Semi–Präemptives Transportieren

Inaugural–Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch–Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Dirk Räbiger
aus Köln

Köln 2005

Berichterstatter:	Prof. Dr. Rainer Schrader
	Prof. Dr. Sven O. Krumke

Tag der mündlichen Prüfung: 13. Juli 2005

Inhaltsverzeichnis

1. Einleitung **1**

2. Grundlagen **7**
2.1. Komplexitätstheorie 7
2.1.1. Algorithmen und Laufzeiten 7
2.1.2. Problemklassen 8
2.1.3. Approximation 9
2.2. Kombinatorische Optimierung 10
2.2.1. Graphentheorie 10
2.2.2. Matroidtheorie 15
2.3. Mathematische Programmierung 16

I. Theorie **19**

3. (d, r)–Arboreszenzen **21**
3.1. d–Bäume 21
3.2. Eigenschaften der (d, r)–Arboreszenz 25
3.2.1. Existenz einer (d, r)–Arboreszenz 29
3.2.2. Wurzelbeschränkte Graphen 29
3.2.3. Exakte r–Arboreszenz 30
3.3. Approximation 33
3.3.1. Approximative Paretokurven 33
3.3.2. Der linear diskrete Fall 37
3.3.3. Approximation von (d, r)–Arboreszenzen 41

4. Transportplanungs–Probleme **45**
4.1. Klassische Transportprobleme 45
4.2. Pickup–And–Delivery auf speziellen Graphenklassen . . . 49
4.2.1. Bisherige Arbeiten 49
4.2.2. Unser Beitrag 52

5. Semi–Präemptives Transportieren auf Pfaden und Kreisen **55**
5.1. Problembeschreibung 55
5.2. Zirkeltraining 56
5.3. Exogen definierte Umladeknoten 66
5.3.1. Arboreszenzen 66
5.3.2. Flußwerte 70
5.3.3. Kombinatorische Lösungsverfahren 74
5.4. Endogen zu bestimmende Umladeknoten 76
5.4.1. Gefärbte Arboreszenzen 77
5.4.2. Gefärbte Bäume 79
5.4.3. Spezielle Instanzen 81
5.4.4. Kombinatorisches Lösungsverfahren 85

6. Semi–Präemptives Transportieren auf Bäumen **99**
6.1. Eigenschaften 100
6.2. Komplexität 101
6.3. Approximation 105
6.3.1. Hilfsgraph 105
6.3.2. Steinerarboreszenzen 106
6.3.3. Exogenes Transportieren auf Bäumen 108
6.3.4. Endogenes Transportieren auf Bäumen 113
6.4. Anwendung auf allgemeine Metriken 119

II. Implementierung **123**

7. Details **125**
7.1. Dynamisches Programm 125
7.2. Gemischt–ganzzahlige Modellierung 126

8. Rechenergebnisse **131**
8.1. Generierung von Instanzen 131
8.2. Laufzeitergebnisse 134

Literaturverzeichnis **139**

Index **145**

Kapitel 1.

Einleitung

Am Anfang steht immer ein Problem. Der Mathematiker David Hilbert hielt 1900 auf dem Internationalen Mathematiker–Kongreß in Paris einen Vortrag, in dem er 23 wichtige Probleme des kommenden Jahrhunderts vorstellte, von denen bis heute einige ungelöst blieben. Er wies darauf hin, daß sich die Mathematik anfangs nur mit von außen gegebenen Problemen beschäftigte, aber schon bald aus der eigenen Theorie heraus künstliche Probleme generierte und zum Untersuchungsgegenstand machte. Tatsächlich hielt man 1974 ein Symposium, das die Fortschritte bei den Hilbert'schen Problemen zum Thema hatte und auf dem 23 weitere Probleme festgehalten wurden.

Hilberts Rede motivierte Alan Turing zu einem Rechenmaschinenkonzept, das bis heute der Theoretischen Informatik zugrunde liegt.

Der Inhalt dieser Arbeit bewegt sich eindeutig nicht in diesen Bedeutungsdimensionen. Trotzdem und immerhin stand am Anfang ein offenes Problem, das es zu lösen galt und auch nachts keine Ruhe gab. Und als endlich eine Lösung gefunden war, taten sich neue Probleme auf. Trotz aller Unsicherheiten während dieser Zeit möchte ich die Freude, die ich dabei empfand, nicht missen. Jetzt, wo sich die Fertigstellung dieser Arbeit abzeichnet, macht sich ein beklemmendes Gefühl bemerkbar, weil es bald gilt, loszulassen.

