Steuerung von Mehrplatz-Aufzügen

Diplomarbeit
bei Prof. Dr. Grötschel

vorgelegt von Steffen Weider
am Fachbereich Mathematik der
Technischen Universität Berlin

Berlin, den 1. März 2000

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides statt.

Steffen Weider
Berlin, den 6. März 2000

Danksagung

Hiermit möchte ich den Personen danken, die diese Arbeit möglich machten. An erster Stelle danke ich meiner Frau, daß sie so wenig gejammert hat, auch wenn ich mich abends noch an den Rechner gesetzt habe, um ein wenig weiter zu schreiben oder zu programmieren. Für die vielen Anregungen und das häufige Korrekturlesen danke ich *Sven Oliver Krumke* und *Jörg Rambau*. *Martin Grötschel* danke ich für die hervorragenden Vorlesungen, die ein Fundament für diese Arbeit bildeten. Außerdem danke ich der gesamten *Abteilung Optimierung* des ZIB für die nette Atmosphäre.

Steffen Weider
Berlin, den 6. März 2000

Inhaltsverzeichnis

Einleitung

Anlaß für die Untersuchung von Aufzügen und Aufzugssystemen ist ein Forschungsprojekt des Konrad-Zuse-Instituts für Informationstechnik Berlin und der Herlitz AG, welches von der Deutschen Forschungsgemeinschaft finanziert wird. In diesem Forschungsprojekt werden die Optimierungmöglichkeiten eines Hochregallagers der Herlitz AG in Falkensee nahe Berlin mit seinen angeschlossenen automatischen Materialfördersystemen untersucht.

Das Materialfördersystem besteht aus fünf Vertikalförderern, die jeweils neun Etagen bedienen. Das Material wird auf Euro-Paletten über ein Ring von Förderbändern zu den Vertikalförderern gebracht. Diese transportieren das Material in das gewünschte Stockwerk, von dort aus wird es über den Ring aus Förderbändern weitertransportiert.

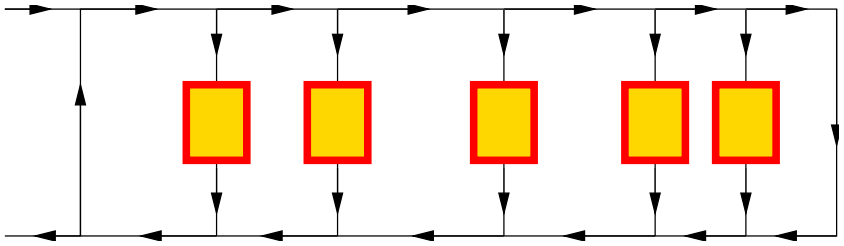

Abbildung 1: schematische Darstellung der Aufzüge und Förderbänder (aus [25]).

Diese Vertikalförderer, im folgenden einfach Aufzüge genannt, sind das Objekt unserer Untersuchungen in dieser Arbeit.

In anderen Arbeiten, wie z.B. [25] oder [24], wurde die Steuerung eines einzelnen Aufzugs, der nur jeweils eine Palette transportieren kann, untersucht. In dieser Arbeit untersuchen wir, welchen Effekt eine höhere Kapazität der Aufzüge auf die Flußzeiten der einzelnen Aufträge und die Fertigstellungszeit aller Aufträge hat. Dazu müssen wir aber zunächst effiziente Algorithmen zur Steuerung der Aufzüge entwickeln, da in der Literatur noch keine Algorithmen für dieses spezielle Problem existieren.

Die Zuweisung und Aufteilung der Aufträge zu den Aufzügen wird gesondert betrachtet und ist nicht Thema dieser Arbeit.

Maximal ist eine Erhöhung des Durchsatzes (also der Anzahl der abgearbeiteten Aufträge je Zeiteinheit) bei Einsatz eines Mehrplatz-Aufzuges um einen Fak-

tor von k im Vergleich zum Einsatz eines Aufzuges mit einfacher Kapazität zu erwarten. Dabei ist k die Kapazität des Mehrplatz-Aufzugs. Tatsächlich wird dieser Faktor geringer sein, da die Kapazität des horizontalen Fördersystems gleichbleibt, und da nicht immer geeignete Aufträge im System vorhanden sein werden, weshalb Leerfahrten oder nicht voll-ausgelastete Fahrten des Aufzuges nötig sein werden. Ein Aufzug mit einer Kapazität von k Objekten hat aber gegenüber k Aufzügen mit einer Kapazität von einem Objekt den Vorteil, daß er weniger Platz in Anspruch nimmt und auch günstiger in der Anschaffung und den Betriebskosten sein wird. Um aber einen konkreten Vergleich anstellen zu können, muß man quantifizieren, wieviel besser ein Aufzug mit Kapazität k gegenüber einem Aufzug mit Kapazität Eins ist.

Gliederung der Arbeit

Zunächst beschreiben wir im Kapitel 1 das untersuchte Problem und die hier verwendeten mathematischen Modelle. Dabei stellen wir die Verwandtschaft des hier vorgestellten Problems, wir nennen es „capacitated Dial-a-Ride-Problem“ (CDARP) zu bekannten Problemen, wie dem „Travelling Salesman Problem“, dem „Dial-a-Ride-Problem“ oder allgemein Transportplanungsproblemen fest.

Im Kapitel 2 beschäftigen wir uns mit der Komplexität des CDARP auf Pfaden und verschiedenen Approximationsalgorithmen und Heuristiken zu seiner Lösung. Zunächst zeigen wir, daß das Problem einen kürzesten Transportplan für einen Aufzug mit einer Kapazität von zwei Objekten zu finden $\mathcal{NP}$-schwer ist. Dazu vollziehen wir einen Beweis von Guan [22] nach, den wir aber leicht modifizieren und damit an einer Stelle genauer fassen. In der Folge werden wir uns mit unteren Schranken für das CDARP beschäftigen. Diese verwenden wir, um zu zeigen, daß der Algorithmus COVERPATH einen Transportplan für ein CDARP mit beliebiger Kapazität findet, der höchstens die dreifache Zeit des optimalen Transportplans benötigt. Der Algorithmus COVERPATH ist damit der erste Approximations-Algorithmus mit konstanter Gütegarantie für das CDARP auf Pfaden.

Den Algorithmus COVERPATH verwenden wir, um zu zeigen, daß für das CDARP auf Kreisen ein Approximationsalgorithmus mit konstanter Gütegarantie existiert und um zu zeigen, daß der Algorithmus INTERVALLGRAPH ebenfalls eine konstante Gütegarantie besitzt. Diesen Algorithmus INTERVALLGRAPH benutzen wir, um Algorithmen für ein erweitertes Modell eines Aufzugs abzuleiten. Die Algorithmen COVERPATH und INTERVALLGRAPH werden wir im Kapitel 4 mit verschiedenen Heuristiken für das CDARP auf Pfaden vergleichen.

Außerdem stellen wir im Kapitel 2 ein Resultat von Atallah und Kosaraju [6] vor. Dieses besagt, daß ein einfaches Modell eines Aufzugs mit Kapazität 1 in polynomialer Zeit optimal lösbar ist. Anschließend vollziehen wir ein Resultat von Guan [22] nach, welches besagt, daß ein einfaches Modell eines Aufzugs mit einer Kapazität von $k > 1$, der einen Auftrag auch vor seinem Zielstockwerk ablegen und andere Aufträge vorziehen darf, auch polynomial lösbar ist.

Im darauffolgenden Kapitel 3 besprechen wir zwei ganzzahlige lineare Programme (ILP) vor, die das CDARP auf Pfaden und auf allgemeinen Graphen beschreiben. Das ILP für CDARP auf Pfaden ermöglicht es Instanzen mit bis zu 20 Aufträgen in wenigen Minuten optimal zu lösen. Das ILP für CDARP auf allgemeinen Graphen ist eine Erweiterung eines ILP, welches von Ruland [28] entwickelt wurde, um ein ähnliches Problem zu lösen.

Zum Abschluß vergleichen wir die hier vorgestellten Algorithmen, indem wir sie zufällig erzeugte Probleminstanzen des CDARP lösen lassen. Außerdem leiten wir aus den hier vorgestellten Algorithmen Online-Algorithmen ab und verwenden diese in einer Simulation von Aufzügen.

Grundlegende Notation und Begriffe sind im Anhang A erklärt.

Kapitel 1

Modelle

Zunächst beschreiben wir Eigenschaften von Aufzügen im allgemeinen und gehen dann auf die Besonderheiten der Aufzüge des innerbetrieblichen Logistiksystems bei der Herlitz AG in Falkensee ein. Anschließend zeigen wir die Verwandtschaft von Aufzügen mit Einheitskapazität mit dem „Dial-a-Ride-Problem“ auf. Einheitskapazität bedeutet dabei, daß der Aufzug genau ein Objekt auf einmal transportieren darf. Darauf aufbauend stellen wir ein Modell für Aufzüge vor, die mehrere Objekte auf einmal transportieren dürfen. Dieses Modell erweitern wir anschließend, um verschiedene Varianten von realen Aufzügen abbilden zu können.

1.1 Aufzüge

In diesem Abschnitt beschreiben wir Aufzüge im Sinne dieser Arbeit und welche mathematisch relevanten Varianten von Aufzügen wir untersuchen. Dabei werden besonders die Eigenschaften der Aufzüge im innerbetrieblichen Logistiksystems bei der Herlitz AG in Falkensee berücksichtigt.

1.1.1 Aufzüge im allgemeinen

Ein Aufzug ist dadurch charakterisiert, daß er mehrere Stockwerke anfahren kann, die übereinander angeordnet sind. Seine Aufgabe ist es, Objekte von einem Stockwerk zu einem anderen zu bringen.

Ein Aufzug kann eine begrenzte Anzahl von Objekten zur selben Zeit transportieren. Diese Anzahl kann durch ein Gesamtgewicht oder -volumen der beförderten Objekte begrenzt sein. So haben z.B. Personenaufzüge immer ein maximal zugelassenes Gesamtgewicht. In automatischen Fördersystemen ist die Kapazität des Aufzugs auch oft durch die Anzahl der Ladehilfsmittel begrenzt, die der Aufzug aufnehmen kann. Insbesondere wenn die Ladehilfsmittel sehr groß sind, wie dies z.B. bei Euro-Paletten der Fall ist, kann der Aufzug oft nur ein einziges beladenes Ladehilfsmittel aufnehmen.

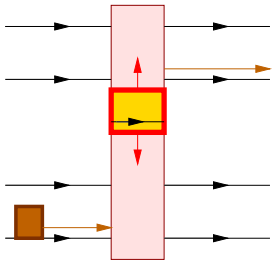

Abbildung 1.1: Horizontale Sicht eines Aufzugs (aus [21]).

Nun stellt sich die Frage, bezüglich welcher Ziele wir die Steuerung der Aufzüge verbessern wollen. Es gibt mehrere sich gegenseitig beeinflussende Zielgrößen für Aufzugssysteme. So kann es ein Ziel sein, die maximale Wartezeit jedes Objekts zu minimieren. Andererseits kann man versuchen, die Auslastung des Aufzugs zu maximieren. D.h. man versucht, die Anzahl Objekte, die der Aufzug in die richtige Richtung transportiert, im Durchschnitt zu maximieren. Diese beiden Ziele können sich widersprechen, da es für eine hohe Auslastung günstig sein kann, daß man ungünstige Objekte zunächst liegen läßt und dafür andere Objekte mit geringerer Wartezeit vorher transportiert.

Es ist in der Praxis wichtig überlange Wartezeiten eines Objektes zu verhindern, da die Vorhersagbarkeit der ungefähren Ankunftszeit eines Objektes ein wichtiges Kriterium für die Planbarkeit der Produktionsabläufe ist. Andererseits führt eine schlechte Auslastung des Aufzugs bei hoher Belastung zwangsweise zu langen Wartezeiten bei den Objekten. Man muß also in der Praxis einen Kompromiss zwischen diesen beiden Zielen finden.

Um einen Aufzug so steuern zu können, daß er bestimmte Ziele bezüglich seiner Fahrzeiten erreicht, muß man untersuchen, wieviel Zeit ein Aufzug benötigt, um von einem Stockwerk zu einem anderen zu fahren. Diese Zeiten setzen sich typischerweise aus verschiedenen Einzelzeiten zusammen:

- einer Zeit zum Beladen des Aufzugs, die von der Anzahl der einzuladenden Objekte abhängig sein kann,
- einer Zeit in der der Aufzug beschleunigt,
- einer Zeit, in der sich der Aufzug mit gleichbleibender Geschwindigkeit bewegt,
- eine Zeit, die der Aufzug zum Abbremsen und Positionieren an seinem Zielstockwerk benötigt, und
- einer Zeit zum Entladen der im Aufzug befindlichen Objekte, die wiederum von der Anzahl der Objekte und ihrer Position im Aufzug abhängig sein kann.

Nun kann man bei Aufzügen noch Anforderungen an die Reihenfolge der Abarbeitung der Aufträge stellen. So können Aufzüge über Warteschlangen an den Stockwerken beladen werden. Dies heißt, daß an jedem Stockwerk nur Aufträge eingeladen werden können, die in der Warteschlange vorne stehen, die als erstes an diesem Stockwerk angekommen sind.

Auch könnte es sein, daß die Aufträge, die sich in dem Aufzug befinden, nur in einer bestimmten Reihenfolge entladen werden können. Hat z.B. ein Aufzug nur eine Tür, durch die er be- und entladen wird, so könnte es sein, daß nur das Objekt entladen werden kann, welches zuletzt in den Aufzug eingeladen wurde.

1.1.2 Aufzüge bei Herlitz

Bei den Aufzügen des Fördersystems bei Herlitz in Falkensee werden die Aufzüge über ein vollautomatisches Fördersystem beladen. Dieses Fördersystem wird dezentral gesteuert. Das führt dazu, daß der Aufzugssteuerung ein Auftrag erst bekannt wird, wenn dieser schon zum Beladen bereit steht. Dabei kann maximal ein Auftrag je Stockwerk auf den Aufzug warten. Dies führt dazu, daß immer nur eine relativ geringe Anzahl von Objekten einem bestimmten Aufzug zugeodnet sind und bei der Erstellung eines Transportplans für diesen Aufzug berücksichtigt werden können. Auch steht wegen dieser kurzfristigen Zuordnung der Objekte zu den Aufzügen beschränkte Rechenzeit (ca. 1 Sekunde) zur Erstellung eines Transportplans zur Verfügung, denn die Rechenzeit muß in der Gesamtzeit, die der Aufzug zur Erledigung der Aufträge benötigt, berücksichtigt werden. Die Steuerung der Aufzüge ist also sowohl ein Echtzeit-Problem (der Algorithmus muß eine Antwort innerhalb der Reaktionszeit seiner Umgebung geben) als auch ein Online-Problem (einige Daten stehen erst zur Laufzeit des Algorithmus zur Verfügung).

Die Entladung der Aufzüge erfolgt auf der gegenüberliegenden Seite der Beladung. Es gibt daher keine Konflikte zwischen Objekten die auf einem Stockwerk eingeladen werden und anderen Objekte die auf demselben Stockwerk entladen werden.

Die Zeit, die der Aufzug benötigt, um von einem Stockwerk zu starten oder zu halten ist ungefähr viermal so groß wie die Zeit, die der Aufzug benötigt, um die Strecke zwischen zwei direkt übereinanderliegenden Stockwerken zu überwinden.

1.1.3 Varianten

Wir werden in dieser Arbeit nicht nur das Herlitz-System und seine Modifizierung mit Aufzügen größerer Kapazität betrachten, sondern auch einige Vereinfachungen und Verallgemeinerungen dieses Problems. Insbesondere betrachten wir folgende Varianten:

- Im ersten Schritt betrachten wir nur die Fahrzeiten des Aufzugs zwischen zwei Stockwerken und vernachlässigen die Zeiten für das Be- und Entladen und das Starten und Stopen des Aufzugs. Auch nehmen wir an, daß

alle zu befördernden Objekte bekannt sind und in beliebiger Reihenfolge ein- und ausgeladen werden dürfen. Dies scheint eine sehr starke Vereinfachung des Problems zu sein, wir werden aber sehen, daß auch dieses Problem für Aufzüge mit einer Kapazität von zwei Objekten und unter Verbot von Zwischenlagerung von Objekten schon $\mathcal{NP}$-schwer ist.

- Wir werden zu diesem Modell Einschränkungen bei den möglichen Reihenfolgen der Objekte zur Beladung der Aufzüge vornehmen.
- Dann wollen wir zusätzlich Zeiten zum Starten und Stoppen des Aufzugs an einem Stockwerk berücksichtigen.
- Zuletzt testen wir in einer Simulation verschiedene Online-Algorithmen für unsere Modelle.

Zur Lösung der Online-Probleme sind die Offline-Algorithmen sehr nützlich, da sich aus den Offline-Algorithmen für das Aufzugsproblem Online-Algorithmen ableiten lassen, die eine Kompetitivität haben, die von der Gütegarantie des Online-Algorithmus abhängig ist. Dies wird in [5] dargelegt.

1.2 Das „General Pickup and Delivery Problem“

Man kann die Steuerung von Aufzügen ganz allgemein als ein Problem betrachten, ein Transportplan für ein Fahrzeug zu finden, welches eine Reihe von Orten unter bestimmten Nebenbedingungen kostenminimal anzufahren hat. Diese Art von Problemen ist in der Literatur als „Vehicle Routing Problem“ (VRP) bzw. als Transportproblem bekannt. Wir stellen vor, welche Arten von Transportproblemen in der Literatur bekannt und erforscht sind. Dazu stellen wir die Klassifikation von Transportproblemen von Savelsbergh und Sol [30] vor.

Savelsbergh und Sol gehen von einem allgemeinen Problem aus und identifizieren einige bekannte Probleme als Spezialfall dieses Problems. Diese allgemeine Problem nennen sie „General Pickup and Delivery Problem“ (GPDP).

Im GPDP soll eine Menge von Routen konstruiert werden, die bestimmte Aufträge bearbeiten. Mehrere Fahrzeuge stehen dazu zur Verfügung. Jedes Fahrzeug hat eine bestimmte Kapazität, einen Anfangsort und einen Endort. Jeder Auftrag benötigt eine bestimmte Kapazität, hat eine Menge von Startknoten und eine Menge von Zielknoten. Jeder Auftrag muß von einem Fahrzeug von der Menge der Startknoten zu der Menge der Zielknoten transportiert werden. Zwischenlagerung (in der Literatur auch „transshipment“ oder Präemption genannt) ist hier nicht erlaubt.

1.2.1 Definition des GPDP

Sei A die Menge der Aufträge, für jeden Auftrag $i \in A$ muß eine Ladung der Größe $\bar{q}_i \in \mathbb{N}$ von der Menge der Startorte V_i^+ zu der Menge der Zielorte V_i^- gebracht

werden. Jede Ladung ist wie folgt unterteilt:

$$\bar{q}_i = \sum_{j \in V_i^+} q_j = - \sum_{j \in V_i^-} q_j.$$

Man kann q_j als Veränderung der Kapazitätsauslastung eines Fahrzeugs interpretieren, wenn es den Ort j anfährt. Sei M die Menge der Fahrzeuge. Jedes Fahrzeug $f \in M$ hat eine Kapazität $k_f \in \mathbb{N}$ einen Startort f^+ und einen Zielort f^-. Sei $V^+ = \bigcup_{i \in A} V_i^+$ die Menge aller Startorte und $V^- = \bigcup_{i \in A} V_i^-$ die Menge aller Zielorte. Sei $V = V^+ \cup V^- \cup \{f^+ | f \in M\} \cup \{f^- | f \in M\}$ die Menge aller Orte. Für alle $i, j \in V$ bezeichnet d_{ij} den Abstand von i und j, t_{ij} die Fahrzeit zwischen i und j und c_{ij} die Kosten einer Fahrt von i nach j.

Definition 1.2.1. Eine *gültige Route* R_f für ein Fahrzeug f ist eine gerichtete Route durch eine Menge von Orten $V_f \subset V$, so daß gilt:

1. R_f beginnt in f^+,
2. $(V_i^+ \cup V_i^-) \cap V_f = \emptyset$ oder $(V_i^+ \cup V_i^-) \cap V_f = V_i^+ \cup V_i^-$ für alle $i \in A$,
3. wenn $V_i^+ \cup V_i^- \subset V_f$, dann werden alle Orte V_i^+ vor allen Orten V_i^- besucht,
4. das Fahrzeug f besucht jeden Ort genau einmal,
5. Die Ladung des Fahrzeugs f übersteigt nie k_f und
6. R_f endet in f^-.

Definition 1.2.2. Ein *Plan* ist eine Menge von Routen $\mathcal{R} = \{R_f | f \in M\}$, so daß

1. R_f ist eine gültige Route für jedes Fahrzeug $f \in M$ und
2. $\{V_f | f \in M\}$ ist eine Partitionierung von V.

1.2.2 Spezialfälle des GPDP

Die Problem, die wir in dieser Arbeit untersuchen, kann man als Spezialfälle des GPDP auffassen. Folgende drei Spezialfälle des GPDP sind gut untersucht:

- Das „*Pickup and Delivery Problem*“ (PDP) hat für jeden Transportauftrag genau einen Start- und Zielknoten und alle Fahrzeuge haben einen gemeinsamen Anfangs- und Endort (das Depot). Diese Problemstellung haben z.B. Savelsbergh und Sol [29] untersucht. In dieser Arbeit wurde ein polyedrischer Ansatz für ein praktisches Problem dieser Art untersucht.

  Ein ähnliches Problem ist mit Lösungsmethoden von Borndörfer, Grötschel und Löbel in [10] beschrieben worden.

- Das „*Dial-a-Ride Problem*“ (DARP) ist ein PDP mit Einheitskapazität der Fahrzeuge und Transportaufträge. Die genaue Beschreibung folgt in Abschnitt 1.3. Ein Beispiel für ein DARP mit mehreren Fahrzeugen, welches ein Problem aus der Praxis als Hintergrund hat, ist z.B. in [9] beschrieben worden.

- Das „*Vehicle Routing Problem*“ (VRP) ist ein GPDP, in dem alle Fahrzeuge ein gemeinsames Depot haben. Dies ist z.B. von Taillard und Laporte [32] untersucht worden.

- Das k-*delivery TSP*, welches von Charikar, Khuller und Raghavachari beschrieben wurde [11] ist ein GPDP mit einem Fahrzeug mit einer Kapazität von k und einem Auftrag. Dieser Auftrag muß von einer Menge von Knoten U zu einer Menge von Knoten W transportiert werden. Dabei sind die $q_j = 1$ für alle $j \in U \cup W$. Das Fahrzeug muss also von jedem Knoten in U ein Objekt der Größe Eins zu einem beliebigen Knoten in W transportieren. In [11] wird ein 5-Approximationsalgorithmus für das Problem die Gesamtlänge der Route des Fahrzeugs zu minimieren angegeben.

In dieser Arbeit wird das „Capacitated Dial-a-Ride Problem“ (cDARP) betrachtet: Dies entspricht dem GPDP mit einem Fahrzeug mit Kapazität $k > 1$. Alle Aufträge haben die Größe 1 und genau einen Start- und Zielort. Start- und Zielort des Fahrzeugs sind gleich. Zusätzlich schränken wir noch die Art des darunterliegenden Graphen ein. Wir untersuchen das Problem nur in bestimmten Arten von Graphen, die für Aufzüge typisch sind.

1.3 Aufzüge mit Einheitskapazität

In diesem Abschnitt stellen wir ein vereinfachtes Modell für einen Aufzug mit einer Kapazität von einem Objekt vor. In seiner Grundform berücksichtigt es nur die Fahrzeiten zwischen verschieden Stockwerken und vernachlässigt Reihenfolgebedingungen der Objekte.

Das hier vorgestellte Problem nennen wir, wie in der Literatur, DARP. DARP steht für „Dial-a-ride-Problem“. Dieser Ausdruck stammt daher, daß dieses Modell ermöglicht, die Situation eines Fahrunternehmens abzubilden, bei dem man telefonisch eine Fahrt von einem Start- zu einem Zielort buchen kann. Das Fahrunternehmen hat das Ziel, mit seinen Fahrzeugen alle Aufträge mit möglichst wenig Leerfahrten zu erledigen. Das DARP ist typischerweise nicht-präemptiv. Das heißt, daß ein Auftrag nur an seinem Zielort abgesetzt werden kann. Es ist also nicht möglich, einen Auftrag zu unterbrechen, um einen anderen „dazwischenzuschieben“.

Für das DARP gibt es polynomiale Algorithmen, die es auf dem Pfad und dem Kreis optimal lösen [6]. Auf Bäumen ist das DARP $\mathcal{NP}$-schwer [15]. Auf speziellen Bäumen, den „caterpillars“, ist das DARP auch $\mathcal{NP}$-schwer [25]. Daß das DARP auf allgemeinen Graphen $\mathcal{NP}$-schwer ist, wurde schon vorher bewiesen [17]. Eine Übersicht über die Forschungsergebnisse findet man in Tabelle 1.1.

Tabelle 1.1: Ergebnisse zum DARP

präemptiv	**Graph**	**Komplexität**	**Quelle**
ja	Pfade und Kreise	$O(m + n)$	[6]
nein	Pfade und Kreise	$O(m + n \log n)$	[12]
ja	Bäume	$O(m + n \log n)$	[15]
nein	Bäume	$\mathcal{NP}$-schwer, approximierbar mit Gütegarantie 5/3	[16]
nein	„caterpillar“	$\mathcal{NP}$-schwer	[25]
nein	allgemein	$\mathcal{NP}$-schwer, approximierbar mit Gütegarantie 9/5	[17]

Der Schwerpunkt dieser Arbeit liegt in der Betrachtung von DARP, in denen ein Fahrzeug mehr als ein Auftrag transportieren kann. Solch ein DARP nennen wir CDARP für „capacitated Dial-a-ride-Problem“. Zunächst einmal untersuchen wir aber das DARP mit Einheitskapazität.

1.3.1 Mathematisches Modell

Wir nennen im Folgenden die Fahrgäste *Objekte*, da mit Fahrgästen in dieser Arbeit in erster Linie Transportgut, wie z.B. Paletten, in einem innerbetrieblichen Logistiksystem gemeint sind.

Gegeben seien ein ungerichteter Graph $G = (V, E)$ und ein Digraph mit gleicher Knotenmenge $D = (V, A)$. Die Mengen der Knoten V in D und G entsprechen den Orten, an denen Objekte abtransportiert und zu denen Objekte befördert werden müssen. Die Menge der Kanten E in G ist die Menge der möglichen Fahrverbindungen zwischen je zwei Knoten. Die Menge der Bögen A in D entspricht der Menge der Objekte, die jeweils von einem Startknoten zu einem Zielknoten transportiert werden müssen. Der Digraph D ist typischerweise nicht einfach. Es kann also mehrere Bögen mit dem gleichen Start- und Zielknoten geben. Start- und Zielknoten eines Bogens $i \in A$ bezeichnen wir mit i^+ bzw. i^-. Man kann einem Bogen i ein Tupel $\langle i^+, i^- \rangle$ zuordnen und die Menge A mit einer Multimenge von Tupeln $\langle u, v \rangle \in V \times V$ identifizieren. Ferner ist eine Kostenfunktion $c : E \to \mathbb{Q}$ gegeben. Sie gibt die Kosten für die Überquerung der Kante E durch das Fahrzeug an. Diese Kostenfunktion erweitern wir auf die Aufträge, indem wir die Kosten $c(a), a \in A$ als die Kosten eines kürzesten Weges von a^+ nach a^- im Graphen (V, E) definieren. Wir schreiben auch $c(v, w)$ für die Kosten eines kürzesten Weges in (V, E) von einem Knoten $v \in V$ zu einem Knoten $w \in V$.

Wir suchen jetzt eine Tour im gemischten Graphen (V, E, A) mit minimalen Fahrtkosten, die jeden Bogen aus A durchläuft, und sowohl im Knoten o startet als auch endet.

Das DARP ist folgendermaßen definiert:

Definition 1.3.1 (DARP). Eine Instanz eines DARP ist durch ein Tupel (V, E, A, c, o) gegeben. Dabei ist V eine Menge von Knoten, E eine Menge von Kanten über V und A eine Menge von Bögen über V. Die Funktion $c : E \cup A \to \mathbb{Q}$ ist eine Kostenfunktion und der Knoten o der Start- und Stopknoten des Fahrzeugs. Wir gehen davon aus, daß $G = (V, E)$ zusammenhängend ist und alle Endpunkte von Bögen aus A enthält. Die Kosten eines Bogens aus A sind gleich den Kosten eines kürzesten Weges in G von einem Endpunkt des Bogens zu dem anderen.

Aufgabenstellung ist es, eine Folge $T = (b_1, b_2, \ldots, b_t)$ von Kanten und Bögen aus $E \cup A$ mit minimalen Kosten $\sum_{i=1}^{t} c(b_i)$ zu finden, so daß

1. T alle Bögen aus A enthält,
2. T eine Tour ist und
3. T im Knoten o beginnt und endet.

Wir betrachten hier nur nicht-präemptive Instanzen des DARP. Zwischenlagerung von Objekten ist also nicht erlaubt.

1.3.2 DARP als Graph-Augmentations-Problem

Das DARP kann äquivalent als Graph-Augmentationsproblem formuliert werden.

Sei W eine gültige Lösung einer Instanz des DARP. Dann induziert W eine Multimenge S von Bögen auf folgende Art: Jedes Mal, wenn ein Kante $(u, v) \in E$ in der Tour W von u nach v überquert wird, fügenwir zu der Multimenge S einen Bogen $\langle u, v \rangle$ hinzu. Der Graph $G[A \cup S]$, der durch die Bögen in $A \cup S$ induziert wird, ist dann eulersch. Wenn wir nämlich parallel zu den Kanten in W die dazugehörigen Bögen im Graphen $G[A \cup S]$ durchlaufen, erhalten wir eine Euler-Tour.

Sei S eine Multimenge von Bögen aus $A(E)$, so daß $G[A \cup S]$ eulersch ist und den Knoten o enthält. Dann erhalten wir eine zulässige Lösung W für unsere Instanz des DARP wie folgt: Wähle eine Euler-Tour C in $G[A \cup S]$, die im Knoten o startet und endet. Die Tour W beginnt dann im Knoten o und folgt C. Wenn der aktuelle Bogen a aus C in A enthalten ist, dann überquert W diesen Bogen, ansonsten traversiert W die Kante aus E, die zu dem Bogen a gehört.

Also haben wir zu jeder gültigen Lösung eines DARP eine zugehörige augmentierende Multimenge S von Bögen, so daß $G[A \cup S]$ eulersch ist und den Knoten o enthält und umgekehrt.

Definition 1.3.2 (DARP als Graph-Augmentations-Problem). Sei eine Instanz eines DARP gegeben, die aus dem Tupel (V, E, A, c, o) wie in der Definition 1.3.1 besteht. Dabei sollen die Kosten jedes Bogens $\langle u, v \rangle \in V \times V$ gleich den Kosten eines kürzesten Pfades von u nach v in $G = (V, E)$ sein.

Finde nun eine Multimenge S von Bögen in $V \times V$ mit minimalen Kosten $c(A \cup S)$, so daß $G[A \cup S]$ eulersch ist und den Knoten o enthält.

Tabelle 1.2: Ergebnisse zum CDARP

präemptiv	**Graph**	**Komplexität**	**Quelle**
egal	allgemein	$\mathcal{NP}$-schwer auch mit $k = 1$	[22], Beweis in [17]
ja	allgemein	approximierbar mit Gütegarantie in $O(\log n \log\log n)$	[22]
egal	Bäume	$\mathcal{NP}$-vollständig	[22]
ja	Bäume	approximierbar mit Gütegarantie 2	[12]
nein	höhenbalancierte Bäume	$8\sqrt{k}$-Approximations-Algorithmus	[12]
nein	Kreise	approximierbar mit Gütegarantie 6	im Abschnitt 2.8
ja	Pfade	$O(m + n)$	[22]
nein	Pfade	$\mathcal{NP}$-vollständig	[22]
		approximierbar mit Gütegarantie 2	[12] ohne Beweis,
		approximierbar mit Gütegarantie 3	im Abschnitt 2.7

1.4 Mehrplatz-Aufzüge

Das „capacitated Dial-a-ride-problem“, kurz CDARP, ist ein verallgemeinertes DARP. Im CDARP darf das Fahrzeug nicht nur ein Objekt auf einmal transportieren sondern auch mehrere. Der praktische Hintergrund sind z.B. Sammeltaxis oder Fördersysteme, die mehrere Objekte auf einmal transportieren können. Dieses Problem ist z.B. in [22] und [11] untersucht worden. Eine Übersicht über Forschungsergebnisse zum CDARP findet man in Tabelle 1.2.

Die Anzahl der Objekte, die auf einmal transportiert werden können, bezeichnen wir mit $k \in \mathbb{N}$. Wollen wir diese Zahl hervorheben, so bezeichnen wir ein CDARP mit Kapazität k auch als k-CDARP. Seien V, D, c und o wie im Abschnitt 1.3 definiert.

1.4.1 Definition des CDARP

Definition 1.4.1 (Bewegung). Wir bezeichnen ein Tripel $(v, w, M) \in V \times V \times \mathcal{P}(A)$ als *Bewegung* des Aufzugs von Knoten v zu Knoten w, bei der die Menge von Objekten M transportiert wird. Jede Bewegung mit $|M| = k$ bezeichnet wir als *vollbeladene* oder auch *vollausgelastete* Bewegung. Eine Bewegung mit $M = \emptyset$ bezeichnen wir als *leere* Bewegung.

Wir sagen, daß ein Objekt i von einem Knoten v zu einem Knoten w mit einer Folge von Bewegungen T *transportiert* wird, wenn für

$$T = ((v, v_1, M_1), (v_1, v_2, M_2), \ldots, (v_{r-1}, w, M_r))$$

gilt, daß i in allen Mengen $M_j, j = 1, \ldots r$ enthalten ist.

Definition 1.4.2 (Transport). Ein *Transport* Q ist eine Folge von Bewegungen,

$$(v_0, v_1, M_1), (v_1, v_2, M_2), \ldots, (v_{r-1}, v_r, M_r).$$

Ein Transport ist *gültig*, wenn zusätzlich gilt:

- $v_0 = v_r$, d.h. der erste und der letzte Knoten des Transports sind derselbe Knoten in V.
- Jedes Objekt wird von einer Teilfolge des Transports von seinem Startknoten zu seinem Zielknoten transportiert und
- $|M_i| \leqslant k$ für alle $i = 1, \ldots, r$, d.h. mit jeder Bewegung werden nie mehr als k Objekte auf einmal transportiert.

Mit $Q(i)$ bezeichnen wir die Teilfolge von Bewegungen eines Transports Q, in der jede Bewegung das Objekt i transportiert. Wir bezeichnen einen Transport als *nicht-präemptiv*, wenn die Teilfolge $Q(i)$, mit der ein Objekt i von seinem Start- zu seinem Zielort transportiert wird, aus direkt hintereinanderliegenden Bewegungen des Transports besteht. In einem nicht-präemptiven Transport dürfen also Objekte nur an ihrem Zielknoten aus der Menge M entfernt werden.

Die Kosten des Transports $c(Q)$ ergeben sich aus der Summe der Kosten der zurückgelegten Strecke, also schreiben wir

$$c(Q) = \sum_{i=1}^{r-1} c(v_i, v_{i+1}).$$

Definition 1.4.3 (CDARP). Ein CDARP ist durch ein Tupel (V, E, A, c, o, k) gegeben. Aufgabe ist es, einen gültigen nicht-präemptiven Transport mit minimalen Kosten zu finden.

In einem präemptiven CDARP ist es die Aufgabe einen gültigen Transport zu finden, der auch präemptiv sein darf. Wir werden im Abschnitt 2.4 sehen, daß das präemptive CDARP polynomial lösbar ist. Ansonsten betrachten wir in dieser Arbeit aber nur das nicht-präemptive CDARP.

1.4.2 CDARP auf Pfaden

Im folgenden betrachten wir den Fall, daß G ein Pfad ist. Denn das CDARP auf Pfaden ist ein vereinfachtes Modell eines Aufzugs mit Kapazität $k \geqslant 1$. Daher bezeichnen wir das Fahrzeug des CDARP auch als Aufzug.

Wir numerieren zur Vereinfachung der Notation die Knoten V von 1 bis n so durch, daß für alle Kanten $(u, v) \in E$ gilt, daß $|u - v| = 1$ ist. Wir sagen, daß ein Knoten u *unter* einem Knoten v liegt, wenn $u < v$ ist. Analog liegt ein Knoten u *über* einem Knoten v, wenn $u > v$ ist. Wir nennen einen Bogen $i \in A$ *Aufwärtsbogen*, wenn $i^+ < i^-$ ist, und wir nennen in *Abwärtsbogen*, wenn $i^+ > i^-$ ist.

Ist ein Digraph $D = (V, A)$ gegeben und wollen wir nur die Aufwärtsbögen in diesem Graphen betrachten, so nennen wir diesen verkleinerten Graphen D^+. Analog nennen wir den Digraphen, der nur die Abwärtsbögen von D enthält D^-. Es gilt also

$$D^+ := (V, \{a \in A : a^+ < a^-\}) \text{ und } D^- := (V, \{a \in A : a^+ > a^-\})$$

Ein Bogen i aus A *überdeckt* eine Kante $(v, v+1) \in e$, wenn $i^+ \leqslant v < v+1 \leqslant i^-$ oder $i^- \leqslant v < v+1 \leqslant i^+$ gilt. Analog sagen wir, daß ein gerichteter Pfad oder eine Menge von Bögen eine Menge von Kanten F überdeckt, wenn jede Kante in F von einem Bogen des Pfades oder einem Bogen aus der Menge von Bögen überdeckt wird.

