S.	Zeit [s]	% unter Opt.	% über Opt.
Semantic	30,0952	–	0	11,48	–
Scenario7	33,7036	35	0	3,7	0
Complex	98,4554	111	1	11,3	0

Heuristik sind in Tabelle 7.6 zu finden.

Tabelle 7.6: Ergebnisse der SWIS-Testfälle bei der Primalen Heuristik

Szenarioname	obere Schranke	Zeit [s]	% unter opt
Semantic	36	0	5,88
Scenario7	36	0	2,86
Complex	140	0	26,13

Die Berechnungsdauer für das exakte Lösen mittels CPLEX liegt mit unter 2 Sekunden deutlich unter der Dauer des Enumerationsverfahren im SWIS Projekt, welches 2 Wochen benötigte. Auch die anderen Verfahren kamen erwartungsgemäß sehr schnell zu ihren Resultaten. Das teilweise schlechtere Abschneiden bei diesen kleinen Testfällen ist nicht überraschend, da die heuristischen Methoden vor allem für den Einsatz bei größeren Instanzen gedacht sind.

Um diese Verfahren an größeren Testfällen zu evaluieren werden Testdaten generiert. Der hierfür verwendete Testdatengenerator wird im nächsten Abschnitt beschrieben.

7.2 Testdatengenerator(en)

Neben den Testfällen aus dem Projekt SWIS sollen generierte Daten eine Evaluation der implementierten Verfahren ermöglichen. Anfänglich wurde ein Testdatengenerator verwendet, der ausgehend von einer angegebenen Anzahl von Knoten und Kanten ein Netzwerk durch Einfügen der Kanten zwischen jeweils zwei zufälligen Knoten erzeugt. Dabei konnte für sehr viele Transporte kein Weg gefunden werden, da es keine Verbindungen zwischen den

Tabelle 7.7: Aufrufoptionen für den Datengenerator

Option	**Parameter**	**Erklärung**
-h		gibt eine Hilfe zu den Optionen aus
-i	ID	verwendet eine vordefinierte Konfiguration ID = {F,G}
-s	#VERBINDUNGEN	Anzahl der sicheren Verbindungen
-u	#VERBINDUNGEN	Anzahl der unsicheren Verbindungen
-n	#KNOTEN	Anzahl der Knoten
-t	#TRANSPORTE	Anzahl der Transporte

jeweiligen Start und Endknoten gab. Daher wurde eine andere Möglichkeit zur Testdatengenerierung gewählt.

Mit Hilfe von *LEDA – The Library of Efficient Data Types and Algorithms* [Alg08] werden daher zwei zusammenhängende Graphen mit der gleichen Knotenmenge generiert – einer für sicher und einer für unsicher Verbindungen. Durch Verwendung dieser Kanten wird gewährleistet, dass jeder Knoten mit jedem anderen verbunden ist. In Folge sind sichere und unsichere Transporte von jedem beliebigen Knoten zu jedem anderen (aus Sicht der Verbindung) möglich. Trotz dieser Herangehensweise werden durch solch einen Testdatengenerator nicht nur gültige Testdaten erzeugt. So wird beispielsweise nicht kontrolliert, ob für jeden Transport ein gültiger Weg mit genügend Kapazitäten und dem begrenzten Delay existiert. In Folge dessen ist es nicht möglich für alle Transporte gültige Wege zu garantieren. Eine Möglichkeit hierbei leitend einzugreifen ist die Festsetzung der Eigenschaften der Kanten und Transporte, wie zum Beispiel Kapazitäten, Delay und Kosten.

Hierbei wurde jeweils eine begrenzte Menge an Möglichkeiten verwendet, da etwa bei Netzwerkverbindungen Kapazitäten von 10MBit/s, 100MBit/s und 1TBit/s möglich sind und nicht jeder beliebige Wert. Für jede Eigenschaft werden 3 Werte angegeben, die durch eine bestimmte Gewichtung in den resultierenden Daten auftreten.

Der Testdatengenerator wird durch *./datagenerator [Options] Network-OutputFile Transport-OutputFile* aufgerufen. Eine Aufzählung der möglichen Optionen wird in Tabelle 7.7 angegeben.

Tabelle 7.8: Konfigurationstabelle für den Datengenerator

Konfiguration	F	G
Kosten für Verbindung	{10, 20, 40}	{1, 2, 4}
Delay der Verbindung	{1, 2, 4}	
Kapazitäten der Verbindungen	{10, 100, 1000}	
maximaler Delay der Transporte	{30, 50, 70}	
Kapazitäten der Transporte	{1, 2, 5}	

7.3 Testinstanzen

Mithilfe des Testdatengenerators wurden Testsets generiert, um die implementierten Ansätze miteinander vergleichen zu können. Dem Testdatengenerator werden jeweils die Anzahl der Knoten, der sicheren und unsicheren Kanten und die Anzahl der Transporte übergeben. Außerdem wird eine Konfiguration (F oder G) angegeben. Diese Konfiguration gibt mögliche Kapazitäten der Verbindungen, deren Kosten und deren Delay sowie mögliche Werte für die Kapazitäten der Transporte und deren Delay an. Der Unterschied zwischen den beiden Konfigurationen liegt im Verhältnis der Kosten für die Verwendung einer Verbindung zu den Protokollkosten. Während in der Konfiguration F für die generelle Verwendung einer Kante zehnfach so viele Kosten anfallen wie für das Protokoll, wird bei Konfiguration G aus dem selben Set an Werten gewählt. Tabelle 7.8 gibt die verwendeten Sets an Möglichkeiten für jede Größe an. Der erste und dritte Wert wird jeweils mit einer Wahrscheinlichkeit von 25% gewählt, die mittlere mit 50%.

Um unterschiedliche Netzwerkgrößen zu testen, wird die Anzahl der Knoten variiert. Getestet wird jeweils mit 25, 30, 40, 50, 100, 500 und 1000 Knoten. Weiters wird festgelegt wie dicht das Netzwerk ist, das heißt wie groß das Verhältnis von Kanten zu Knoten ist. Es wurden jeweils mit den Faktoren 3, 5 und 10 zur Anzahl der Knoten, die Anzahl der sicheren und der unsicheren Kanten bestimmt. In den Testdaten werden jeweils 1000 Transporte generiert, getestet wird dann mit den ersten 100, 200 und allen 1000 Transporten. Tabelle 7.9 gibt einen Überblick über alle verwendeten Parameter.

Die durch den Testdatengenerator erzeugten Testfälle müssen, wie schon in Kapitel 7.2 angeführt, keine gültigen Lösungen aufweisen. Um solche Testfälle auszumustern, muss bewiesen werden, dass keine gültige Lösung existiert. Hierfür wurde zunächst versucht für jeden Testfall eine valide Lösung (siehe Kapitel 5.4) zu finden. Erzeugt die Primale Heuristik keine gültige Lösung, wird versucht das Problem einzugrenzen. Existiert schon für ein Sub-

Tabelle 7.9: Verwendete Parameter bei der Testfallerzeugung

	mögliche Werte
Konfiguration	F, G
Knoten	25, 30, 40, 50, 100, 500, 1000
Verbindungen un-/sichere (Faktoren zu Knotenanzahl)	3, 5, 10
Transporte	100, 200, 1000

set der Transporte keine gültige Lösung, so ist das Problem auch mit allen Transporten unlösbar, da durch das Hinzufügen eines weiteren Transportes keine Kapazitäten frei werden – ganz im Gegenteil. Eine kleinere Menge an Transporten kann in praktikabler Zeit direkt mit CPLEX exakt gelöst werden.

Solch ein kleineres Set an Transporten kann mit der Primalen Heuristik ausgemacht werden, welche Transporte für die kein Weg gefunden werden kann ausgibt. Für alle Testfälle die nicht mit der Primalen Heuristik lösbar waren, konnte ein begrenztes Set an Transporten gefunden werden. Für diese Testfälle wurde jeweils mit CPLEX bewiesen, dass sie keine gültige Lösung aufweisen. Anschließend wurden für nicht lösbare Testfälle mit dem Testdatengenerator neue generiert.

7.4 Testsettings

Durch die Kombination der angegebenen Einstellung – Konfigurationen F oder G; 25, 30, 40, 50, 100, 500 oder 1000 Knoten; Faktor 3, 5 oder 10 für die Kanten; 100, 200 oder 1000 Transporte – wird ein Testset mit $2 \cdot 7 \cdot 3 \cdot 3 = 126$ Testfällen erzeugt. Zu jeder Parameterkombination werden drei Testinstanzen generiert. Als Testrechner dient ein 2 x Dual-Core AMD Opteron(tm) Processor 2214 mit 4GB RAM. Jeder Testfall wird mit jedem der fünf Algorithmen für jeweils maximal 1000 Sekunden getestet.

7.4.1 Parameter-Settings der Algorithmen

Im Folgenden werden die verwendeten Parameter bei den Algorithmen beschrieben. Beim direkten Lösen mit CPLEX wurden die Standardeinstellungen von CPLEX beibehalten. Bei der Lagrange Relaxierung mussten genauere Einstellungen vorgenommen werden.

Lagrange Relaxierung mit Subgradientenverfahren

Bei der Lagrange Relaxierung mit Subgradientenverfahren werden die Parameter und Abbruchbedingungen wie in [Bea93] empfohlen gewählt. Der Parameter π wird mit 2 initialisiert und N mit 30. Wenn für N Iterationen keine Verbesserung eintritt, wird π halbiert. Als Minimalwert für π wird 0,005 verwendet. Ist dieser Wert erreicht werden die Iterationen gestoppt. Außerdem wird das Verfahren nach dem Auffinden einer optimalen Lösung oder bei Überschreitung des Zeitlimits abgebrochen. Nach dem Vorschlag in [Bea93] wird Z_{UB} bei der Berechnung von

$$T = \frac{\pi \cdot (Z_{UB} - Z_{LB})}{\sum_i G_i^2}$$

durch $Z_{UB} \cdot 1{,}05$ ersetzt.

Lagrange Relaxierung mit Volume Algorithmus

Für die Lagrange Relaxierung mit Volume Algorithmus werden die Werte für α und die Multiplikationsfaktoren für die einzelnen Phasen Rot, Gelb und Grün wie in [BA00] vorgeschlagen gewählt. In einer grünen Iteration wird π mit 1,1 multipliziert und nach 20 aufeinander folgenden roten Iterationen π mit 0,66 multipliziert. Als Startwert für α_{max} wird 0,1 gewählt. Wenn 500 Iterationen die Lösung nicht verbessern, wird wie in [HS06] beschrieben das iterative Verfahren abgebrochen. Außerdem findet wie vorgeschlagen nach 250 Iterationen ein Test statt, ob in dieser Zeit die untere Schranke weniger als 1% gestiegen ist. Tritt dieser Fall ein wird α_{max} halbiert.