Im Zentrum des Problems, das in dieser Arbeit betrachtet werden soll, steht ein Roboter, der ein Rad zur Fortbewegung besitzt. Dieses Rad ist auf einer Schiene befestigt, die im *Kreis* verläuft. Entlang dieser Schienen sind Gegenstände positioniert und der Roboter erhält die Aufgabe,

die Objekte von ihrem Ursprungsort zu einem vorher festgelegten Zielort entlang des Kreises zu transportieren. Die dabei zurückgelegte Fahrstrecke soll minimiert werden.

Wenn man sich von dem Bild des Einrad–fahrenden Roboters löst und statt dessen einen Greifarm annimmt, der in der Mitte dieses Kreises befestigt ist und sich entsprechend des Kreises drehen kann, gibt es zu diesem Modell Anwendungen in der Industrie, beispielsweise bei der automatischen Sortierung von Produkten. Die Vorstellung des Roboters macht es aber einfacher, sich eine Verallgemeinerung des Problems vorzustellen. Die zu transportierenden Gegenstände könnten nicht auf einem Kreis, sondern in komplexeren Strukturen angeordnet sein. Wir werden sehen, daß dieses Problem im allgemeinen sehr schwierig zu lösen ist. Neben dem Kreis werden wir noch eine Anordnung auf einem *Pfad* und *Baum* untersuchen. Der Pfad kann als Spezialfall des Kreises betrachtet werden, da ein Pfad ein Kreis ist, der an einer Stelle aufgetrennt wurde. Der Baum ist eine Struktur, die keine Kreise enthält, wodurch eindeutige Wege zwischen je zwei Punkten existieren. Obwohl das Problem auf Bäumen schwierig zu behandeln ist, werden wir diese Eigenschaft ausnutzen, um zumindest einigermaßen gute Lösungen zu produzieren.

Das gerade beschriebene Problem wurde schon vor dieser Arbeit untersucht, insbesondere von Mikhail J. Atallah und S. Rao Kosaraju, deren Artikel von 1988 mich zu der folgenden Verallgemeinerung motivierte. Während der Roboter ein Objekt in seiner Roboterhand hält und von einem Punkt zu einem anderen fährt, passiert er weitere Objekte, die noch bewegt werden müssen. Eventuell ist es in Hinsicht auf die Minimierung der Fahrstrecke günstig, das aktuell gehaltene Objekt abzulegen, dann einen anderen Gegenstand zu befördern und schließlich das vorige Objekt weiter zu transportieren.

Die beiden Autoren betrachteten die Fälle, daß der Roboter entweder an allen beliebigen Orten des Kreises seine Fahrt unterbrechen darf oder an keinem. In dem ersten Fall legt der Roboter ein Objekt vorzeitig ab. Dieses Verhalten nennen wir *präemptiv*, während die zweite Variante *nicht–präemptiv* heißt. In dieser Arbeit werden diese beiden extremen Varianten in einem übergreifenden Konzept, das wir *semi–präemptiv* nennen, aufgelöst: Der Roboter darf an einigen Stellen des Kreises einen Transport

unterbrechen. Diese Umladestellen können *exogen* vorgegeben sein, indem man dem Roboter konkret die Orte benennt. Oder man teilt dem Roboter eine natürliche Zahl k mit und erlaubt ihm, daß er eigenständig k Stellen sucht. Diese zweite *endogene* Version verlagert das Auffinden von günstigen Umladeorten in die Problemstellung hinein.

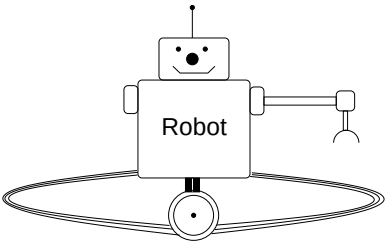
Auf den folgenden Seiten werden wir erfahren, daß die exogene Problemvariante etwas leichter zu modellieren ist als die endogene. Im Sinne der Komplexitätstheorie verhalten sich beide Typen sehr ähnlich. Auf den von uns betrachteten Strukturen sind entweder beide effizient zu lösen oder beide schwierig.

Aus dem Transportproblem ergibt sich als Teilproblem eine Fragestellung aus der Graphentheorie. Dieses werden wir zuerst betrachten, bevor auf das Roboterproblem eingegangen wird. Die Untersuchung wird uns helfen, das Transportproblem auf Bäumen näherungsweise zu lösen.

Schönen Dank!