Sei A eine Menge von Bögen. Diese Bögen überdecken eine Menge von Kanten E. Den durch diese Menge induzierten Graphen $G[E]$ nennen wir den A *unterliegenden* Graphen.

1.5 Erweiterungen des CDARP

Die bisher vorgestellten Modelle reichen nicht aus, um in der Praxis auftretende reale Systeme umfassend abzubilden. Wichtige Einschränkungen wie bestimmte einzuhaltende Reihenfolgen der Objekte oder bisher nicht berücksichtigte Laufzeiten des Aufzugs oder anderer Komponenten des Fördersystems, können die Verwendbarkeit eines Modells für das tatsächliche Problem erheblich beeinflussen. Wir untersuchen nun, wie solche Nebenbedingungen in das CDARP integriert werden können.

1.5.1 FIFO-Präzedenzen

Das FIFO-CDARP ist eine Verallgemeinerung des CDARP bzw. des FIFO-DARP aus [25]. Im FIFO-CDARP werden die möglichen Abarbeitungsreihenfolgen der Objekte eingeschränkt. Dadurch kann man Warteschlangen vor den Aufzügen modellieren.

Im FIFO-cDARP wie auch im FIFO-DARP haben wir zusätzlich zu dem Tupel (V, E, A, c, o, k) aus der Definition 1.4.3 für jeden Knoten $v \in V$ eine partielle Ordnung $\prec_v$ auf den an diesem Knoten liegenden Objekten. Die Objekte auf diesem Knoten dürfen nur in einer Reihenfolge dieser Ordnung abgearbeitet werden.

Es gilt also: Ein gültiger Transport

$$T = ((v, v_1, M_1), (v_1, v_2, M_2), \ldots, (v_{r-1}, w, M_r))$$

respektiert die partielle Ordnung $\prec_v$, wenn je zwei Aufträge a und a', die an einem Knoten v starten und für die $a \prec_v a'$ gilt, entweder in der gleichen Bewegung aufgeladen werden, oder der Auftrag a' in einer Bewegung aufgeladen wird, die hinter der Bewegung liegt, in der der Auftrag a aufgeladen worden ist. In Formeln heißt das:

$$a \prec_v a', a \in M_i, a' \in M_{i'} \Rightarrow i \leqslant i'.$$

Definition 1.5.1 (FIFO-Ordnung). Wir nennen eine partielle Ordnung $\prec$ auf der Menge von Bögen A auf einem Graphen $D = (V, A)$ FIFO-*Ordnung*, wenn aus $a \prec a'$, $a, a' \in A$ folgt, daß daß a und a' an demselben Knoten starten.

Eine FIFO-Ordnung ist die Vereinigung einer Menge von partiellen Ordnungen $\prec_v$, $v \in V$.

Der Hintergrund für die Einführung von FIFO-Ordnungen ist, daß man damit Warteschlangen, an Beladestationen von einem Vertikalförderer modellieren kann, die nur nach dem FIFO-Prinzip abgearbeitet werden können, d.h. Objekte können nur von der Spitze der Warteschlange in den Aufzug befördert werden. Das Aufzugssystem bei Herlitz hat nur Warteschlangen der Länge Eins, aber für den allgemeinen Fall ist dies eine sinnvolle Erweiterung des cDARP.

Wir definieren das FIFO-cDARP wie folgt:

Definition 1.5.2 (FIFO-cDARP). Eine Instanz des FIFO-cDARP ist gegeben durch ein Tupel $(V, E, A, c, o, k, \prec)$. Dabei sind V, E, A, c, o und k wie in 1.4.3 definiert. Zusätzlich ist $\prec = \bigcup_{v \in V} \prec_v$ eine FIFO-Ordnung auf der Menge der Knoten V. Gesucht wird ein gültiger Transport mit minimalen Kosten, der alle partiellen Ordnungen $\prec_v$, $v \in V$ respektiert.

Das Hinzufügen von FIFO-Bedingungen für das Aufnehmen der Aufträge kann die Abarbeitungszeit verlängern, da wir dadurch die Menge der gültigen Transporte einschränken.

1.5.2 Start- und Stopzeiten

Bisher haben wir nur die Zeiten betrachtet, die der Aufzug benötigt, um von einem Stockwerk zu einem anderen zu fahren. Dabei haben wir die Zeiten vernachlässigt, die der Aufzug benötigt um an einem Stockwerk zu halten und Objekte einzuladen und wieder abzufahren. Diese zusätzlich benötigte Zeit wird durch Beschleunigung und Abbremsen des Aufzugs, Positionierzeit und ähnliches verursacht. Wir stellen nun ein Modell vor, welches diese Start- und Stopzeiten des Aufzugs berücksichtigt.

Definition 1.5.3 (CDARP mit Start- und Stopzeiten). Eine Instanz des CDARP mit Start- und Stopzeiten besteht aus denselben Eingangsdaten wie eine Instanz des CDARP ergänzt um Funktionen $p^+, p^- : V \to \mathbb{R}_{\geqslant 0}$ auf der Menge der Knoten V. Dabei gibt $p^+(v)$ die Zeit an, die der Aufzug benötigt, um vom Knoten v zu starten, und $p^-(v)$ gibt die Zeit an, die der Aufzug benötigt, um am Knoten v zu halten. Ziel ist es einen gültigen Transport $T = \{(v_1, v_2, M_1), (v_2, v_3, M_2), \ldots, (v_s, v_{s+1}, M_s)\}$ zu finden, der minimale Kosten

$$c(T) = \sum_{j=1}^{s} c(v_j, v_{j+1}) + \sum_{j=1}^{s} p^+(v_j) + \sum_{j=2}^{s+1} p^-(v_j)$$

verursacht.

Das CDARP mit Start- und Stopzeiten ist eine Verallgemeinerung des PENALTY-DARP aus [25].

Da in einem gültigen Transport Start- und Endknoten des Transports T gleich sind, gilt $v_1 = v_{s+1}$. Also gilt auch $p^-(v_1) = p^-(v_{s+1})$ und damit auch

$$\sum_{j=2}^{s+1} p^-(v_j) = \sum_{j=1}^{s} p^-(v_j). \tag{1.1}$$

Dieses Problem ist äquivalent zum CDARP auf einem etwas größeren Graphen. Und zwar führen wir folgende Transformation durch: Sei eine Instanz des CDARP mit Start- und Stopzeiten $(V, E, A, c, o, p^+, p^-)$ gegeben. Dann konstruieren wir eine Instanz des CDARP (V', E', A', c', o) wie folgt: Wir konstruieren einen neuen Graphen (V', E') aus (V, E), indem wir für jeden Knoten $v \in V$ diesen Knoten v und einen Knoten v' zu V' hinzufügen. Die Kantenmenge E' besteht aus allen Kanten in E und je einer Kante (v, v') für jeden Knoten $v \in V$. Die Kosten dieser neuen Kanten betrage $c'(v, v') := \frac{1}{2}(p^+(v) + p^-(v))$. Ansonsten sei $c'(e) := c(e)$ für alle Kanten $e \in E$. Die Menge der Objekte A' des neuen Problems konstruieren wir aus A, indem wir für jedes Objekt $i \in A$ ein Objekt $j \in A'$ hinzufügen, wobei wir $\langle j^+, j^- \rangle := \langle (i^+)', (i^-)' \rangle$ setzen. Start- und Endknoten der neuen Objekte sind jetzt die Knoten in V', die nicht in V enthalten sind.

Lemma 1.5.4. *Für eine Instanz des* CDARP *mit Start- und Stopzeiten* $P = (V, E, A, c, o, p^+, p^-)$ *und eine Instanz des* CDARP $P' = (V', E', A', c', o)$*, die wie oben aus* P *konstruiert wurde, und jeden für diese Instanz* P *gültigen Transport* $T = \{(v_1, v_2, M_1), (v_2, v_3, M_2), \ldots, (v_s, v_{s+1}, M_s)\}$ *gilt:*

$$\sum_{j=1}^{s} c'(v_j, v_{j+1}) + 2\sum_{j=1}^{s} c'(v_j, v'_j) = \sum_{j=1}^{s} c'(v'_j, v'_{j+1}).$$

Beweis. Wir können einen kürzesten Weg von einem Knoten v'_i zu einem Knoten v'_j, $v'_i, v'_j \in V' \setminus V$ zerlegen in die Kante (v'_i, v_i), den kürzesten Weg von v_i nach v_j und

die Kante (v'_j, v_j), da die Knoten in $V' \setminus V$ nur zu jeweils genau einer Kante adjazent sind. Daher betragen dann auch die Kosten

$$c'(v'_i, v'_j) = c'(v'_i, v_i) + c'(v_i, v_j) + c'(v'_j, v_j).$$

Da nun $v_i = v_{s+1}$ ist, folgt die Behauptung. □

Es gilt für einen minimalen gültigen Transport

$$T = \{(v_1, v_2, M_1), (v_2, v_3, M_2), \ldots, (v_s, v_{s+1}, M_s)\}$$

für die Instanz des CDARP mit Start-/Stopzeiten $P := (V, E, A, c, o, p^+, p^-)$ und einen minimalen gültigen Transport

$$T' = \{(v'_1, v'_2, M_1), (v'_2, v'_3, M_2), \ldots, (v'_s, v'_{s+1}, M_s)\}$$

für die aus P wie oben konstruierte Instanz des CDARP $P' := (V', E', A', c', o)$:

$$\begin{aligned}
c(T) &= \sum_{j=1}^{s} c(v_j, v_{j+1}) + \sum_{j=1}^{s} p^+(v_j) + \sum_{j=2}^{s+1} p^-(v_j), && \text{nach Def. 1.5.3}\\
&= \sum_{j=1}^{s} c(v_j, v_{j+1}) + \sum_{j=1}^{s} (p^+(v_j) + p^-(v_j)), && \text{wegen (1.1)}\\
&= \sum_{j=1}^{s} c(v_j, v_{j+1}) + 2\sum_{j=1}^{s} \frac{p^+(v_j) + p^-(v_j)}{2}\\
&= \sum_{j=1}^{s} c'(v_j, v_{j+1}) + 2\sum_{j=1}^{s} c'(v_j, v'_j), && \text{nach Konstruktion von } c'\\
&= \sum_{j=1}^{s} c'(v'_j, v'_{j+1}), && \text{wegen Lemma 1.5.4}\\
&= c'(T')
\end{aligned}$$

Da nun die oben beschriebene Transformation offensichtlich in polynomialer Zeit durchführbar ist, gilt folgender Satz:

Satz 1.5.5. *Ist* $P := (V, E, A, c, o, p^+, p^-)$ *eine Instanz des* CDARP *mit Start- und Stopzeiten und* $P' := (V', E', A', c', o)$ *eine wie oben aus* P *konstruierte Instanz des* CDARP*, so kann man einen gültigen Transport* T *für* P *in polynomialer Zeit in einen gültigen Transport* T' *für* P' *mit gleichen Kosten transformieren und umgekehrt.*

Das CDARP mit Start- und Stopzeiten ist also kein neues Problem, sondern auch ein CDARP. Probleme ergeben sich nur dadurch, daß z.B. zu einem CDARP auf einem Pfad das zugehörige CDARP mit Start- und Stopzeiten als unterliegenden Graph einen Baum hat. Für dieses ($\mathcal{NP}$-vollständige) Problem aber haben wir keinen Approximationsalgorithmus mit konstanter Gütegarantie gefunden. Das Einführen von Start- und Stopzeiten erschwert uns das Lösen der Probleminstanzen.

1.6 Rundreisen und CDARP

Man kann das CDARP auch als ein Rundreiseproblem, besser bekannt als „Travelling Salesman Problem“ mit zusätzlichen Nebenbedingungen auffassen. Dazu konstruieren wir einen Graphen $G = (V, E)$, indem wir für jeden Auftrag zwei Knoten hinzufügen, jeweils einen Knoten für den Startort und einen Knoten für den Zielort. Dieser Graph sei vollständig. Das Gewicht einer Kante $(i, j) \in E$, sei die Zeit, die das Fahrzeug benötigt, um vom Knoten i zum Knoten j zu fahren. Wir wollen nun eine minimale Tour finden, die in einem gegebenen Knoten beginnt und endet und alle Knoten des Graphen G besucht und dabei einen Endknoten i^- eines Auftrags i nicht vor seinem Startknoten i^+ anfährt.

Wir betrachten zuerst den Fall, daß das Fahrzeug eine beliebig große Kapazität hat.

Dieses Problem ist als „sequential ordering problem“ (SOP) oder als „Traveling Salesman Problem“ mit Präzedenzbedingungen bekannt. Es ist sowohl mit symmetrischer als auch mit asymmetrischer Kostenfunktion untersucht worden. Eine Implementation eines Branch&Cut-Algorithmus für das asymmetrische Problem ist in [3] beschrieben. Das symmetrische SOP ist als „pickup-and-delivery problem“ (PDP) in [28] beschrieben worden.

Man kann das SOP leicht erweitern, so daß es das CDARP auf allgemeinen Graphen beschreibt, indem man Nebenbedingungen für die Kapazitätsbeschränkung des Fahrzeugs einführt. Im Abschnitt 3.2 stellen wir vor, wie man solche Nebenbedingungen formuliert und auch im Rahmen eines Branch&Cut-Algorithmus separieren kann. Auch FIFO-Präzedenzen für die Aufträge stellen kein Problem dar. Diese vergrößern nur den ohnehin benötigten Präzedenzgraphen des SOP.

1.7 Das Herlitzproblem

Wir können einen Aufzug im Materialfördersystem der Lagers von Herlitz in Falkensee, welches wir im Abschnitt 1.1.2 vorgestellt haben als ein FIFO-CDARP auf einem Baum modellieren, wobei wir die Anzahl der Aufträge, die in einem Stockwerk beginnen dürfen auf einen begrenzen. Die Fahrzeiten des Aufzugs haben wir aus [21] entnommen. Sie betragen vier Sekunden für das Fahren von einem Stockwerk zum nächstgelegenen. Das Starten und Stoppen des Aufzugs dauert fünf Sekunden. Diese Zeiten sind also im Verhältnis zu den Fahrzeiten nicht zu vernachlässigen.

Kapitel 2

Komplexität und Algorithmen

Aufbauend auf den in Kapitel 1 vorgestellten Modellen untersuchen wir die Lösbarkeit des CDARP. Dabei zeigen wir, daß das DARP und das präemptive CDARP auf Pfaden polynomial lösbar sind. Das nicht-präemptive CDARP hingegen ist schon mit einer Kapazität von zwei Objekten auf Pfaden $\mathcal{NP}$-schwer.

Anschließend sehen wir uns für das CDARP einige Heuristiken und Approximationsalgorithmen an. Unser wichtigstes Ergebnis ist der Algorithmus COVERPATH für das CDARP auf Pfaden. Er hat eine polynomiale Laufzeit und eine konstante Gütegarantie. Aus diesem leiten wir einen 6-Approximationsalgorithmus für das CDARP auf Kreisen ab.

Mit der Gütegarantie für den Algorithmus COVERPATH werden wir zeigen, daß auch der Algorithmus INTERVALLGRAPH eine Gütegarantie von 3 besitzt. Dieser Algorithmus läßt sich zu einer Heuristik für das FIFO-CDARP erweitern.

2.1 CDARP auf Pfaden ist $\mathcal{NP}$-schwer

In diesem Abschnitt zeigen wir, daß das CDARP $\mathcal{NP}$-schwer ist, daß also, außer falls $\mathcal{NP} = \mathcal{P}$ gilt, kein Algorithmus existiert, der das CDARP in polynomialer Zeit optimal löst.

Wir werden das 3-Erfüllbarkeitsproblem auf das 2/3-Erfüllbarkeitsproblem zurückführen. Dieses läßt sich in polynomialer Zeit auf ein 2-CDARP-Entscheidungsproblem reduzieren. Das 3-Erfüllbarkeitsproblem ist aber nach [19] (3-SAT) $\mathcal{NP}$-vollständig. Daher ist auch das CDARP $\mathcal{NP}$-schwer.

Im wesentlichen vollziehen wir den Beweis von Guan [22] nach. Ein wenig haben wir die Konstruktion der Instanz des CDARP abgewandelt. Dadurch fällt der Beweis leichter, daß man aus einer Lösung unserer Instanz des CDARP auch eine Lösung des 2/3-Erfüllbarkeitsproblems konstruieren kann.

2.1.1 2/3-SAT ist $\mathcal{NP}$-vollständig

Im folgenden nennen wir das Erfüllbarkeitsproblem wie in der englischsprachigen Literatur SAT.

Definition 2.1.1 (SAT). Sei eine Menge von Variablen U und eine Menge von Klauseln C über U gegeben. Jede Variable kann den Wert „wahr“ oder „falsch“ annehmen. Eine Literal ist eine Variable oder eine negierte Variable. Eine Klausel ist eine Disjunktion von Literalen. Innerhalb einer Klausel sind die Literale also mit einem logischen „Oder“ verknüpft, die einzelnen Klauseln sind mit logischem „Und“ verknüpft.

Nun soll entschieden werden, ob eine Belegung der Variablen mit Wahrheitswerten existiert, so daß jede Klausel erfüllt ist.

Eine Lösung einer Instanz des SAT ist eine Zuordnung von Wahrheitswerten zu den Variablen, so daß jede Klausel mindestens ein Literal mit dem Wahrheitswert „wahr“ enthält.

Satz 2.1.2. SAT *ist $\mathcal{NP}$-vollständig.*

Beweis. Siehe etwa [19]. □

Als Symbol für die Oder-Verknüpfung in diesem Zusammenhang benützen wir das „+“. Die Und-Verknüpfung wird durch Multiplikation ausgedrückt. Eine negierte Variable kennzeichnen wir durch einen Querstrich über der Variable. Ein Instanz von SAT sieht beispielsweise folgendermaßen aus:

$$\underbrace{(x_1 + \bar{x}_2)}_{=:c_1}\underbrace{(\bar{x}_1 + x_3 + x_4)}_{=:c_2}\underbrace{(x_2 + \bar{x}_3 + x_4 + x_5)}_{=:c_3}$$

Wir gehen vom 3-SAT aus: In dieser Variante enthält jede Klausel genau drei Literale. Auch das 3-SAT ist $\mathcal{NP}$-schwer [19]. Zunächst führen wir 3-SAT auf eine andere Variante des SAT zurück, diese nennen wir 2/3-SAT. Diese schließlich reduzieren wir auf das 2-CDARP-Entscheidungsproblem. Wir werden zeigen, daß diese beiden Transformationen Polynomialzeitreduktionen sind. Daraus folgt, daß das 2-CDARP-Entscheidungsproblem $\mathcal{NP}$-vollständig ist. Damit ist auch das allgemeine CDARP $\mathcal{NP}$-schwer.

Das Problem 2/3-SAT ist eine eingeschränkte Version des SAT. Es besteht aus den Instanzen, in denen jede Klausel entweder genau zwei oder drei Literale enthält. Ferner darf kein Paar von verschiedenen Literalen in mehr als einer Klausel auftreten.

Lemma 2.1.3. *2/3-SAT ist $\mathcal{NP}$-vollständig.*

Beweis. Es ist klar, daß 2/3-SAT in $\mathcal{NP}$ ist. Das Zertifikat ist eine gültige Zuweisung von Wahrheitswerten an die Variablen. Nun reduzieren wir das 3-SAT in polynomialer Zeit auf das 2/3-SAT, um zu zeigen, daß das 2/3-SAT $\mathcal{NP}$-schwer ist. Sei eine Instanz des 3-SAT gegeben. Jetzt führen wir folgende Schritte zur Transformation durch:

1. Wir eliminieren alle doppelten Klauseln.
2. Wir ersetzen jedes Paar von Klauseln $(u + v + w)$ und $(u + v + \bar{w})$ durch die Klausel $(u + v)$.
3. Für jedes Paar von unterschiedlichen Literalen u und v betrachten wir die Menge der Klauseln $\{(u + v + w_1), (u + v + w_2), \ldots, (u + v + w_t)\}$, die u und v enthalten. Ist $t > 1$ ersetzen wir diese Menge durch die Menge von Klauseln $\{(u + v + y_i), (\bar{y}_i + w_1), (\bar{y}_i + w_2), \ldots, (\bar{y}_i + w_t)\}$. Dabei ist y_i eine neue Variable. So kann kein Paar von verschiedenen Literalen in mehr als einer Klausel auftreten.

Diese Transformation ist offenbar in polynomialer Zeit durchführbar. Nun ist noch zu zeigen, daß es eine gültige Zuweisung für die ursprünglichen Terme genau dann gibt, wenn es auch eine für die neuen Terme gibt.

Angenommen wir haben eine gültige Zuweisung von Wahrheitswerten für die Variablen des 3-SAT. Betrachten wir nun die Klausel $(u + v)$ mit den Literalen aus Schritt 3. Hat diese Klausel mit der gültigen Zuweisung den Wahrheitswert „wahr“, dann wählen wir für die Variable y_i die Zuweisung „falsch“. Ist andererseits $(u + v)$ mit der gültigen Zuweisung „falsch“, wählen wir für y_i den Wahrheitswert „wahr“. Mit dieser Zuweisung folgt dann, daß die Klauseln $\{(u+v+w_1), (u+v+w_2), \ldots, (u+v+w_t)\}$ genau dann wahr sind, wenn auch die Klauseln $\{(u + v + y_i), (\bar{y}_i + w_1), (\bar{y}_i + w_2), \ldots, (\bar{y}_i + w_t)\}$ wahr sind. □

2.1.2 Reduktion von 2/3-SAT auf das cDARP

Wir betrachten nun das cDARP-Entscheidungsproblem P mit einer Kapazität von 2 auf dem gewichteten Pfad $P = (V, E)$, mit einer Menge von Objekten A, einem Startknoten i^+ und einem Zielknoten i^- für jedes Objekt $i \in A$, einem Startknoten $v_0 \in V$ für den Aufzug, der an einem Ende es Pfades P liegt, und Kosten $c_0 := \frac{1}{2}\sum_{i\in A} c(i)$. Die Frage ist, ob ein Transport mit Kosten von höchstens c_0 existiert. Wir können dieses Entscheidungs-Problem P also als ein Tupel (V, E, A, v_0, c, c_0) auffassen.

Idee der Reduktion

Die Kosten c_0 unter denen die Kosten eines Transports der hier konstruierten Instanz des cDARP liegen sollen, sind so gewählt, daß jede Lösung dieser Instanz

des CDARP, die Kosten nicht höher als c_0 hat, nur aus vollbeladenen Bewegungen bestehen kann.

Der Auftragsgraph des CDARP ist nun so konstruiert, daß für jede Klausel des 2/3-SAT ein Transport generiert werden muß, der einen zusammenhängenden Teilgraphen des Graphen P zweimal von oben nach unten und zweimal von unten nach oben durchfahren muß. Der Beginn jedes dieser Transporte kann nur an einem Knoten liegen, der einer Belegung einer Variable des 2/3-SAT zugeordnet ist. Beginnt ein solcher Transport zum Beispiel an dem Knoten x_i^0, so heißt das, daß die Variable x_i den Wahrheitswert „falsch" zugewiesen bekommt. Da jede Klausel mit einer gültigen Belegung „wahr" sein muß, muß in jeder Klausel mindestens ein Literal „wahr" sein. Dieses „wahre" Literal entspricht genau dem Startknoten des Transports für die Klausel. Kann man also im Knoten x_i^0 einen Transport für eine Klausel beginnen, so enthält diese Klausel das Literal $\bar{x}_i$.

Die einzelnen Transporte für die Klauseln, werden in einen Transport eingefügt, der zweimal vom Startknoten s_0 zum Knoten s_n und zurück geht.

Aufbau des CDARP-Entscheidungsproblems

Nun erzeugen wir die Instanz des CDARP-Entscheidungsproblems wie folgt: Der Pfad P habe $2mn+3n+1$ Knoten. Zuerst gibt es $n+1$ Knoten $s_0, s_1, \ldots, s_n$. Zwischen je zwei dieser Knoten s_{i-1} und s_i, $i = 1, \ldots, n$ liegen alle Knoten die einen Bezug zur Variablen x_i haben.

Dies sind einerseits die Knoten, die direkt einer Belegung einer Variablen x_i mit einem Wahrheitswert entsprechen. Für jede Variable $x_i, i = 1, \ldots, n$ gibt es zwei Knoten x_i^0 und x_i^1. Der Knoten x_i^0 steht für die Belegung der Variable x_i mit „wahr", der Knoten x_i^1 für die Belegung mit „falsch".

Andererseits gibt es zwischen je zwei Knoten s_{i-1} und s_i, $i = 1, \ldots, n$ insgesamt $2m$ Knoten, die sowohl einer Variablen x_i, $i = 1, \ldots, n$, als auch einer der m Klauseln c_j, $j = 1, \ldots, m$, zugeordnet sind. Diese $2m$ Knoten entsprechen dem unteren und dem oberen Ende eines Teiltransports für eine Klausel. Die Variablen $l_i^1, l_i^2, \ldots, l_i^m$ stehen für das untere Ende, die Variablen $r_i^1, r_i^2, \ldots, r_i^m$ für das obere. Tatsächlich beginnen und enden nur an einem Teil der Knoten $l_i^1, l_i^2, \ldots, l_i^m$ und $r_i^1, r_i^2, \ldots, r_i^m$ Aufträge. Die anderen Knoten sind überflüssig, vereinfachen aber die Notation. Am Knoten r_i^j beginnt ein Auftrag für die Klausel j, wenn deren Variable mit größtem Index x_i ist. An den Knoten l_i^j endet ein Auftrag für eine Klausel j, wenn deren Variable mit kleinstem Index x_i ist.

Es gibt also insgesamt $2m + 2$ Knoten zwischen jedem Paar von Knoten s_{i-1} und s_i, $i = 1, 2, \ldots, n$. Diese sind von unten nach oben in folgender Reihenfolge angeordnet:

$$s_{i-1}, l_i^1, l_i^2, \ldots, l_i^m, x_i^0, x_i^1, r_i^1, r_i^2, \ldots, r_i^m, s_i.$$

Die Entfernung zwischen zwei benachbarten Knoten betrage Eins. Nun haben wir V und E festgelegt, wir müssen also nur noch die Menge der Bögen bzw. Objekte A konstruieren. Die Objekte $p_1, p_2, \ldots, p_{3n}, f_1, f_2, \ldots, f_{2n}, t_1, t_2, \ldots, t_{2n}$, entsprechen einer Belegung der Variablen in U. Und zwar stehen die Objekte $f_i, i = 1 \ldots 2n$

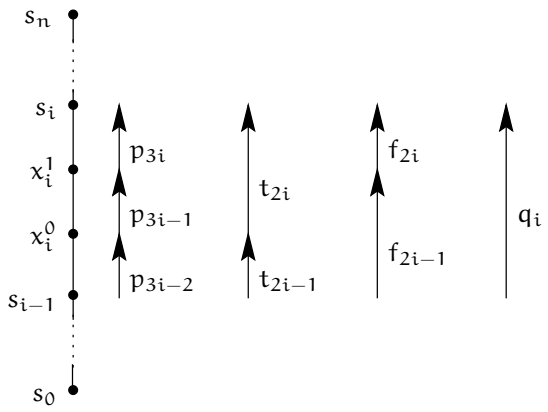

Abbildung 2.1: Schematische Darstellung der Objekte für eine Variable x_i

für die Belegung einer Variable mit dem Wahrheitswert „falsch“ und die Objekte $t_i, i = 1 \dots 2n$ für die Belegung einer Variable mit dem Wahrheitswert „wahr“. Immer, wenn zwei aufeinanderfolgende Objekte t_{2i-1} und t_{2i} bzw. f_{2i-1} und f_{2i} zusammen mit den Objekten p_{3i-2}, p_{3i-1}, p_{3i} in aufeinanderfolgenden Bewegungen in einem optimalen Transport befördert werden, entspricht dies einer Belegung einer Variablen x_i, für $i = 1, \dots, n$ mit „wahr“ bzw. „falsch“.

Die Objekte $q_1, q_2, \dots, q_n$ und die Objekte r_1, r_2, r_3 und r_4 balancieren das Problem.

Jetzt benötigen wir noch Objekte, die in einer Lösung mit Kosten von c einen Teiltransport für eine Klausel bilden. Für $j = 1, 2, \dots, m$ entsprechen Objekte den Klauseln c_j. Die Menge der Objekte für eine Klausel c_j, $j = 1, 2, \dots, m$ bezeichnen wir mit C_j. Für eine Klausel c_j mit zwei Literalen gilt:

$$C_j = \{\alpha_1^j, \alpha_2^j, \beta_1^j, \beta_2^j, \gamma_1^j, \gamma_2^j, \delta_1^j, \delta_2^j, \eta_1^j, \eta_2^j, \eta_3^j, \eta_4^j\}. \tag{2.1}$$

Für eine Klausel c_j mit drei Literalen gilt

$$C_j = \{\alpha_1^j, \alpha_2^j, \beta_1^j, \beta_2^j, \gamma_1^j, \gamma_2^j, \delta_1^j, \delta_2^j, \epsilon_1^j, \epsilon_2^j, \zeta_1^j, \zeta_2^j, \eta_1^j, \eta_2^j, \eta_3^j, \eta_4^j\}. \tag{2.2}$$

Diese Objekte sind so konstruiert, daß immer genau zwei von ihnen gemeinsam an einem der Knoten x_i^0 oder $x_i^1, i = 1, \dots, n$ starten, der einer Belegung einer Variablen x_i mit einem Wahrheitswert derart entspricht, daß das entsprechende Literal in c_j „wahr“ ist. Die Objekte η_1^j, η_2^j, η_3^j, η_4^j balancieren das Problem. Im Folgenden schreiben wir etwas verkürzt $a := \langle u, v \rangle$ statt $\langle a^+, a^- \rangle := \langle u, v \rangle$. Wir wollen aber zu einem Objekt Start- und Endknoten festlegen.

Jedem Objekt weisen wir einen Startknoten und einen Endknoten zu. Und zwar

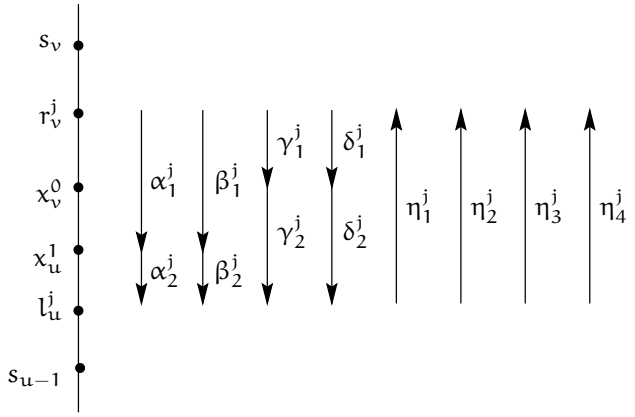

Abbildung 2.2: Schematische Darstellung der Objekte für die Klausel $c_j := (x_u + \bar{x}_v)$

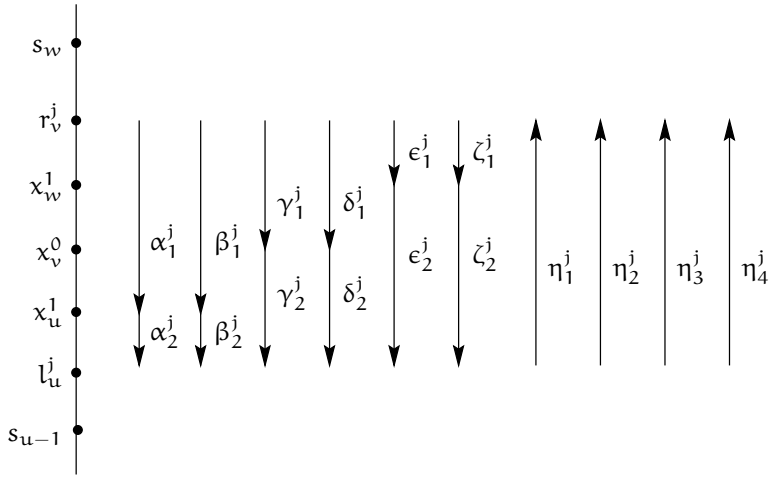

Abbildung 2.3: Schematische Darstellung der Objekte für die Klausel $c_j := (x_u + \bar{x}_v + x_w)$

folgendermaßen wie in Abbildung 2.1.2: Für $i = 1, 2, \ldots, n$ gilt

$$\begin{aligned}
p_{3i-2} &:= \langle s_{i-1}, x_i^0 \rangle, & p_{3i-1} &:= \langle x_i^0, x_i^1 \rangle, & p_{3i} &:= \langle x_i^1, s_i \rangle, \\
f_{2i-1} &:= \langle s_{i-1}, x_i^0 \rangle, & f_{2i} &:= \langle x_i^0, s_i \rangle, & & \\
t_{2i-1} &:= \langle s_{i-1}, x_i^1 \rangle, & t_{2i} &:= \langle x_i^1, s_i \rangle, & & \\
q_i &:= \langle s_{i-1}, s_i \rangle & & & &
\end{aligned}$$

und außerdem gilt

$$\begin{aligned}
r_1 &:= \langle s_n, s_0 \rangle, & r_2 &:= \langle s_n, s_0 \rangle, & r_3 &:= \langle s_n, s_0 \rangle, \\
r_4 &:= \langle s_n, s_0 \rangle & & & &
\end{aligned}$$

Sei z ein Literal, sei i_z der Index der Variable, die zum Literal z gehört. Sei ν_z gleich Eins, wenn z nicht negiert ist. Ist z die Negation einer Variable, sei ν_z gleich

Null. Enthält also eine Klausel das Literal $z = \bar{x}_i$, so ist $i_z = i$ und $\nu_z = 0$. Für jede Klausel $c_j \in C$ mit zwei Literalen u und v bestimmen wir Start- und Endknoten der zugehörigen Objekte wie in Abbildung 2.1.2 und wie folgt: Sei o.B.d.A. $i_u < i_v$.

$$\begin{aligned}
\alpha_1^j &:= \langle r_{i_v}^j, x_{i_v}^{\nu_v} \rangle, & \alpha_2^j &:= \langle x_{i_v}^{\nu_v}, l_{i_u}^{\nu_u} \rangle, \\
\beta_1^j &:= \langle r_{i_v}^j, x_{i_v}^{\nu_v} \rangle, & \beta_2^j &:= \langle x_{i_v}^{\nu_v}, l_{i_u}^{\nu_u} \rangle, \\
\gamma_1^j &:= \langle r_{i_v}^j, x_{i_u}^{\nu_u} \rangle, & \gamma_2^j &:= \langle x_{i_u}^{\nu_u}, l_{i_u}^j \rangle, \\
\delta_1^j &:= \langle r_{i_v}^j, x_{i_u}^{\nu_u} \rangle, & \delta_2^j &:= \langle x_{i_u}^{\nu_u}, l_{i_u}^j \rangle, \\
\eta_1^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, & \eta_2^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, \\
\eta_3^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, & \eta_4^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle
\end{aligned}$$

Für jede Klausel mit drei Literalen u, v und w bestimmen wir Start- und Endknoten der zugehörigen Objekte wie in Abbildung 2.1.2 und wie folgt: Sei o.B.d.A. $i_u < i_v < i_w$.

$$\begin{aligned}
\alpha_1^j &:= \langle r_{i_w}^j, x_{i_w}^{u_w} \rangle, & \alpha_2^j &:= \langle x_{i_w}^{\nu_w}, l_{i_u}^{\nu_u} \rangle, \\
\beta_1^j &:= \langle r_{i_w}^j, x_{i_w}^{\nu_w} \rangle, & \beta_2^j &:= \langle x_{i_v}^{\nu_v}, l_{i_u}^{\nu_u} \rangle, \\
\gamma_1^j &:= \langle r_{i_w}^j, x_{i_v}^{\nu_v} \rangle, & \gamma_2^j &:= \langle x_{i_v}^{\nu_v}, l_{i_u}^j \rangle, \\
\delta_1^j &:= \langle r_{i_w}^j, x_{i_v}^{\nu_v} \rangle, & \delta_2^j &:= \langle x_{i_v}^{\nu_v}, l_{i_u}^j \rangle, \\
\epsilon_1^j &:= \langle r_{i_w}^j, x_{i_u}^{\nu_u} \rangle, & \epsilon_2^j &:= \langle x_{i_u}^{\nu_u}, l_{i_u}^j \rangle, \\
\zeta_1^j &:= \langle r_{i_w}^j, x_{i_u}^{\nu_u} \rangle, & \zeta_2^j &:= \langle x_{i_u}^{\nu_u}, l_{i_u}^j \rangle, \\
\eta_1^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, & \eta_2^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, \\
\eta_3^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle, & \eta_4^j &:= \langle l_{i_u}^j, r_{i_v}^j \rangle
\end{aligned}$$

Nun sei noch der Knoten s_0 der Startknoten und c sei die Hälfte der Abstände vom Startknoten zum Zielknoten jedes Objektes in A, also ist, wie oben erwähnt, $c := \frac{1}{2} \sum_{i \in A} c(i)$.