7.5 Ergebnisse des direkten Lösens mittels CPLEX

Bei den durchgeführten Experimenten mit CPLEX zeigt sich, welchen Einfluss die Größe des Netzwerks und die Anzahl der zu routenden Transporte haben. Je mehr Knoten, Kanten oder Transporte der Testfall enthält, desto seltener konnte CPLEX in der vorgegebenen Zeit eine Lösung finden. Das Diagramm 7.5.1 zeigt diesen Zusammenhang am Beispiel der Knotenanzahl. Es wird dargestellt wie viel Prozent der Testfälle mit der jeweiligen Anzahl an Knoten mit CPLEX optimal gelöst wurden. Die beiden anderen Farben stellen dies für die obere und untere Schranke dar. Wie im Diagramm zu erkennen ist, wurden nur bei den Testfällen mit bis zu 50 Knoten optimale Lösungen gefunden. Bei einer Knotenanzahl von 100 können zumindest

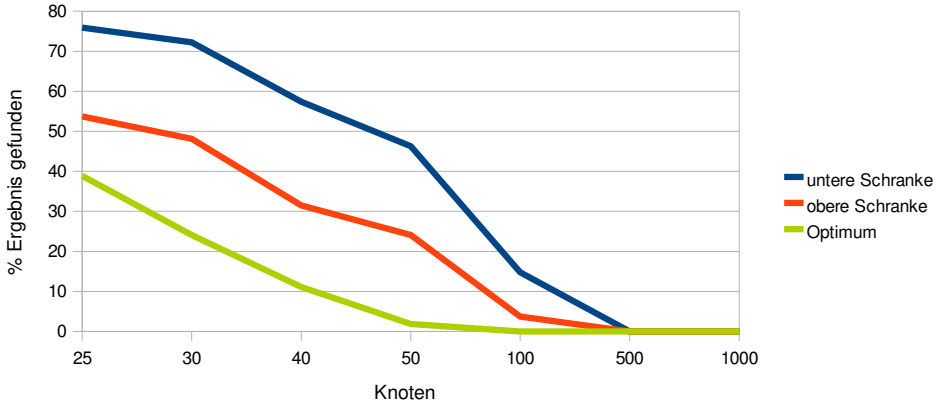

Abbildung 7.5.1: Prozentuale Darstellung der durch CPLEX gelösten Testfälle nach der Anzahl der Knoten gegliedert

noch gelegentlich Schranken gefunden werden. Darüber konnte das direkte Lösen durch CPLEX keine Resultate liefern. Sehr deutlich ist dadurch zu erkennen, dass das direkte Lösen durch CPLEX nur bei kleineren Netzwerken zum Erfolg führt.

Während in Abbildung 7.5.1 ausschließlich der Einfluss der Knotenanzahl dargestellt wird, stellt Abbildung 7.5.2 die optimal gelösten Testfälle noch etwas konkreter dar. Es wird die Transportgröße der jeweiligen Testfälle gezeigt. Durch diese Ansicht zeigt sich, dass neben der Größe des Netzwerkes

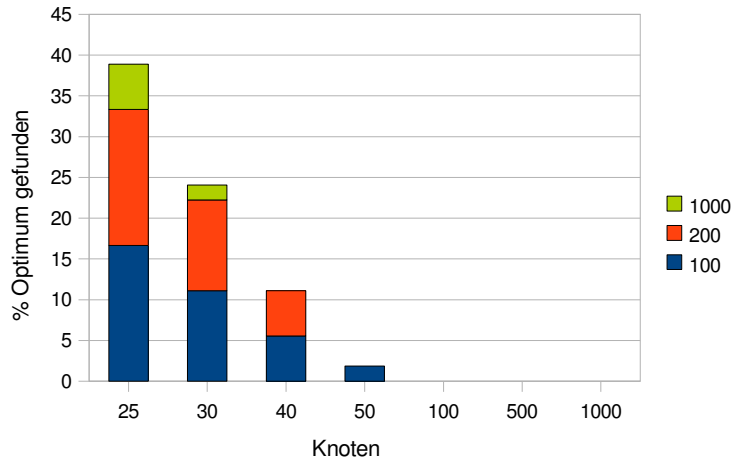

Abbildung 7.5.2: Durch CPLEX optimal gelöste Testfälle nach der Anzahl der Knoten und Transporte gegliedert

auch die Anzahl der Transporte einen entscheidenden Einfluss hat. Testfälle mit 1000 Transporten konnten nur in kleineren Netzwerken mit bis zu 30

Knoten in der beschränkten Zeit gelöst werden. Mit 200 Transporten ist dies noch bis zu 40 Knoten möglich.

Abgesehen von der Komplexität des Netzwerkes und der Anzahl der Transporte hat die Konfiguration großen Einfluss auf das Finden einer Lösung mit CPLEX. Abbildung 7.5.3 zeigt diesen Aspekt durch Aufgliederung in die

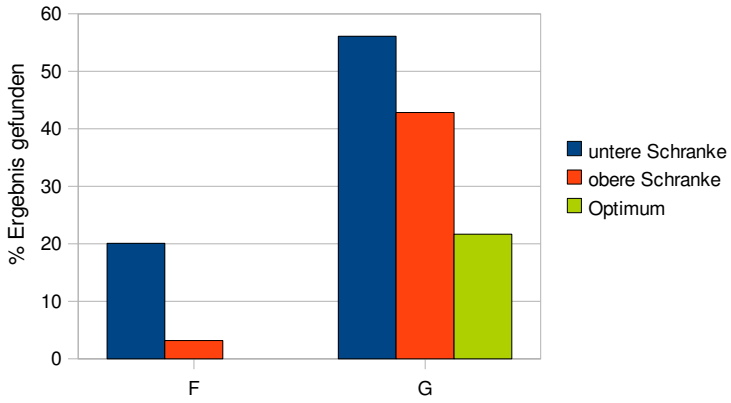

Abbildung 7.5.3: Prozentuale Darstellung der durch CPLEX gelösten Testfälle nach Konfiguration gegliedert

beiden Konfigurationen F und G. Während mit Konfiguration F kein Testfall optimal gelöst wurde, sind es bei G über 20%. Auch die Anzahl der gefunden Schranken ist höher.

Durch ausführlichere Tests mit zusätzlichen Transportgrößen (400, 600, 800) wird ein genaueres Bild vom Einfluss der Anzahl der Transporte und der Netzwerkgröße auf die Lösbarkeit gewonnen. In Abbildung 7.5.4 kann anhand dieses Zusammenhangs festgestellt werden, dass die Netzwerkgröße einen wesentlich höheren Einfluss auf die Lösbarkeit hat als die Anzahl der Transporte. Während eine Verdoppelung der Transporte nur einen geringen Unterschied erkennen lässt, sinkt durch die Verdopplung der Knotenanzahl die Lösbarkeit drastisch.

In weiteren Testläufen wurde die Laufzeit des Algorithmus auf einen Tag erweitert. Dies wäre für das Designproblem bei großen Netzwerken eine praktikable Zeitdauer. Es hat sich gezeigt, dass mit Konfiguration F selbst bei den kleinen Netzwerken mit 25 Knoten und 100 Transporten auch nach einem Tag noch keine optimale Lösung gefunden werden konnte. Anders sieht es bei Konfiguration G aus, hier konnten bis zu einem Netzwerk von 40 Knoten alle Testfälle mit 1000 Transporten in einem Tag gelöst werden.

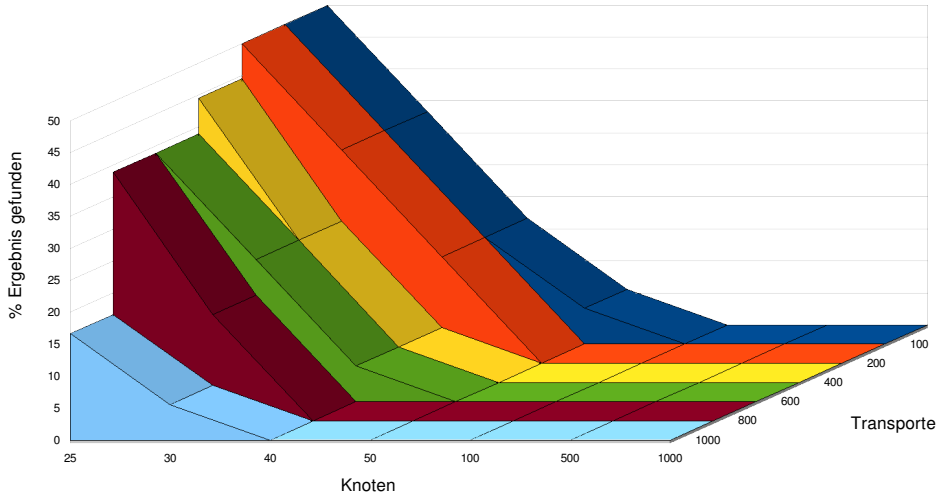

Abbildung 7.5.4: Optimal durch CPLEX gelöste Testfälle aufgegliedert nach Knoten und Transporten

7.6 Ergebnisse der Lagrange Relaxierung

Durch den Einsatz einer Lagrange-Heuristik (Kapitel 4.3) konnte in nahezu allen Fällen neben einer unteren Schranke auch eine obere Schranke erzeugt werden. Nur für weniger als 2% der Testfälle fand das Programm beim Einsatz des Subgradientverfahrens keine obere Schranke, bei Verwendung des Volume Algorithmus waren es etwas mehr als 5%. Hinsichtlich der Qualität der Schranken die in der vorgegebenen Zeit aufgestellt werden, zeigt sich eine deutliche Abhängigkeit von der Größe der Testfälle. In Abbildung 7.6.1 wird dieser Aspekt hinsichtlich der Anzahl der Knoten im Netzwerk aufgezeigt. Es wird der relative Abstand zwischen oberer und unterer Schranke bezüglich oberer Schranke – im Weiteren als *Gap* bezeichnet – prozentuell angegeben. Neben dem Mittelwert des Gaps aller Testfälle mit der jeweiligen Knotenanzahl ist auch die Standardabweichung aus der Abbildung ersichtlich. Außer der Netzwerkgröße hat auch die Konfiguration großen Einfluss auf die Lösungsqualität in der vorgegebenen Zeit. Über all diese Unterscheidungen hinweg liefert der Volume Algorithmus deutlich bessere Resultate als das Subgradientenverfahren. Lediglich einmal wurde durch das Subgradientenverfahren eine bessere untere Schranke erzielt als beim Volume Algorithmus. Allerdings schnitt das Subgradientenverfahren beim Generieren einer obere Schranke in 22,47% der Testfälle besser ab.

Interessant ist neben den Vergleichen der Schranken zu einem bestimmten Zeitpunkt auch die Entwicklung dieser über eine bestimmte Zeitspanne, insbesondere wie sich die beiden Verfahren unterscheiden. In Abbildung 7.6.2

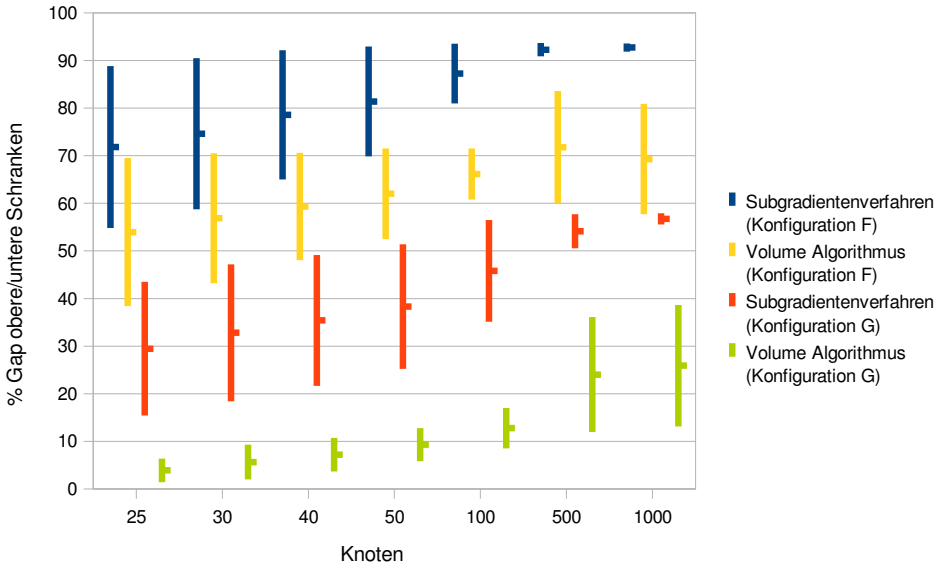

Abbildung 7.6.1: Zusammenhang zwischen Anzahl der Knoten und der Konfiguration hinsichtlich Lösungsgap bei der Lagrange Relaxierung

ist diese Entwicklung anhand eines Netzwerkes mit 1000 Knoten über welches 100 Transporte befördert werden sollen dargestellt. In den ersten 1500 Sekunden ist kein Unterschied zwischen den beiden unteren Schranken zu erkennen. In dieser Zeit liefert jedoch das Subgradientenverfahren bessere obere Schranken als der Volume Algorithmus. Erst bei einer Betrachtung