Ich möchte Prof. Dr. Faigle und Prof. Dr. Schrader danken, daß ich in ihrer Arbeitsgruppe arbeiten durfte. Sie ermöglichten, daß diese Arbeit entstehen konnte. Prof. Dr. Schrader und Prof. Dr. Krumke möchte ich außerdem für viele anregende Gespräche danken.

Viel Dank gebührt der Schar, die sich die Mühe machte, Teile dieser Arbeit zu lesen und mich auf Fehler hinzuweisen. Während meiner Zeit am Institut habe ich viele nette Kollegen, und unendlich wichtiger, *Freunde* gefunden. Ich danke und grüße das Oberstübchen mit Katja Korherr, Bernhard Fuchs, Christian Hagemeier, Britta Wienand und Dominique Andres. Ich denke an Petra „Racepeti“ Fakler, Dr. Greed und Stefan Neuhaus. Ich werde euch täglich vermissen.



Meine Mutter Brunhild, mein Vater Siegfried und mein Bruder Sven haben mir in so vielfältiger Hinsicht ein Leben ermöglicht, daß man mehrere Seiten mit Danksagungen füllen könnte. Danke!

Ganz besonderen Dank gebührt meiner Freundin Nicole Gawron, die mir jeden Tag Liebe schenkt. Ich liebe Dich!

pizzeria über menaggio · vierhändig mit britta · colotec · dösen in grenoble · hintertux · espresso · kirsch–quark–plunder (kqp) · franzen · der gelbe wahnfried · fahrrad touren am rhein · auf berge wandern · cottbus · die lärmstange · oberursel · klagenfurt · machine from hell · milchschnaps · k2 · pistenturbo · vollplaybacktheater · recski · sun–ray · altdeutscher apfelkuchen · lappen grillen · skifahren · crevettenbrötchen am meer · pablo · abendmahl mit jüngern · nebelmaschine · berkheim · salute · die gefrorene wand · tapas · hörspiele · grappa · heizmeier · der herr andres · mexikanisches rührei · schloßhotel kronberg · zelten in holland · return to castle wolfenstein · thailändisch essen · betrunken chair-en · sülzer obststreusel · golf verschrotten · cor{n,r}etti–mann · schnipselraten · piösmes · gemischter–beerenstreusel (gbs) · venedig · madseit · oberstübchen · dampfkessel · andreas+ · segeln · nonnenwerth · gogo–gadgeto robot–arm · der racer · cocktails · kaiserslautern · dienstagsseminar · zelten in konstanz · geröllwandern · joggerknödel · doom · indisch essen · wörthersee · duschbier · frico · keks · pfauenkostüm · flachau · schälchen auseinander! · stefan, der pechvogel · civilization · superberni · golf spielen · jana im comicaze · herr epping · basketball · indonesisch essen · richtööch! · weinprobe zu tous sanctifier · winterberg · r35urr3ct10n 3v1l · zelten in überlingen · ground zero · cocolores · comer see · die lustige mensafrau · picknick am fluß · möppis hütte · frühstücken · italienisch essen · zelten in der eifel · knurrhahn · jourfixe–punkte · das loch von louvain–la–neuf

Kapitel 2.

Grundlagen

2.1. Komplexitätstheorie

Im Kern der Informatik stehen Probleme und Methoden, um diese zu lösen. In diesem Kapitel werden wir konkretisieren, was wir darunter verstehen und außerdem versuchen, beide zu klassifizieren. Wir halten uns inhaltlich und sprachlich an Begriffe, die sich in der Theoretischen Informatik durchgesetzt haben [GJ79], [Weg93].

2.1.1. Algorithmen und Laufzeiten

Wir legen das Modell der *deterministischen Turingmaschine* von Turing bzw. das der *Registermaschinen* zugrunde. Bis auf die Annahme eines unendlichen Speichers entsprechen diese Modelle in ihren Möglichkeiten den heute verwendeten Von–Neumann–Architekturen. Eine *Sprache* ist eine endliche Menge von $\{0,1\}$–Folgen. Zu einer Sprache $L \subseteq \{0,1\}^*$ betrachten wir das *Entscheidungsproblem*, ob eine gegebene $\{0,1\}$–Folge x in L liegt. L ist somit die Menge aller Lösungen. Häufig liegt L nicht explizit vor, sondern wird durch eine Fragestellung und einer *Eingabe* I, die die Parameter festlegt, beschrieben. Ein Problem mit konkreter Parameterausprägung nennen wir *Instanz*.