Beispiel

Wir wollen die Konstruktion eines cDARP aus einem 2/3-SAT an einem Beispiel durchführen. Folgende Probleminstanz haben wir: Die Menge der Variablen sei $U := \{x_1, x_2, x_3\}$. Die Menge der Klauseln ist $C := \{(x_1 + x_2), (x_1 + \bar{x}_2 + x_3)\}$. Damit ergibt sich folgendes cDARP:

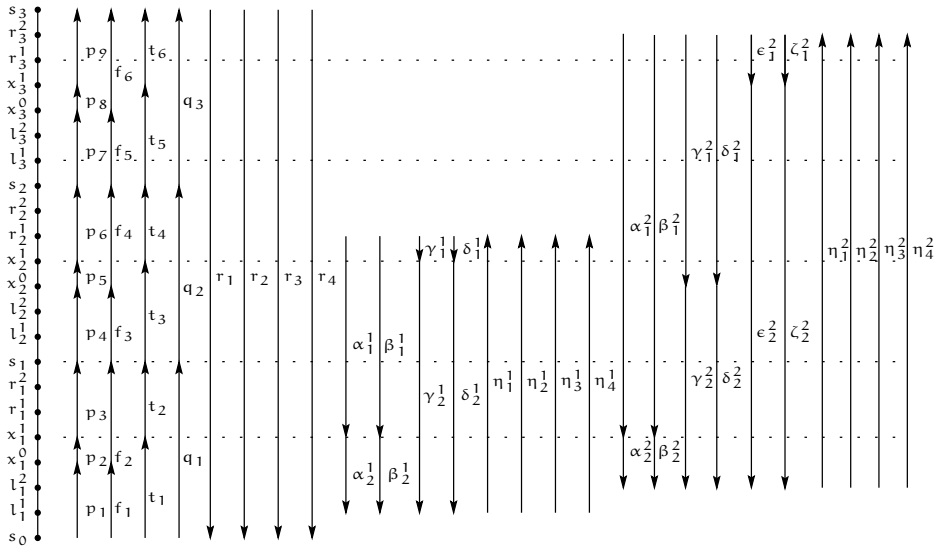

Abbildung 2.4: Beispiel für eine Transformation des Problems $U := \{x_1, x_2, x_3\}$, $C := \{c_1 := (x_1 + x_2), c_2 := (x_1 + \bar{x}_2 + x_3)\}$ in ein CDARP mit Kapazität 2.

Beweis der Korrektheit der Reduktion

Wir wollen in diesem Abschnitt folgenden Satz beweisen:

Satz 2.1.4. *Das* CDARP*-Entscheidungsproblem auf Pfaden ist $\mathcal{NP}$-schwer, selbst für eine Kapazität des Fahrzeugs von zwei und ein Fahrzeug, das an einem Endknoten des Pfades startet.*

Das Problem ist in $\mathcal{NP}$, da ein Transport ein Zertifikat darstellt und jeder Transport, der nicht mehrere leere Bewegungen hintereinander ausführt, höchstens $2m$ Bewegungen enthält. Man kann also in polynomialer Zeit die Kosten dieses Transports ausrechnen.

Um zu beweisen, daß das Problem $\mathcal{NP}$-schwer ist, führen wir das 2/3-SAT auf das CDARP auf Pfaden zurück. Sei eine Instanz des 2/3-SAT gegeben. Dabei sei $U = \{x_1, x_2, \ldots, x_n\}$ die Menge der Variablen und $C = \{c_1, c_2, \ldots, c_m\}$ die Menge der Klauseln über U.

Die oben beschriebene Transformation ist polynomial, da für jede Zuweisung eines Wahrheitswerts zu einer Variablen $8n + 4$ Objekte existieren. Für jede Klausel mit zwei Literalen gibt es zwölf Objekte, für jede Klausel mit drei Literalen 16 Objekte. Also ist die Anzahl der Objekte in $O(n + m)$. Damit ist klar, daß die Transformation in polynomialer Zeit durchführbar ist.

Um den Satz 2.1.4 zu beweisen, müssen wir zeigen, daß das oben konstruierte CDARP-Entscheidungsproblem genau dann mit „Ja“ entschieden werden kann, wenn das entsprechende 2/3-SAT eine Lösung hat. Dazu beweisen wir erst einmal folgendes technische Lemma:

Lemma 2.1.5. *Das* cDARP $P = (V, E, A, v_0, c)$ *hat genau dann einen Transport mit Kosten* c_0*, wenn es eine Lösung gibt, die nur aus vollbeladenen Bewegungen besteht, die alle Objekte in Richtung ihrer Zielknoten transportieren.*

Beweis. Die Summe der Kosten aller Objekte ist $2c_0$. Die Kapazität des Fahrzeugs beträgt 2. Gesamtkosten von $2c_0/2 = c_0$ können also nur erreicht werden, wenn in jeder Bewegung tatsächlich zwei Objekte in die richtige Richtung transportiert werden. Transportiert andererseits jede Bewegung genau zwei Objekte in die richtige Richtung, betragen die Gesamtkosten des gültigen Transports die Hälfte der Gesamtkosten aller Objekte, und das ist c_0. □

Satz 2.1.6. *Wenn ein gültiger Transport* L *der konstruierten Instanz des* cDARP*, die nur aus vollbeladenen Bewegungen besteht und die alle Objekte in Richtung ihrer Zielknoten transportiert, existiert, dann gibt es für jede Klausel* c_j*,* $j = 1, \ldots, m$ *einen Transport* T_j*, für den gilt:*

1. *Mit* T_j *werden nur Objekte aus der Menge* C_j *transportiert. Dabei ist* C_j *wie in den Gleichungen (2.1) und (2.2) definiert.*
2. *alle Bewegungen aus* T_j *sind in der Lösung* L *enthalten,*
3. *die erste Bewegung in* T_j *beginnt an einem Knoten* $x_{i_z}^{v_z}$*, wobei* z *ein Literal in* c_j *ist und*
4. *und die letzte Bewegung in* T_j *endet an demselben Knoten, an dem die erste Bewegung beginnt.*

Diesen Satz wollen wir in mehreren Schritten beweisen:

Lemma 2.1.7. *Eine Bewegung in einem gültigen Transport* L *mit Kosten* c *kann nicht gleichzeitig ein Objekt aus* C_j *enthalten und ein Objekt, welches nicht einer anderen Klausel zugeordnet ist, d.h. ein Objekt der Menge*

$$A' := \{p_1, \ldots, p_{3n}, f_1, \ldots, f_{2n}, t_1, \ldots, t_{2n}, q_1, \ldots, q_n, r_1, \ldots, r_4\}.$$

Beweis. Wir müssen wegen Lemma 2.1.5 nur Bewegungen betrachten, die zwei Objekte enthalten, die beide in die gleiche Richtung transportiert werden müssen.

Angenommen es existiert eine Bewegung in L, die sowohl ein Objekt aus A' als auch ein Objekt aus C_j transportiert. Wir betrachten dann die erste dieser Bewegungen. Diese Bewegung B muß eine Vorgängerbewegung haben, die nur Objekte aus A' oder nur Objekte aus $\bigcup_{j=1}^{m} C_j$ transportiert, da die erste Bewegung in L, welche im Knoten s_0 startet, zwei Objekte aus A' transportieren muß, damit sie vollbeladen ist.

Es existiert kein Knoten, an dem sowohl Aufwärtstransporte aus A' enden als auch Aufwärtstransporte aus $\bigcup_{j=1}^{m} C_j$ beginnen. Es existiert auch kein Knoten, an dem sowohl Abwärtstransporte aus A' enden als auch Abwärtstransporte aus $\bigcup_{j=1}^{m} C_j$ beginnen. Also muß die Bewegung B in die ihrer Vorgängerbewegung entgegengesetzte Richtung fahren. Also müssen in der Vorgängerbewegung von B beide Objekte abgesetzt werden. Daraus folgt aber, daß B entweder

am Knoten s_0 oder s_n beginnen muß, da nur dort gleichzeitig Objekte aus A' mit verschiedenen Richtungen enden und beginnen, oder an einem der Knoten $l_i^j, r_i^j, i = 1, \ldots, n, j = 1, \ldots, m$, da nur dort gleichzeitig Objekte aus $\bigcup_{j=1}^{m} C_j$ mit verschiedenen Richtungen enden und beginnen. An allen diesen Knoten starten aber entweder nur Objekte aus $\bigcup_{j=1}^{m} C_j$ oder nur Objekte aus A'. Also kann auch an diesen Knoten keine Bewegung mit einem Objekt aus $\bigcup_{j=1}^{m} C_j$ und einem aus A' beginnen. Also folgt die Behauptung. □

Lemma 2.1.8. *Eine Aufwärtsbewegung in einem gültigen Transport* L *mit Kosten* c*, der nur aus vollbeladenen Bewegungen besteht, die alle Objekte in die richtige Richtung transportieren, kann nicht gleichzeitig ein Objekt aus* C_j *und ein Objekt aus* $C_{j'}$ *mit* $j \neq j'$ *enthalten.*

Beweis. Angenommen es existiert eine erste Aufwärtsbewegung, die ein Objekt aus der Menge C_j und ein Objekt aus der Menge $C_{j'}, j' \neq j$ transportiert. Wegen Lemma 2.1.5 müssen beide Objekte in die richtige Richtung transportiert werden. Also können nur die Objekte $\eta_1^j, \ldots, \eta_4^j, \eta_1^{j'}, \ldots, \eta_4^{j'}$ transportiert werden. Diese Objekte haben aber nur einen gemeinsamen Startknoten, wenn sie auch aus derselben Menge C_j oder $C_{j'}$ stammen. Also kann jede Aufwärts-Bewegung nur Objekte aus höchstens einer der Mengen $C_j, j = 1, \ldots, m$ transportieren. □

Lemma 2.1.9. *Eine Abwärts-Bewegung in einem gültigen Transport* L *mit Kosten* c*, der nur aus vollbeladenen Bewegungen besteht, die alle Objekte in die richtige Richtung transportieren, kann nicht gleichzeitig ein Objekt aus* C_j *und ein Objekt aus* $C_{j'}$ *mit* $j \neq j'$ *enthalten.*

Beweis. Angenommen es existiert eine Abwärts-Bewegung $B := (u, v, \{o_1, o_2\})$ in L, wobei $o_1 \in C_j$ und $o_2 \in C_{j'}$ mit $j \neq j$ ist. Der Knoten u muß ein Knoten sein, an dem mindestens eines der beiden Objekte o_1 und o_2 beginnt. Und der Knoten v muß ein Knoten sein, an dem mindestens eines der beiden Objekte o_1 und o_2 endet. Diese Bewegung kann dann entweder an einem Knoten $x_i^l, i = 1, \ldots, n, l = 0, 1$, an einem Knoten $r_i^j, i = 1, \ldots, n$ oder $r_i^{j'}, i = 1, \ldots, n$ beginnen.

Angenommen die Bewegung B beginnt am Knoten r_i^j. An diesem Knoten enden aber nur Aufwärtsbögen aus C_j. Also muß der Aufzug am Knoten r_i^j seine Richtung ändern und daher auch zwei Objekte aufnehmen, damit die Bewegung vollbeladen ist. Am Knoten r_i^j liegen aber nur Objekte aus der Menge C_j. Also kann B nur Objekte aus C_j enthalten, dies ist aber ein Widerspruch zur Annahme. Mit der gleichen Argumentation kann die Bewegung B auch nicht an einem Knoten $r_i^{j'}, i = 1, \ldots, n$ beginnen.

Angenommen die Bewegung B endet am Knoten l_i^j. An diesem Knoten kann nur das Objekt o_1 abgelegt werden. Das Objekt o_2 muß weiter abwärts transportiert werden. Am Knoten l_i^j können aber keine Objekte liegen, die abwärts transportiert werden müssen. Also kann die Folgebewegung von B keine vollbeladene Bewegung sein. Dies darf aber nicht auftreten, also kann B nicht an einem Knoten

l_i^j, $i = 1, \ldots, n$ enden. Mit der gleichen Argumentation kann die Bewegung B auch nicht an einem Knoten $l_i^{j'}$, $i = 1, \ldots, n$ enden.

Die Knoten u und v sind also vom Typ x_i^l, $i = 1, \ldots, n, l = 0, 1$.

Die beiden Objekte o_1 und o_2 können nicht sowohl gleiche Start- als auch gleiche Endknoten haben, da ansonsten die Klauseln c_j und $c_{j'}$ zwei gemeinsame Literale haben müßten, denn an den Knoten x_i^l, $i = 1, \ldots, n, l = 0, 1$ beginnen und enden nur Objekte aus C_j, wenn i ein Index einer Variable ist, deren Literal in c_j enthalten ist. Im 2/3-SAT können aber nicht zwei gleiche Literale gleichzeitig in zwei verschiedenen Klauseln enthalten sein. Also müssen die beide Objekte entweder unterschiedliche Startknoten oder unterschiedliche Endknoten haben.

- Wir nehmen an, o_1 und o_2 haben zwei verschiedene Startknoten $x_{i_y}^{\nu_y}$ und $x_{i_z}^{\nu_z}$. Es enden in jedem Knoten x_i^l, $i = 1, \ldots, n, l = 0, 1$, nur Abwärtsbögen, die in einem Knoten r_i^j, $j = 1, \ldots, m$ beginnen. Bögen, die im Knoten $x_{i_y}^{\nu_y}$ enden, beginnen also im Knoten $r_{i_y}^j$, und Bögen die im Knoten $x_{i_z}^{\nu_z}$ enden, beginnen im Knoten $r_{i_z}^j$. In diesen beiden Knoten enden aber keine abwärtsführenden Bögen. Am Knoten $r_{i_y}^j$ oder am Knoten $r_{i_z}^j$ muß also eine Bewegung beginnen, die nur ein Objekt transportiert. Diese Bewegung ist daher nicht vollbeladen.

- Wir nehmen an o_1 und o_2 haben zwei verschiedene Endknoten $x_{i_y}^{\nu_y}$ und $x_{i_z}^{\nu_z}$. Es enden in jedem Knoten x_i^l, $i = 1, \ldots, n, l = 0, 1$,nur Abwärtsbögen, die in einem Knoten l_i^j, $j = 1, \ldots, m$ enden. Bögen, die im Knoten $x_{i_y}^{\nu_y}$ beginnen, enden also im Knoten $l_{i_y}^j$, und Bögen die im Knoten $x_{i_z}^{\nu_z}$ beginnen, enden im Knoten $l_{i_z}^j$. In diesen beiden Knoten beginnen aber keine abwärtsführenden Bögen. Am Knoten $l_{i_y}^j$ oder am Knoten $l_{i_z}^j$ muß also eine Bewegung enden, die nur ein Objekt transportiert. Diese Bewegung ist daher nicht vollbeladen.

Also muß eine Abwärtsbewegung, die mindestens ein Objekt aus C_j transportiert, genau zwei Objekte aus C_j transportieren. □

Mit den Lemmata 2.1.7, 2.1.8 und 2.1.9 folgen die ersten beiden Aussagen aus dem Satz 2.1.6. Die dritte Aussage gilt, da zu den Knoten l_i^j, r_i^j, $i = 1, \ldots, n, j = 1, \ldots, m$ nur Bögen aus C_j inzident sind. Und die vierte Aussage folgt, da die durch C_j induzierten Teilgraphen balanciert sind.

Nun können wir zeigen, daß das 2/3-SAT auf ein spezielles 2-cDARP reduzierbar ist.

Satz 2.1.10. *Es existiert genau dann eine gültige Zuweisung von Wahrheitswerten an die Variablen U für die Klauseln C, wenn das* cDARP, *das durch V, E, A, v_0 und c_0 definiert ist, einen Transport mit Kosten c_0 hat.*

Beweis. 1. Angenommen es existiert eine Zuweisung, so daß alle Klauseln in C wahr sind. Dann erzeugen wir einen Transport mit Kosten c_0 wie folgt:

Das Fahrzeug wird zweimal vom Knoten s_0 zum Knoten s_n und zurück fahren. Beim ersten Mal, werden aber noch zusätzliche Transporte für die Objekte generiert, die für die Klauseln erzeugt wurden. Mit dem ersten Transport von s_0 nach s_n werden die Objekte t_{2i-1} und t_{2i} transportiert, wenn die Variable x_i „wahr" ist. Die Objekte f_{2i-1} und f_{2i} werden mit diesem Transport transportiert, wenn x_i „falsch" ist. Dabei ist $i = 1, \ldots, n$. Zusammen mit diesen Objekten, werden die Objekte $p_i, i = 1, \ldots, 3n$ transportiert.

Ist x_i „wahr", dann werden beim ersten Besuch von x_i^1 für alle Klauseln c_j in denen das Literal x_i enthalten ist, Transporte für die Objekte C_j erzeugt. C_j ist wie in den Gleichungen (2.1) und (2.2) definiert.

Beim ersten Besuch des Knotens x_i^1 werden beide Objekte entladen, daher können die entsprechenden Transporte für die Klauseln eingefügt werden, die das Literal x_i enthalten. Analog wird, wenn x_i = „falsch" ist, beim ersten Besuch des Knotens x_i^0 für alle Klauseln die das Literal $\bar{x}_i$ enthalten, die entsprechenden Transporte konstruiert und eingefügt.

Da jede Klausel ein „wahres" Literal enthält, werden beim ersten Erreichen des Knotens s_n alle Objekte, die für die Klauseln erzeugt wurden, abgearbeitet sein. Also werden nun zwei der Objekte r_1, r_2, r_3, r_4 transportiert, und dann die restlichen Objekte t_p und f_p zusammen mit den Objekten $q_i, i = 1, \ldots, n$ transportiert. Nun müssen nur noch die letzten beide Objekte aus der Menge $\{r_1, r_2, r_3, r_4\}$ nach s_0 gebracht werden, dann sind alle Objekte transportiert worden, und der Transport enthält nur vollbeladene Bewegungen.

2. Existiere ein Transport mit Kosten c. Dieser Transport enthält nur vollbeladene Bewegungen wegen Lemma 2.1.5. Also muß jedes Objekt zusammen mit einem anderen Objekt transportiert werden. Wegen Lemma 2.1.6 müssen, wenn der Transport für Klausel c_j am Knoten $x_{i_z}^0$ startet, p_{3i_z-2} und f_{2i_z-1} in einer gemeinsamen Bewegung transportiert werden. Startet der Transport am Knoten $x_{i_z}^1$ dann müssen einerseits p_{3i_z-2} und t_{2i_z-1}, andererseits p_{3i_z-1} und t_{2i_z-1} in einer gemeinsamen Bewegung transportiert werden. Deshalb bekommt eine Variable x_i den Wahrheitwert „falsch", genau dann, wenn p_{3i-2} und f_{2i-1} in einer gemeinsamen Bewegung transportiert werden. Andernfalls bekommt eine Variable x_i den Wahrheitwert „wahr" zugewiesen. Da nun das CDARP so konstruiert ist, daß ein Transport für eine Klausel nur an Knoten beginnt, die einem Literal entsprechen, welches mit der entsprechenden Belegung den Wahrheitswert „wahr" hat, ist diese Zuweisung gültig für das 2/3-SAT.

□

Damit und mit der Polynomialität der Transformation ist auch der Satz 2.1.4 bewiesen.

2.2 Balancieren

Das Balancieren eines Graphen ist ein wichtiges Konzept zum Ermitteln unterer Schranken für das DARP aber auch das CDARP auf Bäumen und Pfaden. Es beruht auf der Tatsache, daß jede Kante in Bäumen kritisch ist. Deshalb muß jede Kante in einer Tour gleich oft in die eine Richtung wie in die andere Richtung überquert werden. Für das DARP auf Bäumen wurden balancierende Mengen in [6, 16] untersucht und für das CDARP auf Pfaden in [22].

Wir betrachten hier wiederum nur Graphen, die Pfade sind.

Mit $L(l, r)$ bezeichnen wir den induzierten Teilgraphen von G, der die Knoten l und r und alle in dem Weg von dem Knoten l zu dem Knoten r in G enthält. Ist S eine Menge von Objekten, und sind $L(l_1, r_1), L(l_2, r_2)$ zwei disjunkte Teilgraphen, dann ist $f_S([l_1, r_1], [l_2, r_2])$ die Anzahl der Objekte in S, die von einem Knoten in $L(l_1, r_1)$ zu einem Knoten in $L(l_2, r_2)$ transportiert werden müssen. Ist $S = A$, so schreiben wir auch nur f statt f_S. Besteht der Graph $L(l_1, r_1)$ nur aus einem Knoten, gilt also $l_1 = r_1$, so schreiben wir auch statt $f_S([l_1, l_1], [l_2, r_2])$ einfach $f_S(l_1, [l_2, r_2])$, bzw. statt $f_S([l_2, r_2], [l_1, l_1])$ einfach $f_S([l_2, r_2], l_1)$.

Sei für $\nu \in V \setminus \{n\}$

$$\lambda_\nu := \max\{\lceil f([1,\nu],[\nu+1,n])/k\rceil, \lceil f([\nu+1,n],[1,\nu])/k\rceil, 1\}$$

Das ist die minimale Anzahl der Überquerungen der Kante $(\nu, \nu+1)$ durch den Aufzug mit Kapazität k in einem gültigen Transport. Eine untere Schranke für das CDARP ist die *Fluß-Schranke*. Es gilt, daß

$$c_{\text{flow}} := 2\sum_{i=1}^{n-1} \lambda_i$$

eine untere Schranke für das CDARP ist. Darauf gehen wir im Abschnitt 2.5 genauer ein.

Definition 2.2.1. Wir nennen eine Instanz des CDARP auf einem Pfad, die durch das Tupel (V, E, A, c, o, k) gegeben ist, *balanciert*, wenn für jede Kante $(\nu, \nu+1) \in E$ gilt, daß

$$\frac{f([1,\nu],[\nu+1,n])}{k} = \frac{f([\nu+1,n],[1,\nu])}{k} = \lambda_\nu$$

ist.

Jede Kante wird also von einem Vielfachen von k Bögen überdeckt. Die Anzahl der Aufwärtsbögen, die eine Kante überdecken, ist gleich der Anzahl der Abwärtsbögen, die diese Kante überdecken. Und jede Kante wird von mindestens einem Bogen überdeckt.

Man findet leicht zu einer nicht balancierten Instanz eines CDARP auf einem Pfad eine Menge von Bögen S, so daß $(V, E, A \cup S, c, o, k)$ balanciert ist. Man muß nur für jede Kante $(\nu, \nu+1), \nu \in V \setminus \{n\}$ soviele Elementarbögen $\langle \nu, \nu+1\rangle$ oder

$\langle v+1, v\rangle$ hinzufügen, bis für diese Kante $f([1,v],[v+1,n]) = f([v+1,n],[1,v]) = n_1 k, n_1 \in \mathbb{N}_{>0}$ gilt. Bei einer geeigneten Speicherung der Daten können wir also in $O(n)$ die $\lambda_v, v \in V \setminus \{n\}$ bestimmen und dann in $O(m)$, wobei m die Anzahl der Kanten der balancierten Instanz des CDARP ist, die fehlenden Kanten hinzufügen (siehe [22]).

Das Balancieren eines CDARP auf dem Pfad hat also eine Zeitkomplexität von $O(m+n)$. Dabei ist $m = |A \cup S|$ und $n = |V|$.

Wir können die Balanciertheit einer Instanz eines CDARP auch noch anders charakterisieren:

Lemma 2.2.2. *Folgendes ist äquivalent:*

1. $\delta^+(v) = \delta^-(v)$ *für alle* $v \in V$
2. $f([1,v],[v+1,n]) = f([v+1,n],[1,v])$ *für alle* $v \in V \setminus \{n\}$

Beweis. Gelte die 1. Aussage der Behauptung. Wir definieren $x(v,w) = |\{i \in A : \langle i^+, i^-\rangle = \langle v, w\rangle\}|$. Dann gilt

$$f([1,v],[v+1,n]) = \sum_{i=1}^{v} \delta^+(i) - \sum_{i=1}^{v}\sum_{j=1}^{v} x(v,w) \tag{2.3}$$

$$= \sum_{i=1}^{v} \delta^-(i) - \sum_{i=1}^{v}\sum_{j=1}^{v} x(v,w) \tag{2.4}$$

$$= f([v+1,n],[1,v]) \tag{2.5}$$

Für die Rückrichtung definieren wir: Sei $A_v^+ \subset A$ die Menge der Aufwärtsbögen und $A_v^- \subset A$ die Menge der Abwärtsbögen, die nicht zum Knoten v inzident sind und die Kanten $(v-1,v)$ und $(v,v+1)$ überdecken. Sei B_v die Menge der zum Knoten v inzidenten Bögen. Dann gilt für alle Knoten $v \in V \setminus \{1,n\}$:

$$f_{B_v}([1,v-1],[v]) = f_A([1,v-1],[v,n]) - \left|A_v^+\right|, \tag{2.6}$$

$$f_{B_v}([v],[1,v-1]) = f_A([v,n],[1,v-1]) - \left|A_v^-\right|, \tag{2.7}$$

$$f_{B_v}([v],[v+1,n]) = f_A([1,v],[v+1,n]) - \left|A_v^+\right| \text{ und} \tag{2.8}$$

$$f_{B_v}([v+1,n],[v]) = f_A([v+1,n],[1,v]) - \left|A_v^-\right| \tag{2.9}$$

Daraus folgt für alle Knoten $v \in V \setminus \{1,n\}$:

$$\delta^+(v) = f_{B_v}([v],[v+1,n]) + f_{B_v}([v],[1,v-1]) \tag{2.10}$$

$$= f_A([1,v],[v+1,n]) - \left|A_v^+\right| + f_A([v,n],[1,v-1]) - \left|A_v^-\right| \tag{2.11}$$

und

$$\delta^-(v) = f_{B_v}([1,v-1],[v]) + f_{B_v}([v+1,n],[v]) \tag{2.12}$$

$$= f_A([v+1,n],[1,v]) - \left|A_v^+\right| + f_A([1,v-1],[v,n]) - \left|A_v^-\right| \tag{2.13}$$

und für die Knoten 1 und n gilt

$$\delta^-(1) = f_A(1, [2, n]), \tag{2.14}$$
$$\delta^+(1) = f_A([2, n], 1), \tag{2.15}$$
$$\delta^-(n) = f_A(n, [1, n-1]) \text{ und} \tag{2.16}$$
$$\delta^+(n) = f_A([1, n-1], n) \tag{2.17}$$

Gelte nun die 2. Aussage, also für alle $v \in V \setminus \{n\}$

$$f_A([1, v], [v+1, n]) = f_A([v+1, n], [1, v]). \tag{2.18}$$

Dann folgt für alle $v \in V \setminus \{1, n\}$:

$$\delta^+(v) \overset{(2.11)}{=} f_A([1, v], [v+1, n]) - \left|A_v^+\right| + f_A([v, n], [1, v-1]) - \left|A_v^-\right| \tag{2.19}$$
$$\overset{(2.18)}{=} f_A([v+1, n], [1, v]) - \left|A_v^+\right| + f_A([1, v-1], [v, n]) - \left|A_v^-\right| \tag{2.20}$$
$$\overset{(2.13)}{=} \delta^-(v) \tag{2.21}$$

Für $v \in \{1, n\}$ folgt die 1. Aussage aus den Gleichungen (2.14) bis (2.17) und der Annahme (2.18). □

Aus dem Lemma 2.2.2 folgt auch, daß für balancierte Probleme $\delta^+(v) = \delta^-(v)$ für alle $v \in V$ gilt.

Korollar 2.2.3. *Für ein* DARP (V, E, A, c, s) *sind folgende zwei Aussagen äquivalent:*

1. $\delta^+(v) = \delta^-(v)$ *für alle* $v \in V$ *und* $f([1, v], [v+1, n]) > 0$ *für alle* $v \in V \setminus \{n\}$,
2. (V, E, A, c, s) *ist balanciert.*

Beweis. Dies folgt direkt aus dem Lemma 2.2.2. □

2.3 DARP auf Pfaden ist polynomial lösbar

In diesem Abschnitt vollziehen wir ein Resultat von Atallah und Kosaraju [6] nach. Wir betrachten das (nicht-präemptive) DARP, dessen unterliegender Graph $G = (V, E)$ ein Pfad ist. Auf solchen Instanzen des DARP gibt es den polynomialen Algorithmus OPTPATH, der das Problem löst. Genauer gesagt, lösen wir mit diesem Algorithmus das zu einem DARP gehörige Graph-Augmentations-Problem, wie es in der Definition 1.3.2 gegeben ist.

Das DARP interessiert uns, da es einerseits als ein Spezialfall des CDARP gesehen werden kann. Andererseits werden wir später (in Abschnitt 2.7) einen Algorithmus vorstellen, der das CDARP auf Pfaden auf ein DARP auf Pfaden reduziert.

Wir wollen annehmen, daß V nur Knoten enthält, zu denen mindestens ein Bogen aus A adjazent ist. Ist dies nicht der Fall, entfernen wir alle Knoten v aus V, zu denen kein Bogen aus A adajzent ist und ergänzen dann für Bögen (v, u_1) und (v, u_2), $u_1 \neq u_2$ einen Bogen (u_1, u_2). Die Kosten setzen wir auf $c(u_1, u_2) := c(v, u_1) + c(v, u_2)$.

Eine Instanz des DARP ist balanciert (siehe auch Korollar 2.2.3), wenn jede Kante aus E von mindestens einem Bogen und gleich vielen Bögen aus D^+ und D^- überdeckt wird. Wir können jede Instanz eines DARP zu einer balancierten Instanz ergänzen, indem wir eine Menge von Elementarbögen S zu A hinzufügen. Das Balancieren erhöht die Kosten der optimalen Lösung nicht. Ist G ein Pfad so ist das Balancieren trivial, man muß nur für jede Kante $(v, v+1), v \in V \setminus \{n\}$ soviele Elementarbögen $\langle v, v+1\rangle$ oder $\langle v+1, v\rangle$ hinzufügen, bis für diese Kante $f([1, v], [v+1, n]) = f([v+1, n], [1, v]) > 0$ gilt.

Der Algorithmus OPTPATH benutzt den Komponentengraphen $\hat{G}[A \cup S] = (\hat{V}, \hat{E})$. Dieser Graph wird wie folgt konstruiert: Sei $\hat{V}$ die Menge der stark zusammenhängenden Komponenten G_i in $G[A]$. Für jeden Knoten $v \in V$, der nicht in einer der Zusammenhangskomponenten enthalten ist, fügen wir einen Knoten G_i zu $\hat{V}$ hinzu. Eine Kante zwischen zwei Knoten G_i und G_j aus $\hat{V}$ fügen wir zu $\hat{E}$ hinzu, wenn im Graphen $G[A]$ eine Kante existiert, die einen Knoten aus G_i mit einem Knoten aus G_j verbindet. Die Kosten dieser Kanten werden auf die Kosten der kürzesten Kante, die G_i und G_j verbindet, gesetzt.

Die Zusammenhangskomponenten eines Graphen und damit auch den Komponentengraphen $\hat{G}$ kann man mit Tiefensuche in linearer Zeit ermitteln [13].

Die Anzahl der Kanten in $\hat{G}[A]$ ist höchstens $|E|$, also die Anzahl der Kanten im ursprünglichen Graphen G. Ist G ein Baum (oder auch ein Pfad), gibt es also höchstens $n - 1$ Kanten in $\hat{G}[A]$.

Vorbedingung: Gegeben seien eine Menge von Knoten V, eine Menge von Kanten E und eine Menge von Bögen A über V, so daß $G = (E, V)$ ein Pfad ist und zu jedem Knoten in V mindestens ein Bogen aus A adjazent ist. Außerdem haben wir eine Kostenfunktion c auf $E \cup A$.

1: Berechne eine Menge von balancierenden Elementarbögen S.
2: Konstruiere den Graphen der Zusammenhangskomponenten $\hat{G}[A \cup S] = (\hat{V}, \hat{E})$.
3: Berechnen einen minimalen aufspannenden Baum T von $\hat{G}$.
4: Sei $N = \emptyset$. Füge für jede Kante in T ein Paar von antiparallelen Bögen zwischen den Endpunkten der zugehörigen Kante in E zu N hinzu.
5: Gib die Menge $S \cup N$ aus.

Algorithmus 1: OPTPATH

Im folgenden beweisen wir analog zu [6], daß OPTPATH eine minimale Menge $S \cup N$ berechnet, die den Digraphen $D = (V, A)$ eulersch macht.

Definition 2.3.1 (Einbettung). Sei $G = (V, E)$ ein Graph und $D = (V, A)$ ein Digraph. Eine Multimenge B von Bögen ist eine *Einbettung* von B in E, wenn

1. für jeden Bogen $a \in A$ mit $\langle a^+, a^- \rangle = \langle u, v \rangle$ ein Pfad P_a in B von u nach v existiert und
2. die Menge von Bögen B die disjunkte Vereinigung aller dieser Pfade ist, also $B = \bigcup_{a \in A} P_a$.

Lemma 2.3.2. *Ein Digraph ist eulersch, wenn er balanciert und zusammenhängend ist.*

Beweis. Aus Lemma 2.2.2 folgt, daß $\delta^+(v) = \delta^-(v)$ für alle $v \in V$ gilt, wenn der Graph balanciert ist. Wenn er nun noch zusammenhängend ist, ist er nach dem Satz von Euler [13] eulersch. □

Satz 2.3.3. *Der Algorithmus* OPTPATH *findet eine kostenminimale Multimenge S^* von Bögen, so daß $(V, A \cup S^*)$ eulersch ist.*

Beweis. Die Menge $S \cup N \cup A$ ist eulersch, da sie zusammenhängend und balanciert ist. Da sie balanciert ist, gilt nämlich für alle Knoten $v \in V$, daß $\delta^+(v) = \delta^-(v)$ ist. Deshalb erzeugt OPTPATH eine zulässige Lösung.

Es bleibt noch die Optimalität der Lösung zu zeigen. Sei S^* eine bezüglich c minimale Menge von Bögen, so daß $A \cup S^*$ eulersch ist und $S \subset S^*$ gilt. Dann ist $N^* = S^* \setminus S$ eine Menge von Bögen, die die Komponenten von $G[A \cup S]$ zu einem stark zusammenhängenden Graphen ergänzt. Da $A \cup S$ balanciert ist und $G[A \cup S^*]$ eulersch und damit auch balanciert ist, können wir daraus schließen, daß auch N^* balanciert ist. Da $G = (V, E)$ ein Pfad ist, ist jede Kante in E eine kritische Kante. D.h. durch Entfernen einer beliebigen Kante zerfällt G in mehrere Zusammenhangskomponenten. Daraus folgt, daß die Einbettung von $S^* \setminus S$ in E aus Paaren von antiparallelen Bögen besteht. Die Kanten in $\hat{G}[A \cup S]$ traversiert von den Bögen in der Einbettung von $S^* \setminus S$ in E verbinden alle Knoten in $\hat{G}$ und sind damit eine Obermenge eines aufspannenden Baums. Die Kosten von N^* sind deshalb mindestens doppelt so groß, wie die Kosten eines minimalen aufspannenden Baumes von $\hat{G}[A \cup S]$. Da die Multimenge N, die von dem Algorithmus OPTPATH konstruiert wird, genau die doppelten Kosten eine MST haben, ist $S \cup N$ eine optimale Lösung. □

Eine balancierende Menge für $G = (V, E), D = (V, A)$ kann in $O(n + m)$ berechnet werden (Abschnitt 2.2). Ebenso kann der Graph der Zusammenhangskomponenten in $O(n+m)$ Zeit berechnet werden. Das Berechnen eines MST eines Baumes mit q Knoten kann in $O(m \log \beta(n, q))$ Zeit durchgeführt werden [18]. Dabei ist $\beta(n, q) = \min\{i : \log_i n \leqslant n/q\}$ eine sehr langsam wachsende Funktion. Insgesamt ist die Zeitkomplexität von OPTPATH also in $O(n + m \log \beta(n, q))$.

2.4 Präemptives CDARP auf Pfaden

In diesem Abschnitt stellen wir einen Algorithmus von Guan [22] für das präemptive CDARP auf Pfaden vor, der dieses Problem in polynomialer Zeit optimal löst. Dabei sind die Kosten einer optimalen Lösung einer Instanz des präemptiven CDARP auf Pfaden gleich seiner Fluß-Schranke.

2.4.1 Beschreibung des Algorithmus

Man kann für das CDARP einen optimalen präemptiven Transport in polynomialer Zeit berechnen. Dies geschieht für balancierte Instanzen des CDARP durch den in diesem Abschnitt beschrieben Greedy-Algorithmus FIRSTMOVE. Man kann aber auch für nicht-balancierte Instanzen eines CDARP optimale Lösungen ermitteln, indem man zuerst das CDARP balanciert, auf der neuen Instanz mit FIRSTMOVE einen optimalen Transport ermittelt und dann aus diesem Transport die zum Balancieren eingefügten Bögen wieder entfernt. Da das Balancieren die Kosten der optimalen Lösung nicht erhöht, bleibt die so erhaltenen Lösung für nicht-balancierte Instanzen optimal.

Der Algorithmus FIRSTMOVE wählt am aktuellen Knoten eine beliebige Teilmenge der Aufträge, die in die aktuelle Richtung transportiert werden sollen und transportiert diese an ihr Ziel. Gibt es nicht genügend Objekte, die in die aktuelle Richtung transportiert werden sollen, werden alle Objekte abgeladen und die Richtung geändert. Guan [22] zeigt, daß dieser Algorithmus eine Folge von vollbeladenen Bewegungen erzeugt, die alle Objekte nur in die richtige Richtung transportieren, und die im gleichen Knoten anfängt und endet. Diesen Beweis geben wir in leicht modifizierter Form wieder.

Dieser Algorithmus ist, wie wir später erläutern, nicht auf Herlitz übertragbar, sein Verständnis trägt aber zum Verständnis der Strukturen des nicht-präemptiven CDARP bei.

Vorbedingung: Das Problem ist balanciert, v ist der Knoten, an dem der Aufzug startet.
Liefert: $x = v$, das Problem ist balanciert.