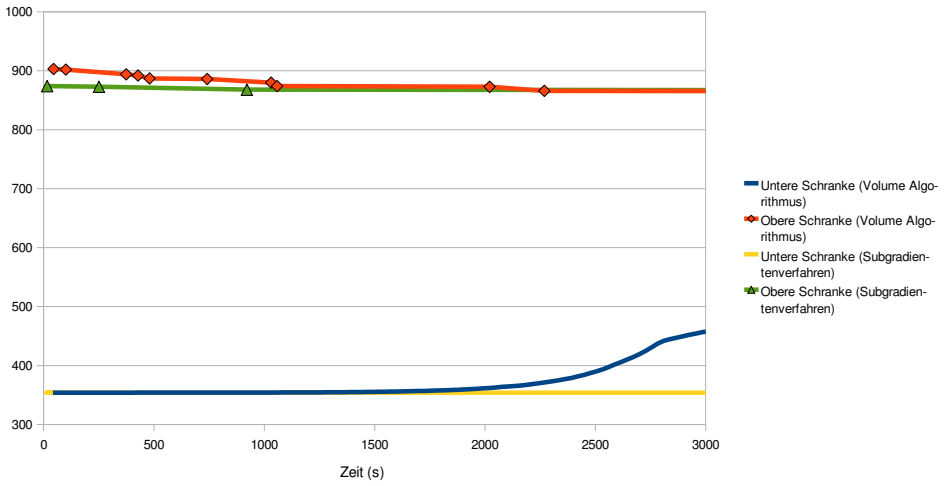

Abbildung 7.6.2: Entwicklung der Schranken bei der Lagrange Relaxierung am Beispiel eines großen Netzwerkes (1000 Knoten) in kurzer Zeit

über eine noch längere Zeitspanne wird das generelle bessere Abschneiden des Volume Algorithmus verständlich. Während das Subgradientenverfahren noch immer kontinuierlich leicht bessere Schranken liefert. Konvergiert der Volume Algorithmus wesentlich schneller. Den weiteren Verlauf der Schranken zeigt Abbildung 7.6.3 über eine deutlich größere Zeitspanne als in der vorherigen Abbildung. Der Vergleich der Entwicklung der Schranken bei un-

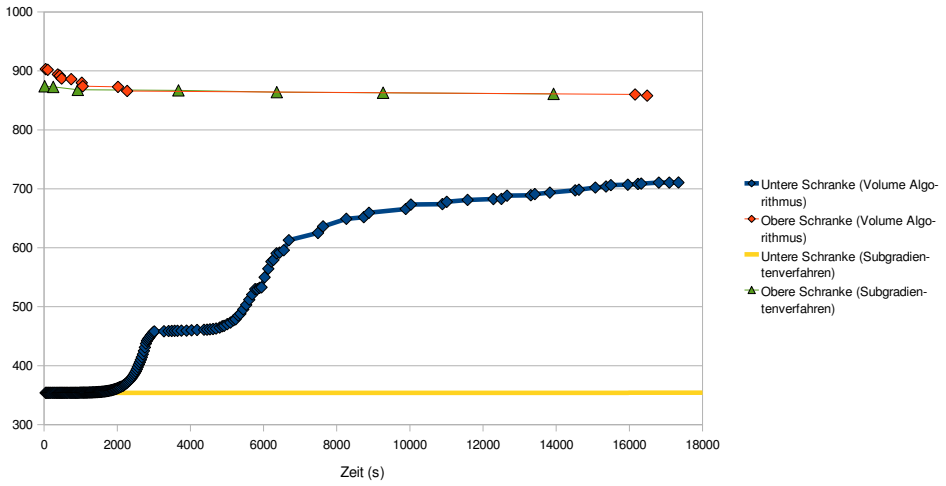

Abbildung 7.6.3: Entwicklung der Schranken bei der Lagrange Relaxierung am Beispiel eines großen Netzwerkes (1000 Knoten) über eine längere Zeitspanne

terschiedlichen Konfigurationen – jeweils mit der gleichen Netzwerkgröße von 25 Knoten – in Abbildung 7.6.4 und 7.6.5 gibt eine Erklärung, warum der Gap vor allem bei Konfiguration F so groß ist. Der Abstand zwischen dem Optimum und der oberen Schranke ist bei Konfiguration F wesentlich größer als bei Konfiguration G. Dadurch wird deutlich, dass der große Gap nicht etwa auf eine schlechte untere Schranke durch die Lagrange Relaxierung zurückzuführen ist, sondern auf die schlechtere obere Schranke. Dieser Unterschied zwischen den Konfigurationen kommt von den unterschiedlichen Designkosten – sie sind bei Konfiguration F durchschnittlich zehnmal so groß wie bei Konfiguration G.

Es stellt sich natürlich die Frage, wie gut die Lagrange-Heuristik ist, beziehungsweise wie oft sie gültige Lösungen findet. Vor allem ist interessant, ob der Gesamtdelay eventuell auch in der Heuristik beachtet werden sollte. Bei den Tests zeigte sich eine große Abhängigkeit der Anzahl der gefundenen Lösungen vom maximalen Gesamtdelay. Wird dieser relativ klein gewählt, so werden durch die Lagrange-Heuristik seltener Lösungen gefunden, wohingegen bei großem Gesamtdelay die Lagrange-Heuristik nie Probleme hat

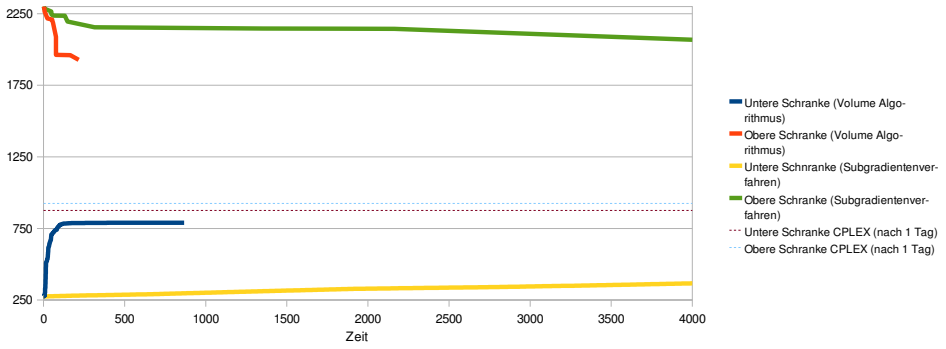

Abbildung 7.6.4: Entwicklung der Schranken bei der Lagrange Relaxierung bei einem kleinen Netzwerk (25 Knoten) der Konfiguration F

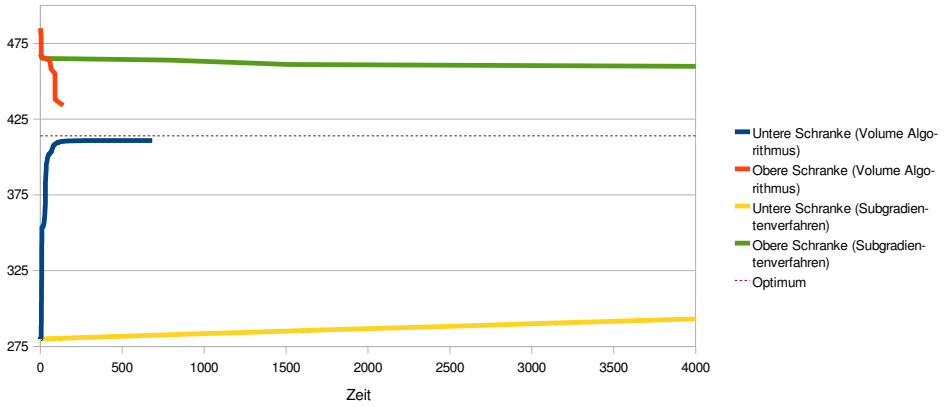

Abbildung 7.6.5: Entwicklung der Schranken bei der Lagrange Relaxierung bei einem kleinen Netzwerk (25 Knoten) der Konfiguration G

eine Lösung zu finden. Bei relativ kleinem maximalen Gesamtdelay liegt die Anzahl der durch die Lagrange-Heuristik behandelten Ergebnisse bei denen wegen dem Gesamtdelay keine Lösung gefunden werden konnte im Bereich von 25-40%.

Eine Adaption der besprochenen Lagrange-Heuristik könnte nun diesem Umstand Rechnung tragen und auch den maximalen Gesamtdelay beachten. Da jedoch in vielen Fällen Lösungen gefunden werden und eine in diese Richtung angepasste Version durch den oftmaligen Aufruf sehr rechenintensiv wäre, wird davon Abstand genommen.

7.7 Ergebnisse der Spaltengenerierung

Nur bei 24,5% der Testfällen lieferte die Spaltengenerierung ein Ergebnis (abgesehen von dem der Startlösung). Es zeigt sich sowohl bei der Netzwerkgröße, als auch bei der Verbindungsdichte und der Transportanzahl ein Gefälle in der Lösbarkeit. Je größer der Testfall, umso geringer die Anzahl der gelieferten Lösungen. Als Beispiel für diese Beobachtung gibt Abbildung 7.7.1 eine Übersicht über die gelösten Testfälle durch Spaltengenerierung in Abhängigkeit von der Anzahl der Knoten und der Transporte. Mit steigender Knotenanzahl sinkt die Anzahl der gelösten Testfälle. Außerdem spielt die Transportgröße eine Rolle. So konnten 1000 Transporte nur in den kleinen Netzwerken mit einer Knotenanzahl von 25 und 30 gelöst werden, während bei kleinere Transportmengen noch bis zu einer Netzwerkgröße von 100 Knoten Lösungen gefunden wurden. Für große Netzwerke mit 500 und 1000 Knoten kam die Spaltengenerierung in der vorgegebenen Zeit nie zu einem Ergebnis. Die Konfiguration der Testfälle spielt auch eine Rolle bei der Lösbarkeit.

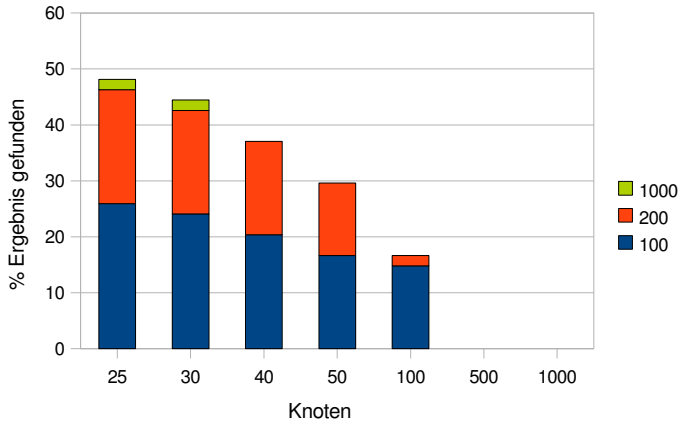

Abbildung 7.7.1: Durch Spaltengenerierung gelöste Testfälle je nach Anzahl der Knoten aufgegliedert nach der Anzahl der Transporte

Diesen Aspekt zeigt Abbildung 7.7.2. Zusätzlich zur Konfiguration wird in dieser nach der Anzahl der Transporte aufgegliedert. Somit ist sichtbar, dass nur bei Konfiguration G Testfälle mit 1000 Transporten lösbar waren. Weniger als 10% der Testfälle mit Konfiguration F konnten gelöst werden. Dies ist auf die bei Konfiguration F gegenüber Konfiguration G erhöhten Verbindungskosten zurückzuführen.