Zu einer gegebenen Eingabe I mit Eingabelänge $|I|$ werden Laufzeiten nach den gebräuchlichen $\mathcal{O}$–, Ω– und Θ–Notationen analysiert [OW96]. Wir sprechen bei einem Algorithmus mit Eingabe I von *polynomieller*

Laufzeit, wenn wir ein Polynom in $|I|$ angeben können, das die Anzahl der Rechenschritte begrenzt. Ein Algorithmus heißt *pseudopolynomiell*, wenn sich seine Rechenzeit durch ein Polynom in $|I|$ und b begrenzen läßt, wobei b die größte in I auftretende Zahl ist. Ein Problem heißt *Zahlproblem*, wenn sich b durch kein Polynom in $|I|$ beschränken läßt.

Für eine Funktion f bezeichnen wir mit $\mathrm{DTIME}(f)$ die Menge aller Entscheidungsprobleme Π, deren Instanzen I auf einer deterministischen Turingmaschine in $\mathcal{O}(f(|I|))$ Zeitschritten lösbar sind.

2.1.2. Problemklassen

In der Komplexitätstheorie, die sich auf die Definition der Klassen $\mathcal{P}$, $\mathcal{NP}$ und $\mathcal{NP}$–vollständig im Sinne von Cook, Karp und Knuth bezieht, werden allein *Entscheidungsprobleme* betrachtet, die als Antwort ein *ja* oder ein *nein* erwarten. $\mathcal{P}$ ist die Klasse aller Entscheidungsprobleme, die in polynomieller Zeit entschieden werden können. $\mathcal{NP}$ ist die Menge aller Entscheidungsprobleme, für die es einen Algorithmus mit exponentieller Laufzeit gibt. Seien Π_1 und Π_2 zwei Entscheidungsprobleme. Dann heißt Π_1 *polynomiell reduzierbar auf* Π_2 (Notation $\Pi_1 \leq_p \Pi_2$), falls es eine in polynomieller Zeit berechenbare Funktion f gibt, so daß π Lösung von Π_1 ist, genau dann wenn $f(\pi)$ Lösung von Π_2 ist. Ein Problem Π_1 heißt $\mathcal{NP}$–vollständig, wenn $\Pi_1 \in \mathcal{NP}$ ist und für alle $\Pi_2 \in \mathcal{NP}$ gilt: $\Pi_2 \leq_p \Pi_1$. Wir fassen in dieser Arbeit die $\mathcal{NP}$–vollständigen Probleme als diejenigen auf, die vermutlich schwierig zu lösen sind.

Wir werden im Laufe dieser Arbeit auch *Optimierungsprobleme* betrachten, etwa: *Gegeben eine Menge von Zahlen $Z \subseteq \mathbb{Z}$, bestimme die kleinste.* Jedes von uns betrachtete Optimierungsproblem läßt sich leicht in ein Entscheidungsproblem überführen: *Gegeben eine Menge von Zahlen $Z \subseteq \mathbb{Z}$ und eine Zahl $z \in \mathbb{Z}$. Existiert in Z eine Zahl kleiner als z?* Aus diesem Grund werden wir manchmal auch Optimierungsprobleme als $\mathcal{NP}$–vollständig bezeichnen und meinen dann das zugehörige Entscheidungsproblem. Streng genommen sollten wir bei Optimierungsproblemen von den Klassen $\mathcal{PO}$ und $\mathcal{NPO}$ sprechen, verweisen für eine Definition dieser Klassen jedoch auf [ACG+99].

Sei I eine Eingabe für ein Problem Π und $|I|$ sei die Kodierungslänge. Ein Problem Π heißt *Zahlproblem*, wenn sich die Kodierungslänge der größten in I vorkommenden Zahl durch kein Polynom in $|I|$ beschränken läßt. Π heißt *stark $\mathcal{NP}$–vollständig*, falls Π für in der Eingabelänge polynomiell beschränkte Instanzen $\mathcal{NP}$–vollständig ist. Solche Probleme sind offensichtlich keine Zahlprobleme. Alle Probleme, die $\mathcal{NP}$–vollständig sind, aber nicht im starken Sinne, heißen *schwach $\mathcal{NP}$–vollständig*.

2.1.3. Approximation

Im Falle von $\mathcal{P} \neq \mathcal{NP}$ gibt es für $\mathcal{NP}$–vollständige Probleme keine in diesem Sinne effizienten Algorithmen. Wir werden solche Probleme in ihrer Optimierungsvariante untersuchen und Algorithmen betrachten, denen wir nicht das Aufspüren einer optimalen Lösung unterstellen wollen. Vielmehr werden wir abschätzen, inwieweit die von ihnen produzierte Lösung unter ungünstigsten Bedingungen von der einer Optimallösung entfernt ist. Dieses Verhältnis bezeichnen wir als *Güte* einer Lösung bzw. des Algorithmus. Hier verlassen wir dann endgültig die Entscheidungsprobleme und wenden uns den Optimierungsvarianten zu, weil wir die Kosten von generierten Lösungen untersuchen.