1: $x := v; M := \emptyset; d :=$ „aufwärts“; Dabei ist x der Knoten an dem sich der Aufzug befindet, M die Menge der Objekte im Aufzug und d die aktuelle Richtung des Aufzugs.
2: **Solange** es Objekte am Knoten v gibt **:**
3: Sei T eine Menge von Objekten am Knoten x, die in Richtung d transportiert werden sollen;
4: **Wenn** $|T \cup M| \geqslant k$ **Dann**
5: Wähle $k - |M|$ Objekte aus T und füge sie zu M hinzu;
6: Sei W die Teilmenge von Objekten aus M, deren gemeinsamer Zielknoten y am nähesten bei x liegt;
7: Erzeuge eine Bewegung (x, y, M);
8: Lege die Objekte aus W bei y ab; $M := M \setminus W; x := y$;
9: **Sonst**
10: Lege alle Objekte aus M bei x ab; $M = \emptyset$;
11: Ändere die Richtung d;
12: **Ende Wenn**
13: **Ende Solange**

Algorithmus 2: FIRSTMOVE (v)

Dieser Algorithmus kann nun benutzt werden, um einen optimalen Transport,

also eine Folge von Bewegungen mit minimalen Kosten, zu erzeugen, die alle Objekten von ihren Start- zu ihren Zielknoten bringt.

Dies erfolgt durch den Algorithmus STARTATENDS. Er ruft FIRSTMOVE für die Knoten $v = 1, 2, \ldots, n-1$ auf, wenn Objekte am Knoten v transportiert werden müssen. Die nacheinander erzeugten Bewegungen W werden in den Transport Q eingefügt.

```
1: Q := ∅;
2: Von v = 1 bis n − 1 :
3:   Wenn Objekte am Knoten v existieren Dann
4:     rufe FIRSTMOVE (v) auf;
5:     Sei W die Folge von Bewegungen, die von FIRSTMOVE (v) erzeugt wurde;
6:     Füge W zu Q hinzu;
7:   Ende Wenn
8: Ende Von
```

Algorithmus 3: STARTATENDS

Das Einfügen der Folge von Bewegungen W in den Transport Q im Schritt 6 des Algorithmus STARTATENDS muß noch genauer erläutert werden. W wird wie folgt in Q eingefügt: Ist $Q = \emptyset$, dann sei $Q := W$, ansonsten sei (x_i, x_{i+1}, M_i) die letzte Bewegung welche am Knoten v vorbeiführt. Ersetze dann (x_i, x_{i+1}, M_i) durch (x_i, v, M_i) und (v, x_{i+1}, M_i), wenn $x_{i+1} \neq v$. Füge den Transport W nach der Bewegung (x_i, v, M_i) ein.

2.4.2 Korrektheit

Wir zeigen zunächst, daß FIRSTMOVE Folgen von vollbeladenen Bewegungen konstruiert, mit denen alle Objekte in die richtige Richtung transportiert werden.

Sei

$$\mu_v^+ = f([1, v], [v+1, n]) \text{ und}$$

$$\mu_v^- = f([v+1, n], [1, v])).$$

Dies sind Zähler, wieviele Bögen aus A eine Kante $(v, v+1) \in E$ überdecken. Dabei ist μ_v^+ die Anzahl der Aufwärtsbögen, die $(v, v+1)$ überdecken und μ_v^- die Anzahl der Abwärtsbögen.

Lemma 2.4.1. *Wenn das Problem balanciert ist, dann gilt für jeden Knoten $v \in V$*

1. $f([1, v-1], v) > f(v, [v+1, n]) \Rightarrow f(v, [1, v-1]) \geqslant k$ *und*
2. $f([v+1, n], v) > f(v, [1, v-1]) \Rightarrow f(v, [v+1, n]) \geqslant k$

Beweis. Seien $e_1 = (v-1, v)$ und $e_2 = (v, v+1)$ die beiden zu v inzidenten Kanten. Dann folgt nach Definition

$$\mu^+_{v-1} = f([1, v-1], v) + f([1, v-1], [v+1, n]) \tag{2.22}$$
$$\mu^-_{v-1} = f(v, [1, v-1]) + f([v+1, n], [1, v-1]) \tag{2.23}$$
$$\mu^+_{v} = f(v, [v+1, n]) + f([1, v-1], [v+1, n]) \tag{2.24}$$
$$\mu^-_{v} = f([v+1, n], v) + f([v+1, n], [1, v-1]). \tag{2.25}$$

Da wir nur balancierte Instanzen des Problems betrachten gilt $\mu^+_v = \mu^-_v = k\lambda_v$ für jede Kante $(v, v+1) \in E$. Deshalb gilt

$$\begin{aligned} f([1, v-1], v) + f([1, v-1], [v+1, n]) &= f(v, [1, v-1]) \\ &\quad + f([v+1, n], [1, v-1]) = k\lambda_{v-1} \end{aligned} \tag{2.26}$$
$$\begin{aligned} f(v, [v+1, n]) + f([1, v-1], [v+1, n]) &= f([v+1, n], v) \\ &\quad + f([v+1, n], [1, v-1] = k\lambda_v \end{aligned} \tag{2.27}$$

Nun ziehen wir die Gleichung (2.27) von der Gleichung (2.26) ab:

$$\begin{aligned} f([1, v-1], v) - f(v, [v+1, n]) &= f(v, [1, v-1]) - f([v+1, n], v) \\ &= k(\lambda_{v-1} - \lambda_v) \end{aligned} \tag{2.28}$$

Ist $f(v, [1, v-1]) > f(v, [v+1, n])$, dann folgt aus (2.28):

$$\begin{aligned} f(v, [1, v-1]) - f([v+1, n], v) &= k(\lambda_{v-1} - \lambda_v) \\ &= f([1, v-1], v) - f(v, [v+1, n]) \\ &> 0 \end{aligned} \tag{2.29}$$

Ist $f(v, [1, v-1]) > f(v, [v+1, n])$, dann folgt aus (2.28) multipliziert mit (-1):

$$\begin{aligned} -f(v, [1, v-1]) + f([v+1, n], v) &= k(-\lambda_{v-1} + \lambda_v) \\ &= -f([1, v-1], v) + f(v, [v+1, n]) \\ &> 0 \end{aligned} \tag{2.30}$$

Da nun nach Definition $f([v+1, n], v) \geqslant 0$ und $f([1, v-1], v) \geqslant 0$, folgt aus den Ungleichungen (2.29) und (2.30) das Lemma. □

Lemma 2.4.2. *Ist die Instanz des Problems balanciert, dann erzeugt der Algorithmus* FIRSTMOVE *(v) für Knoten v mit $f(v, [1, v-1]) \geqslant k$ oder $f(v, [v+1, n]) \geqslant k$ eine nicht leere Folge von vollbeladenen Bewegungen, die am Knoten v startet und endet. Außerdem wird kein Objekt in die falsche Richtung transportiert.*

Beweis. Die Folge von Bewegungen ist nicht leer, da nach Voraussetzung entweder $f(v, [1, v-1])$ oder $f(v, [v+1, n])$ beim Aufruf des Algorithmus nicht kleiner als k ist.

Wir nehmen nun an, daß im Verlaufe des Algorithmus die Richtung d gleich „abwärts“ sei und der aktuelle Knoten $x \neq \nu$. Der Algorithmus FIRSTMOVE erzeugt nur vollbeladene Bewegungen. In jeder Bewegung (x, y, M), die der Algorithmus erzeugt, gibt es eine Teilmenge $\emptyset \neq W \subset M$ von Objekten, deren Zielknoten y ist.

Angenommen es tritt der Fall ein, daß in Schritt 4 des Algorithmus FIRSTMOVE bei der „Wenn“-Abfrage $|T \cup M| < k$ ist. Das bedeutet, daß am Knoten x und im Aufzug zusammen nicht genügend Objekte vorhanden sind, um den Aufzug mit Objekten, die in die aktuelle Richtung d transportiert werden müssen, voll zu beladen. Dieser Fall tritt auf, wenn $|W| > |T|$ ist (W ist die Menge aus Schritt 6 des Algorithmus FIRSTMOVE), wenn also mehr Objekte aus dem Aufzug am Knoten x abgeladen werden müssen, als dort zum Weitertransport zur Verfügung stehen. Dies bedeutet also, daß zu diesem Zeitpunkt $f([x+1, n], x) > f(x, [1, x-1])$ gilt.

Wäre nun das Problem zu diesem Zeitpunkt balanciert, würde mit dem Lemma 2.4.1 folgen, daß $f(x, [x+1, n]) \geqslant kn_1$ für ein $n_1 \in \mathbb{N}_{>0}$ gilt. Das heißt, wenn $|T \cup M| < k$ gilt, müssen mindestens k Objekte am Knoten x liegen und aufwärts transportiert werden.

Am Anfang ist das Problem balanciert. Nach einigen Bewegungen muß das Problem jedoch nicht mehr balanciert sein. Angenommen wir befinden uns nach einigen Bewegungen am Knoten x. Dann gilt für alle Kanten im Teilgraphen $L(\nu, x)$, daß $\mu_\nu^+ = \mu_\nu^- - k$ ist, da der Aufzug die Kante $(\nu, \nu+1)$ einmal mehr aufwärts als abwärts überquert hat und jede Bewegung des Aufzugs vollbeladen ist. Wenn wir also eine Menge S von k Objekten mit Startknoten ν und Zielknoten x zum Problem hinzufügen würden, dann wäre das Problem wieder balanciert.

Nun kann aber keines der k Objekte, die wegen Lemma 2.4.1 am Knoten x liegen müssen, eines der Objekte in S sein, da alle Objekte in S am Knoten ν liegen. Also können wir das Lemma 2.4.2 tatsächlich anwenden.

Analog können wir zeigen, daß mindestens k Objekte am Knoten x liegen, die abwärts transportiert werden müssen, wenn $x \neq \nu$ und d gleich „aufwärts“ ist. Deshalb muß auch am Ende von FIRSTMOVE x gleich ν sein.

Da der Aufzug nur vollbeladene Bewegungen durchgeführt hat, die kein Objekt in die falsche Richtung transportiert haben, und am Ende wieder an seinem Startknoten angekommen ist, muß für (V, E, A) nach Abarbeiten des Algorithmus wieder gelten, daß ein $n \in \mathbb{N}$ existiert, so daß $\lceil f([1, \nu], [\nu+1, n]) \rceil = \lceil f([\nu+1, n], [1, \nu]) \rceil = nk$ für alle $\nu \in V \setminus \{n\}$ gilt. □

Nun sehen wir uns den in Abbildung 3 definierten Algorithmus STARTATENDS an.

Satz 2.4.3. *Angenommen, der Startknoten o ist einer der Endpunkte des Pfades G. Eine Instanz eines präemptiven* CDARP *auf einem Pfad hat genau dann als optimale Lösung einen Transport, der nur aus vollbeladenen Bewegungen besteht und der jedes Objekt von seinem Startknoten in Richtung seines Zielknoten transportiert, wenn die Instanz balanciert ist.*

Beweis. Wenn das Problem einen Transport als Lösung hat, der nur aus vollbeladenen Bewegungen besteht und der jedes Objekt von seinem Startknoten in Richtung seines Zielknoten transportiert, dann gilt für jede Kante $(v, v+1) \in E$, daß $\mu_v^+ = \mu_v^- = k\lambda_v$ ist. Also ist das Problem balanciert.

Angenommen das Problem sei balanciert. Dann erzeugt der Algorithmus FIRSTMOVE wegen Lemma 2.4.2 nur Bewegungen mit den geforderten Eigenschaften. Also ist es klar, daß auch der Algorithmus STARTATENDS nur solche Bewegungen erzeugt.

Da jeder gültige Transport jedes Objekt von seinem Start- zu seinem Zielknoten transportieren muß ist

$$\frac{c(A)}{k}$$

eine untere Schranke für die Kosten jeder Lösung. Da dies aber auch genau die Kosten eines Transportes sind, der nur aus vollbeladenen Bewegungen besteht, die kein Objekt in die falsche Richtung transportieren, ist die Lösung, die der Algorithmus STARTATENDS findet, optimal. □

Den Beweis für die Polynomialität (eigentlich sogar Linearität) des Algorithmus führen wir ein wenig anders als in [22].

Lemma 2.4.4. *Nach Ausführung des Algorithmus* FIRSTMOVE *in Schritt 4 des Algorithmus* STARTATENDS *ist* $\lambda_{v'} = 0$ *für alle Knoten* $v' \leqslant v$.

Beweis (mit vollständiger Induktion). Ist $v = 1$, dann liegen nach Auführung des Algorithmus FIRSTMOVE (v) keine Objekte mehr am Knoten 1, da der Algorithmus nach Lemma 2.4.2 am Knoten v endet, und der Algorithmus nur terminiert, wenn keine Objekte mehr am aktuellen Knoten liegen. Da das Problem auch nach der Ausführung des Algorithmus FIRSTMOVE balanciert ist, dürfen auch keine Objekte mehr zum Knoten 1 transportiert werden. Also ist $\lambda_1 = 0$.

Induktionsschritt ($v \to v+1$): Nach Induktionsvorraussetzung ist $\lambda_{v'} = 0$ für alle $v' < v$. Es werden also keine Objekte zu Knoten transportiert oder von Knoten abgeholt, die unterhalb des Knotens v liegen. Da nach Beenden des Algorithmus FIRSTMOVE (v) keine Objekte mehr am Knoten v liegen und wegen der Balanciertheit des Problems auch keine Objekte mehr zum Knoten v transportiert werden, ist auch $\lambda_v = 0$. □

Satz 2.4.5. *Wir finden eine optimale Lösung des präemptiven* CDARP *auf einem Pfad in* $O(n+m)$ *Zeit, wenn der Startknoten einer der beiden Endknoten des Pfades ist.*

Beweis. Ein Problem kann in $O(n+m)$ Zeit balanciert werden. Für die balancierte Version finden wir mit dem Algorithmus STARTATENDS eine optimale Lösung. Eine optimale Lösung des ursprünglichen Problems finden wir dann aus der Lösung des balancierten Problems durch Löschen der Objekte, die nicht im ursprünglichen Problem enthalten sind. Die Anzahl der erzeugten Bewegungen des Algorithmus STARTATENDS ist höchstens $m + n - 1$, da in jeder Bewegung entweder mindestens ein Objekt zu seinem Zielknoten gebracht wird, oder die Bewegung durch

das Einfügen einer Folge von Bewegungen entstanden ist, was höchstens $(n-1)$-mal geschieht. Das Ermitteln einer Bewegung kann in konstanter Zeit durchgeführt werden, wenn wir zu Beginn des Algorithmus die Aufträge an jedem Knoten mit Bucket-Sort (also in $O(m)$ Zeit) nach Zielknoten sortieren. Also ist die Laufzeit des Algorithmus in $O(n+m)$. □

2.4.3 Anwendbarkeit auf das Herlitzproblem

Dieser Algorithmus ist auf das Herlitzproblem nicht unmittelbar anwendbar. Einerseits kann er nur innerhalb der Aufträge, die in eine Richtung transportiert werden müssen, im Schritt 5 des Algorithmus FIRSTMOVE auswählen, welche Objekte transportiert werden sollen. Durch die FIFO-Präzedenzen könnte es aber passieren, daß der vorderste Auftrag an einem Stockwerk in die falsche Richtung transportiert werden muß. Dann scheitert der Algorithmus.

Ein anderes Problem ist die Präemption. Wird ein Objekt ausgeladen, so kann es im Herlitz-System nur auf einer Seite abgeladen werden und kann erst wieder eingeladen werden, wenn es auf die andere Seite des Aufzuges transportiert worden ist.

Außerdem muß in einem präemptiven Transportplan der Aufzug häufig anhalten und Objekte ein- und ausladen. Bei den Aufzügen des Herlitz-Systems dauert aber das Anhalten und Anfahren des Aufzugs deutlich länger als das Überwinden der Entfernung zwischen zwei Stockwerken. Die Lösung des präemptiven Problems ist also für das eigentliche Problem ohne Präemption und mit Start-/Stopzeiten ziemlich schlecht.

2.5 Untere Schranken

Wir haben im Abschnitt 2.1 gesehen, daß das 2-CDARP auf einem Pfad bereits $\mathcal{NP}$-schwer ist. Da deshalb kein polynomialer Algorithmus zu erwarten ist, der das CDARP optimal löst, wollen wir zunächst untersuchen, welche Kosten ein gültiger Transport eines CDARP auf einem Pfad mindestens verursacht. Wir werden in diesem Abschnitt eine untere Schranke für das CDARP auf einem Pfad, die Fluß-Schranke, vorstellen.

Jede Kante $(v, v+1)$ von G muß in einem gültigen Transport mindestens $2\lambda_v$-mal überquert werden. Dies liegt daran, daß λ_v die Mindestanzahl der Überquerungen der Kante $(v, v+1)$ in eine bestimmte Richtung ist und der Aufzug jede Kante genauso oft aufwärts wie abwärts überqueren muß, da Start- und Zielknoten eines gültigen Transports gleich sind.

Dadurch ergibt sich eine untere Schranke für das präemptive wie auch das nicht-präemptive CDARP. Diese Schranke nennen wir *Fluß*-Schranke. Sie ist definiert durch

$$c_{\text{flow}} := 2 \sum_{v=1}^{n-1} \lambda_v c(v, v+1).$$

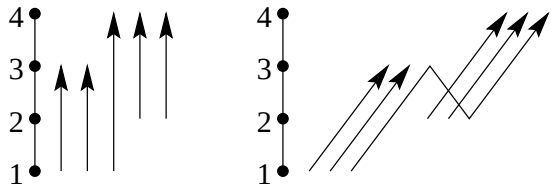

Abbildung 2.5: Beispiel für eine Instanz und ihre Lösung des nicht-präemptiven 3-CDARP, deren minimale Kosten durch Balancieren steigen.

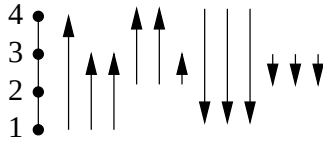

Abbildung 2.6: Instanz des 3-CDARP, die aus derjenigen in Abbildung 2.5 durch Balancieren entsteht.

In einem balancierten Problem wird diese Fluß-Schranke durch einen Transport, der nur aus vollbeladenen Bewegungen besteht, die Objekte nur in die richtige Richtung transportieren, genau erreicht. Da aber ein solcher Transport für das präemptive CDARP auf einem Pfad immer konstruiert werden kann, wie wir in Abschnitt 2.4 gsehen haben, ist die Fluß-Schranke auch immer gleich den Kosten einer optimalen Lösung. Das Balancieren erhöht also die Kosten eines optimalen präemptiven Transports nicht.

Dies gilt aber nicht für den nicht-präemptiven Fall. Dies liegt im wesentlichen an zwei Gründen. Einerseits gibt es Instanzen des nicht-präemptiven CDARP, deren minimale Kosten sich durch Balancieren erhöhen. In der Abbildung 2.5 sieht man ein Beispiel für diesen Fall. Dies liegt daran, daß im optimalen Transport Bewegungen enthalten sind, die Objekte, deren Zielknoten über dem Startknoten liegen, abwärts transportieren. Objekte werden also sozusagen in die „falsche“ Richtung transportiert. Dadurch werden diese Objekte mehrmals über dieselbe Kante transportiert. Will man aber in einer balancierten Instanz des CDARP einen Transport mit Kosten finden, die nicht höher als die Fluß-Schranke sind, so darf dieser Transport nur Bewegungen enthalten, die Objekte in die richtige Richtung transportieren.

Andererseits gibt es Instanzen mit mehreren Zusammenhangskomponenten im Digraphen D. In diesen kommen für einen nicht-präemptiven Transport zu den

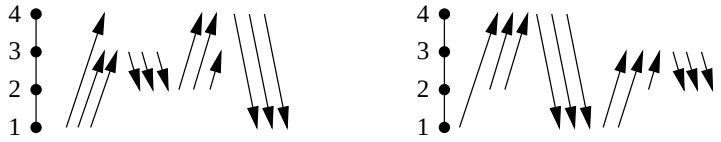

Abbildung 2.7: mögliche optimale Lösungen für Abbildung 2.6

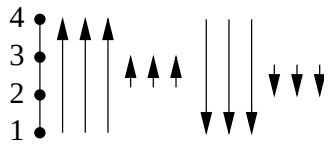

Abbildung 2.8: 3-CDARP mit zwei Zusammenhangskomponenten.

Kosten der Fluß-Schranke noch die Kosten für die Fahrten zwischen den Zusammenhangskomponenten hinzu. Im präemptiven balancierten Instanzen des CDARP benötigt man solche Fahrten nicht, da man Objekte an jedem beliebigen Knoten abladen kann, also Bewegungen an jedem beliebigen Knoten beenden kann. Und in balancierten Instanzen ist der unterliegende Graph immer zusammenhängend. Im nicht-präemptiven CDARP hingegen, kann man aber Objekte nur an ihrem Endknoten ablegen. Ein Beispiel für diesen Fall ist in der Abbildung 2.8 dargestellt.

Nun betrachten wir nur stark zusammenhängende balancierte Teilgraphen des Auftragsgraphen. Angenommen eine Bewegung eines Transports endet an dem Knoten v. Ist nun $\lambda_{v-1} > \lambda_v$, so enden am Knoten v mehr Aufträge, als dort beginnen. Tritt nun der Fall auf, daß während eines nicht-präemptiven Transports am Knoten v weniger als k Objekte abgeladen werden, aber nicht genügend Objekte für eine vollbeladene Bewegung in die aktuelle Richtung zur Verfügung stehen, so überschreiten die Kosten dieses Transports die Fluß-Schranke. Es gibt aber Instanzen des nicht-präemptiven CDARP, wo sich diese Situation nicht vermeiden läßt. In der Abbildung 2.8 ist dieser Fall dargestellt. Startet der Aufzug am Knoten 1 mit einer vollbeladenen Bewegung, so lädt er am Knoten 3 zwei Aufträge ab, kann aber keine neuen in Aufwärtsrichtung aufladen. Also ist hier die Fluß-Schranke nicht scharf. Entweder muß die Kante $(1, 2)$ mit einer nicht voll-ausgelasteten Bewegung überquert werden oder die Kante $(3, 4)$. Die Objekte in die Rückrichtung können dabei beliebig verteilt sein.

2.6 Heuristiken für CDARP

Die in den vorigen Abschnitten untersuchten optimale Algorithmen für das DARP und das präemptive CDARP auf Pfaden reichen, wie wir gesehen haben, nicht aus, um gute Transportpläne für Mehrplatz-Aufzüge zu ermitteln. Daher untersuchen wir nun Heuristiken, die einen gültigen Transport für das CDARP auf Pfaden ermitteln.

Eine Möglichkeit einen Transport zu erzeugen besteht darin, die Kapazität auf Eins zu setzen, und das Problem als DARP zu lösen. Denn jeder gültige Transport für das DARP ist automatisch ein gültiger Transport für das CDARP.

Das DARP auf Pfaden können wir mit dem Algorithmus OPTPATH optimal lösen (siehe Abschnitt 2.3. Ein gültiger Transport mit minimalen Kosten für eine Instanz des CDARP ist aber höchstens k-mal so groß, wie ein gültiger Transport mit minimalen Kosten für das k-CDARP mit ansonsten den glei-

chen Eingabe-Daten wie das DARP. Deshalb ist der Algorithmus OPTPATH ein k-Approximationsalgorithmus für das CDARP.

2.6.1 Abstimm-Heuristik

Ein Problem, welches bei nicht-präemptiven im Gegensatz zu präemptiven Problemen auftaucht, ist, daß am Ende einer Bewegung am aktuellen Knoten nicht genügend Aufträge in die aktuelle Richtung liegen, um den Aufzug voll auszulasten. Das Ändern der Richtung führt aber auch nicht zu einer voll ausgelasteten Bewegung, da im Aufzug noch Aufträge sein können, die in die aktuelle Richtung weitertransportiert werden müssen. Die Idee dieses Algorithmus ist es in solch einem Fall in die Richtung weiterzufahren, die die beste Auslastung des Aufzuges mit Aufträgen in die richtige Richtung gewährleistet. Die Objekte, die sich im Aufzug befinden und diejenigen, welche im aktuellen Stockwerk starten, stimmen also quasi ab, in welche Richtung sie fahren wollen. Dieser Algorithmus ist in Abbildung 4 formal beschrieben.

Dieser Algorithmus besitzt keine Gütegarantie besser als k.

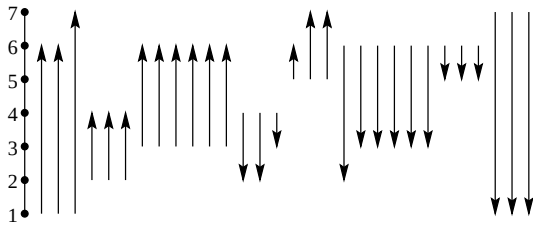

Abbildung 2.9: Instanz eines CDARP für das Gegenbeispiel zur Abstimm-Heuristik

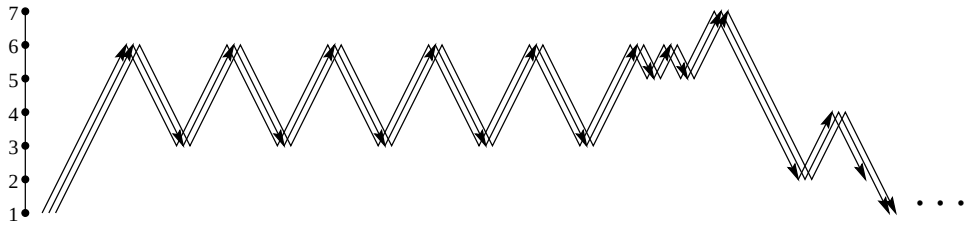

Abbildung 2.10: Ein Transport mit $kc(\mathcal{OPT})$ Kosten

In der Abbildung 2.9 ist eine Instanz des CDARP dargestellt, für die die Abstimm-Heuristik mindestens $kc(\mathcal{OPT})$ Kosten verursacht, wobei $c(\mathcal{OPT})$ die Kosten einer Optimallösung sind. In der Abbildung 2.10 ist der Transport dargestellt, den die Abstimm-Heuristik berechnet hat. Der Aufzug pendelt hier zwischen den Stockwerken 3 und 5 und transportiert dabei nur jeweils ein Objekt an sein Ziel. Dies liegt daran, daß der Aufzug eine Kapazität von 3 hat und ein Objekt enthält, welches aufwärts transportiert werden muß und ein Objejkt enthält, welches abwärts transportiert werden muß. An den Knoten, an denen er dann ein

1: $x := 1$, $d :=$ aufwärts, $M := \emptyset$
2: **Solange** noch Objekte existieren, die transportiert werden müssen **:**
3: Sei T eine Menge von Objekten am Knoten x, die in Richtung d transportiert werden sollen und T' die Menge von Objekte am Knoten x, die in die d entgegengesetzte Richtung transportiert werden sollen;
4: Sei M^+ die Menge der Objekte in M die in Richtung d transportiert werden sollen, sei $M^- := M \setminus M^+$;
5: **Wenn** $|T \cup T' \cup M| = 0$ **Dann**
6: Sei y der nächstgelegene Knoten, an dem ein Auftrag liegt;
7: Erzeuge eine Bewegung $(x, y, \emptyset)$;
8: x:=y;
9: **Ende Wenn**
10: **Wenn** $0 < |T \cup M| < k$ **Dann**
11: **Wenn** $|T' \cup M^-| > |T \cup M^+|$ **Dann**
12: Ändere die Richtung d;
13: $M := M \cup T'$
14: **Sonst**
15: $M := M \cup T$
16: **Ende Wenn**
17: **Sonst**
18: $M := M \cup T$
19: **Ende Wenn**
20: Sei W eine Teilmenge von Objekten aus M, deren Ziel y am nähesten bei x liegt. Ist $M = \emptyset$ sei y der nächste Knoten, an dem ein zu transportierendes Objekt liegt;
21: Erzeuge eine Bewegung (x, y, M);
22: Lege die Objekte aus W bei y ab; $M := M \setminus W$; $x := y$;
23: **Ende Solange**
24: **Wenn** $x \neq 1$ **Dann**
25: Erzeuge eine Bewegung $(x, 1, \emptyset)$;
26: **Ende Wenn**

Algorithmus 4: Abstimm-Heuristik

Objekt ablegt, findet er nur Objekte in die der aktuellen Fahrtrichtung entgegengesetzten Richtung.

Dies könnte man beliebig lange fortsetzen, also nähern sich für längere Sequenzen von Aufträgen $\langle 3, 5\rangle, \langle 5, 3\rangle$ die Kosten des durch den Abstimm-Algorithmus erzeugten Transportes $kc(\mathcal{OPT})$. An diesem Beispiel kann man auch sehen, daß mit einer besseren Auswahl der Menge T in diesem Algorithmus dieses Beispiel nicht zu verbessern ist, da an den entscheidenden Knoten nur gleichartige Aufträge oder genau k Aufträge starten.

2.6.2 Auf-Ab-Heuristik

In dieser Heuristik ändert der Aufzug seine Richtung nur, wenn er die Richtung nicht mehr beibehalten kann, weil er am Ende des Schachtes angelangt ist, oder weil in der aktuellen Richtung nichts mehr zu tun ist, also keine Objekte abgeholt oder abgeliefert werden müssen. Ansonsten lädt der Aufzug an jedem Knoten, an dem er vorbeikommt, soviele Aufträge wie möglich auf, nachdem er alle dort zu beendenden Aufträge abgeladen hat.

Diese Strategie führt zu einem Auf- und Abpendeln des Aufzuges.

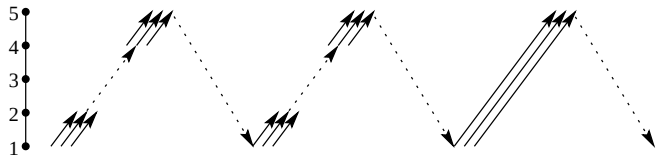

Abbildung 2.11: Ein Transport, der von der Auf-Ab-Heuristik gefunden wurde

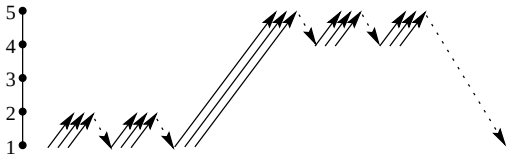

Abbildung 2.12: Optimale Lösung des Problems aus Abbildung 2.11

In der Abbildung 2.11 sieht man ein Beispiel für eine Instanz des CDARP,in der die Auf-Ab-Heuristik beliebig schlechte Transport erzeugt. In der optimalen Lösung in Abbildung 2.12 wird nämlich nur einmal vom 2. zum 4. Stockwerk gefahren, in der Lösung die von der Auf-Ab-Heuristik erzeugt wird, wird hingegen abhängig von der Anzahl der Aufträge oft diese Entfernung überwunden. Dieser Algorithmus hat aber als triviale obere Grenze $\max_{v \in V \setminus \{n\}} \lambda_v c(1, n)$, da der Aufzug über die Kante $(v, v+1)$ mit maximalen λ_v immer vollbeladen (außer eventuell beim letzten Mal) fahren wird.

Dieser Algorithmus ist einfach für das FIFO-CDARP zu erweitern, indem man bei der Erzeugung einer Bewegung immer nur die Objekte auflädt, welche die FIFO-Ordnung respektieren.

2.6.3 First-Fit-Heuristik

Diese Heuristik funktioniert wie die Auf-Ab-Heuristik. Nur wird, wenn der Aufzug leer ist, nicht einfach weiter in die aktuelle Richtung gefahren, sondern zum nächsten Knoten, an dem ein Objekt liegt.

Die Kante zwischen den Stockwerken 4 und 7 wird von der Lösung der First-Fit-Heuristik in Abbildung 2.13 k-mal überquert, von der optimalen Lösung in Ab-

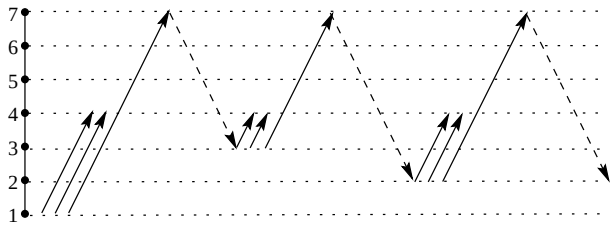

Abbildung 2.13: Ein Transport, der von der First-Fit-Heuristik gefunden wurde

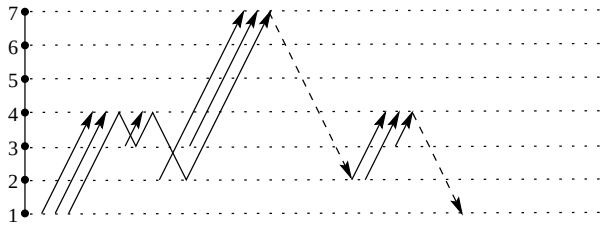

Abbildung 2.14: Optimale Lösung des Problems in Abbildung 2.13

bildung 2.14 hingegen nur einmal. Setzt man die Kantengewichte für diese Stockwerke im Verhältnis zu den anderen Kantengewichten sehr groß, nähert sich der Faktor mit dem sich die Kosten der optimalen Lösung von den Kosten der Lösung des First-Fit-Algorithmus unterscheiden, dem Wert k.

Aber auch für First-Fit gelten die Beobachtungen für die Auf-Ab-Heuristik: Und zwar ist $\max_{v \in V \setminus \{n\}} \lambda_v c(1, n)$ eine obere Schranke für die Kosten eines mit der First-Fit-Heuristik ermittelten Transports und die First-Fit-Heuristik läßt sich ebenfalls einfach für das FIFO-CDARP erweitern.

2.7 Ein Approximationsalgorithmus für CDARP auf Pfaden

Charikar und Raghavachari behaupten in [12] ohne Beweis, daß ein Algorithmus für das CDARP auf Pfaden mit Gütegarantie 2 existiert. Wir haben einen 3-Approximationsalgorithmus gefunden und werden diese Gütegarantie hier beweisen.

Wir haben in Abschnitt 2.5 gesehen, das ein präemptiver Transport mindestens $c_{\text{flow}} = \sum_{v=1}^{n-1} \lambda_v$ Kosten verursacht. Diese Kosten entstehen, wenn in jeder Bewegung des Vertikalförderers seine Kapazität vollständig ausgelastet wird und nie ein Objekt in die falsche Richtung transportiert wird. Diese Schranke ist gleichzeitig eine untere Schranke für die Lösung des nicht-präemptiven CDARP. Sie reicht aus, um einen 3-Approximationsalgorithmus für den nicht-präemptiven CDARP zu finden. Der hier vorgestellte Algorithmus läuft im wesentlichen zweistufig ab.

1. Finde jeweils zwei Folgen von Bewegungen, die über jede Kante, über die noch Objekte transportiert werden müssen, zusammen mindestens k Ob-

jekte transportieren, also die Kapazität des Aufzugs im Schnitt mindestens zur Hälfte ausnützen. Der Aufzug ändert während dieser Bewegungen seine Richtung nicht. Alle diese Objekte werden also in dieselbe Richtung transportiert. Dabei wird kein Objekt in die falsche Richtung transportiert. Diese Folgen von Bewegungen verursachen zusammen also höchstens die doppelten Kosten der Flußschranke. Diesen Teilalgorithmus nennen wir PATHFINDER.

2. Füge die Teiltransporte zu einem gültigen Transport zusammen.

Wir sehen später, daß das Ergebnis des ersten Teil des Algorithmus als nichtpräemptives DARP interpretiert werden kann. Der Algorithmus PATHFINDER reduziert das Problem also auf den Fall mit Kapazität 1, welches bekanntermaßen in polynomialer Zeit optimal zu lösen ist, wie wir in Abschnitt 2.4 bewiesen haben.

2.7.1 Idee des Algorithmus PATHFINDER

Zuerst wollen wir uns mit dem ersten Teil des Algorithmus, also mit dem Algorithmus PATHFINDER beschäftigen. Wir geben hier nur eine informelle Beschreibung des Algorithmus. Den Beweis der Korrektheit und der Gütegarantie wollen wir später führen.

Da immer nur Objekte, die in dieselbe Richtung transportiert werden müssen, in einem Durchgang von dem Algorithmus PATHFINDER bearbeitet werden, teilen wir den Auftragsgraphen D in zwei Teilgraphen auf, die nur Aufträge enthalten, die alle in dieselbe Richtung transportiert werden müssen. Der eine Teilgraph, D^+, enthält alle Objekte, die nach oben transportiert werden müssen, also

$$D^+ := (V, \{i \in A : i^+ < i^-\})$$

und der andere,

$$D^- := (V, i \in A : i^+ > i^-\}),$$

enthält nur die Objekte, die nach unten transportiert werden müssen.

Wir wollen zunächst nur den Algorithmus PATHFINDER für den Teilgraphen D^+ beschreiben, der die Aufträge enthält, die von unten nach oben, also aufwärts, transportiert werden müssen.

Wir benutzen folgende Notation: Den Startknoten, also den untersten Knoten eines Pfades P, bezeichnen wir mit $\alpha(P)$, den Endknoten, also den obersten Knoten mit $\omega(P)$. Ebenso bezeichnen wir für eine Multimenge $\emptyset \neq M_V \subset V \times V$ von Bögen mit $\alpha(M_V)$ den kleinsten Startknoten eines Bogens aus M_V und mit $\omega(M_V)$ den größten Endknoten eines Bogens aus M_V. Also ist

$$\alpha(M_V) = \min\{u \in V | \exists v \in V : \langle u, v\rangle \in M_V\}$$

und

$$\omega(M_V) = \max\{v \in V | \exists u \in V : \langle u, v\rangle \in M_V\}$$

Für $M_V = \emptyset$ definieren wir $\alpha(M_V) := \min\{v \in V\}$ und $\omega(M_V) := \max\{v \in V\}$.