Das Gros der in der festgelegten Zeit unlösbaren Testfälle kommt nicht über den iterativen Vorgang des LP-relaxierten Lösens und dem Hinzufügen von Variablen mit reduzierten Kosten hinaus. Lediglich bei einem Testfall wurde zwar der Vorgang des iterativen Hinzufügens von Spalten beendet,

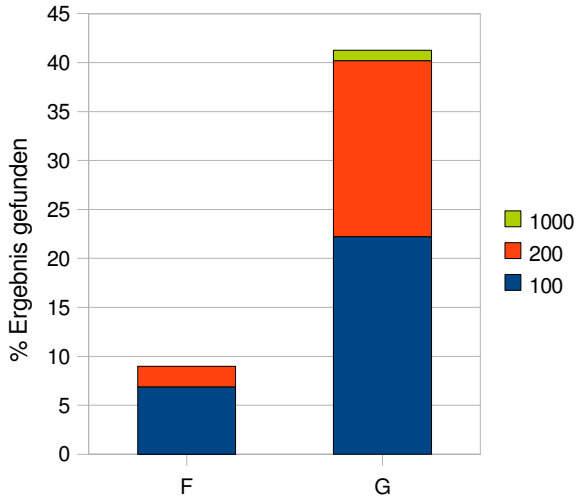

Abbildung 7.7.2: Durch Spaltengenerierung gelöste Testfälle je nach Konfigurationen und Transportanzahl

jedoch daraus noch keine obere Schranke in der vorgegebenen Zeit gefunden. Der Teil des Findens einer Lösung aus der LP-relaxierten Lösung ist demnach nicht der kritische Part. Es werden sehr viele Iterationen benötigt bis keine Variablen mit reduzierten Kosten mehr gefunden werden können. Dies ist ein bekanntes Problem bei großen ILPs wie Amor und Desrosiers beschreiben:

> *"Difficulties arise in column generation/cutting plane algorithms when they are used to solve large-scale degenerate problems. It is well accepted that primal degeneracy, among other factors, is responsible for the tail effect of column gernation. Moreover, instability in the behavior of the values of the dual variables becomes more frequent when problems get larger."*[AD06]

Eine Stabilisierung der Spaltengenerierung wie in [AD06, ADdC06] diskutiert, sieht auch für unser Problem vielversprechend aus, um den Lösungsprozess zu verkürzen und dadurch öfter Resultate zu erhalten. Dies sprengt jedoch den Umfang dieser Arbeit, kann allerdings als vielversprechender Aspekt etwaiger weiterführenden Arbeiten angesehen werden.

7.8 Ergebnisse für die Primale Heuristik

Für die Spaltengenerierung wird als Ausgangslösung eine gültige Startlösung benötigt (oder es muss zumindest eine Lösung geben, wenn die Ganzzahligkeitsbedingung weggelassen wird). Da das schnelle Finden (irgend-)einer

Lösung selbst ein nicht-triviales Problem darstellt, wird das Erzeugen einer gültigen Lösung gesondert betrachtet. Wie in Kapitel 5.4 besprochen, wird nicht notwendigerweise mit der dort beschriebenen Primalen Heuristik eine gültige Lösung erzeugt. Bei den behandelten Testfällen konnten nur 1,32% (5 Testfälle) durch dieses Verfahren nicht gelöst werden. Lediglich bei einem dieser Testfälle wurde mit einem anderen Verfahren eine gültige Lösung gefunden – mit der Spaltengenerierung. Die Anzahl der gefundenen Lösungen und der Testfällen bei denen keine gefunden wurde, zeigt Abbildung 7.8.1. Bei genauerer Betrachtung der fehlgeschlagenen Testfälle zeigte sich, dass der (willkürlich gewählte) maximale Gesamtdelay das Hindernis für die Erzeugung einer gültigen Lösung war.

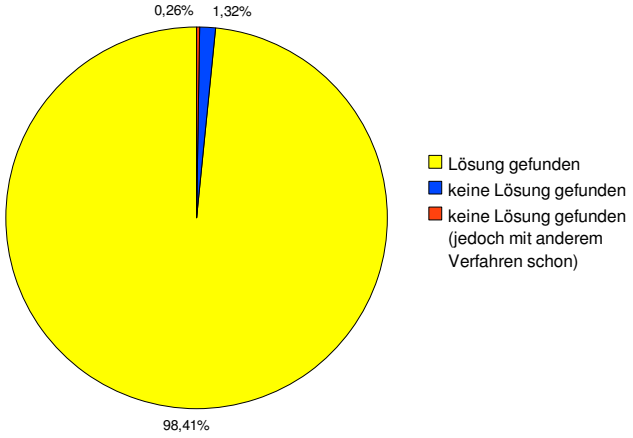

Abbildung 7.8.1: Durch die Primale Heuristik gefundene Lösungen

Da mit dieser Heuristik sehr schnell Lösungen gewonnen werden – durchschnittlich in 1,12 Sekunden und längstens in 17 Sekunden –, stellt sich die Frage nach der Güte der durch diese heuristischen Lösungen gewonnenen oberen Schranken. Als einzige Vergleichsmöglichkeit steht die beste obere Schranke aller auf den Testfall angewandter Verfahren zur Verfügung. In Abbildung 7.8.2 werden die gefundenen validen Lösungen mit denen der jeweils besten gefundenen (aus allen fünf Verfahren) verglichen. Durchschnittlich liegt diese gefundene Lösung 15,1% über der besten oberen Schranke. Die Kurve ist erwartungsgemäß fallend, da mit zunehmender Größe des Netzwerkes die Qualität der besten oberen Schranke durch das Zeitlimit abnimmt. Außerdem sind bei den größeren Testfällen häufiger die durch dieses Verfahren erzeugten validen Lösungen die besten oberen Schranken. Nichtsdestotrotz zeigt die Kurve, dass mit dieser schnellen Methode gute Ergebnisse erreicht werden können. Ein durchschnittlicher Gap von 15,15% (Standardabweichung von 8,85%) bei 25 Knoten mit Konfiguration G (bei diesen Testfällen liefer-

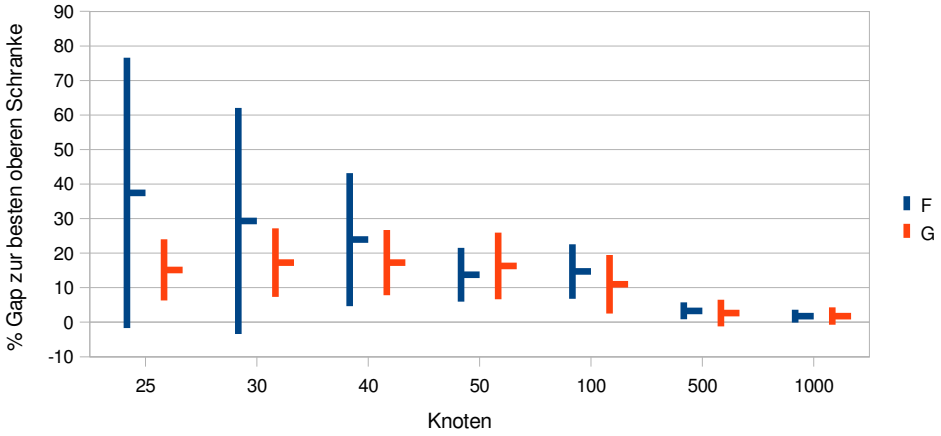

Abbildung 7.8.2: Primale Heuristik im Vergleich zur besten oberen Schranke nach Konfiguration; Mittelwert und Standardabweichung

ten die anderen Verfahren ausreichend gute Lösungen in der vorgegebenen Zeit, um einen guten Vergleichswert abzugeben) ist ein guter Wert für ein so schnelles Verfahren. Gegenüber den anderen in dieser Diplomarbeit besprochenen Verfahren skaliert die Primale Heuristik sehr gut. Probleme ergeben sich allerdings bei einem kleinen maximalen Gesamtdelay, da bei dieser einfachen Heuristik nur im verbleibenden Netzwerk (ohne die schon an Transporte vergebenen Kapazitäten) nach neuen zeitlich kürzeren Wegen für jeden Transporte gesucht wird. Aus diesem Grund konnten wie beschrieben nicht für alle Testfälle Lösungen durch die Primale Heuristik gefunden werden.

7.9 Vergleich der Resultate der fünf Verfahren

In diesem Abschnitt sollen nun die Resultate der verschiedenen angewandten Verfahren verglichen werden. Zuerst werden die Ergebnisse bezüglich der Erzeugung einer unteren Schranke beleuchtet. Danach werden die Lösungen hinsichtlich oberer Schranke erläutert. Abschließend werden sowohl untere als auch obere Schranken zusammenfassend betrachtet.

7.9.1 Vergleich bezüglich unterer Schranken

Zum Ermitteln der unteren Schranke wurden vier Verfahren angewandt: direktes Lösen des ILPs mit CPLEX, Lagrange Relaxierung mit Subgradientenverfahren, Lagrange Relaxierung mit Volume Algorithmus und Spalten-

generierung. Bei der Spaltengenerierung ist das Ergebnis des LP-relaxierten Lösens nach Hinzufügen aller Variablen mit reduzierten Kosten eine untere Schranke für das ursprüngliche Problem. In den folgenden vergleichenden Abbildungen ist dargestellt, welches Verfahren jeweils die beste untere Schranke erzeugte – wurde die gleiche von mehreren Verfahren gefunden, so erzeugte das schnellere Verfahren die beste Schranke. Einen Gesamtüberblick über den Erfolg der einzelnen Verfahren bei der Berechnung einer unteren Schranke zeigt Abbildung 7.9.1. Diese zeigt, dass alle Methoden in bestimmten Fällen die beste untere Schranke liefern konnten. Es werden die meisten unteren Schranken mit CPLEX oder Lagrange Relaxierung mit Volume Algorithmus erzeugt. Spaltengenerierung und Lagrange Relaxierung mit Subgradientenverfahren liefern nur selten die beste untere Schranke.

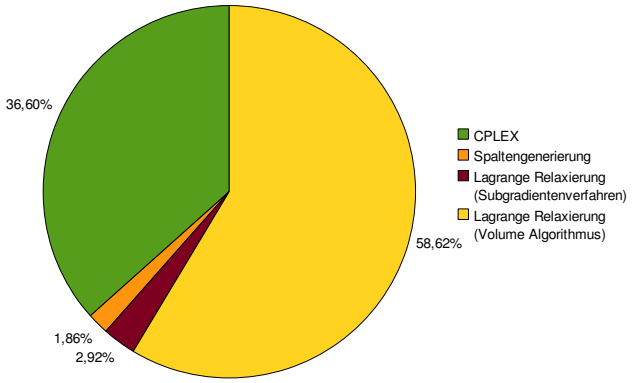

Abbildung 7.9.1: Verfahren mit der besten untere Schranke

Im Folgenden sollen nach dieser etwas allgemeinen Darstellung verschiedene Aspekte genauer betrachtet werden. Zunächst zeigt die Aufteilung nach Konfigurationen in Abbildung 7.9.2, wie CPLEX bei Konfiguration G die meisten Lösungen liefert. In Anbetracht der in Kapitel 7.5 präsentierten Resultate – es wurden wesentlich mehr Testfälle mit Konfiguration G durch CPLEX gelöst – ist dies nicht verwunderlich. Wurde ein Testfall mit CPLEX gelöst, so ist dies immer auch die beste untere Schranke. In 1,06% der Testfälle konnte die Spaltengenerierung immerhin das gleiche Resultat für die untere Schranke erzielen. Die Spaltengenerierung selbst liefert sehr selten Lösungen – wie auch in diesem Fall bei der Aufgliederung nach Konfigurationen sichtbar. Dies ist nicht etwa auf schlechte untere Schranken bei der Spaltengenerierung zurückzuführen. (So entspricht im Gegenteil die Lösung der Spaltengenerierung in einigen Fällen der durch CPLEX gefundenen). Als Hauptgrund für das “schlechte” Abschneiden der Spaltengenerierung ist ihr seltenes Terminieren – nur 25,4% der Testfälle ließen sich in der vorgegebe-

nen Zeit lösen. Die meisten für CPLEX unlösbaren Testfälle wurden durch die Lagrange Relaxierung mit Volume Algorithmus am Besten gelöst.