Sei Π ein Minimierungsproblem und D_Π die Menge aller Instanzen von Π. Außerdem sei f eine Kostenfunktion, die jeder *Lösung* $\pi(I)$ für eine Instanz $I \in D_\Pi$ einen Wert $f(\pi(I)) \in \mathbb{R}$ zuordnet. Dann ist eine *Optimallösung* $\pi^*(I)$ eine Lösung mit minimalem Kostenwert $f(\pi^*(I))$. Sei A ein Algorithmus, dann ist die *Güte einer Lösung* $A(I)$ für eine Instanz I eines Minimierungsproblems definiert als das Verhältnis

$$R_A(I) = \frac{f(A(I))}{f(\pi^*(I))}$$

und bei einem Maximierungsproblem betrachten wir den Kehrwert. Wir beschreiben die *Güte* R_A *eines Approximationsalgorithmus* A als

$$R_A = \inf\{r \geq 1 | R_A(I) \leq r \text{ für alle } I \in D_\Pi\},$$

d.h. R_A beschreibt den Faktor, um den der Zielfunktionswert von A schlechter als der der Optimallösung ist, und zwar über alle mögliche

Instanzen betrachtet. Ein $(1+\varepsilon)$–*Approximationsalgorithmus* A garantiert für beliebige Instanzen eine Güte $R_A \leq 1+\varepsilon$.

Ein *polynomielles Approximationsschema* (PTAS, polynomial time approximation scheme) für ein Optimierungsproblem Π ist ein Algorithmus A, der auf Eingabe einer Instanz $I \in D_\Pi$ sowie einer Genauigkeitsvorgabe $\varepsilon > 0$ eine Lösung für Π mit Güte $1+\varepsilon$ berechnet und dessen Laufzeit polynomiell in $|I|$ beschränkt ist. Ein *voll polynomielles Approximationsschema* (FPTAS, fully polynomial time approximation scheme) berechnet zu I und $\varepsilon > 0$ eine Lösung mit Güte $1+\varepsilon$ und ist in seiner Laufzeit polynomiell in $|I|$ und ε^{-1} beschränkt. $\mathcal{APX}$ ist die Menge aller Optimierungsprobleme, so daß für ein $\varepsilon \geq 0$ ein $(1+\varepsilon)$–Approximationsalgorithmus mit polynomieller Laufzeit existiert.

2.2. Kombinatorische Optimierung

In diesem Abschnitt definieren wir die in der vorliegenden Arbeit verwendete Notation und Begriffe aus der Graphen– und Matroidtheorie. Immer, wenn wir von Mengen sprechen, meinen wir *endliche Mengen*. Insbesondere sind die von uns betrachteten Graphen endlich.

2.2.1. Graphentheorie

Wir unterscheiden zwischen ungerichteten und gerichteten Graphen.

ungerichtete Graphen: Ein *ungerichteter Graph* $G = (V, E)$ besteht aus der *Knotenmenge* V und der *Kantenmenge* $E \subseteq \mathcal{P}_2(V)$, wobei $\mathcal{P}_2(V)$ die zweielementige Potenzmenge von V ist. Eine *ungerichtete Kante* $e \in E$ wird als Menge $e = \{u, v\}$ notiert. Wir sagen, e ist zu u und v *inzident*.

Ein ungerichteter Graph $P = (\{v_1, \ldots, v_{k+1}\}, \{e_1, \ldots, e_k\})$ heißt *ungerichteter Kantenzug*, wobei $e_i = \{v_i, v_{i+1}\}, i = 1, \ldots, k$. Er heißt *geschlossen* oder *Kreis*, falls $v_1 = v_{k+1}$. Ein *ungerichteter Pfad* ist ein Kantenzug, in dem die Knoten paarweise verschieden sind.