Der Algorithmus PATHFINDER konstruiert in der i-ten Iteration für $i = 1, \ldots, k$ zwei disjunkte Mengen P_i^1 und P_i^2 von Bögen aus D^+, die zusammen jede Kante eines Teilgraphen $L(w_0, w_1)$ von G überdecken. Dieser Teilgraph hat die Eigenschaft, daß w_0 der kleinste Knoten ist, für den $\lambda_{w_0} > 0$ ist, und $w_1 > w_0$ der kleinste Knoten ist, für den $\lambda_{w_1} = 0$ ist, oder falls ein solcher nicht existiert, gilt $w_1 = n$. Der Algorithmus PATHFINDER betrachtet also nur eine Teilmenge von Bögen aus D^+. Der darunterliegende Graph dieser Teilmenge enthält keine Kante, die nicht von einem Bogen aus D^+ überdeckt wird.

Jede der beiden Mengen P_i^1 und P_i^2 läßt sich durch das Hinzufügen von Bögen zu einem Pfad von dem Knoten w_0 zu dem Knoten w_1 ergänzen. Es dürfen also in keiner der beiden Mengen zwei Bögen enthalten sein, die die gleiche Kante $(v, v+1), v \in G$ überdecken. Anschaulich bedeutet daß, daß sich keine zwei Bögen in einer Menge „überlappen".

Dies wird dadurch erreicht, daß die Mengen P_i^1, P_i^2 folgendermaßen konstruiert werden:

1. Sind beide Mengen leer, wähle P_i^1, ansonsten wähle die Menge, welche nicht den Bogen mit maximalen Endknoten enthält, d.h. wähle P_i^ϕ, so daß $\omega(P_i^\phi) < \omega(P_i^\psi), (\phi, \psi) \in \{(1,2),(2,1)\}$ gilt.

2. füge zu dieser Menge P_i^ϕ die Bogenmenge eines Pfades Q aus D^+ hinzu. Dieser Pfad Q hat folgende Eigenschaften:

   (a) $\alpha(Q) > \omega(P_i^\phi)$, d.h. der Pfad Q beginnt hinter dem Knoten, an dem der Bogen mit höchstem Endknoten aus P_i^ϕ endet.

   (b) $\alpha(Q) < \omega(P_i^\psi)$, d.h. der Pfad Q beginnt vor dem Knoten, an dem der Bogen mit höchstem Endknoten aus der anderen der beiden Mengen endet.

   (c) Q ist der Pfad mit maximalen Endknoten für den die Eigenschaften 2a und 2b gelten.

3. Entferne die Bögen aus Q aus dem Digraphen D^+.

Später werden wir zeigen, daß stets ein Pfad mit diesen Eigenschaften existiert. Dies wird solange durchgeführt, bis eine der beiden Mengen einen Bogen enthält, der an dem Knoten w_1 endet.

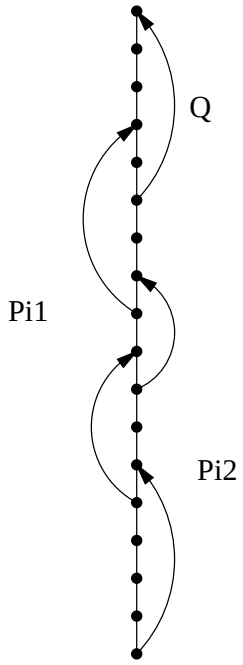

Abbildung 2.15: Die Pfeile in dieser Abbildung entsprechen den Pfaden Q, die zu den Bogenmengen P_i^{ϕ} und P_i^{ψ} hinzugefügt werden.

Nach k Iterationen haben wir also $2k$ Mengen P_i^1, P_i^2, $i = 1, \ldots, k$ konstruiert, die sich zu Pfaden vom Knoten w_0 zum Knoten w_1 ergänzen lassen. Wir können jeweils k dieser Mengen zu einem Transport zusammenfassen, der nur Bewegungen enthält, die maximal k Objekte auf einmal transportieren.

Diese Transporte konstruieren wir, indem wir die Mengen von Aufträgen P_i^1, $i = 1, \ldots, k$ und P_i^2, $i = 1, \ldots, k$ zu zwei Transporten zusammenfügen. Dies machen wir mit dem Algorithmus „Transporte konstruieren(aufwärts)“.

Dazu generieren wir die Bewegungen $(v, v+1, M)$, $v = w_0, \ldots, w_1 - 1$. Die Menge der transportierten Objekte M ist am Anfang leer. Dann nehmen wir für jeden Bogen (v, w) aus D^+, der in einem der Pfade P_i^1, $i = 1, \ldots, k$ enthalten ist, das entsprechende Objekt, welches nach w transportiert werden muß, zu der Menge M hinzu. Ist ein Objekt in M, welches am Knoten v abgelegt werden muß, wird es aus M entfernt.

Da immer die Bögen aus den beiden Mengen P_i^1 und P_i^2 zusammen alle Kanten aus $L(w_0, w_1)$ überdecken, müssen die Bewegungen in beide Teiltransporte zusammen im Durchschnitt mindestens zur Hälfte ausgelastet sein.

Den Algorithmus PATHFINDER können wir jetzt so oft durchführen, bis der Digraph D^+ keine Bögen mehr enthält. Dadurch erhalten wir eine Menge von Transporten, die wir als Aufträge in einem DARP interpretieren können, welches wir optimal mit dem Algorithmus OPTPATH aus Abschnitt 2.3 lösen können.

1: Wähle eine Menge $\mathcal{P}$ von k Mengen aus den Mengen $P_i^1, i = 1, \ldots, k$ und $P_i^2, i = 1, \ldots, k$.
2: Sei $v \in V$ der unterste Knoten zu dem ein Bogen aus einer der Mengen in $\mathcal{P}$ inzident ist. Sei $M := \emptyset$.
3: **Solange** v nicht der oberste Knoten in V ist, zu dem ein Bogen aus einer der Mengen $\mathcal{P}$ inzident ist **:**
4: Füge zu M die Bögen aus A hinzu, die zu v inzident sind und in einer der Mengen in $\mathcal{P}$ enthalten sind.
5: Sei w der kleinste Knoten zu dem ein Bogen aus M inzident ist.
6: Erzeuge eine Bewegung (v, w, M)
7: $v := w$, Entferne aus M alle Bögen, die zu w inzident sind.
8: **Ende Solange**
9: Führe die „Solange"-Schleife nocheinmal für die restlichen k Mengen von Bögen aus.

Algorithmus 5: Transporte konstruieren(aufwärts)

2.7.2 Die Algorithmen PATHFINDER und COVERPATH

Wir betrachten hier wiederum zunächst nur den Teilgraphen D^+. Die Ergebnisse für diesen Teilgraphen lassen sich aber direkt auf den Digraphen D^- übertragen.

Da die Knoten V von 1 bis n in der Reihenfolge, in der sie im Pfad auftreten, durchnumeriert sind, ist $v < w$ für $v, w \in V$ äquivalent zu „v liegt unterhalb von w".

Der Algorithmus PATHFINDER konstruiert noch keine gültigen Transporte, sondern Mengen von Bögen, aus denen sich mit dem Algorithmus in Abbildung 7 Transporte konstruieren lassen.

Diese Transporte, wir nennen sie Auf- und Abwärts-Teiltransporte, erfüllen bestimmte Eigenschaften.

Definition 2.7.1 (Aufwärts-Teiltransport). Wir bezeichnen eine Folge von Bewegungen als (nicht-präemptiven) *Aufwärts-Teiltransport*, wenn sie

1. im untersten Knoten w_0 des Graphen D beginnt, für den $\lambda_{w_0} > 0$ ist,
2. am untersten Knoten $w_1 > w_0$ endet, für den $\lambda_{w_1} = 0$ ist oder, falls für alle $w = 1, \ldots, n$ gilt, daß $\lambda_{w_1} > 0$ ist, am obersten Knoten n in D^+ endet,
3. in jeder Bewegung maximal k Objekte befördert, und
4. wenn jedes Objekt nur an seinem Zielknoten abgelegt wird.

Einen Abwärts-Teiltransport definieren wir analog, nur daß er von oben nach unten verläuft.

Definition 2.7.2. Wir nennen einen Pfad P aus einem Digraphen D *maximal*, wenn in D kein Pfad $Q \neq P$ existiert, der P vollständig enthält.

Der Algorithmus PATHFINDER findet nun solche Mengen von Bögen, die jede Kante mindestens k-fach überdecken. Dazu fügen wir zu einer Menge von Bögen jeweils einen *maximalen* Pfad mit den Eigenschaften wie in Abbildung 2.15 hinzu. Aus diesen Mengen von Bögen konstruieren wir dann mit dem Algorithmus 5 Teiltransporte.

Vorbedingung: D^+ ist ein Digraph, der nur nach oben gerichtete Bögen enthält.
Liefert: $P_i^1, P_i^2, i = 1, \ldots, k$ sind paarweise disjunkte Teilmengen von $A(D^+)$.

1: Sei w_0 der unterste Knoten im Graphen D^+, an dem Objekte liegen;
2: Sei $w_1 > w_0$ der unterste Knoten im Graphen D^+ mit $\lambda_{w_1} = 0$ oder, falls für alle $w = 1, \ldots n$ gilt, daß $\lambda_w > 0$ ist, sei w_1 der oberste Knoten in D^+;
3: **Von** $i = 1$ bis k **:**
4: Seien $P_i^1 = \emptyset$; $P_i^2 = \emptyset$;
5: Sei $\phi = 1$ und $\psi = 2$;
6: **Solange** P_i^ψ vor dem Knoten w_1 endet oder leer ist **:**
7: Existieren Kanten $(w, w+1), w \in \{w_0, \ldots, w_1 - 1\}$, über die keine Bögen aus D^+ laufen, dann füge jeweils einen entsprechenden Bogen $\langle w, w+1 \rangle$ zu D^+ hinzu;
8: Suche den maximalen Pfad Q in D^+ mit höchstem Endknoten, der $(\omega(P_i^\psi), \omega(P_i^\psi)+1)$ überdeckt oder, falls $P_i^\psi = \emptyset$ ist, (w_0, w_0+1) überdeckt;
9: Füge Q zu P_i^ϕ hinzu;
10: Entferne die Bögen in Q aus D^+;
11: Vertausche ϕ und ψ;
12: **Ende Solange**
13: **Ende Von**

Algorithmus 6: PATHFINDER (aufwärts)

Auf die gleiche Art, wie die Aufwärts-Teiltransporte, werden auch die Abwärts-Teiltransporte konstruiert, nur daß hier der Digraph D^- betrachtet wird und entsprechend die Pfade von oben nach unten konstruiert werden.

Zweite Phase

Nachdem wir eine Menge von Teiltransporten mit dem Algorithmus PATHFINDER gefunden haben, müssen wir diese noch zu einem gültigen Transport zusammenfügen.

Die Kosten eines Aufwärts-Teiltransportes T sind genau die Kosten für das Durchfahren des Graphen G vom Knoten $\alpha(T)$ zum Knoten $\omega(T)$, da ein Teiltransport nur Bewegungen in eine Richtung enthält. Also definieren wir $c(T) := c(\alpha(T), \omega(T))$. Analoges gilt für die Abwärts-Teiltransporte. Betrachten wir nun den gemischten Graphen $G' = (V, E, \mathcal{T})$, wobei $\mathcal{T}$ die Menge der Bögen ist, die durch die Teiltransporte induziert werden, also

$$\mathcal{T} := \{\langle \alpha(T), \omega(T) \rangle : T \text{ ist ein Teiltransport}\},$$

dann ist durch $(G', c, 1)$ ein DARP auf einem Pfad gegeben, welches wir in polynomialer Zeit optimal lösen können.

Der komplette 3-Approximationsalgorithmus sieht also folgendermaßen aus:

1: $i = 1$
2: **Solange** D^+ Bögen enthält **:**
3: PATHFINDER (aufwärts);
4: Erzeuge Aufwärts-Teiltransporte $T^+_{1,i}$ und $T^+_{2,i}$ aus den Pfaden $P^1_j, P^2_j, j = 1, \ldots, k$.
5: PATHFINDER (abwärts);
6: Erzeuge Abwärts-Teiltransporte $T^-_{1,i}$ und $T^-_{2,i}$ aus den Pfaden $P^1_j, P^2_j, j = 1, \ldots, k$.
7: $i := i + 1$.
8: **Ende Solange**
9: Füge die Aufwärts- und Abwärtsteiltransporte zu einem Transport zusammen.

Algorithmus 7: COVERPATH

2.7.3 Beispiel

Nun führen wir den Algorithmus PATHFINDER am Beispiel des Problems in Abbildung 2.16 vor. Ein Pfeil in der Abbildung entspricht dabei einem zu transportierenden Objekt. Der Aufzug hat eine Kapazität von Drei. Dieses Problem ist balanciert.

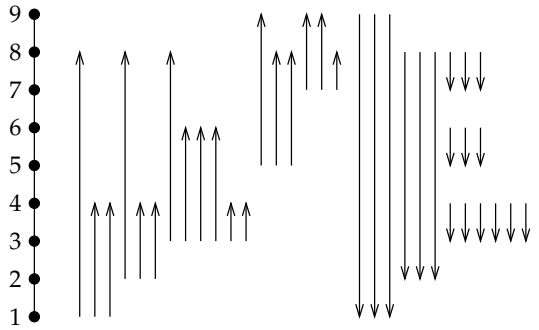

Abbildung 2.16: Ein CDARP mit $k = 3$, 9 Stockwerken und 36 Aufträgen

Im ersten Aufruf der Algorithmen PATHFINDER (aufwärts) und PATHFINDER (abwärts) werden aus dem Graphen D, der oben in der Abbildung 2.17 dargestellt ist, die unten links in der Abbildung dargestellten Mengen P^1_i und P^2_i, $i = 1, \ldots, k$ erzeugt. Jeweils übereinanderliegende Objekte gehören zu einer Menge. Von links nach rechts gehören die Objekte zu den Mengen P^1_1, P^2_1, P^1_2, P^2_2, P^1_3 und P^2_3. Die gepunkteten Aufträge auf der linken Seite der Abbildung sind die, welche aus dem Digraphen D entfernt werden. Es gibt auch sechs Mengen für die Aufträge abwärts. Die Mengen $P^2_i, i = 1, 2, 3$, die durch PATHFINDER (abwärts) konstruiert werden, sind aber jeweils leer, da die Mengen $P^1_i, i = 1, 2, 3$, jeweils schon den

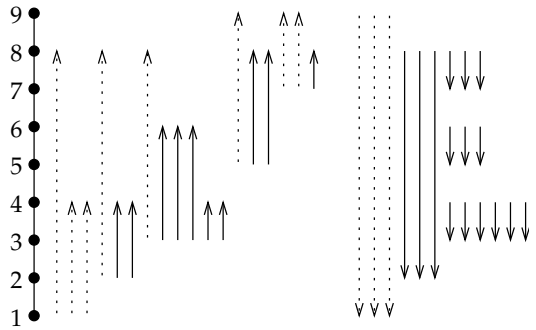

Abbildung 2.17: Der erste Durchlauf von COVERPATH. – Die gepunkteten Pfeile werden aus dem Auftragsgraphen gelöscht, und zu Mengen von Bögen zusammengefaßt, aus denen Teiltransporte konstruiert werden, diese Mengen sind in der Abbildung 2.18 dargestellt.

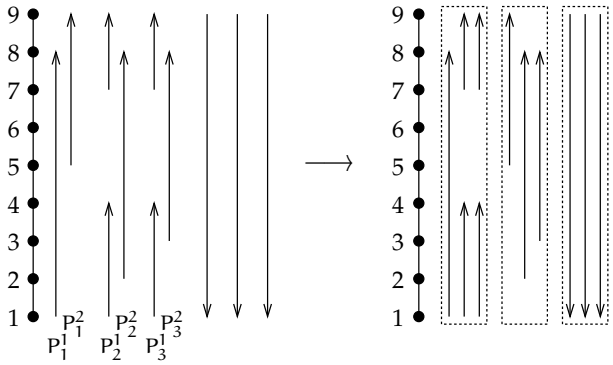

Abbildung 2.18: Die Mengen $P_i^j, i = 1, 2, 3, j = 1, 2$ nach dem ersten Durchlauf und die daraus resultierenden Teiltransporte

gesamten Graphen $L(1, 9)$ überdecken. Auf der rechten Seite der Abbildung sind die Teiltransporte dargestellt. Jeder Kasten enthält einen Teiltransport. Das Zusammenfügen der Teiltransporte zum engültigen Transport führen wir erst am Schluß durch.

Die gestrichelten Aufträge in der Abbildung 2.19 sind diejenigen, welche im Schritt 7 von PATHFINDER deshalb hinzugefügt werden, weil über die entsprechende Kante kein Auftrag mehr führte. Die gepunkteten Aufträge sind wiederum diejenigen, welche entfernt werden. Hier wird nur noch der Graph $L(2, 8)$ betrachtet, da über die Kanten $(1, 2)$ und $(8, 9)$ keine Aufträge mehr laufen.

In Abbildung 2.21 sieht man die Situation nach dem 2. Durchlauf. Aus jeweils drei Aufträgen mit gleichen Start- und Zielknoten werden Teiltransporte erzeugt. Der zweite Teiltransport ist jeweils leer, daher werden wir ihn bei der Konstruktion des Transports aus den Teiltransporten weglassen.

Nun müssen die erzeugten Teiltransporte noch zu einem Transport zusammengefaßt werden. Wie der vollständige Transport dann aussieht, sieht man in Ab-

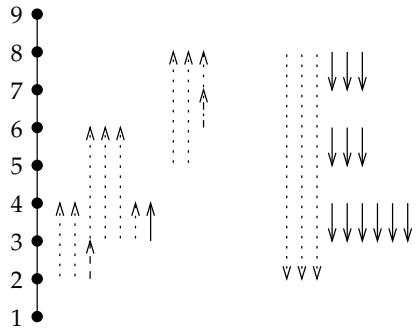

Abbildung 2.19: Die gepunkteten Pfeile sind wiederum diejenigen, die zu Teiltransporten werden. Zusätzlich werden die gestrichelten Aufträge ergänzt, da ansonsten keine Überdeckung des Graphen möglich wäre. Die resultierenden Teiltransporte sind in Abbildung 2.20 abgebildet.

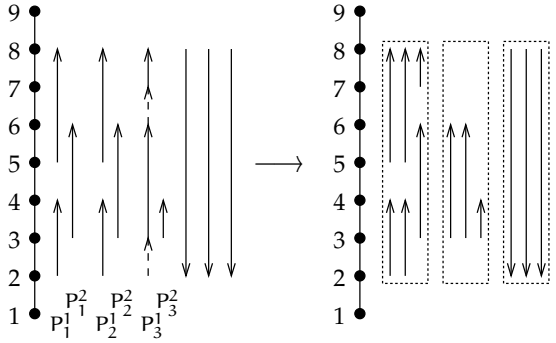

Abbildung 2.20: Die Mengen $P_i^j, i = 1,2,3, j = 1,2$ nach dem zweiten Durchlauf und die daraus resultierenden Teiltransporte

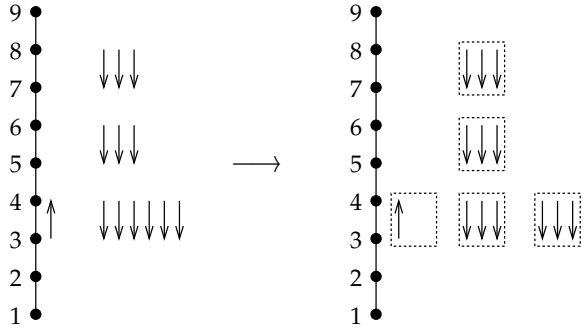

Abbildung 2.21: letzter Durchlauf – die letzten Aufträge werden eingesammelt

bildung 2.23. Die Aufträge, die wegen der Kanten ohne hinüberführende Aufträge hinzugekommen sind, wurden in dieser Abbildung wieder entfernt. Angenommen die Kosten für eine Fahrt von einem Knoten zu seinem Nachbarn betragen 1, dann verursacht dieser Transport Kosten von 50. Die Fluß-Schranke ist 36, der Wert der optimalen Lösung dagegen 42.

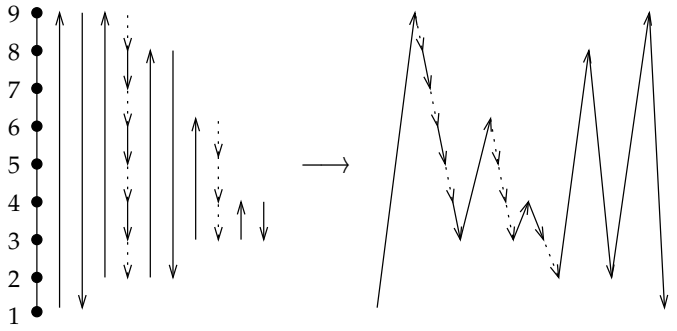

Abbildung 2.22: Erzeugen eines Transports aus den Teiltransporten. Jeder durchgezogene Pfeil entspricht einem Teiltransport, die gepunkteten Pfeile entsprechen leeren Bewegungen, die zum Balancieren des Problems eingefügt wurden.

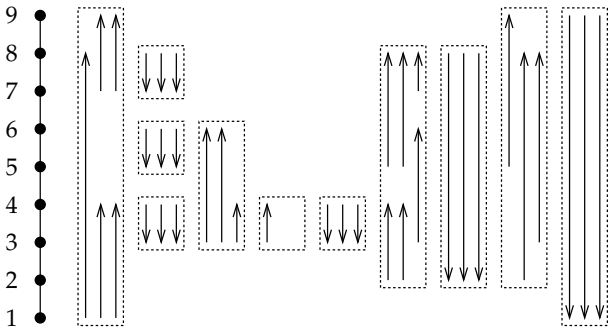

Abbildung 2.23: Der Transport, der dem in Abbildung 2.22 gefundenen entspricht.

2.7.4 Korrektheit des Algorithmus PATHFINDER

Die Korrektheit des Algorithmus PATHFINDER folgt im wesentlichen aus der Tatsache, daß die Konstruktion der Mengen $P_i^1, P_i^2, i = 1, \ldots, k$ immer so erfolgt, daß die beiden Mengen P_i^1 und P_i^2 zusammen den von den übrigen Objekten induzierten Graphen überdecken, und daß sich jede dieser beiden Mengen zu einem Pfad ergänzen läßt.

Der Algorithmus terminiert, weil in jedem Durchlauf der inneren Schleife des Algorithmus PATHFINDER der höchste Endknoten eines Bogens beider Mengen größer wird. Genauer gesagt, wird der Menge von P_i^1 und P_i^2 mit dem kleineren höchsten Endknoten eines Bogens eine Menge von Bögen hinzugefügt, die den nächsten *Engpaß* überdecken.

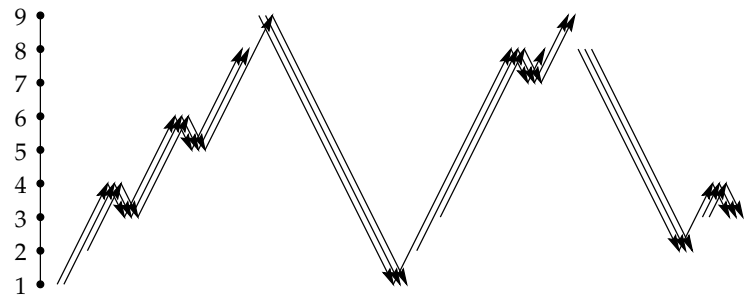

Abbildung 2.24: optimale Lösung des Problems mit einem ganzzahligen linearen Programm und CPLEX gefunden (siehe Abschnitt 3.1).

Um zu definieren, was ein Engpaß ist, benötigen wir Zähler für die Anzahl der Objekte, die über eine Kante führen. Sei $\mu_v := f([1, v], [v + 1, n])$ die Anzahl der Objekte, die über die Kante $(v, v + 1)$ von unten nach oben transportiert werden müssen.

Definition 2.7.3 (Engpaß). Ein *Engpaß* ist eine Kante $(v, v + 1)$ für die ein Knoten $w > v$ existiert, so daß gilt:

1. v ist nicht der unterste Knoten und w ist nicht der oberste Knoten und $\mu_{v-1} > \mu_v = \mu_{v+1} = \cdots = \mu_{w-2} = \mu_{w-1} < \mu_w$ oder
2. v ist der unterste Knoten und $\mu_v = \mu_{v+1} = \cdots = \mu_{w-2} = \mu_{w-1} < \mu_w$ oder
3. w ist der oberste Knoten und $\mu_{v-1} > \mu_v = \mu_{v+1} = \cdots = \mu_{w-2} = \mu_{w-1}$.

Ein Engpaß ist also, bezogen auf die Anzahl von Objekten, die über die Kanten des Graphen transportiert werden müssen, ein lokales Minimum.

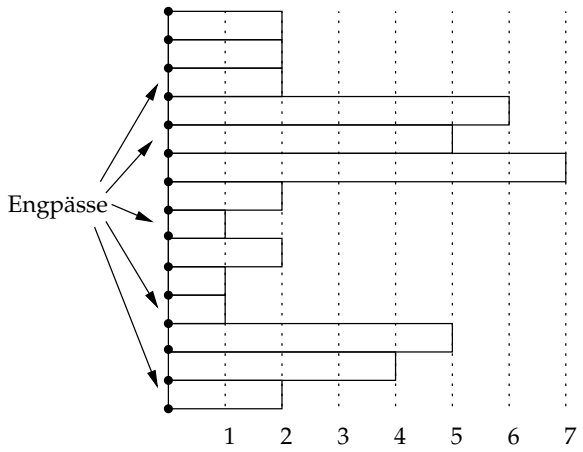

Abbildung 2.25: Engpässe. Die Länge eines Balkens entspricht der Anzahl von Objekten, die über die entsprechende Kante transportiert werden müssen.

Aus dieser Definition folgt auch:

Beobachtung 2.7.4. *Ist $\mu_{v-1} > \mu_v$, so existiert ein Engpaß $(w, w+1)$ mit $w \geqslant v$.*

Beweis. Ist nämlich $\mu_{v-1} > \mu_v$, so tritt entweder der 1. oder der 3. Fall der Definition 2.7.3 ein. □

Beobachtung 2.7.5.
1. *Ist $\mu_{v-1} \leqslant \mu_v$, dann folgt, daß man für jeden Auftrag, der am Knoten v beginnt, einen (Vorgänger-)Auftrag finden kann, der im Knoten v endet.*
2. *Nur wenn $\mu_{v-1} > \mu_v$ gilt, kann am Knoten v ein maximaler Pfad P enden.*
3. *Ist $\mu_{v-1} \geqslant \mu_v$, dann folgt, daß man für jeden Auftrag, der am Knoten v endet, einen (Folge-)Auftrag finden kann, der im Knoten v beginnt.*
4. *Nur wenn $\mu_{v-1} < \mu_v$ gilt, kann am Knoten v ein maximaler Pfad beginnen.*

Beweis.
1. Die Bögen, die sowohl die Kante $(v, v-1)$ als auch die Kante $(v, v+1)$ überdecken, werden mit μ_{v-1} und mit μ_v gezählt. Hingegen werden Bögen, die am Knoten v enden, nur mit μ_{v-1} gezählt und Bögen, die am Knoten v beginnen, nur mit μ_v. Also folgt aus $\mu_{v-1} \geqslant \mu_v$, daß am Knoten v mehr Aufträge enden als beginnen. Daraus folgt die Behauptung.
2. Würde $\mu_{v-1} > \mu_v$ nicht gelten, könnte man wegen 1. einen Bogen finden, der am Knoten v beginnt. Dies wäre ein Widerspruch dazu, daß der Pfad P maximal ist.
3. und 4. folgen analog.

□

Lemma 2.7.6. *Gegeben sei ein Digraph $D^+ = (V, A)$. Sei $(w, w+1)$ ein Engpaß in D und sei x der kleinste Knoten in V für den $x > w$ und $\mu_x > \mu_w$ gilt, falls dieser existiert. (Der Knoten x ist also quasi das „obere Ende" des Engpasses.) Ansonsten sei x der oberste Knoten in V. Falls nun $\mu_w > 0$ gilt, dann existiert immer ein gerichteter Pfad in D^+, der die Kanten $(w-1, w)$ und $(x-1, x)$ überdeckt.*

Beweis. Wegen $\mu_w > 0$ muß mindestens ein Bogen den Engpaß $(w, w+1)$ überdecken. Da für alle Knoten x' für die $w < x' < x$ gilt, $\mu_{x'} = \mu_w$ sein muß, nach Definition 2.7.3, folgt aus der Beobachtung 2.7.4, daß jede Kante des Graphen $L(w, x)$ mit einem Pfad P überdeckt werden kann.

Da nun wegen der Engpaßeigenschaften $\mu_w > \mu_{w+1}$ gilt, folgt aus der Beobachtung 2.7.5, 3., daß man, falls der Pfad P am Knoten v beginnt, auch einen Bogen findet, der am Knoten v endet. Der Pfad P zusammen mit diesem Bogen überdeckt dann die Kanten $(w-1, w)$ und $(x-1, x)$. □

Wir wollen nun folgende Invariante für den Algorithmus PATHFINDER beweisen:

Lemma 2.7.7. *Sei $\omega(P_i^\phi) < \omega(P_i^\psi)$. Dann hat im Schritt 8 der Pfad Q folgende Eigenschaften:*

1. *Der Pfad Q überdeckt mindestens einen Engpaß $(w, w+1)$, der über $\omega(P_i^\psi)$ beginnt.*
2. *Der Startknoten des Pfades $\alpha(Q)$ liegt unter dem höchsten Endknoten eines Bogens in P_i^ψ, also $\alpha(Q) \leqslant \omega(P_i^\psi)$.*
3. *Der Startknoten $\alpha(Q)$ liegt hinter dem höchsten Endknoten eines Bogens in P_i^ϕ, also $\alpha(Q) \geqslant \omega(P_i^\phi)$.*

Insgesamt muß also $\omega(P_i^\phi) < \alpha(Q) < \omega(P_i^\psi) < \nu < \omega(Q)$ gelten. Diese Situation ist in der Abbildung 2.26 dargestellt.

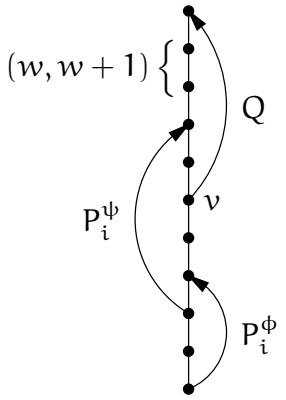

Abbildung 2.26: Darstellung der Situation: $\omega(P_i^\phi) < \alpha(Q) < \omega(P_i^\psi) < \nu < \omega(Q)$.

Beweis.

- Am Knoten $\omega(P_i^\psi)$ beginnen keine Bögen, da ansonsten in der letzten Iteration zu der Menge P_i^ψ nicht der maximale Pfad mit höchstem Endknoten hinzugefügt worden wäre. Also muß aufgrund der Beobachtung 2.7.5 gelten, daß $\mu_{\omega(P_i^\psi)-1} > \mu_{\omega(P_i^\psi)}$ ist. Aus der Beobachtung 2.7.4 folgt, daß ein Engpaß $(w, w+1)$ mit $w \geqslant \omega(P_i^\psi)$ existieren muß. Sei nun $(w, w+1)$ der Engpaß mit niedrigstem Knoten w für den $w \geqslant \omega(P_i^\psi)$ gilt.

  Nun finden wir einen Knoten x, mit den Eigenschaften des Knotens x aus Lemma 2.7.6. Insbesondere ist $x > w$. Da nun wegen Schritt 7 des Algorithmus jede Kante aus $L(w_0, w_1)$ von mindestens einem Bogen überdeckt wird, ist $\mu_w > 0$ und wir können das Lemma 2.7.6 anwenden. Wir finden also einen Pfad, der die Kanten $(w-1, w)$ und $(x-1, x)$ und damit auch den Engpaß $(w, w+1)$ überdeckt. Wir finden einen maximalen Pfad Q, der diesen Pfad enthält. Dieser beginne am Knoten ν. Ist $\nu < \omega(P_i^\psi)$, so haben wir schon den gesuchten Pfad gefunden.

  Nun nehmen wir an, daß $\nu > \omega(P_i^\psi)$ sei. Da der Pfad maximal ist, enden am Knoten ν keine Bögen, also gilt nach Beobachtung 2.7.5, daß $\mu_{\nu-1} < \mu_\nu$ ist.

Außerdem beginnen am Knoten $\omega(P_i^\psi)$ keine Bögen, also gilt

$$\mu_{\omega(P_i^\psi)-1} > \mu_{\omega(P_i^\psi)}.$$

Also liegt im Widerspruch zur Annahme zwischen den Knoten v und $\omega(P_i^\psi)$ ein weiterer Engpaß. Also muß $v < \omega(P_i^\psi)$ gelten.

- Jetzt müssen wir noch zeigen, daß der Pfad Q über dem Knoten $\omega(P_i^\phi)$ beginnt. Dazu zeigen wir, daß, in der letzten Iteration des Algorithmus PATHFINDER in Schritt 8 ein Fehler begangen worden wäre, falls $\alpha(Q) < \omega(P_i^\phi)$ gilt.

  Wir nehmen an, daß der Pfad Q unter dem Knoten $\omega(P_i^\phi)$ beginnt. Der Pfad Q überdeckt die Kante $(\omega(P_i^\psi), \omega(P_i^\psi)+1)$, da er mindestens einen Engpaß über dem Knoten $\omega(P_i^\psi)$ überdeckt. Also ist $\omega(Q) > \omega(P_i^\psi)$. Der Pfad Q hätte also einen Endknoten, der über dem Knoten $\omega(P_i^\psi)$ liegt. Dann hätte aber der Pfad Q einen höheren Endknoten als der Pfad Q′, der in der letzten Iteration zur Menge P_i^ψ hinzugefügt worden ist. Dies steht im Widerspruch zum Schritt 8 des Algorithmus PATHFINDER. Also muß der Pfad Q hinter dem Knoten $\omega(P_i^\phi)$ beginnen.

  Es gilt also

$$\omega(P_i^\phi) < \alpha(Q) < \omega(P_i^\psi) < w < \omega(Q).$$

□

Lemma 2.7.8. *In Schritt 6 des Algorithmus* PATHFINDER *gilt entweder*

- $P_i^\phi = P_i^\psi = \emptyset$ *oder*
- $\omega(P_i^\phi) > \omega(P_i^\psi)$

Beweis. Beim ersten Durchlaufen von Schritt 6 sind beide Mengen leer. Ansonsten gilt: Durch Hinzufügen zur Menge P_i^ϕ des Pfades Q im Schritt 8 des Algorithmus gilt, daß $\omega(P_i^\phi) < \omega(P_i^\psi)$ ist, da wegen Lemma 2.7.7 gilt, daß $\omega(Q) > \omega(P_i^\psi)$ ist. Da im Schritt 11 die Werte der Variablen ϕ und ψ vertauscht werden, folgt die Behauptung. □

Außerdem gilt in Schritt 6 des Algorithmus PATHFINDER:

Korollar 2.7.9. *Im Schritt 6 des Algorithmus* PATHFINDER *überdecken die Bögen in* $P_i^1 \cup P_i^2$ *jede Kante im Graphen* $L(w_0, \omega(P_i^\phi))$ *mindestens einmal und höchstens zweimal.*

Beweis. Die Behauptung folgt direkt aus dem Lemma 2.7.7: Denn dort zeigen wir, daß der Pfad Q, der in jeder Iteration hinzugefügt wird, hinter dem Ende der Menge beginnt, zu der er hinzugefügt wird. Also überdeckt jede der beiden Mengen P_i^1, P_i^2 alleine für sich genommen jede Kante höchstens einmal. Da aber der Pfad Q immer an einem Knoten unter dem Endknoten der anderen Menge beginnt, muß von beiden Mengen zusammen jede Kante mindestens einmal überdeckt werden. □

Nun gilt also für alle k Mengen $P_i^1, P_i^2, i = 1, \ldots, k$ zusammen folgender Satz:

Satz 2.7.10. *Bei einem Aufruf findet der Algorithmus* PATHFINDER $2k$ *Mengen von Bögen* P_i^1, P_i^2, $i = 1, \ldots, k$ *mit folgenden Eigenschaften:*

1. *Sie enthalten nur Bögen, deren Anfangsknoten größer oder gleich* w_0 *ist und*
2. *deren Endknoten kleiner oder gleich* w_1 *ist.*
3. *Zusammen enthalten diese* $2k$ *Mengen Bögen, die jede Kante aus dem Teilgraphen* $L(w_0, w_1)$ *mindestens* k*-mal überdecken*
4. *und die Mengen* P_i^1, P_i^2, $i = 1, \ldots, k$ *enthalten keine sich gegenseitig „überlappenden" Bögen, das heißt für je zwei Bögen* a_1, a_2 *aus der gleichen Menge gilt* $\alpha(a_1) \geqslant \omega(a_2)$ *oder* $\alpha(a_2) \geqslant \omega(a_1)$.

Beweis. Da in jeder Iteration Bögen zu den Mengen hinzugefügt werden, endet irgendwann eine der beiden Mengen am Knoten w_1. Auch in diesem Fall gilt das Lemma 2.7.7. Da es für jeweils die Mengen $P_i^1, P_i^2, i = 1, \ldots, k$ gilt, folgt der Satz. □

Man beachte aber, daß die Bögen in den Mengen $P_i^1, P_i^2, i = 1, \ldots, k$ nicht nur aus D^+ stammen, sondern daß auch, falls im Laufe des Algorithmus PATHFINDER eine Kante nicht mehr durch Bögen überdeckt wird, neue Bögen erzeugt werden. Diese neu erzeugten Bögen erhöhen aber, wie wir im nächsten Abschnitt sehen werden, die Kosten nicht.

Der Algorithmus PATHFINDER terminiert, weil in jedem Durchlauf der Schleife mindestens ein Bogen zu einer der beiden Mengen hinzugefügt wird und jede Menge maximal n Bögen enthalten kann.