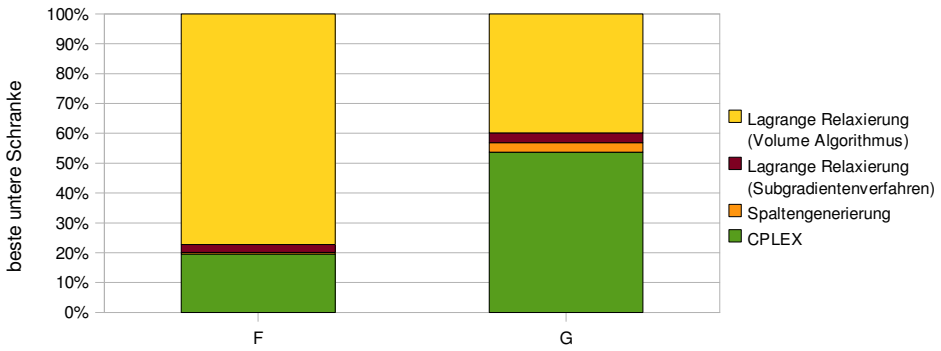

Abbildung 7.9.2: Beste untere Schranke nach Konfiguration gegliedert

Ein anderer Blickwinkel auf die Ergebnisse wirft ein klareres Bild auf den Zusammenhang zwischen der Größe der Testfälle (hinsichtlich dem Netzwerk und den Transporten) und dem Abschneiden der verschiedenen Verfahren. In Abbildung 7.9.3 wird dies anhand der Aufteilung nach der Anzahl der Knoten dargestellt. Bei Transporten und auch bei der Verbindungsanzahl ergibt sich ein ähnliches Bild – jedoch ist dies bei den Knoten am deutlichsten zu sehen. Bei kleineren Netzwerken bis zu einer Anzahl von 40 Knoten dominiert die

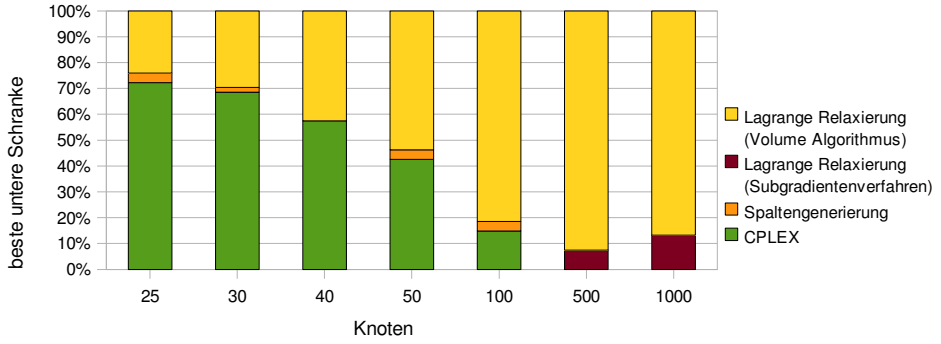

Abbildung 7.9.3: Verfahren mit der jeweils besten unteren Schranke nach Knoten aufgegliedert

Lösung durch CPLEX deutlich. In diesem Bereich kann auch die Spaltengenerierung ab und zu punkten. Die größeren Testfällen werden nur noch von der Lagrange Relaxierung gelöst. Es wird deutlich, dass die Lagrange Relaxierung mit Subgradientenverfahren nur im oberen Bereich gegenüber der Lagrange Relaxierung mit Volume Algorithmus bessere Resultate liefert. Bei dieser Knotenanzahl finden Spaltengenerierung und CPLEX keine Lösungen.

Bei der Aufgliederung der Verbindungsdichte in Abbildung 7.9.4 zeigen sich vor allem Unterschiede durch die Netzwerkgröße wie schon in der vorigen

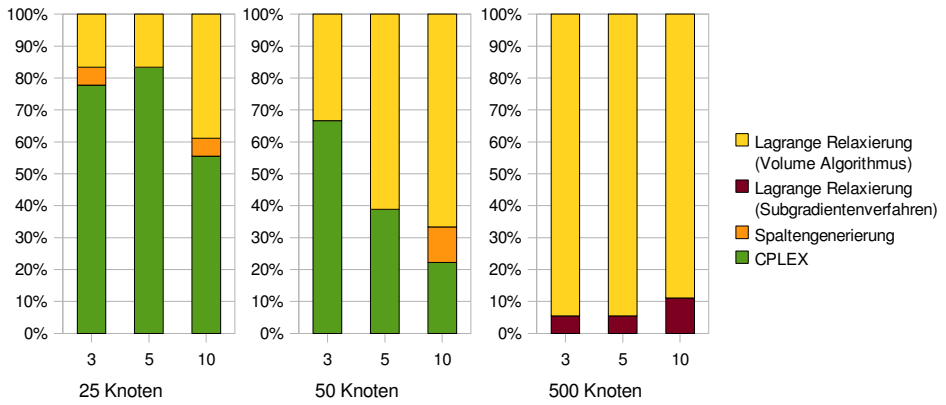

Abbildung 7.9.4: Verfahren mit der jeweils besten unteren Schranke nach Verbindungsdichte für 25, 50 und 500 Knoten

Abbildung 7.9.3 . Innerhalb einer Netzwerkgröße zeigt sich der Einfluss der Verbindungsdichte deutlich. CPLEX liefert bei dichteren Netzwerken seltener die beste untere Schranke. Dies ist vor allem darauf zurückzuführen, dass CPLEX bei größeren und dichteren Netzwerken in der vorgegebenen Zeit weniger oft eine Lösung findet.

Eine ähnliche Aufteilung zeigt sich bei der Anzahl der Transporte in Abbildung 7.9.10 wieder mit 25, 50 und 500 Knoten. CPLEX kann vor allem die kleineren Netzwerke mit 100 und 200 Transporten am Besten lösen. Je größer das Netzwerk und je mehr Transporte umso öfteren findet die Lagrange Relaxierung in der vorgegebenen Zeit die beste untere Schranke.

7.9.2 Vergleich bezüglich oberer Schranken

Eine etwas andere Verteilung der besten Verfahren als bei der besten unteren Schranke, zeigt sich bei der besten oberen Schranke. Hierbei kamen fünf Verfahren zum Einsatz: direktes Lösen mit CPLEX, Lagrange Relaxierung mit Subgradientenverfahren, Lagrange Relaxierung mit Volume Algorithmus, Spaltengenerierung und die Primale Heuristik. Einen Überblick bietet Abbildung 7.9.6, wobei wenn mehrere Verfahren die selben Lösungen liefern das schnellere Verfahren aufscheint. CPLEX und die Lagrange Relaxierung mit Volume Algorithmus sind nicht mehr eindeutig die erfolgreichsten Verfahren. Algorithmen wie die Spaltengenerierung und die Lagrange Relaxierung mit Subgradientenverfahren konnten öfter die beste obere Schranke als die untere beste Schranke liefern. Die Primale Heuristik ist vor allem dort am Besten,

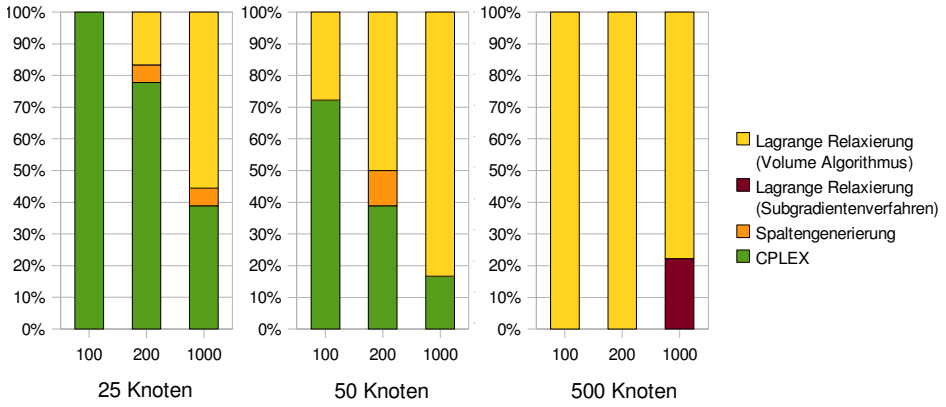

Abbildung 7.9.5: Verfahren mit der jeweils besten unteren Schranke nach der Anzahl der Transporte für 25, 50 und 500 Knoten

wo alle anderen Verfahren in der gegebenen Zeit keine Lösung mehr erzeugen. Wurde durch zwei Verfahren die selbe beste obere Schranke gefunden,

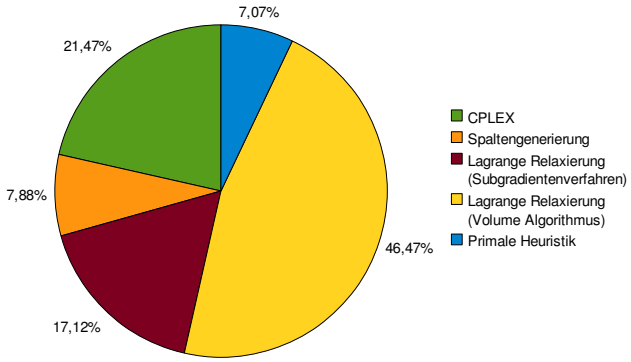

Abbildung 7.9.6: Überblick über die Verfahren, welche die beste obere Schranke liefern

so wird das schnellere Verfahren gegenüber dem langsameren bevorzugt.

Bei der Gegenüberstellung der beiden Konfigurationen in Bezug auf die beste obere Schranke in Abbildung 7.9.7 ist in erster Linie wieder das bessere Abschneiden vom CPLEX bei Konfiguration G hervorzuheben. Die Lösungen der Primalen Heuristik und der Spaltengenerierung sind bei beiden Konfigurationen etwa gleich stark vertreten. Die Verfahren der Lagrange Relaxierung liefern beide bei Konfiguration F öfter die beste obere Schranke. Diese unterschiedlichen Verteilungen sind vor allem darauf zurückzuführen, dass die Testfälle mit Konfiguration G wesentlich öfter direkt durch CPLEX (häufig sogar optimal) gelöst wurden. Findet CPLEX eine Lösung so ist diese meist auch die beste. Andernfalls finden die Verfahren der Lagrange Relaxierung

die beste obere Schranke finden, weshalb die Lagrange Relaxierung bei Konfiguration F öfter die beste obere Schranke liefert.

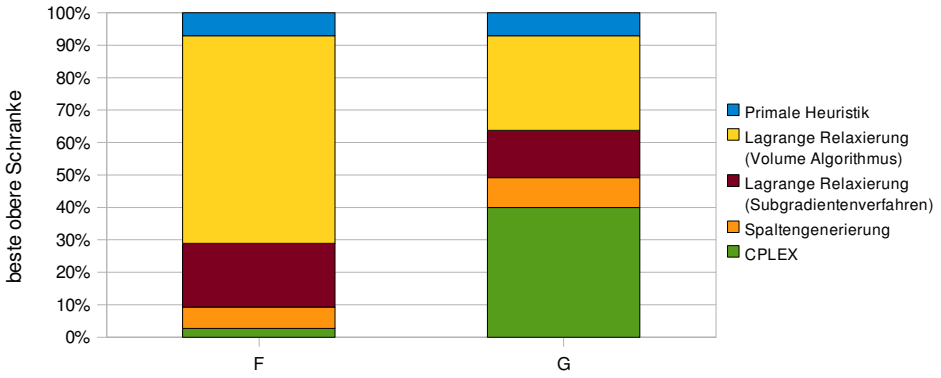

Abbildung 7.9.7: Verfahren mit der jeweils besten oberen Schranke nach Konfigurationen

Eine Aufgliederung der besten jeweiligen Methode nach Verbindungsdichte und ein Vergleich für unterschiedliche Netzwerkgrößen (25, 50 und 500 Knoten) zeigt Abbildung 7.9.8. Bei kleineren Netzwerken liefert das direkte Lösen durch CPLEX die besten Ergebnisse, jedoch ist dies bei steigender Verbindungsdichte seltener der Fall. Die Primale Heuristik ermöglicht erst bei größeren Netzwerken das Finden der besten oberen Schranke. Die Lagrange Relaxierung liefert vor allem dort die beste obere Schranke wo CPLEX keine Lösungen mehr findet.