Dabei heißt v_1 *Anfangs–* und v_{k+1} *Endknoten* von P. P heißt auch (v_1, v_{k+1})–Pfad. Ein *Hamilton–Kreis* ist ein Kreis, der alle Knoten des zugrunde liegenden Graphen genau einmal besucht. $G = (V, E)$ heißt *zusammenhängend*, falls es für alle $v, w \in V$ ein (v, w)–Pfad existiert. Teilmengen $V' \subseteq V$ heißen *Zusammenhangskomponenten von* G, falls für alle $v, w \in V'$ ein (v, w)–Pfad existiert. Bilden einzelne Knoten eine Zusammenhangskomponente, so nennen wir sie auch *isoliert*.

gerichtete Graphen: Ein *gerichteter Graph* $D = (V, A)$ besteht aus der *Knotenmenge* V und der Kantenmenge $A \subseteq V \times V$. Eine *gerichtete Kante* $e \in A$ wird mit dem Tupel $e = (u, v)$ beschrieben. In diesem Fall nennen wir u den *Schwanz* (bzw. *Anfangsknoten*) und v den *Kopf* (*Endknoten*) von e. Auch im gerichteten Fall nennen wir e zu u und v *inzident*. Mit dem D *zugrunde liegenden ungerichteten Graphen* bezeichnen wir den Graphen, der entsteht, wenn in A die Ordnung aufgehoben wird.

Ein gerichteter Graph $P = (\{v_1, \ldots, v_{k+1}\}, \{e_1, \ldots, e_k\})$ heißt *gerichteter Kantenzug*, wobei $e_i = (v_i, v_{i+1}), i = 1, \ldots, k$. Ein *gerichteter Pfad* ist ein Kantenzug, in dem die Knoten v_i paarweise verschieden sind. Dabei heißt v_1 *Anfangs–* und v_{k+1} *Endknoten* von P. Ein *gerichteter Kreis* bzw. *Zyklus* ist ein geschlossener und gerichteter Kantenzug. $D = (V, A)$ heißt *schwach zusammenhängend*, falls der zugrunde liegende ungerichtete Graph zusammenhängend ist. D heißt *stark zusammenhängend*, falls für je zwei Knoten v, w ein gerichteter (v, w)–Pfad und ein gerichteter (w, v)–Pfad besteht.

Sei $G = (V, E)$ ein gerichteter oder ungerichteter Graph. Zwei Knoten mit einer gemeinsamen Kante heißen *adjazent* oder *benachbart*. Eine Kante ist zu ihrem Anfangs– und Endknoten *inzident*.

Falls $E = \mathcal{P}_2$ bzw. $A = V \times V$, so heißt G bzw. D *vollständig*. $G' = (V', E')$ ist ein *Teilgraph* von $G = (V, E)$, falls $V' \subseteq V$, $E' \subseteq E$ und V' alle Knoten enthält, die zu einer Kante $e \in E'$ inzident sind. Sei $E' \subseteq E$ gegeben. Mit $V(E')$ bezeichnen wir die durch E' induzierten Knoten. Falls $V(E') = V$, *spannt* E' den Graphen G' *auf*.

Ein geschlossener und gerichteter Kantenzug in G, der alle Kanten des Graphen G enthält, heißt *eulersch*. Ein schwach zusammenhängender Graph heißt *eulersch*, wenn er einen solchen Kantenzug enthält.

Für einen ungerichteten Graphen $G = (V, E)$ und $v \in V$ sei der *Grad* $\delta(v)$ die Anzahl der zu v inzidenten Kanten. Für gerichtete Graphen definieren wir den *Außengrad* $\delta^+(v)$ (bzw. *Innengrad* $\delta^-(v)$) als die Anzahl der Kanten in G, die v als Schwanz (bzw. Kopf) besitzen. Falls für einen Knoten $v \in V$ der Außengrad gleich dem Innengrad ist, so heißt v *gradbalanciert*. Ein Knoten mit (Innen–)grad eins heißt *Blatt*. Für eine Knotenmenge $X \subseteq V$ ist $\delta^+(X) := |\{(x, y) \in E : x \in X, y \in V \setminus X\}|$ und $\delta^-(X) := |\{(y, x) \in E : x \in X, y \in V \setminus X\}|$.

Lemma 2.1 (Euler 1736 [Die96]). *Ein schwach zusammenhängender gerichteter Graph* $G = (V, E)$ *ist genau dann eulersch, wenn* $\delta^-(v) = \delta^+(v)$ *für alle* $v \in V$.

Ein ungerichteter Graph $G = (V, E)$ heißt *bipartit*, wenn sich V in zwei disjunkte Teilmengen V_1, V_2 partitionieren läßt, so daß $V_1 \cap V_2 = \emptyset, V = V_1 \dot\cup V_2$ und die durch V_1 und V_2 induzierten Teilgraphen leer sind.