2.7.5 Gütegarantie und Polynomialität

Um die Laufzeit des gesamten Algorithmus zu untersuchen, müssen wir genauer erklären, wie wir die Pfade im Digraphen D^+ suchen. Man kann den Digraphen D^+ mit Tiefensuche durchsuchen und sich dabei die längsten Pfade merken. Dabei wird jeder Bogen genau einmal durchlaufen. Mit Tiefensuche kann man für jeden Knoten ermitteln, welche Knoten von ihm aus über welche Bögen erreichbar sind. Die gesamten Suchoperationen benötigen daher $O(m)$ Laufzeit. Die „Solange"-Schleife durchläuft der Algorithmus maximal $n/2$ mal, da in jedem Durchlauf an einem der beiden Pfade Bögen angefügt werden, die den nächsten Engpaß überdecken und Engpässe immer mindestens zwei Knoten Abstand voneinander haben. Der Algorithmus PATHFINDER hat also eine Laufzeit von $O(knm)$.

Nun müssen wir noch zeigen, daß der Algorithmus COVERPATH ein 3-Approximationsalgorithmus ist. Dazu zeigen wir zunächst, daß der Algorithmus PATHFINDER für jede Kante $(v, v+1)$ höchstens $2\lambda_v$ Aufwärts-Teiltransporte erzeugt, die diese Kante überqueren. Damit sind die Kosten für die Teiltransporte höchstens doppelt so hoch wie c_{flow}, da das Argument für Abwärts-Teiltransporte analog gilt.

Lemma 2.7.11. *Über jede Kante* $(\nu, \nu+1)$ *gehen höchstens* $2\lambda_\nu$ *Aufwärts-Teiltransporte und höchstens* $2\lambda_\nu$ *Abwärts-Teiltransporte.*

Beweis. Nach Satz 2.7.10 findet der Algorithmus PATHFINDER in jedem Aufruf k Bögen, die die Kante $(\nu, \nu+1)$ überdecken. Ist vor Beginn des Algorithmus PATHFINDER $\mu_\nu \geqslant 2k$, dann werden nur Bögen aus D^+, welche die Kante $(\nu, \nu+1)$ überdecken, ausgewählt und keine Bögen im Schritt 7 ergänzt. Also reduziert sich λ_ν um mindestens Eins bei der Erzeugung der zwei Teiltransporte.

Ist aber vor Beginn des Algorithmus PATHFINDER $\mu_\nu < 2k$, dann kann es geschehen, daß alle Bögen, die die Kante $(\nu, \nu+1)$ überdecken, in die Menge $P_i^1, i = 1, \ldots, k$ und $P_i^2, i = 1, \ldots, k$ aufgenommen werden. Nur in diesem Fall werden neue Bögen erzeugt. Diese werden aber gleich wieder entfernt. Also reduziert sich λ_ν von Eins auf Null. Ist aber $\lambda_\nu = 0$, so führen über die Kante $(\nu, \nu+1)$ keine Bögen mehr, also ist $w_1 \leqslant \nu$ oder $w_0 \geqslant \nu+1$, es werden also nur noch Teiltransporte erzeugt, die nicht mehr über die Kante $(\nu, \nu+1)$ führen. Für Abwärtsteiltransporte gilt die Aussage analog. □

Nun müssen wir nur noch die Kosten für die Leerfahrten berücksichtigen, die durch das Zusammenfügen der Teiltransporte zu einem gültigen Transport entstehen. Wie in Kapitel 2.7.1 beschrieben, lösen wir dazu das DARP auf Pfaden und zwar das auf dem durch die Menge der Bögen $\mathcal{T} := \{\langle \alpha(T), \omega(T)\rangle : T \text{ ist ein Teiltransport}\}$ induzierten Digraphen. Betrachten wir den Algorithmus OPTPATH von Atallah und Kosaraju [6] aus Abschnitt 2.3, der dieses Problem optimal löst, dann sehen wir, daß dieser Algorithmus den Digraphen balanciert und dann zu einem stark zusammenhängenden Graphen erweitert. Das Balancieren verschlechtert die Gütegarantie nicht, da wir nach Lemma 2.7.11 wissen, daß über jede Kante $(\nu, \nu+1)$ höchsten $2\lambda_\nu$ Aufwärts- und höchstens $2\lambda_\nu$ Abwärts-Teiltransporte führen. Nach dem Balancieren führen zwar gleichviele Aufwärts- und Abwärts-Teiltransporte über $(\nu, \nu+1)$, das Maximum aber bleibt gleich. Das Balancieren führt auch dazu, daß für alle Knoten $\nu \in V$ gilt, daß $\delta^+(\nu) = \delta^-(\nu)$ ist. Das Erweitern zu einem stark zusammenhängenden Graphen aber kostet höchstens $2c(1, n)$ zusätzlich, da das Hinzufügen der Elementarbögen $\{\langle \nu, w\rangle : \nu = w+1, w \in V\backslash\{n\} \text{ oder } \nu = w-1, w \in V\backslash\{1\}\}$ jeden Graph stark zusammenhängend macht. Also gilt folgender Satz:

Satz 2.7.12. *Für die Kosten eines Transports* T*, der durch den Algorithmus* COVERPATH *erzeugt wird gilt:*

$$c(T) \leqslant 2c_{\mathit{flow}} + 2c(1, n) \leqslant 3c_{\mathit{flow}} \leqslant 3c_{\mathcal{OPT}}.$$

Damit ergibt sich insgesamt eine Gütegarantie von 3.

Der Algorithmus OPTPATH zum Lösen des DARP auf Pfaden ist in $O(|\mathcal{T}|\, n^2)$ (siehe Abschnitt 2.3). Die Laufzeit des Algorithmus PATHFINDER ist, wie wir oben gesehen haben, in $O(knm)$. Da die Anzahl aller Teiltransporte, also $|\mathcal{T}|$, kleiner oder gleich der Anzahl der Bögen in D ist, ist die Laufzeit der Einfügeoperation in $O(mn^2)$. Das Balancieren der Digraphen D^+ und D^- ist in $O(m+n)$, also ist die Laufzeit des gesamten Algorithmus COVERPATH in $O(knm^2)$ und damit polynomial.

2.8 Ein Algorithmus für CDARP auf Kreisen

Aus dem im vorherigen Abschnitt 2.7 vorgestellten Algorithmus COVERPATH läßt sich ein 6-Approximationsalgorithmus für das CDARP, dessen unterliegender Graph ein Kreis ist, ableiten. Dazu verwenden wir eine Idee von Karp und Bartal zur Abschätzung einer komplizierten Metrik durch eine einfachere Metrik [7], [8].

Das DARP auf Kreisen ist zuerst von Atallah und Kosaraju [6] betrachtet worden. Hintergrund für dieses Problem ist ein Roboter, der seinen Arm um eine feststehende Achse kreisförmig bewegen kann und mit diesem Arm Objekte von einem Ort in Reichweite des Arms zu einem anderen Ort befördern soll. Könnte dieser Roboter mehrere Objekte zur selben Zeit transportieren, würde dieses Problem dem CDARP auf Kreisen entsprechen.

Sei eine Instanz eines CDARP durch das Tupel $P = (V, E, A, c, o, k)$ gegeben. Dabei sei $G = (V, E)$ ein Kreis.

Wir definieren $C := \sum_{e \in E} c(e)$. Wir wählen nun mit einer Wahrscheinlichkeit von $p_e = \frac{c(e)}{C}$ die Kante $e \in E$ und konstruieren eine Menge $E_e = E \setminus \{e\}$. Die Menge E_e induziert einen Graphen $G_e := (V, E_e)$. Wir definieren die Kosten $c_e(v, w), v, w \in V$ als die Kosten des kürzesten Weges vom Knoten v zum Knoten w im Graphen G_e. Dann ist also $P_e = (V, E_e, A, c_e, o, k)$ eine Instanz des CDARP auf einem Pfad und auf Pfaden existiert ein 3-Approximationsalgorithmus für das CDARP (siehe voriger Abschnitt).

Jetzt gilt folgendes Lemma:

Lemma 2.8.1. *Sei das* CDARP P, *die Familie von* CDARP $P_e, e \in E$ *und* p_e *wie oben definiert. Dann gilt:*

$$\sum_{e \in E} p_e c_e(v, w) \leqslant 2c(v, w).$$

Dabei kann man die linke Seite dieser Gleichung als Erwartungswert der Zufallsvariablen $c_e(v, w)$ interpretieren.

Beweis. Sei $v_1 := v, v_k := w$ und $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ der (eindeutige) Pfad von v nach w in G_e. Dann gilt:

$$\begin{aligned}\sum_{e \in E} p_e c_e(v, w) &\leqslant \sum_{e \in E} p_e \left(\sum_{j=1}^{k-1} c_e(v_j, v_{j+1}) \right) \\ &= \sum_{j=1}^{k-1} \sum_{e \in E} p_e c_e(v_j, v_{j+1})\end{aligned}$$

Also genügt es zu zeigen, daß für benachbarte Knoten v_j, v_{j+1} gilt:

$$\sum_{e \in E} p_e c_e(v_j, v_{j+1}) \leqslant 2c(v_j, v_{j+1}).$$

Wir wissen, daß für $e = (v_j, v_{j+1}), j = 1, \ldots, n-1$ gilt: $c_e(v, v+1) = C - c(v, v+1)$ und $p_e = \frac{c(e)}{C}$. Und für $e \neq (v_j, v_{j+1}), j = 1, \ldots, n-1$ gilt: $c_e(v_j, v_{j+1}) = c(v_j, v_{j+1})$.

Also gilt für alle $j = 1, \ldots, n-1$

$$\begin{aligned}
\sum_{e \in E} p_e c_e(v_j, v_{j+1}) &= p_{(v_j, v_{j+1})}(C - c(v_j, v_{j+1})) + \sum_{e \in E \setminus \{(v_j, v_{j+1})\}} p_e c(v_j, v_{j+1}) \\
&= \frac{c(v_j, v_{j+1})(C - c(v_j, v_{j+1}))}{C} + \\
&\quad c(v_j, v_{j+1}) \left(1 - \frac{c(v_j, v_{j+1})}{C}\right) \\
&= c(v_j, v_{j+1}) - \frac{c^2(v_j, v_{j+1})}{C} + c(v_j, v_{j+1}) - \frac{c^2(v_j, v_{j+1})}{C} \\
&= 2c(v_j, v_{j+1}) - \frac{2c^2(v_j, v_{j+1})}{C} \\
&\leqslant 2c(v_j, v_{j+1})
\end{aligned}$$

□

Sei nun $T = ((v_0, v_1, M_1), (v_1, v_2, M_2), \ldots, (v_{r-1}, w, M_r))$ der gültige Transport mit minimalen Kosten der Probleminstanz P und $c(T) = \sum_{i=0}^{r-1} c(v_i, v_{i+1})$ die Kosten dieses optimalen Transports. Seien $c(T_e)$ die Kosten des gültigen Transports mit minimalen Kosten der Probleminstanz P_e für alle $e \in E$. Dann gilt:

$$\begin{aligned}
c(T) &= \sum_{i=0}^{r-1} c(v_i, v_{i+1}) \\
&\geqslant \sum_{i=0}^{r-1} \frac{1}{2} \sum_{e \in E} p_e c_e(v_i, v_{i+1}) \qquad \text{wegen Lemma 2.8.1} \\
&= \frac{1}{2} \sum_{e \in E} p_e \sum_{i=0}^{r-1} c_e(v_i, v_{i+1}) \\
&\geqslant \frac{1}{2} \sum_{e \in E} p_e c(T_e) \qquad (*) \\
&\geqslant \frac{1}{2} \left(\min_{e \in E} c(T_e)\right) \left(\sum_{e \in E} p_e\right) \\
&= \frac{1}{2} \min_{e \in E} c(T_e)
\end{aligned}$$

Die Ungleichung (*) gilt, da jeder gültige Transport für ein Problem P auch ein gültiger Transport für $P_e, e \in E$ ist und da $c_{e_1}(e_2) \geqslant c(e_2)$ für alle $e_1, e_2 \in E$ gilt.

Lösen wir also alle Probleminstanzen P_e mit dem Algorithmus COVERPATH und wählen dann die Lösung mit minimalen Kosten $c_{\text{COVERPATH}}$, dann gilt:

$$c_{\text{COVERPATH}} \leqslant \min_{e \in E} 3c(T_e) \leqslant 6c(T).$$

Dies ist in polynomialer Zeit durchführbar, da der Algorithmus COVERPATH n-mal durchgeführt wird und COVERPATH polynomial in der Ausführungszeit ist.

Es gilt also:

Satz 2.8.2. *Es existiert ein 6-Approximationsalgorithmus für das* CDARP *auf Kreisen.*

2.9 Der Algorithmus INTERVALLGRAPH

Wir stellen in diesem Abschnitt einen Algorithmus vor, der eine mindestens so gute Gütegarantie wie der Algorithmus COVERPATH besitzt, der aber leichter erweiterbar ist als der Algorithmus COVERPATH.

Auch dieser Algorithmus, wir nennen ihn INTERVALLGRAPH, arbeitet in zwei Phasen. In der ersten werden wie beim Algorithmus COVERPATH Teiltransporte gefunden, die in der zweiten Phase zu einem Transport zusammengefügt werden. Die Art und Weise, wie diese Teiltransporte gefunden werden unterscheidet sich aber von der beim Algorithmus COVERPATH.

Diesmal suchen wir nämlich in jeder Iteration eine Menge von Bögen mit maximalen Kosten aus D^+ bzw. D^-, so daß jede Kante des darunterliegenden Pfades maximal k-fach überdeckt wird. Diese Mengen kann man, wie wir zeigen werden, in polynomialer Zeit finden.

2.9.1 Beschreibung des Algorithmus INTERVALLGRAPH

Zunächst betrachten wir nur die Aufträge, die von unten nach oben transportiert werden müssen. Das heißt also, wir betrachten nur den Digraphen D^+. Der Algorithmus für den Digraphen D^- funktioniert analog.

Wir wollen das Problem MAXTEILTRANSPORT (A, c) lösen:

Definition 2.9.1 (MAXTEILTRANSPORT). Folgendes Problem bezeichnen wir mit MAXTEILTRANSPORT (A, c): Gegeben seien

- ein Digraph $D^+ = (V, A)$, wobei für A gilt: $i \in A \Rightarrow i^+ < i^-$ (D^+ ist also azyklisch) und
- eine Kostenfunktion $c : A \to \mathbb{R}$.

Nun suchen wir eine Menge A' von Bögen aus A, so daß die Kosten $c(A')$ maximal sind, und die Menge der Bögen in A', die eine Kante $e \in E$ überdecken, eine Kardinalität nicht größer als k haben. Es soll also

$$\left|\{i \in A' : i^+ \leqslant v < i^-\}\right| \leqslant k \qquad \text{für alle } v = 1, \ldots, n-1 \tag{2.31}$$

gelten.

Dieses Problem entspricht dem Problem, eine Menge A' von Kanten mit maximalen Kosten in einem Intervallgraphen $D = (V, A)$ zu finden, so daß die größte Clique im Graphen $D' = (V, A')$ nicht größer als k ist.

Wir können dieses Problem als ein ganzzahliges lineares Programm formulieren. Die Menge A enthalte m Bögen, also sei $m := |A|$. Diese Bögen numerieren wir von 1 bis m.

Seien $x_i \in \{0, 1\}, i = 1, \ldots, m$ die Entscheidungsvariablen, ob der Bogen i in A' ist oder nicht. Also x_i nimmt den Wert Eins an, wenn der Auftrag i in der Menge A' enthalten ist, ansonsten sei $x_i = 0$. Dann ergibt sich folgendes ganzzahlige lineare Programm (ILP):

$$\max \sum_{i=1}^{m} c(i) x_i, \tag{2.32}$$

so daß

$$\sum_{i \in A : i^+ \leqslant v < i^-} x_i \leqslant k \qquad \text{für alle } v \in V \setminus \{n\} \tag{2.33}$$

$$0 \leqslant x_i \leqslant 1 \qquad \text{für alle } i = 1, \ldots, m \tag{2.34}$$

$$x_i \in \mathbb{N} \qquad \text{für alle } i = 1, \ldots, m \tag{2.35}$$

Dieses ganzzahlige lineare Programm läßt sich, wie wir später zeigen werden, in polynomialer Zeit lösen, da die Matrix der Koeffizienten der Nebenbedingungen total unimodular ist.

Bemerkung. Man kann dieses Problem auch als ein Min-Cost-Flow-Problem modellieren. Dazu konstruieren wir ein Netzwerk, das für jeden Auftrag in A einen Bogen mit Kapazität Eins und Kosten Null enthält. Zusätzlich fügen wir für jede Kante $(v, v+1) \in V$ einen Bogen $\langle v, v+1 \rangle$ mit Kapazität k und Kosten von Eins hinzu. Die Quelle s dieses Netzwerks ist adjazent zum Knoten 1 und der Bogen $\langle s, 1 \rangle$ hat eine Kapazität von k. Der Knoten n ist die Senke. Ein maximaler Fluß mit minimalen Kosten entspricht dann einem maximalen Teiltransport. Jeder Bogen, der einem Auftrag aus A entspricht und der im Fluß enthalten ist entspricht einem Auftrag in A', der Lösung des Problems MAXTEILTRANSPORT.

Unser Algorithmus INTERVALLGRAPH löst jetzt das Problem MAXTEILTRANSPORT, entfernt alle Bögen aus A, die in A' enthalten sind und konstruiert aus den Bögen aus A' einen Teiltransport, wie im Abschnitt 2.7.1 beschrieben. Anschließend beginnt er wieder von vorne, bis die Menge A keine Bögen mehr enthält.

Wir haben zwar das Problem MAXTEILTRANSPORT (A, c) bisher nur für eine Menge A definiert, welche nur Aufwärtsbögen enthält. Wir können aber analog das Problem MAXTEILTRANSPORT (A, c), A enthält nur Abwärtsbögen definieren.

Vorbedingung: A ist eine Menge von Aufträgen mit Kardinalität m. Die Kapazität des Aufzugs betrage k

1: **Solange** A nicht leer ist **:**
2: **Wenn** A einen Auftrag i mit $i^+ < i^-$ enthält **Dann**
3: sei $B = A(D^+)$
4: **Sonst**
5: sei $B = A(D^-)$
6: **Ende Wenn**
7: Sei A' die Lösung des Problems MAXTEILTRANSPORT (B, c);
8: entferne alle Bögen aus A, die in A' enthalten sind;
9: konstruiere aus A' Teiltransporte und füge sie zu der Menge der Teiltransporte $\mathcal{T}$ hinzu;
10: **Ende Solange**
11: konstruiere aus der Menge der Teiltransporte $\mathcal{T}$ einen Transport.

Algorithmus 8: INTERVALLGRAPH

2.9.2 Gütegarantie

Wir zeigen in diesem Abschnitt, daß die Gütegarantie des Algorithmus INTERVALLGRAPH mindestens so gut ist, wie die Gütegarantie des Algorithmus COVERPATH. Das heißt, wir zeigen, daß der Algorithmus INTERVALLGRAPH ein 3-Approximationsalgorithmus ist. Dazu werden wir die Kosten der Lösungsmenge A' des Problems MAXTEILTRANSPORT aus Schritt 7 des Algorithmus INTERVALLGRAPH mit den Kosten der Mengen, die der Algorithmus COVERPATH aus Abschnitt 2.7 findet, vergleichen.

Wir könnten im Schritt 7 des Algorithmus INTERVALLGRAPH, anstatt das Problem MAXTEILTRANSPORT zu lösen, durch mehrfache Anwendung des Algorithmus COVERPATH $2k$ Mengen von Bögen finden und daraus k Mengen auswählen. Die Vereinigung dieser k Mengen erfüllt dann die Gleichung (2.31).

Wir betrachten jetzt, wie wir den Algorithmus COVERPATH anwenden müssen, damit wir ihn als Ersatz für das Lösen des Problems MAXTEILTRANSPORT verwenden können. Der Algorithmus COVERPATH operiert nämlich nur auf Teilgraphen $L(w_0, w_1) \subset D^+ = (V, A)$, wobei $w_0 \in V$ der unterste Knoten ist, an dem Bögen aus D^+ starten, also $\lambda_{w_0} > 0$ ist und $w_1 > w_0$ der unterste Knoten in V ist, für den $\lambda_{w_1} = 0$ gilt. Aber wir können D^+ in Teilgraphen $D_0^+, D_1^+, \ldots, D_s^+$ zerlegen, so daß für jede Bogenmenge eines dieser Teilgraphen der unterliegende Graph zusammenhängend ist (siehe Abbildung 2.9.2).

Dazu betrachten wir die Menge der Kanten $(v, v+1)$ für die $\mu_v^+ = 0$ gilt. Sei

$$M = \{(v, v+1) \in E : \mu_v^+ = 0\} = \{(v_1, v_1+1), (v_2, v_2+1), \ldots, (v_s, v_s+1)\}$$

diese Menge. O.B.d.A. gelte $v_1 < v_2 < \cdots < v_s$. Außerdem sei $v_0 = 1$ und $v_{s+1} = n$. Dann betrachten wir die Teilgraphen $D_i^+, i = 0, \ldots, s$ von D. Ein Digraph D_i^+ enthält die Aufwärtsbögen, die einen Startknoten u mit $v_i \leqslant u < v_{i+1}$ haben.

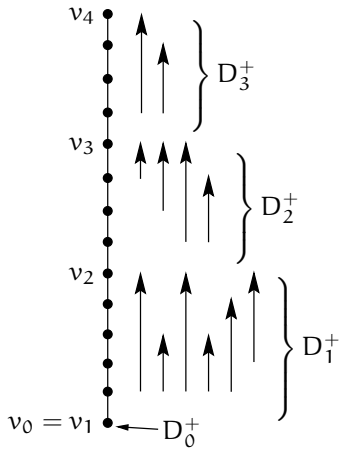

Abbildung 2.27: Aufteilung der Menge A in Mengen $D_0^+, D_1^+, \ldots, D_s^+$.

Es kann keine Bögen in D^+ geben, die ihren Startknoten in einem Digraphen D_i^+ und ihren Zielknoten in einem Digraphen $D_j^+, i \neq j$ haben. Dies liegt daran, daß zwischen je zwei Digraphen $D_i^+, D_j^+, i \neq j$ mindestens eine Kante aus M liegt, über die keine Bögen aus D^+ laufen. Es gilt weiterhin $\bigcup_{j=0}^{s+1} D_j^+ = D^+$.

Nun können wir den Algorithmus COVERPATH auf jeden dieser Teilgraphen $D_0^+, D_1^+, \ldots D_s^+, D_{s+1}^+$ anwenden.

Wenn wir aus den Mengen $P_i^1, P_i^2, i = 1, \ldots, k$, die der Algorithmus COVERPATH aus dem Digraphen $D_j^+, j \in \{0, \ldots, s+1\}$ konstruiert hat, k Mengen auswählen und diese vereinigen, so ist die Gleichung (2.31) erfüllt. Dies folgt direkt aus dem Satz 2.7.10.

Sei E_j die Menge der Kanten, die von Bögen in D_j^+ überdeckt wird, also gilt

$$E_j := \{(v, v+1) \in E : i \in A(D_j^+) \text{ mit } i^+ \leqslant v < i^-\}.$$

Lemma 2.9.2. *Für die Mengen A', die Lösungen des Problems* MAXTEILTRANSPORT *(A, c) im Schritt 7 des Algorithmus* INTERVALLGRAPH *sind, gilt:*

$$c(A') > \sum_{j=0}^{s} \frac{k}{2} c(E_j).$$

Beweis. Wir unterteilen den Digraphen $D^+ = (V, A)$ wie oben beschrieben in Teilgraphen D_i^+, $i = 0, \ldots, s$. Auf jeden dieser Teilgraphen wenden wir den Algorithmus COVERPATH an. Wir wissen aus Lemma 2.7.10, daß die $2k$ Mengen von Bögen, die der Algorithmus COVERPATH findet, jede Kante $(v, v+1)$ des D_i^+ unterliegenden Graphen $L(v_i, v_{i+1})$ für die $\mu_v > 0$ gilt, mindestens k-fach überdeckt. Daraus folgt, daß die Bögen in diesen $2k$ Mengen zusammen mindestens Kosten von $kc(E_i)$ verursachen. Wählen wir nun die Hälfte der Mengen mit größeren Kosten, so sind die Kosten der Bögen in dieser Hälfte der Mengen mindestens halb so groß, wie die

Kosten aller Mengen. Da nun $A(D^+)$ disjunkte Vereinigung der $A(D_i^+), i = 0, \ldots, s$ ist, folgt das Lemma. □

Das Lemma 2.9.2 bedeutet also, daß in jeder Iteration Teiltransporte erzeugt werden, deren Kosten $c(E_j)$ betragen und in denen das Fahrzeug mindestens zur Hälfte ausgelastet ist. Daraus folgt aber sofort, daß die Kosten der Teiltransporte $c(\mathcal{T}) < \frac{c_{flow}}{2}$ sind. Dann folgt mit der gleichen Argumentation, wie für den Satz 2.7.12, der folgende Satz:

Satz 2.9.3. *Der Algorithmus* COVERPATH *hat eine Gütegarantie von 3.*

2.9.3 Die Koeffizientenmatrix ist unimodular

Wir zeigen jetzt, daß der Algorithmus COVERPATH polynomial in seiner Zeitkomplexität ist. Dazu müssen wir zeigen, daß das Problem MAXTEILTRANSPORT polynomial lösbar ist.

Betrachten wir die Matrix der Koeffizienten, so entspricht jede Spalte einem Auftrag und jede Zeile einer gerichteten Kante $\langle v, v+1\rangle$ für $v = 1, \ldots, n-1$. Sortieren wir die Zeilen der Matrix nach den Startknoten der gerichteten Kanten, so hat die sich ergebende Koeffizientenmatrix die sogenannte „consecutive ones"-Eigenschaft. Das heißt, daß in jeder Spalte genau ein Block mit „1"-Einträgen enthalten ist und ansonsten nur Nullen.

Lemma 2.9.4. *Die Matrix der Koeffizienten A im ganzzahligen linearen Programm (ILP) hat mit einer geeigneten Permutation der Zeilen die Eigenschaft, daß $A \in \{0,1\}^{m\times 2(n-1)}$ in jeder Spalte genau einen Block mit Einsen enthält. Das heißt, gilt $A_{i,j} = 1$ und $A_{i',j} = 1, i' > i$, dann müssen auch alle Elemente zwischen diesen beiden Eins sein, also gilt dann: $A_{l,j} = 1$ für alle $i < l < i'$. Diese Eigenschaft nennen wir* „consecutive ones"-*Eigenschaft.*

Beweis. Ordnen wir die Zeilen wie oben beschrieben, so befinden sich die Zeilen für übereinanderliegende Stockwerke auch in der Koeffizientenmatrix A übereinander. jede Spalte entspricht genau einem Auftrag. Ein Auftrag i muß aber aller Kanten $(j, j+1)$ mit $i^+ \leqslant j < i^-$ überqueren. Also hat er genau in den Zeilen für diese Knoten j einen Eintrag 1 und in den Anderen den Eintrag 0. □

Lemma 2.9.5. *Matrizen mit der „consecutive ones"-Eigenschaft sind total unimodular.*

Beweis. Wir fügen eine Zeile, die nur Nullen enthält oben an die Matrix an. Anschließend ziehen wir von jeder Zeile die darüberliegende Zeile ab. Nun sind in jeder Spalte genau ein Eintrag 1 und ein Eintrag -1. Daraus folgt mit [31], Example 2, S. 274, daß die so erzeugte Matrix total unimodular ist. Da aber unsere Transformation die Eigenschaft der totalen Unimodularität nicht stört, folgt die Behauptung. □

Lemma 2.9.6 (Lemma aus [27], [31]). *Eine $\{-1, 0, +1\}$-Matrix M, die in jeder Spalte genau eine $+1$ und eine -1 enthält, ist total unimodular.*

Beweis aus [33], [31] (vollständige Induktion). Für 1×1-Matrizen gilt das Lemma trivialerweise. Nun schließen wir von einer $n \times n$-Matrix auf eine $(n+1) \times (n+1)$-Matrix. Jede quadratische Teilmatrix N von M enthält entweder eine Spalte mit höchstens einem Eintrag, der nicht Null ist, oder jede Spalte von N enthält sowohl eine $+1$ als auch eine -1. Im ersten Fall entwickelt man die Determinante nach der Spalte, die nur einen Eintrag ungleich Null enthält. Induktiv folgt dann, daß die Determinante von N $-1, 0$ oder $+1$ ist. Im zweiten Fall ist die Determinante von N Null. □

Daraus folgt, daß das Polyeder, welches durch $\text{conv}(x : Ax \leqslant k)$ definiert ist, nur ganzzahlige Eckpunkte hat. Dieses Polyeder ist wegen der Ungleichungen (2.34) beschränkt. Man kann also mit der Ellipsoid-Methode oder Innere-Punkte-Methoden in polynomialer Zeit eine optimale Lösung des MAXTEILTRANSPORT-Problems finden, indem man eine optimale Lösung des IP's findet.

Der Vorteil dieses Algorithmus ist es, daß man relativ leicht Erweiterungen, wie z.B. Start-/Stopzeiten oder Ordnungen auf den Aufträgen, des Algorithmus implementieren kann, indem man zusätzliche Nebenbedingungen zu dem ILP hinzufügt. Dabei geht aber in die Regel die polynomiale Lösbarkeit verloren.

2.9.4 FIFO

Man kann auf der Basis des Algorithmus INTERVALLGRAPH nun einen erweiterten Algorithmus angeben, der FIFO-Präzedenzen für Aufträge auf den gleichen Stockwerken berücksichtigt.

Dazu schränkt man die Menge der Bögen A, auf der man das Problem MAXTEILTRANSPORT (A, c) löst, auf Bögen ein, die minimal bezüglich der FIFO-Ordnung sind. Außerdem setzt man die Kapazität auf 1 und löst das Problem k-mal hintereinander.

Man kann dann aber im allgemeinen anschließend nicht mehr einen optimalen Transport aus den Teiltransporten bestimmen, da Ändern der Reihenfolge der Teiltransporte die FIFO-Präzedenzbedingung verletzen könnte.

Welche Gütegarantie dieser Algorithmus hat ist nicht klar, er liefert aber in praktischen Versuchen Transporte deren Kosten nicht weit von der Flußgrenze entfernt sind.

2.9.5 Start-/ Stopzeiten

Wir leiten jetzt aus dem Algorithmus INTERVALLGRAPH einen Algorithmus ab, der auch Start-/Stopzeiten berücksichtigt.

Äquivalent zu obigem (ILP) können wir auch alle Teiltransporte auf einmal berechnen. Dazu führen wir einen Zeitindex für die Variablen x_i ein. Wir benutzen also jetzt Variablen x_{ti}. Diese werden 1, wenn der Auftrag i im t. Teiltransport transportiert wird, ansonsten 0.

Damit ergibt sich folgendes neues IP:

$$\min \sum_{t=1}^{T} \sum_{i=1}^{m} tc(i) x_{ti}, \tag{2.36}$$

so daß

$$\sum_{i \in A : i^+ \leqslant v < i^-} x_{ti} \leqslant k \qquad \text{für alle } v \in V \setminus \{n\}, \text{für alle } t = 1, \ldots, T \tag{2.37}$$

$$\sum_{i \in A : i^+ \geqslant v > i^-} x_{ti} \leqslant k \qquad \text{für alle } v \in V \setminus \{n\}, \text{für alle } t = 1, \ldots, T \tag{2.38}$$

$$\sum_{t=1}^{T} x_{ti} = 1 \qquad \text{für alle } i = 1, \ldots, m \tag{2.39}$$

$$0 \leqslant x_i \leqslant 1 \qquad \text{für alle } i = 1, \ldots, m \tag{2.40}$$

$$x_i \in \mathbb{N} \qquad \text{für alle } i = 1, \ldots, m \tag{2.41}$$

Die neue Nebenbedingung 2.39 zerstört die totale Unimodularität nicht, da sie nur über jede Intervallgraphenmatrix eine Einheitsmatrix setzt. Die Koeffizienten Matrix A sieht also folgendermaßen aus:

$$A := \begin{pmatrix} E_n & E_n & \ldots & E_n \\ -E_n & -E_n & \ldots & -E_n \\ S & & & 0 \\ & S & & \\ & & \ddots & \\ 0 & & & S \end{pmatrix}$$

Dabei ist S die Koeffizienten-Matrix von (ILP) aus Abschnitt 2.9.1.

In dieses ganzzahlige lineare Programm können wir nun Nebenbedingungen für Start-/Stop-Zeiten einfügen.

Dazu müssen wir die Anzahl der Variablen erhöhen. Dabei geht aber die totale Unimodularität verloren. Das entstehende lineare ganzzahlige Programm läßt sich aber dennoch in relativ geringer Zeit lösen.

Dazu führen wir folgende Variablen ein:

$z^+_{start,t,j}$	$1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n$	Der Vertikalförderer startet im t. Durchlauf im Stockwerk j mit Aufträgen nach oben.
$z^+_{stop,t,j}$	$1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n$	Der Vertikalförderer stoppt im t. Durchlauf im Stockwerk j mit Aufträgen nach oben.
$z^-_{start,t,j}$	$1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n$	Der Vertikalförderer startet im t. Durchlauf im Stockwerk j mit Aufträgen nach unten.
$z^-_{stop,t,j}$	$1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n$	Der Vertikalförderer stoppt im t. Durchlauf im Stockwerk j mit Aufträgen nach unten.

Nun können wir folgende Ungleichungen zu (ILP) hinzufügen:

$$z^{+}_{start,t,i^{+}} - x_{ti} \geqslant 0 \quad \text{für alle } t = 1,\dots,T, \text{ für alle } i = 1,\dots,m \text{ mit } i^{+} < i^{-} \tag{2.42}$$

$$z^{+}_{stop,t,i^{-}} - x_{ti} \geqslant 0 \quad \text{für alle } t = 1,\dots,T, \text{ für alle } i = 1,\dots,m \text{ mit } i^{+} < i^{-} \tag{2.43}$$

$$z^{-}_{start,t,i^{+}} - x_{ti} \geqslant 0 \quad \text{für alle } t = 1,\dots,T, \text{ für alle } i = 1,\dots,m \text{ mit } i^{+} > i^{-} \tag{2.44}$$

$$z^{-}_{stop,t,i^{-}} - x_{ti} \geqslant 0 \quad \text{für alle } t = 1,\dots,T, \text{ für alle } i = 1,\dots,m \text{ mit } i^{+} > i^{-} \tag{2.45}$$

Setzen wir in der Zielfunktion die Koeffizienten für die x_{it} auf 0 und für die $z^{+/-}_{start/stop}$ auf 1, so minimieren wir die Anzahl der Starts und der Stops. Das IP liefert aber auch nur eine Näherungslösung, da die Starts und Stops für die Verbindungsfahrten nicht berücksichtigt werden. Da sich aber durch Verbindungfahrten die Anzahl der Starts- und Stops höchstens verdoppelt haben wir hier eine Gütegarantie von 2. Da wir aber auch das Auf und Abwärtsfahren trennen, ergibt sich nur noch eine Gütegarantie von 4. Eine konstante Gütegarantie für die Fahrzeiten ergeben sich aber nicht mehr.

In dieses (ILP) kann man auch die FIFO-Präzedenzbedingungen integrieren:

$$x_{t,i} + x_{t',j} \leqslant 1 \quad \text{für alle } t < t', j \prec i \tag{2.46}$$

Wir haben also einen Algorithmus gefunden, der nicht mehr polynomial beschränkte Rechenzeit hat, aber dessen Lösung höchstens 4-mal so häufig an Stockwerken hält, wie die Lösung mit minimaler Anzahl von Starts und Stops. Für die gesamte Fahrtzeit konnten wir keine Gütegarantie angeben. Dieser Algorithmus ist also nur für Probleme geeignet, in der der Anteil der Zeiten für Anfahren und Anhalten im Vergleich zu den Fahrzeiten relativ groß sind.

Kapitel 3

Polyedrische Ansätze

Da wir gesehen haben, daß, wenn $\mathcal{NP} \neq \mathcal{P}$ gilt, keine polynomialen Algorithmen existieren, die das CDARP selbst auf Pfaden optimal lösen, untersuchen wir nun ganzzahlige lineare Programme, die das CDARP beschreiben. Wir stellen zwei verschiedene Ansätze vor. Der erste Ansatz ist nur für CDARP auf Pfaden oder einfachen Bäumen zu gebrauchen. Dafür enthält er verhältnismäßig wenig Nebenbedingungen und ist daher, wenn man ein Programm zum Lösen von ganzahligen linearen Programmen, wie z.B. CPLEX hat, einfach zu implementieren.

Wir stellen noch ein zweites ganzzahligeslineares Programm (ILP) vor, welches auf der polyedrischen Formulierung des „Travelling Salesman Problem“ von Dantzig, Fulkerson und Johnson [14] basiert, und demzufolge exponentiell viele Nebenbedingungen in der Anzahl der Aufträge benötigt. Um solch ein ILP zu lösen muß man Techniken wie Schnittebenenverfahren (siehe z.B. [31]) anwenden. Das macht die Implementation komplizierter. Dieses ILP beschreibt dafür das CDARP bzw. sogar das FIFO-CDARP auf allgemeinen Graphen.

3.1 ILP für das CDARP auf Pfaden

In diesem Abschnitt stellen wir ein ganzzahliges lineares Programm (ILP) vor, welches das CDARP auf Pfaden vollständig beschreibt. Eine optimale Lösung dieses (ILP) entspricht also einer optimalen Lösung eines CDARP.