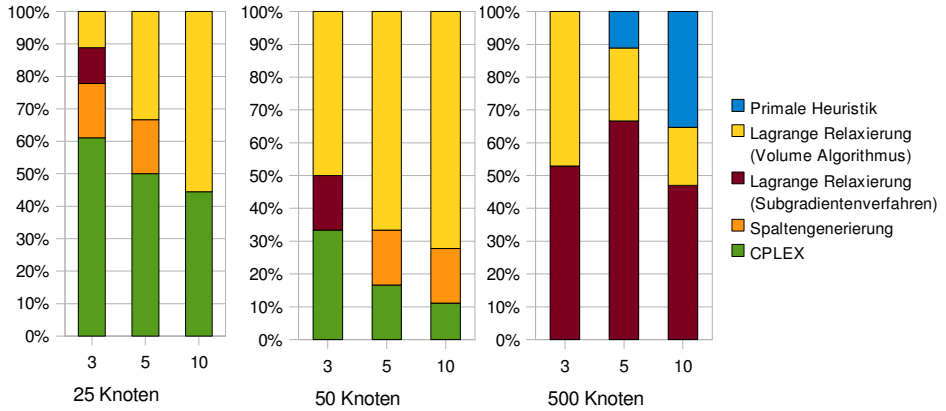

Abbildung 7.9.8: Verfahren mit der jeweils besten oberen Schranke nach der Dichte der Verbindungen für 25, 50 und 500 Knoten

Ein ähnliches Bild wie bei der Gliederung nach der Verbindungsanzahl ergibt sich bei der Auffächerung nach der Anzahl der Knoten, wie Abbildung

7.9.9 veranschaulicht. Die Anzahl der durch CPLEX erzeugten besten oberen Schranken sinkt stetig bis hin zu 100 Knoten, während der Anteil der Lagran-

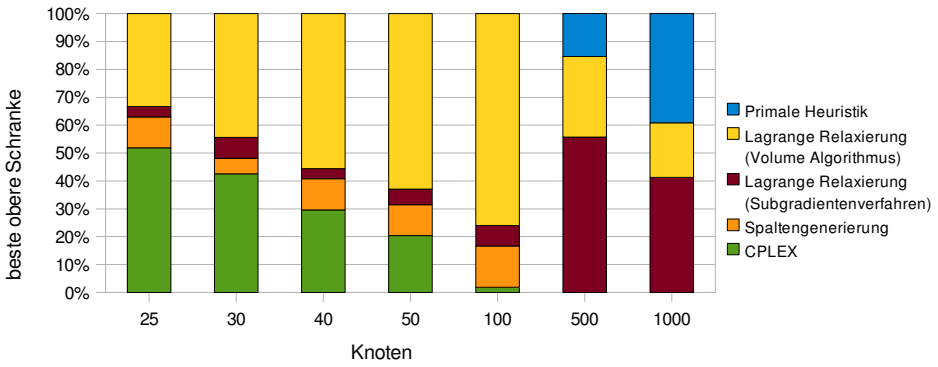

Abbildung 7.9.9: Verfahren mit der jeweils besten oberen Schranke nach Knotenanzahl

ge Relaxierung mit Volume Algorithmus in diesem Bereich etwa im gleichen Maße steigt. Über dieses Feld hinweg sind Spaltengenerierung und Lagrange Relaxierung mit Subgradientenverfahren immer ungefähr durch den selben Anteil vertreten. Ein anderes Bild zeigt sich bei 500 und 1000 Knoten. Die Primale Heuristik und die Lagrange Relaxierung mit Subgradientenverfahren sind deutlich stärker bei der besten oberen Schranke vertreten. Das gute Abschneiden der Primalen Heuristik in diesem Teil geht vor allem darauf zurück, dass hier andere Verfahren kein Ergebnis in der vorgegebenen Zeit zurückliefern, während eine valide Lösung sehr schnell erzeugt werden kann.

Bei der Aufgliederung nach Transporten in Abbildung 7.9.10 liefert wie bei der Aufgliederung nach der Verbindungsdichte direktes Lösen mittels CPLEX bei kleineren Netzwerken die besten Resultate. Erst bei großen Netzwerken und vielen Transporten kann teilweise nur noch die Primale Heuristik Ergebnisse erzielen. Die Lagrange Relaxierung kann bei allen Netzwerkgrößen und Transportgröße immer wieder die beste obere Schranke generieren, jedoch vor allem dort, wo CPLEX keine beziehungsweise nicht so viele Lösungen liefert.

7.9.3 Vergleich der besten oberen und unteren Schranken

Abschließend sollen nun die besten gefundenen oberen und unteren Schranken in Relation gesetzt werden, um einen Überblick über die Lösungsgüte zu geben. In Abbildung 7.9.11 wird der Gap – relativer Abstand – der besten gefundenen Schranken gegliedert nach der Anzahl der Knoten dargestellt.

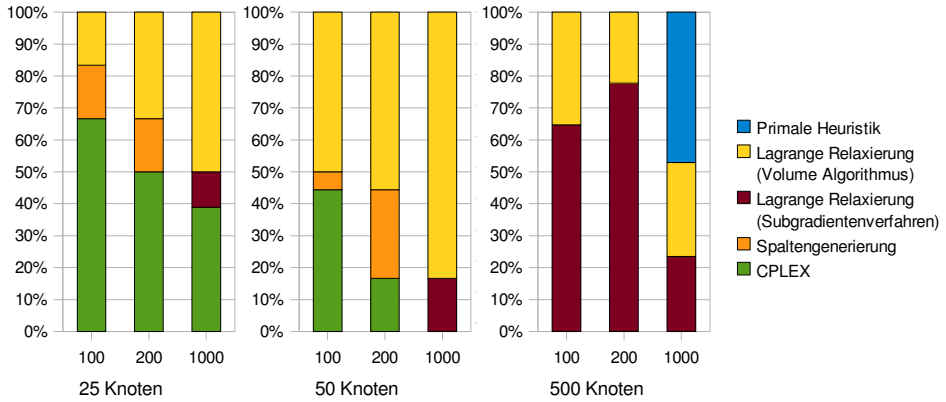

Abbildung 7.9.10: Verfahren mit der jeweils besten oberen Schranke nach der Anzahl der Transporte für 25, 50 und 500 Knoten

Der durchschnittliche Gap liegt bei 36,44%, wobei ein Verschlechterung des Gaps bei größeren Netzwerken auftritt. In diesem Zusammenhang ist aber

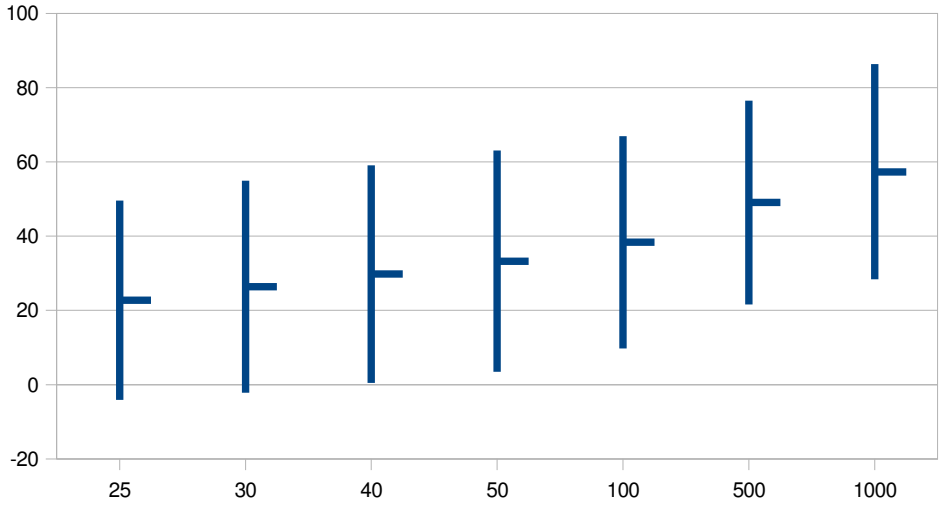

Abbildung 7.9.11: Darstellung des durchschnittlichen Gaps (mit Standardabweichung) zwischen der besten oberen und besten unteren gefundenen Schranke

zu bedenken, dass bei den Testläufen jedem Verfahren nur maximal 1000 Sekunden zur Verfügung standen, um eine Lösung zu generieren. Gibt man den jeweiligen Methoden ein größeres Zeitkontingent, werden auch für solch große Instanzen bessere Ergebnisse möglich. Da es sich bei dem bearbeiteten Problem um ein Design Problem handelt, wären längere – mehr als 1000 Sekunden – Laufzeiten unproblematisch.

Tabelle 7.10: Gelöste Instanzen des kleineren Testsets bei unterschiedlichen Laufzeiten (1000 und 10000 Sekunden) für CPLEX und Spaltengenerierung

Testfälle				CPLEX		Spaltengen.	
Konfig.	Knoten	Transp.	Linkf.	1000s	10000s	1000s	10000s
F	25	100	3	-	-	✓	✓
F	25	100	10	-	-	-	✓
F	50	100	3	-	-	-	✓
F	50	100	10	-	-	-	-
F	500	100,1000	3,10	-	-	-	-
G	25	100	3,10	✓	✓	✓	✓
G	25	1000	3,10	✓	✓	-	-
G	50	100	3,10	-	✓	✓	✓
G	50	1000	3,10	-	-	-	-
G	500	100	3,10	-	-	-	✓
G	500	1000	3,10	-	-	-	-

7.10 Ergebnisse für längere Laufzeiten

Mit einem kleineren Testset – 25, 50 und 500 Knoten, 3 und 10 als Linkfaktor und 100 und 1000 Transporte – wurden die Test auch mit der zehnfachen Laufzeit (10000 Sekunden) ausgeführt. Durch die Steigerung der Laufzeit konnten erwartungsgemäß mehr Testfälle optimal gelöst werden. Tabelle 7.10 zeigt einen Vergleich der gelösten Testfälle für das direkte Lösen mit CPLEX und für die Spaltengenerierung. Wie schon erwähnt kann CPLEX Testfälle der Konfiguration F auch nicht innerhalb eines Tages lösen. Bei Tests mit Konfiguration G konnten durch das Verlängern der Laufzeit auch etwas größere Netzwerke gelöst werden. Mit der Spaltengenerierung konnten generell nur die Testfälle mit 100 Transporten in der gegebenen Zeit gelöst werden. Bei Konfiguration F konnten durch die verlängerte Laufzeit neben dünnen Netzwerken mit 25 Knoten, auch für dichtere mit 25 Knoten und dünne Netzwerke mit 50 Knoten Ergebnisse erzielt werden. Ein ähnliches Phänomen lässt sich bei Konfiguration G beobachten, statt 25 und 50 Knoten werden in 10000 Sekunden auch Netzwerke mit 500 Knoten gelöst.

Die Verfahren der Lagrange Relaxierung konnten durch eine längere Laufzeit den durchschnittlichen Gap zwischen oberer und unterer Schranke sichtbar verbessern. Dieser Zusammenhang ist in Abbildung 7.10.1 zu sehen, hierbei wird zwischen den unterschiedlichen Konfigurationen unterschieden. Durch diese Ergebnisse bei längeren Laufzeiten wird deutlich, dass durch

mehr Zeit vor allem auch größere Testfälle gelöst beziehungsweise bessere Lösungen gefunden werden können.