Wir belegen häufig die Kanten eines Graphen $G = (V, E)$ mit Kosten, indem wir eine irrationale *Kostenfunktion* $c : E \to \mathbb{R}$ auf den Kanten definieren. Für eine Kantenmenge $E' \subseteq E$ definieren wir die Kosten als $c(E') = \sum_{e \in E'} c(e)$.

In späteren Kapiteln werden wir Graphen konstruieren oder bestehende Graphen modifizieren. Ausgehend von einem Basisgraphen $G = (V, E)$ werden wir dann sowohl die Knotenmenge als auch die Kantenmenge *erweitern* oder *reduzieren*. Beim Hinzufügen einer Kante e zu E entsteht eine neue Menge $E' = E \cup \{e\}$. In Algorithmen wird die Kantenmenge eines Graphen schrittweise konstruiert werden. Dabei nennen wir die neu entstehende Menge E' erneut E, um die Anzahl benutzter Indizes zu verringern. Wenn aus dem Kontext eindeutig hervor geht, ob e ein Knoten oder eine Kante ist, schreiben wir auch abkürzend $G + e$ für das Hinzufügen bzw. $G - e$ für das Löschen eines Elements.

Starker Zusammenhang

Die Algorithmen und Datenstrukturen in dieser Arbeit nutzen Eigenschaften von starkem Zusammenhang in gerichteten Teilgraphen intensiv aus. Das folgende Lemma charakterisiert stark zusammenhängende Graphen über die Lage von Kanten. Darauf aufbauend beweisen wir eine hinreichende Bedingung für starken Zusammenhang, die das Verhältnis von Innen– und Außengrad der Knoten ausnutzt.

Lemma 2.2 (s. [KV02], S. 19). *Sei* G *ein gerichteter Graph.* G *ist genau dann stark zusammenhängend, wenn* G *schwach zusammenhängend ist und jede Kante in* G *auf einem gerichteten Kreis liegt.*

Lemma 2.3. *Sei* $G = (V, A)$ *ein gerichteter Graph. Falls* $\delta^-(v) = \delta^+(v)$ *für alle* $v \in V$ *gilt, dann sind alle schwachen Zusammenhangskomponenten auch stark zusammenhängend.*

Beweis. Sei $Z \subseteq V$ eine schwache Zusammenhangskomponente von G und es gelte $\delta^-(v) = \delta^+(v)$ für alle $v \in Z$. Dann ist der Teilgraph G_Z, der durch Z induziert wird, nach Lemma 2.1 eulersch und alle Kanten liegen auf einem gerichteten Kreis. Mit Lemma 2.2 ist G_Z stark zusammenhängend. □

Multigraphen

In den oben definierten Konzepten von ungerichteten und gerichteten Graphen gibt es keine parallelen Kanten. In einem (gerichteten oder ungerichteten) *Multigraph* sind parallele Kanten erlaubt.

Bäume und Arboreszenzen

Ein *Wald* ist ein kreisfreier ungerichteter Graph. Ein *Baum* ist ein zusammenhängender Wald. Ein gerichteter Graph $G = (V, E)$ ist ein *Branching*, falls der zugrunde liegende ungerichtete Graph ein Wald ist und

jeder Knoten maximal eine eingehende Kante besitzt, d.h. $\delta^-(v) \leq 1$ für alle $v \in V$. Ein zusammenhängendes Branching ist eine *Arboreszenz*.

Da wir uns im Laufe dieser Arbeit intensiv mit speziellen Arboreszenzen auseinander setzen werden, geben wir alternative Definitionen von Arboreszenzen an, die alle äquivalent sind.

Lemma 2.4 (s. [KV02] S. 18). *Sei $G = (V, E)$ ein gerichteter Graph mit n Knoten. Die folgenden Aussagen sind äquivalent.*

(i) G *ist eine Arboreszenz mit* Wurzel $r \in V$.

(ii) G *ist ein Branching mit* $n - 1$ *Kanten und* $\delta^-(r) = 0$.

(iii) G *besitzt* $n-1$ *Kanten und für alle* $v \in V$ *existiert ein gerichteter* (r, v)*–Pfad.*

(iv) G *ist ein minimaler Graph mit* $\delta^+(X) \geq 1$ *für alle* $X \subsetneq V$ *mit* $r \in X$.

Die letzte Definition betrachtet alle Teilmengen $X \subset V$, die die Wurzel r enthalten. Für ein solches X nennen wir die Kantenmenge $X^+ := \{(u, v) \in E | u \in X, v \notin X\}$ einen r*–Schnitt*.