Die Idee dieses ganzzahligen linearen Programms ist es, mehrere Netzwerke zu koppeln. Wir betrachten die Bewegung des Aufzugs als einen Fluß. Jede mögliche Folge von Bewegungen entspricht dann einem gültigen Fluß für den Aufzug. Zusätzlich haben wir für jeden Auftrag ein Netzwerk.

Probleme mit ca. 20 Aufträgen und 9 Stockwerken werden in ungefähr einer Minute mit CPLEX gelöst. Probleme mit 36 Aufträgen und 5 Stockwerken werden in ein bis zwei Stunden gerechnet.

Wir wollen nun das ILP für eine Instanz des CDARP $P = (V, E, A, c, 1, k)$ auf einem Pfad modellieren. Dabei ist die Menge der Knoten V wie im Abschnitt 2.7 so

von 1 bis n numeriert, daß für je zwei Knoten $u, v \in V$ mit $|u - v| = 1$ eine Kante $(u, v) \in E$ existiert.

3.1.1 Die Konstanten

Folgende Konstanten verwenden wir in diesem ganzzahligen linearen Programm:

n	Anzahl der Stockwerke,
m	Anzahl der Objekte,
k	die Kapazität des Vertikalförderers in Objekten,
T	die maximale Anzahl von Durchläufen, wobei gilt: $T \leqslant 3 \lceil m/k \rceil$ (s.u.).

3.1.2 Die Variablen

Wir haben eine Menge von Variablen für den Aufzug. Jede Variable entspricht einer Überquerung einer Kante im Graphen G durch den Aufzug zum t-ten Mal, oder einem t-ten Richtungswechsel. Außerdem haben wir eine Menge von Variablen für die Aufträge analog zu den Aufzugsvariablen. Im einzelnen haben wir folgende Variablen:

$x^{+}_{t,i,j}$	$1 \leqslant t \leqslant T, 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n-1$	Das Objekt i wird im t-ten Durchlauf vom j-ten in das $(j+1)$-te Stockwerk transportiert.
$x^{-}_{t,i,j}$	$1 \leqslant t \leqslant T, 1 \leqslant i \leqslant m, 2 \leqslant j \leqslant n$	Das Objekt i wird im t-ten Durchlauf vom j-ten in das $(j-1)$-te Stockwerk transportiert.
$x^{+-}_{t,i,j}$	$1 \leqslant t \leqslant T, 1 \leqslant i \leqslant m, 2 \leqslant j \leqslant n$	Das Objekt i ändert im t-ten Durchlauf im j-ten Stockwerk seine Richtung von aufwärts in abwärts.
$x^{-+}_{t,i,j}$	$1 \leqslant t \leqslant T-1, 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n-1$	Das Objekt i ändert im t-ten Durchlauf im j-ten Stockwerk seine Richtung von abwärts in aufwärts.
$y^{+}_{t,j}$	$1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n-1$	Der Aufzug überquert im t-ten Durchlauf die Kante vom j-ten zum $(j-1)$-ten Stockwerk aufwärts.
$y^{-}_{t,j}$	$1 \leqslant t \leqslant T, 2 \leqslant j \leqslant n$	Der Aufzug überquert im t-ten Durchlauf die Kante vom j-ten zum $(j-1)$-ten Stockwerk abwärts.
$y^{+-}_{t,j}$	$1 \leqslant t \leqslant T, 2 \leqslant j \leqslant n$	Der Aufzug ändert im t-ten Durchlauf im j-ten Stockwerk seine Richtung von aufwärts in abwärts.
$y^{-+}_{t,j}$	$1 \leqslant t \leqslant T-1, 1 \leqslant j \leqslant n-1$	Der Aufzug ändert im t-ten Durchlauf im j-ten Stockwerk seine Richtung von abwärts in aufwärts.

Alle Variablen sind $(0,1)$-Variablen.

3.1.3 Zielfunktion

Wir wollen die Kosten der Bewegung des Aufzugs minimieren.

$$\textit{minimiere} \sum_{t=1}^{T} \sum_{i=1}^{n} (c(j, j+1) y^{+}_{t,j} + c(j+1, j) y^{-}_{t,j}) \tag{3.1}$$

3.1.4 Bedingungen

Netzwerkflußbedingungen für den Aufzug

Die folgenden Nebenbedingungen modellieren die Bewegungsmöglichkeiten des Aufzugs. Für jeden Knoten v_t, der einem Knoten im Graphen $G = (V, E)$ zum Zeitpunkt t entspricht benötigen wir eine Nebenbedingung, die angibt, zu welchem Knoten der Aufzug von v_t aus gelangen kann und von welchem Knoten er kommen kann. Dies ist in der Abbildung 3.1 dargestellt.

$$y^{+}_{1,1} = 1 \tag{3.2}$$

$$y^{+}_{t,j} + y^{-+}_{t-1,j} - y^{+}_{t,j+1} - y^{+-}_{t,j+1} = 0 \quad \text{für alle } 2 \leqslant t \leqslant T,\ 1 \leqslant j \leqslant n-1 \tag{3.3}$$

$$y^{+}_{1,j} - y^{+}_{1,j+1} - y^{+-}_{1,j+1} = 0 \quad \text{für alle } 1 \leqslant j \leqslant n-1 \tag{3.4}$$

$$y^{+}_{t,n-1} - y^{+-}_{t,n} = 0 \quad \text{für alle } 1 \leqslant t \leqslant T \tag{3.5}$$

$$y^{-}_{t,j} + y^{+-}_{t,j} - y^{-}_{t,j-1} - y^{-+}_{t+1,j} = 0 \quad \text{für alle } 1 \leqslant t \leqslant T-1,\ 2 \leqslant j \leqslant n \tag{3.6}$$

$$y^{-}_{T,j} + y^{+-}_{T,j} - y^{-}_{T,j-1} = 0 \quad \text{für alle } 2 \leqslant j \leqslant n \tag{3.7}$$

$$y^{-}_{t,1} - y^{-+}_{t+1,1} \leqslant 1 \quad \text{für alle } 1 \leqslant t \leqslant T-1 \tag{3.8}$$

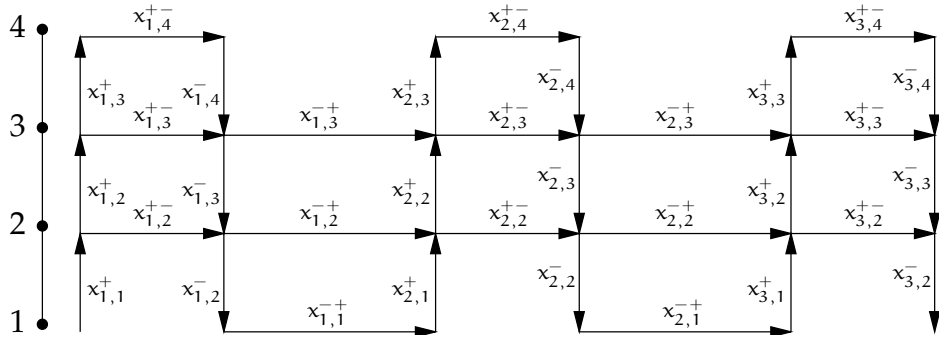

Abbildung 3.1: Netzwerk für den Aufzug mit Kanten- und Knotenkapazität von 1

Netzwerkflußbedingungen für die Aufträge

Es fehlen noch die Nebenbedingungen für die Aufträge. Ein Auftrag kann sich genau so bewegen, wie der Aufzug. Es ist aber nicht sinnvoll, einen Auftrag $i \in A$, mit $i^+ < i^-$ unter seinen Startknoten i^+ oder über seinen Endknoten i^- zu transportieren, wenn wir keine Kosten für das Auf- und Abladen berücksichtigen. Diese Nebenbedingungen sind analog wie die Netzwerkflußbedingungen für den Aufzug aufgebaut und in Abbildung 3.2 dargestellt.

Für alle $1 \leqslant i \leqslant m$ gilt

$$x^+_{t,i,j} + x^{-+}_{t-1,i,j} - x^+_{t,i,j+1} - x^{+-}_{t,i,j+1} = 0 \quad \text{für alle } 2 \leqslant t \leqslant T,\ i^+ + 1 \leqslant j \leqslant i^- \tag{3.9}$$

$$x^+_{1,i,j} - x^+_{1,i,j+1} - x^{+-}_{1,i,j} = 0 \quad \text{für alle } 2 \leqslant t \leqslant T,\ i^+ + 1 \leqslant j \leqslant i^- \tag{3.10}$$

$$x^+_{t,i,n-1} - x^{+-}_{t,i,n} = 0 \quad \text{für alle } 1 \leqslant t \leqslant T \tag{3.11}$$

$$x^-_{t,i,j} + x^{+-}_{t,i,j} - x^-_{t,i,j-1} - x^{-+}_{t+i,1,j} = 0 \quad \text{für alle } 1 \leqslant t \leqslant T-1,\ i^+ + 1 \leqslant j \leqslant i^- \tag{3.12}$$

$$x^-_{T,i,j} + x^{+-}_{T,i,j} - x^-_{T,i,j-1} = 0 \quad \text{für alle } 2 \leqslant j \leqslant n \tag{3.13}$$

$$x^-_{t,i,1} - x^{-+}_{t+1,i,1} \leqslant 1 \quad \text{für alle } 1 \leqslant t \leqslant T-1 \tag{3.14}$$

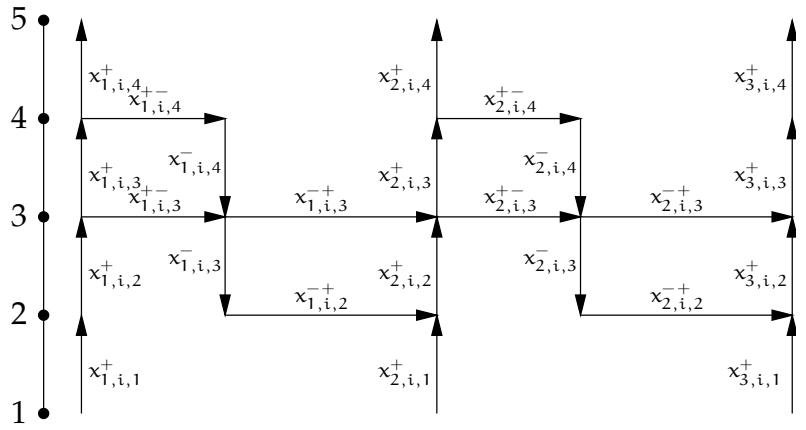

Abbildung 3.2: Netzwerk für den Auftrag i vom Knoten 1 zum Knoten 5

Kopplung von Aufträgen mit dem Aufzug

Nun müssen wir noch modellieren, daß der Aufzug nicht mehr als k Aufträge auf einmal abarbeiten kann, und daß ein Auftrag nur bewegt werden kann, wenn sich

gleichzeitig der Aufzug bewegt.

$$\sum_{i=1}^{m} x^{+}_{t,i,j} \leqslant k y^{+}_{t,j} \quad \text{für alle } 1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n \tag{3.15}$$

$$\sum_{i=1}^{m} x^{-}_{t,i,j} \leqslant k y^{-}_{t,j} \quad \text{für alle } 1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n \tag{3.16}$$

$$\sum_{i=1}^{m} x^{+-}_{t,i,j} \leqslant k y^{+-}_{t,j} \quad \text{für alle } 1 \leqslant t \leqslant T, 1 \leqslant j \leqslant n-1 \tag{3.17}$$

$$\sum_{i=1}^{m} x^{-+}_{t,i,j} \leqslant k y^{-+}_{t,j} \quad \text{für alle } 1 \leqslant t \leqslant T, 2 \leqslant j \leqslant n \tag{3.18}$$

Diese Ungleichungen sorgen dafür, daß die Kapazität des Aufzugs nicht überschritten wird, und daß ein Auftrag nur dann zu einem bestimmten Zeitpunkt über eine bestimmte Kante befördert wird, wenn auch der Fahrstuhl zu diesem Zeitpunkt diese Kante überquert.

FIFO-Präzedenzbedingungen

In dieses (ILP) kann man auch FIFO-Präzedenzbedingungen integrieren, wenn man das FIFO-CDARP lösen will:

$$x_{t,i,j} + x_{t',i',j} \leqslant 1 \quad \text{für alle } t < t', i' \prec i, j = i^{+} \tag{3.19}$$

3.1.5 Beobachtungen

Dieses Modell läßt dich durch leichte Modifikationen in ein Modell für präemptive CDARP ändern. Dazu entfernt man einfach die Ungleichungen (3.17) und (3.18). Außerdem kann man dann auch die Möglichkeit für den Abwärtstransport der Aufwärtsaufträge entfernen und vice versa.

3.1.6 Wie groß muß T sein?

Nun stellt sich noch die Frage wie viele Richtungsänderungen macht der Aufzug insgesamt. Eine einfache obere Grenze für die Anzahl der Richtungsänderungen ist die Anzahl der Aufträge, da eine Richtungsänderung immer nur dann sinnvoll ist, wenn gerade ein Auftrag beendet worden ist.

Eine andere Abschätzung der Richtungsänderungen ergibt sich aus den Werten $\lambda_v, v \in V \setminus \{n\}$. Überquert der Aufzug die Kante $(v, v+1)$ nämlich λ_v-mal, so muß er mindestens so oft die Richtung wechseln. Also gilt:

$$T \leqslant \max_{v \in V \setminus \{n\}} \lambda_v.$$

Eine andere Abschätzung für T, die in der Praxis am besten funktioniert ergibt sich aus folgendem Lemma.

Lemma 3.1.1. *Die Anzahl der Richtungsänderungen des Aufzugs in einer optimalen Lösung einer Instanz des präemptiven* CDARP *ist nicht größer als* $3\left\lceil \frac{m}{k} \right\rceil$.

Beweis durch Widerspruch. Angenommen wir haben eine Lösung mit mehr als $3\left\lceil \frac{m}{k} \right\rceil$ Richtungsänderungen des Aufzugs. Daraus folgt, daß die Bewegungen zwischen je zwei Richtungsänderungen im Durchschnitt weniger als $\frac{k}{3}$ Aufträge enthalten. Damit verursacht dieser Transport aber größere Kosten als $3c_{flow}$ und kann also nicht optimal sein, da der Algorithmus PATHFINDER immer eine Lösung mit Kosten nicht größer als $3c_{flow}$ findet. □

Praktische Versuche haben gezeigt, daß die Rechenzeit für das hier vorgestellte ganzzahlige Programm wesentlich von T abhängt. Dies liegt daran, daß für große T die Lösung der LP-Relaxierung des ILPs nur noch sehr wenige ganzzahlige Einträge hat. Wählt man hingegen T recht klein, so müssen schon in der Lösung der LP-Relaxierung viele Variablen ganzzahlige Werte annehmen. Die Gefahr besteht aber darin, daß durch die Wahl eines zu kleinen Wertes für T die optimale Lösung des CDARP für das ILP nicht mehr gültig ist.

In praktischen Versuchen hat sich gezeigt, daß für $T = 3$ ungefähr 99% alle Instanzen optimal lösbar waren und die Rechenzeit noch verhältnismäßig gering war.

3.2 ILP für das CDARP auf allgemeinen Graphen

Hier stellen wir ein ganzzahliges lineares Programm (ILP) vor, welches das CDARP auf allgemeinen Graphen vollständig beschreibt. Dieses ILP ist eine Erweiterung des ILPs für das „Traveling Salesman Problems“ von Dantzig, Fulkerson und Johnson [14].

Die hier vorgestellte Formulierung stammt im wesentlichen von K. S. Ruland [28]. Dort ist das Problem mit unbeschränkter Kapazität untersucht worden. Wir haben noch eine Klasse von Ungleichungen hinzugefügt, die die Kapazität beschränken. Eine Untersuchung der Lösbarkeit des IPs wurde nicht durchgeführt. Sie wäre aber interessant, da in dieser Formulierung FIFO-Präzedenzen zwischen Aufträgen und Start-/Stopzeiten eines Aufzuges ohne Hinzufügen neuer Klassen von Ungleichungen modelliert werden könnte. Die Verwandtschaft mit gut untersuchten Problemen wie dem „Traveling Salesman Problem“ oder dem „Sequential Ordering Problem“ [3], einem asymmetrischen TSP mit Präzedenz-Bedingungen für die Knoten, von denen viele facettendefinierende Ungleichungen und Separationsalgorithmen bekannt sind, sollte es ermöglichen einen Branch&Cut-Algorithmus zu entwickeln, der auch das CDARP in vernünftiger Zeit löst.

3.2.1 Graphentheoretische Formulierung des Problems

Sei A die Menge der Aufträge, ein Auftrag $i \in A$ hat dann einen Startort i^+ und einem Zielort i^-. Die Menge der Startorte $\bigcup_{i \in A} i^+$ bezeichnen wir mit A^+ und die

Menge der Zielorte $\bigcup_{i \in A} i^-$ mit A^-. Der Startort des Fahrzeugs sei 0^+ und der Zielort 0^-. Sei V die Menge aller Start- und Zielorte von Aufträgen vereinigt mit dem Start- und Zielort des Fahrzeugs. Diese Menge V betrachten wir als die Knotenmenge eines vollständigen Graphen $G = (V, E)$.

Definition 3.2.1. Eine Kantenmenge $R \subset E$ nennen wir *gültige Route*, wenn gilt:

1. $(0^+, 0^-) \in R$ (Orientierung der Route Vervollständigung zum Hamilton-Kreis).

2. $|\delta(v) \cap R| = 2$ für alle $v \in V$, wobei $\delta(v)$ die Menge der zu v inzidenten Kanten ist (jeder Knoten wird genau einmal besucht).

3. $G(R)$, der von R induzierte Untergraph von G ist 2-zusammenhängend (dies verhindert die Bildung von mehreren Kreisen).

4. i^+ liegt auf dem Pfad von 0^+ nach i^- in $R \setminus (0^+, 0^-)$ für alle Aufträge $i \in A$. (Präzedenzbedingung)

Wollen wir noch zusätzlich fordern, daß das Fahrzeug höchstens k Aufträge auf einmal transportieren darf, fügen wir folgende Bedingung dazu:

5. $|\delta(Q)| \geqslant 4$ für alle $Q \in \{U \in \mathcal{U} : |U^+| - |U^-| > k\}$

Dies bedeutet, daß eine Teilmenge von $\mathcal{U}$, in der die Differenz zwischen aufgeladenen und abgeladenen Objekte größer als k ist, kein Pfad sein darf.

3.2.2 Formulierung als ILP

In der Formulierung als IP sieht das folgendermaßen aus:

Variablen

$$x_e = \begin{cases} 0, & e \text{ ist nicht in der Route.} \\ 1, & e \text{ ist in der Route.} \end{cases} \quad \text{für alle } e \in E.$$

Bedingungen

Sei $\mathcal{U} := \{U \subset V \mid 0^+ \in U\}$

$$x(0^+, 0^-) = 1 \tag{3.20}$$

$$x(\delta(v)) = 2 \quad \text{für alle } v \in V \tag{3.21}$$

$$x(\delta(U)) \geqslant 2 \quad \text{für alle } U \in \mathcal{U} \tag{3.22}$$

$$x(\delta(U)) \geqslant 4 \quad \text{für alle } U \in \mathcal{U} \text{ für die } 0^- \notin U \text{ und ein } i \text{ existiert mit } i^- \in U \text{ und } i^+ \notin U \tag{3.23}$$

$$x(e) \geqslant 0 \quad \text{für alle } e \in E \tag{3.24}$$

$$x(e) \leqslant 1 \quad \text{für alle } e \in E \tag{3.25}$$

$$x(e) \in \mathbb{N} \quad \text{für alle } e \in E \tag{3.26}$$

Die ersten drei Bedingungen sichern, daß die x_e einen Hamiltonkreis bilden, die Bedingung (3.23) sichert, daß für alle $i \in A$ i^+ auf der Route vor i^- liegt. Die Kapazitätsbeschränkung wird durch folgende Ungleichungen modelliert:

$$x(\delta(Q)) \geqslant 4 \quad \text{für alle } Q \in \{U \in \mathcal{U} : |A^+ \cap U| - |A^- \cap U| > k \text{ und } \nexists i \in A, i^+ \notin U, i^- \in U\} \tag{3.27}$$

Die Ungleichungen (3.27) sind äquivalent zur 5. Eigenschaft aus Abschnitt 3.2.1. Sie sorgen dafür, daß Mengen, die zwar den Präzedenzbedingungen (3.23) genügen, die aber mehr als k Objekte in das Fahrzeug laden, keinen Pfad bilden können.

Dieses ILP ist eine Verallgemeinerung des „Traveling Salesman Problems“ (TSP). Betrachten wir nur die Bedingungen (3.21) (3.22) und (3.24) bis (3.26), so haben wir die Formulierung des TSP von Dantzig, Fulkerson und Johnson [14].

3.2.3 Elimination redundanter Ungleichungen

Diese Formulierung ist nicht irredundant, d.h. einige der Ungleichungen sind Linearkombination anderer Ungleichungen. Wir wollen ein irredundantes System von Ungleichungen aufstellen.

Dazu definieren wir zwei Mengen von Knotenmengen, die gültige „subtour elimination constraints“ und Präzendenzungleichungen, also Ungleichungen vom Typ (3.22) und (3.23), definieren.

Wir ersetzen (3.22) und (3.23) durch

$$x(\delta(U)) \geqslant 2 \quad \text{für alle } U \in \mathcal{U}_s \tag{3.22'}$$

und

$$x(\delta(U)) \geqslant 4 \quad \text{für alle } U \in \mathcal{U}_p \tag{3.23'}$$

Dabei sind $\mathcal{U}_s$ und $\mathcal{U}_p$ wie folgt definiert:

$$\mathcal{U}_s := \{U \in \mathcal{U} | 0^-, \in U, 3 \leqslant |U| < |V| - 2, \forall i \in A(i^+ \in U \Leftrightarrow i^- \in U)\} \tag{3.28}$$

$$\mathcal{U}_p := \{U \in \mathcal{U} | 3 \leqslant |U| < |V| - 3, \exists! i \in A, i^+ \notin U, i^- \in U\} \tag{3.29}$$

Dann ist das neue System von Ungleichungen nach [28] Satz 3.11 irredundant und beschreibt das PDP vollständig.

3.2.4 Kapazitätsbeschränkung

Betrachten wir nun das Problem mit einer endlichen Kapazität k. Wir fügen also Ungleichungen vom Typ (3.27) hinzu. Diese enthalten keine Ungleichungen vom Typ (3.23′) nach Konstruktion und sind auch keine Linearkombination von Ungleichungen vom Typ (3.22′), da für die definierenden Mengen $U \in \mathcal{U}_s$ gilt, daß $|U^+| - |U^-| = 0$ ist. Also bleibt das ILP auch bei Hinzufügen der Ungleichung (3.27) irredundant.

3.2.5 Separation der Ungleichungen

Die Ungleichungen (3.20) bis (3.26) und ihre Separation sind bereits von Ruland in [28] untersucht wirden. Wir betrachten daher in erster Linie die Unlgeichungen (3.27) zur Beschränkung der Kapazität.

Wir interpretieren die gefundene gebrochene Lösung als Kantengewichte. Damit haben wir einen gewichteten Graphen. Nun betrachten wir je zwei Knoten als Quelle und Senke und berechnen den minimalen Schnitt zwischen diesen beiden Knoten. Diese Klasse von Ungleichungen läßt sich in polynomialer Zeit separieren. Man kann entweder mit dem Gomory-Hu Algorithmus [20] einen Baum finden, der jeweils einen minimalen Schnitt zwischen je zwei Knoten beschreibt. Dieser Algorithmus benötigt $|V| - 1$ Berechnungen minimaler Schnitte. Mit diesem Algorithmus findet man jedoch nicht alle verletzten Ungleichungen, da immer nur ein minimaler Schnitt zwischen je zwei Knoten berechnet wird, obwohl mehrere minimale existieren können. Ein ähnlicher Algorithmus von Gusfield [23] findet alle Schnitte und damit auch alle verletzten Ungleichungen. Haben wir einen Schnitt zwischen zwei Knoten u und v gefunden, (wir interpretieren u als Quelle und v als Senke), der kleiner ist als 2, so ist eine der Ungleichungen (3.22′) verletzt. Der Schnitt gibt dabei die Menge U an. Ist der Schnitt kleiner als 4, und $u > v$, so ist eine der Ungleichungen (3.23′) verletzt.

Beschreibung des Separationsalgorithmus

Für jeweils ein Paar von Knoten u, v suchen wir den maximalen (u, v)-Fluß in V. Dann wählen wir die Knotenmenge Q aus V, deren Knoten adjazente Kanten mit Flußwert größer als Null haben. Ist dann $x(\delta(Q)) < 4$, und $|Q^+| - |Q^-| > k$, so haben wir eine durch Q definierte verletzte Ungleichung vom Typ (3.27) gefunden.

Nun stellt sich die Frage, ob wir alle verletzten Nebenbedingungen finden, wenn wir alle maximalen (u, v)-Flüsse überprüfen

Wir stellen also fest für welche Knotenmengen Q verletzte Ungleichungen vom Typ (3.27) auftreten können.

Lemma 3.2.2. *Ist* $x(\delta(Q)) < 4$, *dann gilt für jeden Schnitt* $[U : W]$, $U \cup W = Q$, *daß* $x([U : W]) > 0$ *ist.*

Beweis. Es gilt

$$x([U : W]) = \frac{x(\delta(U)) + x(\delta(W)) - x(\delta(Q))}{2}.$$

Zur Verdeutlichung siehe auch Abbildung 3.3. Da nach Bedingung (3.22) sowohl $x(\delta(U)) \geqslant 2$ als auch $x(\delta(W)) \geqslant 2$ ist, folgt

$$x([U : W]) > \frac{2 + 2 - 4}{2} = 0.$$

□

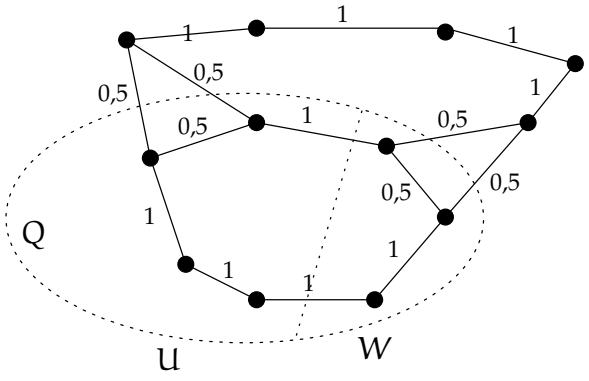

Abbildung 3.3: Die Mengen U, W und Q aus dem Lemma 3.2.2.

Wir brauchen also tatsächlich nur zusammenhängende Mengen von Knoten Q zu untersuchen.

Wir finden also z.B. verletzte Ungleichungen, indem wir von jedem Knoten startend den Graphen mit Tiefensuche oder Breitensuche durchgehen. Dies ist in polynomialer Zeit durchführbar. Nach jedem neu entdeckten Knoten wird überprüft, ob $|U^+| - |U^-| > k$ ist. Ist dies der Fall, haben wir eine verletzte Ungleichung gefunden.

Leider findet man mit dieser Methode nicht alle verletzten Ungleichungen, weil man sich dazu alle zusammenhängenden Teilmengen der Knoten des Graphen ansehen müsste. Ein Graph kann aber exponentiell in der Anzahl der Knoten viele solcher Teilmengen enthalten. In einem vollständigen Graph ist z.B. jede Teilmenge zusammenhängend.

Die so separierten Nebenbedingungen reichen aber in einem Branch&Cut-Algorithmus aus, eine optimale Lösung zu finden. Ist nämlich bereits eine ganzzahlige Lösung gefunden worden, so entspricht diese Lösung, wegen der Nebenbedingungen (3.22) („subtour elimination constraints“) einem Hamilton-Kreis. Durchsucht man diesen mit Tiefensuche, so geht man einfach die Knoten in der Reihenfolge, in der sie im Hamilton-Kreis enthalten sind, durch. Gilt $|U^+|-|U^-| > k$, dann ist die Kapazität des Fahrzeugs überschritten, also hat man eine verletzte Ungleichung gefunden.

Will man Start-/Stopzeiten eines Aufzuges in diesem Problem modellieren, so müssen die Koeffizienten der Zielfunktion nur entsprechend angepaßt werden. FIFO-Präzedenzen zwischen Aufträgen kann man modellieren, indem man die Nebenbedingungen (3.23′) auf einer anderen Menge definiert. Und zwar müßte man die Menge $\mathcal{U}_s$ wie folgt definieren:

$$\begin{aligned}\mathcal{U}_s := \{U \in \mathcal{U} : 0^- \in U, 3 \leqslant |U| < |V| - 2,\\ \forall i \in A(i^+ \in U \Leftrightarrow i^- \in U) \text{ oder } \forall i \prec j(i^+ \in U \Leftrightarrow j^+ \in U)\}.\end{aligned}$$

Die Ungleichungen (3.23′) für diese Mengen haben zur Folge, daß für je zwei Aufträge $i, j \in A$ für die $i \prec j$ gilt, die Lösung keinen Pfad $(0, \ldots, j^+, \ldots, i^+)$ enthalten kann, der einem Aufladen des Auftrags j vor dem Auftrag i entsprechen würde.

Kapitel 4

Simulation und numerische Ergebnisse

4.1 Numerische Ergebnisse

Wir wollen nun die in dieser Arbeit vorgestellten Algorithmen auf Pfaden vergleichen. Unsere Testprobleme bestehen aus Pfaden $G = (V, E)$ mit $n = 5, 9$ bzw. 20 Knoten. Die Kosten für eine Kante zwischen zwei benachbarten Knoten in G haben wir auf 1 gesetzt. Also ist $c(v, v+1) = 1$ für alle $v \in V \setminus \{n\}$.

Dazu haben wir zufällige Mengen A von $n = 20$ Aufträge erzeugt. Dabei sind die Start- und Zielknoten der Aufträge gleichverteilt. Dann haben wir die verschiedenen Algorithmen die entsprechenden cDARP $(V, E, A, c, 1, k)$ mit $k = 2, 3, 5$ oder 10 lösen lassen. Wir haben jeweils 1000 Instanzen des cDARP gelöst.

Wir haben folgende Algorithmen getestet:

ABSTIMM Eine modifizierte Abstimm-Heuristik aus Abschnitt 2.6.1 für cDARP auf Pfaden. Dabei wird aber nur die Richtung geändert, wenn zusätzlich zu der Bedingung für Richtungsänderung aus dem Abschnitt 2.6.1 sich entweder keine Aufträge im Aufzug befinden, oder sich durch die Richtungsänderung ein λ_v verringert.

AUFAB Die Auf-Ab-Heuristik aus Abschnitt 2.6.2 für cDARP und FIFO-cDARP auf Pfaden.

COVERPATH aus Abschnitt 2.7 für das cDARP auf Pfaden.

INTERVALL Den Algorithmus INTERVALLGRAPH aus Abschnitt 2.9 für cDARP und FIFO-cDARP auf Pfaden.

IP Das ganzzahlige lineare Programm aus Abschnitt 3.1 für das cDARP auf Pfaden.

OPTPATH Wir haben den Algorithmus OPTPATH aus Abschnitt 2.3, der das DARP auf Pfaden optimal löst, zum Vergleich laufen lassen, um herauszufinden, um welchen Faktor sich die Kosten im Vergleich zur Kapazität 1 verringern.

Außerdem haben wir jeweils die Flußschranke berechnet.

Nun haben wir zufällig Aufträge mit gleichverteilten Start- und Zielknoten generiert. Die Spalte „gap" gibt den größten auftretenden Quotienten $\frac{\text{Fluß-Schranke}}{t}$ an, wobei t die Anzahl der Stockwerke ist, die der Aufzug überquert bis er alle Objekte an ihr Zielstockwerk gebracht hat. Dieser Quotient unterscheidet sich in unseren Versuchen in den meisten Fällen nicht von dem maximalen Quotienten $\frac{c(\mathcal{OPT})}{t}$, wobei $\mathcal{OPT}$ eine optimale Lösung der jeweiligen Instanz ist. Wir haben die Qualität der Lösungen aber mit der Fluß-Schranke verglichen, weil diese im Gegensatz zu der optimalen Lösung auch für große Instanzen schnell berechenbar ist.

Tabelle 4.1: 20 Aufträge, 5 Stockwerke

Algorithmus	k=2		k=3		k=5		k=10	
	t	gap	t	gap	t	gap	t	gap
Fluß-Schranke	27,02		19,01		13,18		8,086	
OPTPATH	50,01	2	48,81	3	50,18	5	49,37	8,75
ABSTIMM	32,95	1,93	23,53	1,88	16,70	2	8,34	1,75
AUFAB	31,43	1,64	21,90	1,67	15,45	1,6	8,18	1,4
COVERPATH	27,72	1,27	19,61	1,29	14,13	1,4	8,16	1,2
INTERVALLGRAPH	27,84	1,27	19,67	1,29	14,12	1,4	8,16	1,4
IP	27,13	1,18	19,03	1,14	13,21	1,2	8,09	1

Tabelle 4.2: 20 Aufträge, 9 Stockwerke

Algorithmus	k=2		k=3		k=5		k=10	
	t	gap	t	gap	t	gap	t	gap
Fluß-Schranke	45,95		33,12		23,44		16,07	
OPTPATH	83,77	2	83,72	2,94	84,57	4,5	84,39	7,38
ABSTIMM	62,89	2,19	45,82	2,47	32,76	2,54	17,94	2,63
AUFAB	56,78	1,73	41,07	1,71	29,19	1,78	16,24	1,78
COVERPATH	49,12	1,35	36,28	1,47	26,90	1,56	16,18	1,44
INTERVALLGRAPH	49,01	1,39	36,07	1,38	26,75	1,5	16,18	1,44
IP	46,25	1,36	33,47	1,44	23,70	1,5	16,07	1

Tabelle 4.3: 20 Aufträge, 20 Stockwerke

Algorithmus	k=2		k=3		k=5		k=10	
	t	gap	t	gap	t	gap	t	gap
Fluß-Schranke	99,18		72,3		51,96		37,73	
OPTPATH	179,08	1,94	177,44	2,76	180,57	4,14	177,95	7,95
ABSTIMM	152,37	2,8	113,49	2,83	81,16	2,93	47,74	4,32
AUFAB	126,12	1,81	92,21	1,75	66,58	1,79	38,07	1,9
COVERPATH	110,07	1,30	82,0	1,5	61,76	1,55	37,95	1,5
INTERVALLGRAPH	108,75	1,35	80,64	1,33	61,08	1,64	37,95	1,45

Erwartungsgemäß waren die Lösungen der beiden Algorithmen COVERPATH und INTERVALLGRAPH etwa gleich gut, während die einfacheren Heuristiken schlechter abschnitten.

Wenn man nun vergleicht welche Verbesserung sich bei höherer Kapazität gegenüber einer Kapazität von einem Objekt ergibt, zeigt sich, daß eine Verdopplung der Kapazität keine Halbierung der Transportzeiten nach sich zieht und daß die Verbesserung der Transportzeiten immer geringer ausfällt, wenn sich die Kapazität weiter erhöht.

In der Tabelle 4.4 ist aufgeführt, wie das Verhältnis der Kosten des optimalen Transports für Kapazitäten von $k = 2, 3, 5$ zum Verhältnis eines optimalen Transports mit Kapazität 1 verhalten. Für $k = 10$ haben wir zum Vergleich die Kosten eines mit dem Algorithmus INTERVALLGRAPH ermittelten Transports herangezogen, da die Berechnung eines optimalen Transports mit dem im Abschnitt 3.1 vorgestellten ILP zu viel Zeit in Anspruch nahm.

Tabelle 4.4: Quotienten der Kosten für optimale Lösungen für $k = 1$ und $k > 1$

Stockwerke	5	9	20
$k = 2$	1,84	1,81	1,65
$k = 3$	2,56	2,50	2,20
$k = 5$	3,80	3,57	2,96
$k = 10$	6,10	5,25	4,69

Die Verbesserung der Fertigstellungszeit eines Aufzug mit Kapazität k im Vergleich zu einem Aufzug mit Einheitskapazität ist aber auch von der Anzahl der Aufträge abhängig. Stehen viele Aufträge zur Verfügung, so Verringern sich die Kosten eines k-CDARP im Vergleich zum DARP um nahezu k. Stehen hingegen relativ wenig Aufträge zur Verfügung, so fällt die Verbesserung geringer aus.

Zuletzt haben wir noch den Einfluß des Einführens einer FIFO-Ordnung für die Abarbeitung der Aufträge untersucht. Hier sieht man, daß bei höherer Kapazität

Tabelle 4.5: 20 Aufträge, 9 Stockwerke, FIFO

Algorithmus	k=2		k=3		k=5		k=10	
	t	gap	t	gap	t	gap	t	gap
Fluß-Schranke	46,29		33,23		23,21		16,03	
AUFAB	63,74	2,11	47,13	2,5	35,11	2,91	29,74	3,63
INTERVALLGRAPH	57,19	1,68	43,08	2,07	32,52	2,75	29,57	3,38

der Aufzüge die Lücke der Lösung des Algorithmus INTERVALLGRAPH zur Fluß-Schranke immer größer wird. Einen optimalen gültigen Transport haben wir nicht berechnet, da die Rechenzeit des IPs für das FIFO-CDARP zu lang war.

Die Algorithmen wurden auf einer Sun Ultra-1 m140 mit 128 MB Speicher unter Solaris 5.4 getestet. Die Rechenzeiten betrugen für die verschiedenen Algorithmen mit polynomialer Laufzeit nur Sekundenbruchteile. Das ILP für ein CDARP auf einem Pfad mit fünf Knoten und zwanzig Aufträgen wurde im Durchschnitt in zwei Sekunden gelöst. Erhöht man die Knotenanzahl auf neun, dauerte es durchschnittlich 15 Sekunden, bei einigen Instanzen aber bis zu drei Minuten. Mit 20 Knoten dauerte das Lösen einer Instanz im Durchschnitt schon deutlich über einer Minute. Ein CDARP mit 40 Aufträgen und acht Stockwerken wurde in ca. 4 Stunden gelöst.