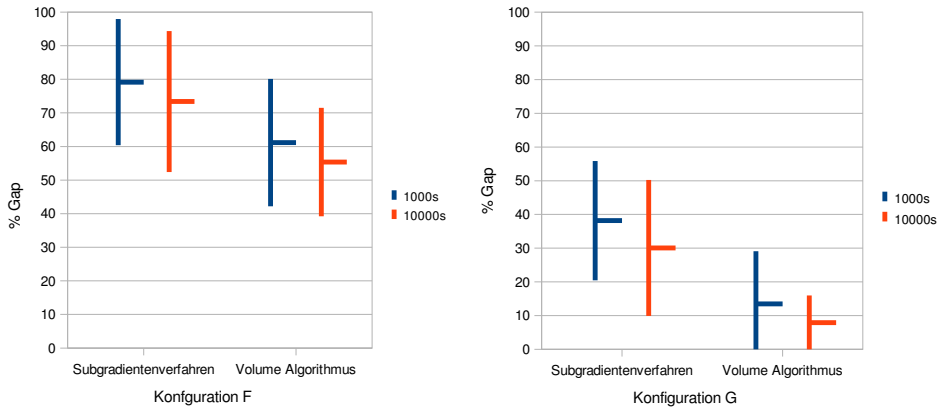

Abbildung 7.10.1: Darstellung des durchschnittlichen Gaps (mit Standardabweichung) zwischen der oberen und unteren Schranke bei der Lagrange Relaxierung mit unterschiedlichen Laufzeiten

7.11 Ergebnisse hinsichtlich Lösungsqualität

Es ist schwer allgemeine Aussagen über die Lösungsqualität der Verfahren zu geben, da nur für eine sehr kleine Teilmenge der Testfälle in der vorgegebenen Zeit für alle Verfahren eine gültige Lösung lieferten. In die folgenden Auswertungen wurden daher nur jene Testfälle einbezogen, bei denen zumindest CPLEX eine gültige Lösung erzeugt. Diese Testfälle haben alle Konfiguration G, maximal 50 Knoten und maximal 1000 Transporte.

Die Lagrange Relaxierung mit Subgradientenverfahren liefert Ergebnisse die durchschnittlich 14,07% (Standardabweichung 5,95) über dem Optimum lagen. Deutlich besser sind die Resultate der Lagrange Relaxierung mit Volume Algorithmus mit durchschnittlich 4,9% (Standardabweichung 2,56) über dem Optimum. Noch bessere Ergebnisse ermöglichte die Spaltengenerierung, die jedoch nicht für alle durch CPLEX gelösten Testfälle Resultate fand, mit 3,42% (Standardabweichung 1,94). Die sehr schnelle Primale Heuristik lieferte durchschnittlich Ergebnisse um 18,52% (Standardabweichung 7,07) über dem Optimum.

Für einen Teil der Instanzen, die optimal mittels CPLEX gelöst werden konnten, gibt Tabelle 7.11 einen Überblick über den durchschnittlichen prozentuellen Gap der jeweiligen unteren beziehungsweise oberen Schranke zum Optimum. Hierbei wurden nur Instanzen der Konfiguration G angeführt, da

Tabelle 7.11: Durchschnittliche %-Gaps (und Standardabweichung) der unteren (LB) und oberen (UB) Schranken in Bezug auf die optimale Lösung für Konfiguration G jeweils für (Knoten, Transporte, Linkfaktor)

	Lagrange Rel. mit Vol.				Spaltengenerierung			
Instanzen	LB [%]		UB [%]		LB[%]		UB[%]	
25,100,3	-0,99	(0,4)	4,43	(0,41)	-0,97	(0,4)	4,37	(1,01)
25,100,5	-1,03	(0,31)	5,53	(3,43)	-33,96	(57,18)	11,15	(15,07)
25,100,10	-0,82	(0,36)	5,53	(3,37)	-0,8	(0,35)	2,19	(0,49)
25,200,3	-0,99	(0,43)	4,36	(1,75)	-33,82	(57,31)	5,9	(3,57)
25,200,5	-0,71	(0,28)	5,18	(0,55)	-0,69	(0,27)	3,19	(1,2)
25,200,10	-0,21	(0,2)	3	(0,54)	-0,19	(0,18)	1,44	(1,75)
30,100,3	-1,33	(0,11)	5,1	(2,73)	-1,31	(0,13)	3,78	(2,06)
30,100,5	-1,39	(1,04)	4,79	(1,01)	-1,38	(1,02)	3,31	(1,61)
30,200,3	-0,72	(0,19)	3,56	(1,17)	-0,71	(0,2)	2,79	(1,06)
30,200,5	-0,74	(0,33)	6,38	(3,27)	-0,72	(0,32)	6,05	(4,1)

für die Konfiguration F selbst innerhalb von einem Tag nicht das Optimum gefunden werden konnte. Es werden die Instanzen jeweils nach Anzahl der Knoten, Transporten und der Linkfaktor gegliedert.

Bei größeren Instanzen kann das Optimum mittels CPLEX nicht mehr in der gegebenen Zeit gefunden werden, deshalb werden für die größere Instanzen die in Tabelle 7.12 angeführt sind, die durchschnittlichen Gaps zwischen den unteren und oberen Schranken angegeben. Deutlich ist zu sehen, dass die Spaltengenerierung für größere Instanzen keine Ergebnisse außer die Startlösung mehr in der vorgegebenen Zeit liefert – das Verfahren kommt nicht über den iterativen Prozess des LP-relaxierten Lösens hinaus.

7.12 Zusammenfassende Ergebnisse

Die Primale Heuristik kann in wenigen Sekunden für fast alle Testinstanzen gültige Lösungen erzeugen. Es ist jetzt möglich Netzwerke der Konfiguration G bis zu einer Größe von 40 Knoten für 1000 Transporte innerhalb eines Tages exakt zu lösen. Für Netzwerke bis zu einer Größe von 100 Knoten kann für alle Testfälle mit Konfiguration G innerhalb von 1000 Sekunden eine Lösung generiert werden, die nicht mehr als 25% über dem Optimum liegt.

Tabelle 7.12: Durchschnittliche Gaps (und Standardabweichung) zwischen unterer und oberer Schranke für größere Instanzen mit Konfiguration G jeweils für (Knoten, Transporte)

Instanzen	Lagrange R. mit Vol.-Gap [%]		Spaltengenerierung -Gap [%]		total Gap [%]	
50,100,3	9,68	(1,84)	7,43	(1,34)	7,43	(1,34)
50,100,5	9,78	(4,17)	6,5	(3,74)	6,5	(3,74)
50,100,10	8,94	(2,48)	6,46	(2,07)	6,46	(2,07)
50,200,3	12,4	(0,99)	38,57	(53,23)	9,42	(3,32)
50,200,5	11,97	(1,23)	7,85	(1,1)	7,85	(1,1)
50,200,10	11,84	(0,97)	37,37	(54,23)	7,63	(2,73)
100,100,3	14,4	(1,12)	8,88	(1,41)	8,88	(1,41)
100,100,5	12,28	(2,87)	6,36	(2,27)	6,36	(2,27)
100,100,10	6,98	(1,36)	36,15	(55,29)	4,77	(1,04)
100,200,3	16,86	(1,34)	70,79	(50,58)	14,91	(2,27)
100,200,5	16,83	(2,87)	100	(0)	16,83	(2,87)
100,200,10	15,21	(0,66)	100	(0)	15,21	(0,66)
500,100,3	12,92	(2,54)	100	(0)	12,92	(2,54)
500,100,5	14,6	(2,47)	100	(0)	14,6	(2,47)
500,100,10	15,17	(7,13)	100	(0)	15,17	(7,13)
500,200,3	20,17	(3,48)	100	(0)	20,17	(3,48)
500,200,5	20,91	(0,12)	100	(0)	20,56	(0,28)
500,200,10	24,75	(0,16)	100	(0)	24,75	(0,16)

Zusammenfassend ist zu sagen, dass jeder der vorgestellten Algorithmen in unterschiedlichen Bereichen gute Resultate erzielen konnte. Während CPLEX vor allem bei kleineren Testinstanzen erfolgreich war, sind durch die Lagrange Relaxierung auch Ergebnisse bei großen Testfällen möglich. Die Primale Heuristik ermöglicht schnell eine gültige Lösung zu erhalten. Etwas problematischer ist die Spaltengenerierung, da sie leider oft zu keinem Ergebnis kommt, hierbei sind Stabilisierungsfaktoren [AD06, ADdC06] vielversprechend, die jedoch den Rahmen der Diplomarbeit sprengen würden.

Kapitel 8

Zusammenfassung

In dieser Arbeit wurden Algorithmen zum Design eines sicherheits-, zeit- und kostenkritischen Kommunikationsnetzwerkes vorgestellt und evaluiert. Das Diplomarbeitsthema entstand aus einer Forschungskooperation des Instituts für Softwaretechnik und Interaktive Systeme mit Frequentis und Austro Control unter der Bezeichnung SWIS (System-Wide Information Sharing Network). Dieses Projekt ist Teil des fakultätsübergreifenden Forschungsschwerpunkts "Komplexe Systeme". In den vorangegangenen Forschungsarbeiten stand die in dieser Arbeit behandelte Optimierung jedoch nicht im Vordergrund.

Zunächst wurde das Problem modelliert und als ganzzahliges Lineares Programm formuliert. Ausgehend davon wurden bisherige (verwandte) Arbeiten auf diesem Gebiet präsentiert und analysiert. Aus diesen Betrachtungen ging hervor, dass die Schwierigkeit vor allem in den zusätzlichen Bedingungen hinsichtlich Zeit und Sicherheit bei der Übertragung liegt.

Weiters wurden algorithmische Ansätze zur Lösung des Problems erarbeitet. Neben der Lagrange Relaxierung mit Subgradientenverfahren und Volume Algorithmus wurde die Spaltengenerierung auf das Problem angewandt. Beide Verfahren beinhalten das exakte Lösen eines Teilproblems, nämlich das Finden eines kürzesten Weges mit zusätzlichen Beschränkungen. Dieses Problem wird in der Literatur als Constrained Shortest Path Problem beschrieben.

Die besprochenen Ansätze wurden in C++ implementiert und mittels Testdaten evaluiert. Bei vergleichsweise kleinen Instanzen erweist sich das direkte Lösen des ganzzahligen Linearen Programms mit CPLEX als durchaus praktikabler Ansatz. Größere Instanzen sind als ILP in der vorgegebenen Zeit jedoch unlösbar. In diesem Bereich liefert die Lagrange Relaxierung gute untere und obere Schranken. Schranken können bei unbekanntem Optimum den Bereich der optimalen Lösung eingrenzen und somit etwas über

die Qualität der gefundenen Lösung aussagen. Wie in der Literatur beschrieben, konnte der Volume Algorithmus gegenüber dem Subgradientenverfahren bessere Resultate liefern. Durch letzteres konnten sehr oft gute oberen Schranken gefunden werden. Der Spaltengenerierungsansatz konnte nur in Ausnahmefällen (die besten) Lösungen finden. Sehr oft wurde kein Ergebnis erzielt, da das Verfahren nicht über das iterative LP-relaxierte Lösen hinaus kam. Der Einsatz von Stabilisierungsfaktoren wäre hierbei vielversprechend.