4.2 Online Probleme

Bisher haben wir nur die Situation untersucht, daß alle Daten einer Probleminstanz vollständig gegeben sind. In der Realität aber sind nicht alle Aufträge schon von Anfang an bekannt. Es können dem Aufzug Aufträge erst zu einem späteren Zeitpunkt bekannt gemacht werden. Solche Probleme nennen wir *Online*-Probleme. Eine Einführung über verschiedene Online-Algorithmen findet man z.B. in [1]. Eine Übersicht über Online-Algorithmen in der Praxis ist [4].

Wir benutzen nun ein *Zeitstempel-Modell*, um unser Problem zu modellieren.

Definition 4.2.1 (Online-CDARP). Sei eine Instanz eines CDARP durch das Tupel $P = (V, E, A, c, o, k)$ gegeben. Das Online-CDARP ist nun ein CDARP, wobei zusätzlich eine Funktion $t : A \to \mathbb{Q}$ existiert, die jedem Auftrag $i \in A$ einen Zeitpunkt $t(i)$ zuweist zu dem der Auftrag i dem Algorithmus bekannt wird.

Man kann nun eine kürzeste Fertigstellungzeit für alle Aufträge in A suchen. In der Praxis ist diese aber nicht relevant, da ein Aufzugssytem ununterbrochen arbeitet. Das heißt, die Zahl der Aufträge kann beliebig groß werden. Uns interessiert daher die maximale bzw. durchschnittliche Bearbeitungszeit der Aufträge, das heißt, die maximale bzw. durchschnittliche Zeitspanne eines Auftrags zwischen Bekanntwerden des Auftrags und seiner Ablieferung an seinem Zielknoten.

Zum Testen der Algorithmen haben wir die ereignisgesteuerte diskrete Simulation eines Aufzugs benutzt, welche auf der Bibiliothek von C/C++-Funktionen AMSEL [2] basiert. Diese Simulation ist genauer in [21] beschrieben.

Man kann drei verschiedene Herangehensweisen unterscheiden:

1. Man plant nach jedem neu bekanntwerdenden Auftrag neu. Angenommen zum Zeitpunkt t wird ein neuer Auftrag bekannt. Dann löst man das entsprechende Offline-Problem mit allen zum Zeitpunnkt t bekannten aber noch nicht abgearbeiteten Aufträgen und verwendet dessen Lösung als neuen Transportplan. Diese Methode nennen wir REPLAN.

   Bei dieser Herangehensweise müssen viele Probleminstanzen gelöst werden, da bei jedem neu auftauchenden Auftrag der Lösungsalgorithmus neu abgearbeitet werden muss. Außerdem ist das hier zu lösende Problem ein anderes als das FIFO-CDARP, da zum Zeitpunkt t eines neu bekanntwerdenden Auftrags der Aufzug nicht an einem Endpunkt des unterliegenden Graphen stehen muß. Auch kann der Aufzug zum Zeitpunkt t noch Aufträge enthalten. Erledigt man aber die Aufträge im Aufzug und fährt an einen Endpunkt des Graphen verschenkt man Fahrzeit.

2. Man plant immer dann neu, wenn ein vorhandener Transportplan komplett abgearbeitet ist und unbearbeitete Aufträge bekannt sind. Diese Methode nennen wir IGNORE.

3. Jeder neue Auftrag wird in den vorhandenen Transportplan so eigefügt, daß er möglichst wenig Kosten verursacht. Diese Methode nennen wir GREEDY.

Wir haben folgende Online-Algorithmen in einer Simulation eines Aufzugs verwendet:

GREEDY Dieser Algorithmus fügt jeden neu ankommenden Auftrag möglichst früh in seinen Transportplan ein, wenn dies ohne Erhöhung der Kosten möglich ist. Ansonsten wird der Auftrag an das Ende des Transportplans angehängt.

GREXP Dieser Algorithmus fügt jeden neu ankommenden Auftrag möglichst früh in seinen Transportplan ein, auch wenn dies die Kosten erhöht.

IGNIV Dieser Algorithmus ist vom Typ **IGNORE**. Er berechnet immer dann einen neuen Transportplan, wenn keiner vorhanden ist, aber Aufträge im System existieren. Dieser Transportplan wird mit dem Algorithmus INTERVALLGRAPH aus Abschnitt 2.9 berechnet. Dieser Transportplan wird dann abgearbeitet.

IGNGRIV Dieser Algorithmus arbeitet wie der Algorithmus **IGNIV**, nur daß er Aufträge, die dem System bekannt aber nicht im aktuellen Transportplan eingeplant sind, befördert, wenn dies ohne zusätzliche Kosten möglich ist.

REPLANIV Dieser Algorithmus berechnet für jeden neu ankommenden Auftrag einen neuen Transportplan mit dem Algorithmus INTERVALLGRAPH.

REPLAN Ein Replan-Algorithmus für $k = 1$, der mit einem Branch&Bound-Algorithmus einen meist optimalen Transportplan für das entsprechende DARP berechnet. Für Details siehe [21].

Die in Kapitel 3 vorgestellten ganzzahligen linearen Programme sind in der Praxis bisher nicht einsetzbar, da ihre Rechenzeit auch für verhältnismäßig kleine Probleminstanzen noch zu groß sind. Die Rechenzeit muß ja wegen der Echtzeitsituation zu den Fahrzeiten der Aufzüge addiert werden.

Zur theoretischen Beurteilung von Online-Algorithmen benutzt man die *Kompetitivität* eines Online-Algorithmus.

Definition 4.2.2 (Kompetitivität). Gegeben sei ein Online-Algorithmus A für das (Minimierungs-)Problem Π. Seien $c_{\text{online}}(y)$ die Kosten der Lösung einer Instanz y des Problems Π, die der Algorithmus A ermittelt hat. Und seien $c_{\mathcal{OPT}}(y)$ die Kosten einer optimalen Lösung des dazugehörigen Offline-Problems.

Der Online-Algorithmus A ist k-*kompetitiv*, wenn für jede Instanz y eine Konstante c existiert, so daß gilt:

$$c_{\text{online}}(y) \leqslant k c_{\mathcal{OPT}}(y) + c.$$

Satz 4.2.3 (Ascheuer, Krumke und Rambau). *Der Algorithmus* IGNORE *ist 5/2-kompetitiv für das Online-*CDARP *mit beliebiger Kapazität, wenn der Algorithmus* IGNORE *das entsprechende Offline-Problem optimal löst.*

Beweis. Für den Beweis siehe [5], Satz 4.4. □

Da wir nur einen Approximationsalgorithmus mit Gütegarantie 3 in **IGNIV** benutzen ist **IGNIV** 15/2-kompetitiv bezüglich der Fertigstellungszeit. Damit ist auch **IGNGRIV** 15/2-kompetitiv, da die Fertigstellungszeit des Algorithmus **IGNGRIV** nicht größer sein kann, als die des Algorithmus **IGNIV**. Für die restlichen Algorithmen sind keine theoretischen Resultate bekannt.

Zunächst haben wir die Fahrzeiten zwischen den Stockwerken auf 1 Sekunde und die Start- und Stopzeiten auf 0 Sekunden gesetzt. Wir haben nun die Aufträge poissonverteilt erzeugt. Der Parameter t_ϵ gibt dabei den Erwartungswert des Abstandes zwischen zwei Aufträgen in Sekunden an.

An den Ergebnissen sehen wir, daß die Greedy-Algorithmen am Besten abschneiden, sowohl was die durchschnittliche als auch die maximale Fluß-Zeit betrifft. Der Algorithmus **REPLIV** ist ähnlich gut wie die Greedy-Algorithmen. In einigen Instanzen ist er zwar in den durschnittlichen Fluß-Zeiten etwas besser als der **GREEDYEXPAND**-Algorithmus, dafür ist er aber in der maximalen Flußzeit oft schlechter. Dieser Effekt liegt nahe, da der **GREEDYEXPAND**-Algorithmus jeden Auftrag so früh wie möglich abarbeitet, die Aufträge dabei aber in der Reihenfolge ihrer Ankunft einplant. Der Algorithmus **REPLIV** dagegen berücksichtigt die

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	30	33	27	89	98	76	4028	4052	4015
GREXP	26	30	24	90	104	73	4016	4031	4003
IGNIV	265	349	203	712	921	538	4492	4606	4331
IGNGRIV	262	349	209	690	921	466	4470	4606	4275
REPLIV	29	34	24	175	245	127	4017	4029	4003
REPLAN	247	285	186	1131	1777	609	4552	4734	4357

Tabelle 4.6: Ergebnisse der Simulation mit 8 Stockwerken, $t_\epsilon = 10$ und $k = 2$

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	12	13	11	40	49	35	4012	4022	3998
GREXP	11	13	11	43	56	33	4009	4017	3998
IGNIV	13	15	12	46	51	40	4011	4026	3997
IGNGRIV	13	15	12	49	62	40	4011	4026	3997
REPLIV	12	13	11	47	58	43	4008	4021	3998
REPLAN	12	14	12	65	78	50	4010	4021	3998

Tabelle 4.7: Ergebnisse der Simulation mit 8 Stockwerken, $t_\epsilon = 20$ und $k = 2$

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	22	23	20	73	82	61	4024	4045	4008
GREXP	19	20	18	69	83	60	4021	4033	4012
IGNIV	36	42	31	95	115	71	4048	4081	4029
IGNGRIV	35	43	29	110	152	76	4043	4075	4020
REPLIV	19	20	19	91	145	61	4022	4033	4011
REPLAN	217	249	171	887	1180	737	4404	4575	4257

Tabelle 4.8: Ergebnisse der Simulation mit 8 Stockwerken, $t_\epsilon = 10$ und $k = 3$

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	12	13	11	37	43	30	4009	4014	4007
GREXP	11	12	11	32	34	30	4009	4010	4007
IGNIV	13	14	12	36	42	29	4013	4019	4007
IGNGRIV	13	14	12	36	42	29	4013	4019	4007
REPLIV	11	12	11	42	48	33	4009	4010	4007
REPLAN	13	14	12	68	101	46	4010	4016	4004

Tabelle 4.9: Ergebnisse der Simulation mit 8 Stockwerken, $t_\epsilon = 20$ und $k = 3$;

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	237	322	182	562	681	435	4394	4521	4325
GREXP	192	269	163	492	622	400	4310	4383	4205
IGNIV	984	1233	743	2495	2955	2134	5854	6167	5577
IGNGRIV	916	1233	549	2391	2896	2004	5730	6067	5247
REPLIV	191	258	159	1554	2638	1028	4308	4372	4207
REPLAN	797	950	643	3205	3535	2899	5665	5823	5490

Tabelle 4.10: Ergebnisse der Simulation mit 30 Stockwerken, $t_\epsilon = 20$ und $k = 3$

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	71	79	61	225	270	180	4104	4119	4089
GREXP	57	65	47	183	201	163	4085	4123	4066
IGNIV	138	270	81	387	735	197	4211	4445	4106
IGNGRIV	145	270	88	430	735	308	4243	4445	4106
REPLIV	57	69	46	305	388	189	4061	4079	4051
REPLAN	179	257	98	1293	2239	456	4348	4505	4245

Tabelle 4.11: Ergebnisse der Simulation mit 30 Stockwerken, $t_\epsilon = 30$ und $k = 3$

Algorithmus	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
	ø	max	min	ø	max	min	ø	max	min
GREEDY	37	40	34	114	142	90	4018	4038	3991
GREXP	34	36	32	105	139	83	4015	4035	3987
IGNIV	40	47	35	120	136	102	4018	4038	3987
IGNGRIV	38	43	34	111	136	97	4018	4038	3987
REPLIV	32	34	31	120	169	95	4013	4024	3987
REPLAN	38	46	35	201	237	174	4012	4019	3987

Tabelle 4.12: Ergebnisse der Simulation mit 30 Stockwerken, $t_\epsilon = 50$ und $k = 3$

Reihenfolge der Ankunft der Aufträge überhaupt nicht und versucht nur die Auslastung des Aufzugs zu maximieren.

Erstaunlicherweise sind die beiden **IGNORE**-Varianten deutlich schlechter als die anderen getesteten Algorithmen. Offensichtlich verlieren sie zu viel Zeit durch nicht-vollausgelastete Bewegungen, die durch Ignorieren von Aufträgen entstehen, während die Vergleichsalgorithmen die Aufzüge besser auslasten.

Man sieht an den Ergebnissen auch, wie deutlich eine Erhöhung der Kapazität des Aufzugs von einem auf zwei Objekte die maximalen Fluß-Zeiten unter hoher Last reduziert. (von ca. 250s auf 25s im Durchschnitt siehe Tabelle 4.6). Aber auch die durchschnittlichen Fluß-Zeiten verkürzen sich um den Faktor 10. Dabei war der **REPLAN**-Algorithmus, wie in [21] gezeigt, schon recht gut für $k = 1$.

Die weitere Erhöhung der Kapazität von 2 auf 3 zeigt dagegen nur eine geringere Verkürzung der Flußzeiten. Auch bei einer größeren Anzahl von Stockwerken ändern sich die Ergebnisse im Prinzip nicht.

Zum Abschluß haben wir Aufzüge mit Fahrzeiten wie denjenigen bei Herlitz simuliert. Der Aufzug benötigt für die Fahrt von einem Stockwerk zu einem nächstgelegenen 4 Sekunden. Zum Starten und Stoppen benötigt er jeweils 5 Sekunden.

	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
Algorithmus	ø	max	min	ø	max	min	ø	max	min
GREEDY	133	151	115	449	535	356	25271	25324	25237
GREXP	115	130	98	431	503	349	25279	25373	25217
REPLIV	133	157	105	1010	1339	536	25303	25408	25217
IGNIV	820	1400	411	2372	3875	1281	26534	27781	25560
IGNGRIV	798	1400	411	2286	3875	1281	26472	27781	25560
REPLAN	883	1249	617	4808	7628	3365	26700	28109	25830

Tabelle 4.13: Ergebnisse der Simulation mit 9 Stockwerken, $t_e = 50$ und $k = 2$

	ø-Fluß			Maximal-Fluß			Fertigstellungszeit		
Algorithmus	ø	max	min	ø	max	min	ø	max	min
GREEDY	56	57	56	199	242	165	25235	25279	25196
GREXP	54	56	54	184	195	167	25231	25279	25186
REPLIV	55	58	53	236	287	188	25237	25293	25196
IGNIV	63	65	62	245	313	215	25245	25293	25196
IGNGRIV	63	65	62	245	313	215	25245	25293	25196
REPLAN	64	65	62	348	431	303	25251	25331	25196

Tabelle 4.14: Ergebnisse der Simulation mit 9 Stockwerken, $t_e = 80$ und $k = 2$

Im Prinzip ändert sich nichts an den Ergebnissen der Algorithmen, nur fallen die Unterschiede geringer aus, da die Algorithmen alle ähnliche Zeiten zum Starten und Stoppen benötigen und sich nur in den Fahrzeiten des Aufzugs unterscheiden.

Interessant wäre es zu untersuchen, wie sich die Zeiten ändern, wenn man einen Algorithmus verwendet, der auch die Start- und Stopzeiten minimiert.

Kapitel 5

Zusammenfassung

Wir haben gezeigt, die Steuerung von Aufzügen ein keineswegs triviales Problem ist und haben für ein recht einfaches Modell eines Mehrplatz-Aufzugs, nämlich das CDARP auf Pfaden ein neues theoretisches Ergebnis gefunden. Im Abschnitt 2.7 haben wir nämlich einen 3-Approximationsalgorithmus für dieses Problem vorgestellt, der auch in numerischen Versuchen in Kapitel 4 gute Ergebnisse erzielte. Er lag im Durchschnitt nur 5% über der optimalen Lösung. Für das FIFO-CDARP haben wir eine Heuristik im Abschnitt 2.9 vorgestellt, für die wir keine Gütegarantie zeigen konnten, die aber dennoch in numerischen Versuchen recht gut abschnitt.

Ferner haben wir für das CDARP auf Pfaden ein ganzzahliges lineares Programm in Abschnitt 3.1 entwickelt. Mit dessen Hilfe kann man optimale Lösungen des CDARP auf Pfaden für Instanzen mit ca. 10 Stockwerken und 20 Aufträgen in 10 Sekunden bis 3 Minuten berechnen.

Im Abschnitt 3.2 haben wir ein ganzzahliges lineares Program für das CDARP auf allgemeinen Graphen vorgestellt, welches auch für das FIFO-CDARP auf allgemeinen Graphen geeignet ist. Eine Implementation eines Branch&Cut-Alogrithmus zum Lösen dieses ganzzahligen linearen Programms wurde nicht vorgenommen.

Eine weitere Untersuchung des Online-CDARP wäre von Interesse, da einfache Ansätze zum Einsatz der Offline-Algorithmen für das Online-CDARP nicht die gewünschten Ergebnisse erbrachten. Dabei stellt sich aber wieder die Frage, wie man die Güte von Online-Algorithmen misst.

Auch ist die Frage, ob für das FIFO-CDARP oder das CDARP mit Start- und Stopzeiten ein Approximationsalgorithmus mit konstanter Gütegarantie existiert, weiterhin offen.

Die praktischen Versuche mit der Simualtion im Abschnitt 4.2 haben gezeigt, daß unter Vernachlässigung der Start- und Stopzeiten eines Aufzugs eine höhere Kapazität unter hoher Last eine deutliche Verbesserungen bei den Fluß-Zeiten der Aufträge bringt. Bei geringerer Last fällt diese Verbesserung naturgemäß geringer aus.

Berücksichtigt man die Start- und Stopzeiten der Aufzüge, so ist eine Verbesserung der Fluß-Zeiten weiterhin nachweisbar, sie fällt aber geringer aus. Hier stellt sich die Frage nach Offline- und Online-Algorithmen, die diese Zeiten berücksichtigen und auch die Anzahl der Starts und Stops an den Stockwerken minimieren.

Anhang A

Notation und Begriffe

A.1 Graphen

Ein *Graph* $G = (V, E)$ ist ein Tupel einer Menge von Knoten V und einer Menge von Kanten E. In dieser Arbeit sind alle Graphen *einfach*. Das heißt, daß zwischen je zwei Knoten $u, v \in V, u \neq v$ maximal eine Kante existiert. Eine Kante zwischen den Knoten $u, v \in V$ bezeichnen wir mit (u, v). Die Kante (u, v) ist gleich der Kante (v, u).

Eine Kante $e \in E$ ist *inzident* zu einem Knoten $v \in V$, wenn einer der beiden Endpunkte der Kante gleich v ist.

Ein Graph ist *vollständig*, wenn zwischen je zwei Knoten aus V eine Kante existiert.

Ein *Weg* in einem Graphen ist eine Folge $(v_1, e_1, v_2, e_2, \ldots, e_{s-1}, v_s)$ von Knoten und Kanten, so daß jede Kante $e_i, 1 \leqslant i < s$ in diesem Weg inzident zu den Knoten v_i und v_{i+1} ist. Ein *Kreis* ist ein Weg, in dem Startknoten v_1 und Endknoten v_s gleich sind, und der keine Kante doppelt enthält. Eine *Tour* in einem Graphen $G = (V, E)$ ist ein Kreis, der jeden Knoten aus V genau einmal enthält.

Ein Graph ist *zusammenhängend*, wenn zwischen je zwei Knoten ein Weg existiert.

Eine Kante in einem zusammenhängenden Graphen ist *kritisch*, wenn durch ihr Entfernen der Graph nicht mehr zusammenhängend ist.

Ein *Digraph* $D = (V, A)$ ist ein gerichteter Graph. Ein gerichteter Graph besteht einer Menge von Knoten V und einer Menge von Bögen A. Jedem Bogen $i \in A$ ordnen wir einen Startknoten i^+ und einen Zielknoten i^- zu. Das Tupel von Start- und Zielknoten eines Bogens i bezeichnen wir auch mit $\langle i^+, i^- \rangle$. Dieses Tupel von Start- und Zielknoten ist ein anderes Objekt als der eigentliche Bogen. Es können nämlich in Digraphen mehrere unterschiedliche Bögen mit gleichem Start- und Zielknoten existieren. Wenn wir also von einem Bogen $\langle u, v \rangle, u, v \in V$ reden, ist damit ein Bogen i gemeint, für den $\langle i^+, i^- \rangle = \langle u, v \rangle$ gilt.

Mit $G[E]$ bezeichnen wir den durch die Kantenmenge E *induzierten* Graphen. $G[E] = (V, E)$ ist der Graph, dessen Knotenmenge V genau die Knoten enthält, die Endknoten von Kanten in E sind und dessen Kantenmenge E ist. Einen durch eine Bogenmenge A induzierten Digraphen $G[A]$ definieren wir analog.

Ein *Pfad* ist ein zusammenhängender Graph, für den eine Sortierung $v_1, v_2, v_3, \ldots, v_n$ seiner Knotenmenge V existiert, so daß für alle Kanten $e \in E$ gilt, daß ein $i = 1, \ldots, n$ existiert, so daß $e = (v_i, v_{i+1})$ ist.

Der *Grad* $\delta(v)$ eines Knotens $v \in V$ in einem Graphen, ist die Anzahl der zu diesem Knoten inzidenten Kanten. In einem gerichteten Graphen unterscheiden wir zusätzlich den Grad eines Knotens bezüglich der in diesen herausgehenden und hereingehenden Bögen. Wir schreiben $\delta^+(v)$ für die Anzahl der Kanten $i \in A$ mit $i^- = v$ und $\delta^-(v)$ für die Anzahl der Bögen $i \in A$ mit $i^+ = v$.

Ein *Schnitt* $[U : W]$ eines Graphen $G = (V, E)$ für zwei disjunkte Knotenmengen $U, W \subset V$ ist die Menge der Kanten, die sowohl zu einem Knoten aus U als auch zu einem Knoten aus W inzident sind.

A.2 Komplexitätstheorie

Wir wollen hier, teilweise etwas informell, die wichtigsten Grundbegriffe der Komplexitätstheorie einführen, soweit wir sie in dieser Arbeit benötigen. Für eine ausführlichere Darstellung siehe [19].

Um die Laufzeit oder den Speicherbedarf von Algorithmen zu vergleichen führen wir Klassen von Funktionen ein. Wir sagen eine Funktion f ist in der Klasse $O(g)$, wobei g eine Funktion ist, bzw. $f \in O(g)$, wenn gilt: Es existiert eine Konstante $c > 0$ und eine Konstante n_0, so daß $f(n) \leqslant cg(n)$ für alle $n \geqslant n_0$ gilt.

Sei Σ eine endliche Menge von Symbolen, also ein *Alphabet*. Ein *Wort* z aus dem Alphabet Σ ist eine endliche Folge von Symbolen aus Σ. Mit Σ^* bezeichnen wir die Menge aller möglichen Wörter aus dem Alphabet Σ.

Ein (mathematisches) *Problem* ist gegeben durch eine Teilmenge Π von $\Sigma^* \times \Sigma^*$, wobei Σ ein Alphabet ist. Sei nun ein Wort $z \in \Sigma^*$ und folgende Aufgabe gegeben: Finde ein Wort $y \in \Sigma^*$, so daß $(z, y) \in \Pi$ liegt, oder stelle fest, daß kein solches Wort y existiert.

Ist $y = \emptyset$ für jedes Tupel $(z, y) \in \Pi$, so nennen wir das Problem *Entscheidungs*-Problem. Das Wort z bezeichnen wir als *Probleminstanz*. Ein Entscheidungs-Problem ist also ein Problen, in dem wir nur die Zulässigkeit von z feststellen müssen.

Wir sagen ein Algorithmus *löst* das Problem Π, wenn er für jede Probleminstanz z eine gültige Lösung y findet oder entscheidet, daß das Problem nicht lösbar ist.

Die Laufzeit eines Algorithmus zur Lösung eines Problems ist im allgemeinen über die Bewegung eines Schreib-/Lesekopfes einer universellen Turing-Maschine

bei gegebene Eingangsdaten definiert. Wir wollen aber ein etwas einfacheres Rechnermodell verwenden. Wir kodieren alle Zahlen binär. Ein Zahl $n \in \mathbb{Z}$ hat also eine Kodierungslänge von $\lceil \log_2 |n| + 1 \rceil + 1$ und eine Zahl $x = \frac{p}{q} \in \mathbb{Q}, p, q \in \mathbb{Z}$ eine Kodierungslänge von $\lceil \log_2 |p| + 1 \rceil + \lceil \log_2 |q| + 1 \rceil$. Wir bezeichnen die Kodierungslänge von x mit $size(x)$.

Im Allgemeinen gehen wir davon aus, daß die Kodierungslänge $size(x)$ einer Zahl $x = \frac{p}{q} \in \mathbb{Q}, p, q \in \mathbb{Z}$ in $O(\log |p| + \log |q|)$ ist.

Unser Algorithmus A soll nun eine Instanz y des Problems Π lösen. Die Anzahl der Speicherplätze die der Algorithmus benutzt, nennen wir den Speicherbedarf von A zur Lösung von y. Die *Laufzeit* von A zur Lösung von y ist die Anzahl der elementaren Operationen von A multipliziert mit der Kodierungslänge der bearbeiteten Daten. Die elementaren Operationen sind: Lesen, Schreiben, Addieren, Subtrahieren, Multiplizieren, Dividieren und Vergleichen von rationalen und ganzen Zahlen. Wir definieren die Laufzeit des Algorithmus A für die Probleminstanz y als die Funktion $f : \mathbb{N} \to \mathbb{N}$ mit

$$f(\sigma) = \max_{y, size(y)} (\text{Laufzeit des Algorithmus } A \text{ für die Instanz } y).$$

Da Lesen und Schreiben des Speichers zu den elementaren Operationen gehört, ist der Speicherbedarf von A in $O(\text{Laufzeit})$. Wir werden also nur die Zeitkomplexität von Algorithmen betrachten.

Wir sagen ein Algorithmus A hat eine polynomiale Zeitkomplexität oder kurz: ist *polynomial*, wenn ein $k \in \mathbb{N}$ existiert, so daß $f(\sigma) \in O(\sigma^k)$ liegt.

Ein Problem ist *polynomial lösbar*, wenn für dieses Problem ein polynomialer Algorithmus existiert.

Die Klasse der Probleme, die polynomial lösbar ist, bezeichnen wir mit $\mathcal{P}$. Eine andere, wahrscheinlich größere Klasse von Problemen ist die Klasse $\mathcal{NP}$. Dies steht für „nicht-deterministisch polynomial“. Ein Entscheidungsroblem gehört zur Klasse $\mathcal{NP}$, wenn für jede Instanz dieses Problems für die eine gültige Lösung existiert, ein Objekt Q existiert, mit dessen Hilfe die Existenz dieser Lösung mit einem polynomialen Algorithmus überprüft werden kann.

Offensichtlich gilt $\mathcal{P} \subset \mathcal{NP}$. Allgemein wird aber angenommen, daß $\mathcal{P} \neq \mathcal{NP}$ ist.

Ein Entscheindungsproblem Π heißt *$\mathcal{NP}$-vollständig*, wenn jedes andere Problem aus $\mathcal{NP}$ in polynomialer Zeit in Π transformiert werden kann.

Es existieren $\mathcal{NP}$-vollständige Probleme nach einem grundlegenden Satz von Cook (1971).

Satz A.2.1 (Cook's theorem). *Das Erfüllbarkeitsproblem und das 3-Erfüllbarkeitsproblem sind $\mathcal{NP}$-vollständig.*

Für die Definition von „Erfüllbarkeitsproblem“ und „3-Erfüllbarkeitsproblem“ siehe Abschnitt 2.1. Für einen Beweis des Satzes siehe [19] oder [26].

Wir nennen ein Problem $\mathcal{NP}$*-schwer*, wenn das dazugehörige Enscheidungsproblem $\mathcal{NP}$-vollständig ist.

Außer wenn $\mathcal{P} = \mathcal{NP}$ ist, existieren keine polynomialen Algorithmen für $\mathcal{NP}$-vollständige Probleme. Man muß also für diese Klasse von Problemen auf Näherungslösungen ausweichen oder Algorithmen finden, die zwar im Allgemeinen eine Laufzeit haben, die exponentiell zu der Eingabegrösse sind, aber für die zu lösenden Instanzen des Problems eine vertretbare Laufzeit haben.

A.3 Approximationsalgorithmen

Wie wir gesehen haben gibt es für manche Probleme (außer wenn $\mathcal{P} = \mathcal{NP}$ gilt) keine Algorithmen, die in polynomialer Zeit eine Lösung berechnen können. Wir betrachten nun *Optimierungs*-Probleme.

Sei $L \subset \Sigma^*$ die Menge der *gültigen Lösungen* eines Problems. Das heißt, ist ein Problem Π und eine Probleminstanz y dieses Problems gegegen, so gilt für alle $z \in L$, daß (y, z) in Π liegt.

Ein *Optimierungs-Problem* Π ist ein Problem zusammen mit einer reell-wertigen Funktion f auf der Menge der gültigen Lösungen L bezüglich der Instanz y. Gefragt ist eine gültige Lösung z mit maximalem oder minimalem Wert $f(z)$. Diese Lösung z nennen wir eine *Optimallösung*. Sei $f(\mathcal{OPT}_y)$ der Wert $f(z)$ einer Optimallösung der Instanz y.

Ein k-*Approximationsalgorithmus* für ein Optimierungs-Problem Π ist ein Algorithmus, der für jede Instanz y dieses Problems eine zulässige Lösung z'_y liefert, für deren Lösungswert $f(z'_y)$ gilt (bei Minimierung): Es existiert eine Konstante c, so daß $f(z'_y) \leqslant kf(\mathcal{OPT}_y) + c$ für alle Instanzen y. Bei Maximierung dreht sich das Ungleichungszeichen um.

Anhang B

Symbole

A	Bogenmenge eines Digraphen, bzw. Menge der Aufträge eines DARP oder cDARP	
c	Kostenfunktion	
c_{flow}	Flußschranke für das cDARP	in 2.5
D^+	Der Teilgraph von D der nur die Aufwärtsbögen enthält	in 2.7.1
D^-	Der Teilgraph von D der nur die Abwärtsbögen enthält	in 2.7.1
E	Kantenmenge eines Graphen	
f_A	Anzahl der Objekte in A die von einem Teilgraphen zu einem anderen Teilgraphen transportiert werden müssen	in 2.2
k	Kapazität des Aufzugs	
$L(u, v)$	der durch die Kanten, die in einem Weg von u nach v enthalten sind, induzierte Teilgraph eines Pfades P	in 2.2
m	Anzahl der Aufträge	
$\mathbb{N}$	Die Menge der ganzen Zahlen inklusive der Null	
n	Anzahl der Knoten in einem Graphen, bzw. Anzahl der Stockwerke	
o	Startknoten eines DARP oder cDARP	
$\mathbb{Q}$	Menge der rationalen Zahlen	
$\mathbb{R}$	Menge der reellen Zahlen	
(u, v, M)	Bewegung eines Aufzugs vom Knoten u zum Knoten v mit den Objekten M	in 1.4.1
V	Knotenmengen eines Graphen oder Digraphen	
$\mathbb{Z}$	Menge der ganzen Zahlen	

$\lceil a \rceil$	die kleinste ganze Zahl, die grösser als a ist	
$\lfloor a \rfloor$	die größte ganze Zahl, die kleiner als a ist	
$\lvert a \rvert$	Ist a eine Menge so bezeichnet $\lvert a \rvert$ die Anzahl der Elemente in dieser Menge, ansonsten ist $\lvert a \rvert = a$ für $a \geqslant 0$ und $\lvert a \rvert = -a$ für $a < 0$.	
$\delta(v)$	Grad des Knoten v	in A.1
$\delta^+(v)$	Anzahl der v verlassenden Bögen	in A.1
$\delta^-(v)$	Anzahl der in v mündenden Bögen	in A.1
λ_v	minimale Anzahl der Überquerungen der Kante $(v, v+1)$ durch den Aufzug in einem gültigen Transport	in 2.2
μ_v^+	Anzahl der Aufwärtsbögen, die die Kante $(v, v+1)$ überdecken	in 2.4.2
μ_v^-	Anzahl der Abwärtsbögen, die die Kante $(v, v+1)$ überdecken	in 2.4.2

Literaturverzeichnis

[1] S. Albers, *Competitive online algorithms*, OPTIMA **54** (1997), 2–8.

[2] N. Ascheuer, *Amsel—a modeling and simulation environment library (v 1.6)* [1], 1999.

[3] N. Ascheuer, M. Fischetti, and M. Grötschel, *Solving the asymmetric travelling salesman problem with time windows by branch-and-cut*, Preprint SC 99-31[2], Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

[4] N. Ascheuer, M. Grötschel, and Jörg Rambau, *Combinatorial online optimization in practice*, OPTIMA – Mathematical Programming Society Newsletter (1998), no. 57, 1–6.

[5] N. Ascheuer, S. O. Krumke, and J. Rambau, *Online dial-a-ride problems: Minimizing the completion time*, Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, 2000, To appear.

[6] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with application to minimizing robot arm travel*, SIAM Journal of Computing **5** (1988), no. 17, 849–869.

[7] Y. Bartal, *Probalistic approximations of metric spaces and its algorithmic applications*, Proc. 37th FOCS, 1996, pp. 184–193.

[8] ———, *On approximating arbitrary metrics by tree metrics*, Proc. 30th STOC, 1998, pp. 161–168.

[9] R. Borndörfer, M. Grötschel, F. Klostermeier, and Ch. Küttner, *Telebus berlin: Vehicle scheduling in a dial-a-ride system*, Proc. of the 7th Int. Workshop on Computer-Aided Scheduling of Public Transport, Center for Transp. Studies, MIT, Cambridge, MA, 1997, pp. 291–311.

[10] R. Borndörfer, M. Grötschel, and A. Löbel, *Optimization of transportation systems*, ACTA FORUM ENGELBERG 98 and The Future of Mobility & Transportation in a Moving World, VDF Hochschulverlag an der ETH Zürich, Schweiz, 1998.

[1] Avail. at URL `http://www.zib.de/optimization/index.de.html`

[2] Avail. at URL `http://www.zib.de/ZIBbib/Publications/`

[11] M. Charikar, S. Khuller, and B. Raghavachari, *Algorithms for capacitated vehicle routing*, Proceeding of the 30th Annual ACM Symposium on the Theory of Computing (STOC'98), ACM, 1998, pp. 349–358.

[12] M. Charikar and B. Raghavachari, *The finite capacity dial-a-ride problem*, Proceedings of the 39th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'98), 1998.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[14] G. Dantzig, D. Fulkerson, and S. Johnson, *Solution of a large-scale traveling-salesman problem*, Operations Research **2** (1954), 393–410.

[15] G. N. Frederickson and D. J. Guan, *Preemptive ensemble motion planning on a tree*, SIAM Journal of Computing **21** (1992), no. 6, 1130–1152.

[16] _____, *Nonpreemptive ensemble motion planning on a tree*, Journal of Algorithms **15** (1993), no. 1, 29–60.

[17] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, SIAM Journal of Computing **2** (1978), no. 7, 178–193.

[18] H. N. Gabow, Z. Galil, T. H. Spencer, and R. E. Tarjan, *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*, Combinatorica **6** (1986), 109–122.

[19] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness*, W. H. Freeman and Company, New York, 1979.

[20] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, SIAM Journal of applied mathematics **9** (1961), no. 4, 551–570.

[21] M. Grötschel, D. Hauptmeier, S. O. Krumke, and J. Rambau, *Simulation studies for the online dial-a-ride problem*, Preprint SC 99-09[3], Konrad-Zuse-Zentrum für Informationstechnik Berlin, March 1999.

[22] D. J. Guan, *Routing a vehicle of capacity greater then one*, Discrete Applied Mathematics **81** (1998), no. 1, 41–57.

[23] D. Gusfield, *Very simple methods for all pairs network flow analysis*, SIAM Journal of computing **19** (1990), no. 1, 143–155.

[24] D. Hauptmeier, *Online algorithms for industrial transport systems*, Diplomarbeit, Technische Universität Berlin, 1999, [4].

[25] D. Hauptmeier, S. O. Krumke, J. Rambau, and H.-C. Wirth, *Euler is standing in line*, Preprint SC 99-06, Konrad-Zuse-Zentrum für Informationstechnik Berlin, mar 1999.

[3] Avail. at URL `http://www.zib.de/ZIBbib/Publications/`

[4] Avail. at URL `http://www.zib.de/groetschel/students/diplomhauptm.pdf`

[26] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization*, Prentice-Hall, Inc., 1982.

[27] H. Poincaré, *Second complément à l'analysis situs*, Proceedings of the London Mathematical Society **32** (1900), 277–308.

[28] K. S. Ruland, *Polyhedral solution to the pickup and delivery problem*, Ph.D. thesis, Washington University, Sever Institute of Technology, Department of Systems Science and Mathematics, August 1995.

[29] M. Savelsbergh and M. Sol, *Drive: Dynamic routing of independent vehicles*, Tech. report, Georgia Institute of Technology and Eindhoven University of Technology, 1996.

[30] M. W. P. Savelsbergh and M. Sol, *The general pickup and delivery problem*, Transportation Science **29** (1995), 17–29.

[31] A. Schrijver, *Theory of linear and integer programming*, Wiley, 1998.

[32] Éric D. Taillard and Gilbert Laporte, *Vehicle routing with multiple use of vehicles*, Tech. Report CRT-95-19, Centre de recherche sur les transports, Universite' de Montreal, 1995.

[33] O. Veblen and Ph. Franklin, *On matrices whose elements are integers*, Annals of mathematics **23** (1921-2), 1–15.