Mit den implementierten Verfahren konnten verschiedene Aspekte abgedeckt werden. Zum einen kann durch die Primale Heuristik innerhalb weniger Sekunden eine gültige Lösung für das Problem erzeugt werden. Anderseits kann für Netzwerke bis zu einer Größe von 40 Knoten und für 1000 Transporte innerhalb eines Tages die optimale Lösung gefunden werden. Für größere Netzwerke können mittels Lagrange Relaxierung gute Ergebnisse in vertretbarer Zeit generiert werden. Die Testfälle des SWIS-Projektes können jetzt in maximal 2 Sekunden exakt gelöst werden. Das stellt eine erhebliche Verbesserung gegenüber der Laufzeit von 2 Wochen des zuvor verwendeten Enumerationsverfahrens dar.

Insgesamt zeigen sich demnach vor allem zwei Vorteile der hier behandelten Ansätze gegenüber den bisher beim SWIS Projekt verwendeten Verfahren. Zum einen bei der Geschwindigkeit gegenüber dem Enumerationsverfahren und zum anderen bei der Abschätzung der Lösungsgüte gegenüber dem heuristischen Ansatz. Die in dieser Diplomarbeit erarbeiteten Ansätze und Ergebnisse werden auch in [CMR08] besprochen. Die gewonnenen Erkenntnisse werden insbesondere bei Folgeprojekten und Varianten Verwendung finden.

Anhang A

Preprocessing Algorithmus

Algorithmus A.1 Preprocessing Algorithmus

Transport k	von Quelle s zu Senke t
	v_k Einheiten zu transportieren
$\boldsymbol{d}_{\max}$	maximal erlaubter Delay für Transport k
$\boldsymbol{c}_{ij}$	Kosten der Kante (i, j)
$\boldsymbol{d}_{ij}$	Delay der Kante (i, j)
costs(Pfad)	Pfadkosten: die Kosten des Pfades
delay(Pfad)	Pfaddelay: der Delay des Pfades
PathU	Pfad der oberen Schranke
U	obere Schranke (Kosten)
$\mathbf{U}_0$	erste obere Schranke
L	untere Schranke

1: **Step 0 – Initialisierung:**
2: Setze obere Schranke U; $U_0 = U$; $L = 0$
3: **Step 1 – kürzeste Kostenpfade:**
4: C_{si} bezeichnet den kostengünstigsten Pfad von der Quelle s zu einem Knoten i
5: **if** (C_{st} nicht existiert) **then** // es existiert kein kostengünstigsten Pfad von der Quelle s zur Senke t
6: **if** ($U \neq U_0$) **then** // in einem früheren Durchgang wurde eine obere Schranke gefunden
7: **return** PathU // Pfad der oberen Schranke U
8: **else**
9: **return** Infeasible, es gibt keinen Pfad
10: **end if**
11: **end if**

12: **if** (delay(C_{st}) $\leq$ maxDelay) **then** // Delay des kostengünstigsten Pfades von der Quelle s zur Senke t ist im erlaubten Rahmen
13: **if** ($U \neq U_0$) **then** // in einem vorigen Durchgang wurde eine obere Schranke U gefunden
14: **return** PathU // Pfad der oberen Schranke
15: **else**
16: **return** C_{st} // aktueller Pfad
17: **end if**
18: **end if**
19: **if** (costs(C_{st}) $\geq U$) **then** // Kosten des kostengünstigsten Pfades von der Quelle s zur Senke t ist größer als die obere Schranke
20: **return** PathU // Pfad der oberen Schranke ist optimal
21: **else**
22: L = costs(C_{st}) // Setze untere Schranke
23: **end if**
24: C_{it} bezeichnet den kostengünstigsten Pfad von der Senke t zu einem Knoten i
25: **Step 2 – kürzeste Delaypfade:**
26: D_{si} bezeichnet den günstigsten Pfad hinsichtlich Delay von der Quelle s zu einem Knoten i
27: **if** (delay(D_{st}) $> d_{\max}$) **then** // Delay des kürzesten Pfades hinsichtlich Delay von der Quelle s zur Senke t ist größer als der maximal erlaubte Delay
28: **if** ($U \neq U_0$) **then** // in einem vorigen Durchgang wurde eine obere Schranke U gefunden
29: **return** PathU // Pfad der oberen Schranke
30: **else**
31: **return** Infeasible, es gibt keinen Pfad
32: **end if**
33: **end if**
34: **if** (delay(D_{st}) $\leq d_{\max}$ AND costs(D_{st}) $< U$) **then** // Delay des günstigsten Pfades hinsichtlich Delay von der Quelle s zur Senke t ist im erlaubten Bereich und die aktuellen Kosten sind besser als die bisherige untere Schranke
35: L = costs(D_{st}) // Setze untere Schranke
36: PathU = aktueller Pfad D_{st}
37: **end if**
38: D_{it} bezeichnet den kürzesten Pfad hinsichtlich Delay von der Senke t zu einem Knoten i

39: **Step 3 – Prüfe Knoten und Kanten:**
40: **for all** $(i \in V \setminus \{s,t\})$ **do** // für alle Knoten außer Quelle und Senke
41: **if** (delay(D_{si}) + delay(D_{it}) > $d_{\max}$) **then** // über Knoten i kann kein gültiger Pfad führen, da der maximale Delay verletzt wäre
42: lösche Knoten i und inzidenten Kanten
43: **end if**
44: **if** (costs(C_{si}) + costs(C_{it}) $\geq U$) **then** // über Knoten i kann kein günstigster Pfad führen, da sicher über der oberen Schranke
45: lösche Knoten i und inzidenten Kanten
46: **end if**
47: **end for**
48: **for all** Kanten (i,j) **do**
49: **if** (delay(D_{si}) + d_{ij} + delay(D_{jt}) > $d_{\max}$) **then**
50: lösche Kante (i,j)
51: **else if** (costs(C_{si}) + c_{ij} + costs(C_{jt}) $\geq U$) **then**
52: lösche Kante (i,j)
53: **else if** (delay (C_{si}) + d_{ij} + delay(C_{jt}) $\leq$ maxDelay) **then**
54: U = costs(C_{si}) + c_{ij} + costs(C_{jt}); // obere Schranke wird gesetzt
55: PathU = Pfad C_{si} + (i,j) + C_{jt}
56: **end if**
57: **end for**
58: **Step 4 – Wiederhole wenn nötig:**
59: **if** hat sich etwas geändert **then** // wurde eine Kante oder ein Knoten gelöscht oder die untere Schranke geändert
60: gehe zu Step 1
61: **end if**

Literaturverzeichnis

[AD06] Hatem Ben Amor and Jacques Desrosiers. A proximal trust-region algorithm for column generation stabilization. *Compututers & Operations Research*, 33(4):910–927, 2006.

[ADdC06] Hatem Ben Amor, Jacques Desrosiers, and Jos Manuel Valrio de Carvalho. Dual-optimal inequalities for stabilized column generation. *Computers & Operations Research*, 54(3):454–463, 2006.

[Alg08] Algorithmic Solutions Software GmbH. LEDA - The Library of Efficient Data types and Algorithms, 2008. `http://www.algorithmic-solutions.com/leda` (20.08.2008).

[AMOR95] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M.R. Reddy. Applications of network optimization. In C. Monma G. Nemhauser M. Ball, T. Magnanti, editor, *Handbooks in Operations Research and Management Science, Volume 7: Network Models*, chapter 1, pages 1–84. 1995.

[Aus08] Austro Control. Firmenwebsite, 2004–2008. `http://www.austrocontrol.co.at` (02.10.2008).

[BA00] Francisco Barahona and Ranga Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[Bea93] John E. Beasley. Lagrangean relaxation. pages 243–303, New York, NY, USA, 1993. John Wiley & Sons, Inc.

[CMR08] Andreas M. Chwatal, Nina Musil, and Günther R. Raidl. Solving a Multi-Constrained Network Design Problem by Lagrangean Decomposition and Column Generation. 2008.

[CN98] Shigang Chen and Klara Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. In *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, 1998.

[DB03] Irina Dumitrescu and Natashia Boland. Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem. In *Networks*, volume 42, pages 135–153, 2003.

[DL08] Complex Systems Design and Engineering Lab. Forschungsprojekt SWIS. Website, 2008. online verfügbar `http://www.informatik.tuwien.ac.at/csde/` (15.05.2008).

[Fre08] Frequentis. Firmenwebsite, 2002–2008. `http://www.frequentis.com` (07.05.2008).

[GCF98] Bernard Gendron, Teodor Crainic, and Antonio Frangioni. Multicommodity capacitated network design. In *Telecommunications Network Planning*, pages 1–19. B. Sans'o and P. Soriano (eds.), 1998.

[GG61] Paul Gilmore and Ralph Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GPSS08] Luis Gouveia, Ana Paias, Dushyant Sharma, and Dushyant Sharma. Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers and Operations Research*, 35(2):600–613, 2008.

[HK95] Richard V. Helgason and Jeffery L. Kennington. Primal simplex algorithms for minimum cost network flows. In C. Monma G. Nemhauser M. Ball, T. Magnanti, editor, *Handbooks in Operations Research and Management Science, Volume 7: Network Models*, chapter 2, pages 85–133. 1995.

[HS06] Mohamed Haouari and Jouhaina Chaouachi Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers and Operations Research*, 33:1274–1288, 2006.

[ILO08] ILOG. CPLEX, 2008. `http://www.ilog.com/products/cplex/` (28.08.2008).

[JLK78] David S. Johnson, Jan K. Lenstra, and Alexander H.G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.

[JLX05] Xin Jin, Xiande Liu, and Shiyuan Xiao. An effective algorithm for delay constrained least cost unicast routing. In *ISCAS (4)*, pages 3251–3254, 2005.

[JV06] Zhanfeng Jia and Pravin Varaiya. Heuristic methods for delay constrained least cost routing using k-shortest-paths. In *Automatic Control, IEEE Transactions on*, volume 51, pages 707–712, 2006.

[KF06] Jiri Kubalik and Jan Faigl. Iterative prototype optimisation with evolved improvement steps. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 154–165, Budapest, Hungary, 10 - 12 April 2006. Springer.

[KLHG07] Mark P. Kleeman, Gary B. Lamont, Kenneth M. Hopkinson, and Scott R. Graham. Solving Multicommodity Capacitated Network Design Problems using a Multiobjective Evolutionary Algorithm. In *IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)*, pages 33–41. IEEE Press, April 2007.

[Kli06] John G. Klincewicz. Optimization issues in quality of service. In M. G. C. Resende and P. M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, chapter 17, pages 435–458. 2006.

[KM07] Jiri Kubalik and Richard Mordinyi. Optimizing events traffic in event-based systems by means of evolutionary algorithms. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 1101–1107, Washington, DC, USA, 2007. IEEE Computer Society.

[KMKK02] Fernando Kuipers, Piet Van Mieghem, Turgay Korkmaz, and Marwan Krunz. An overview of constraint-based path selection algorithms for QoS routing. *Communications Magazine, IEEE*, 40(12):50–55, 2002.

[LD05] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[MZ00] Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In *Proc. 8th European Symposium on Algorithms (ESA), Springer LNCS*, pages 326–337. Springer, 2000.

[Res00] Eric Rescorla. *HTTP Over TLS. RFC 2818.* May 2000.

[RS00] Douglas S. Reeves and Hussein F. Salama. A distributed algorithm for delay-constrained unicast routing. *IEEE/ACM Trans. Netw.*, 8(2):239–250, 2000.

[Tec07] Technische Universität Wien. Forschungsdokumentation: Systemweites Informationsverteilungssystem, 2007. online verfügbar `http://tuwis.tuwien.ac.at/ora/tuwis/bokudok/search_project.show_project?project_id_in=4696` (05.06.2008).

[THJ07] Jiri Trdlicka, Zdenek Hanzalek, and Mikael Johansson. Optimal flow routing in multi-hop sensor networks with real-time constraints through linear programming. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 924–931, 2007.

[TZ06] George Tsaggouris and Christos Zaroliagis. QoS-aware Multicommodity Flows and Transportation Planning. In Riko Jacob and Matthias Müller-Hannemann, editors, *ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways*, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[WC96] Zheng Wang and Jon Crowcroft. Quality of service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.