Problem 2.5 (kostenminimale r–Arboreszenz)

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit Wurzel $r \in V$. Eine Kantenbewertung $c : E \to \mathbb{R}$.

Gesucht: Eine kostenminimale aufspannende Arboreszenz mit Wurzel r.

Eine kostenminimale r–Arboreszenz kann effizient durch den Algorithmus von Chu und Liu [CL65] oder den unabhängig entwickelten Algorithmus von Edmonds für Branchings [Edm67] bestimmt werden. Mittels einer Implementation über Fibonacci Heaps kann die Laufzeit optimiert werden.

Satz 2.6 (Gabow et al. 1986 [GGST86]). *Es existiert ein Algorithmus zur Bestimmung einer minimalen r–Arboreszenz, der als Datenstruktur Fibonacci Heaps benutzt, mit einer Laufzeit von $\mathcal{O}(m + n \log n)$.*

Bemerkung 2.7. *Wir notieren Teilgraphen eines Graphen* $G = (V, E)$ *wie z.B. Bäume und Arboreszenzen häufig nur mit ihrer Kantenmenge* $T \subseteq E$ *und meinen damit den durch* T *induzierten Teilgraphen* $G' = (V(T), T)$.

2.2.2. Matroidtheorie

Ein Matroid $\mathcal{M} = (E, \mathcal{I})$ besteht aus einer endlichen *Grundmenge* E und einer Menge $\mathcal{I}$ von Teilmengen von E, die man *unabhängige* Mengen nennt. Sie erfüllen die Eigenschaften (M1) bis (M3).

(M1) $\emptyset \in \mathcal{I}$.

(M2) Falls $X \in \mathcal{I}$ und $Y \subseteq X$ ist, dann gilt $Y \in \mathcal{I}$.

(M3) Falls $X \in \mathcal{I}$ und $Y \in \mathcal{I}$ mit $|X| = |Y| + 1$, dann existiert ein $x \in X \setminus Y$, so daß $Y \cup \{x\} \in \mathcal{I}$ ist.

Eine Teilmenge, die nicht in $\mathcal{I}$ liegt, heißt *abhängig*. Eine *Basis* von $\mathcal{M}$ ist eine maximale unabhängige Menge. Die *Menge aller Basen* von $\mathcal{M}$ bezeichnen wir mit $\mathcal{B}(\mathcal{M})$. Ein *Kreis* ist eine minimale abhängige Menge. Matroide lassen sich auf viele äquivalente Arten beschreiben, z.B. auch über ihre Kreise. Für eine ausführliche Behandlung des Themas verweisen wir auf [Wel76] oder [Oxl92]. Statt dessen stellen wir spezielle Mengensysteme vor, die als Matroide bekannt sind [Law76]:

Graphisches Matroid Sei $G = (V, E)$ ein ungerichteter Graph und $\mathcal{I}$ die Menge aller Wälder in G. Dann beschreibt $(E, \mathcal{I})$ ein Matroid.

Partitionsmatroid Sei E eine endliche Menge, $d_1, \ldots, d_p \in \mathbb{N}$ und $E_1, \ldots, E_p$ eine Partitionierung von E, d.h.

$$E = \bigcup_{i=1}^{p} E_i \text{ und } \bigcap_{i=1}^{p} E_i = \emptyset.$$

Dann ist $(E, \mathcal{I})$ ein Matroid, wobei $\mathcal{I} = \{I \subseteq E : |I \cap E_i| \leq d_i, i = 1, \ldots, p\}$.

Die Eigenschaft (M3) sorgt dafür, daß eine kleinere unabhängige Menge stets durch ein Element einer größeren unabhängigen Menge erweitert werden kann. Für Basen existiert eine ähnliche Aussage:

Proposition 2.8 (Symmetrischer Basisaustausch [Whi86]). *Seien* B_1 *und* B_2 *Basen. Dann existiert für jedes Element* $x \in B_1$ *ein Element* $y \in B_2$*, so daß* $(B_1 - x) + y$ *und* $(B_2 - y) + x$ *beides Basen sind.*

Algorithmus 1 Gewichtsgreedy$(\mathcal{M} = (E, \mathcal{I}), c)$

Initialisiere $B := \emptyset$
while $T := \{x \in E \setminus B | B \cup \{x\} \in \mathcal{I}\} \neq \emptyset$ **do**
  Wähle ein $x \in T$ mit $c(x) \leq c(y)$ für alle $y \in T$.
  Erweitere $B := B \cup \{x\}$.
end while

Wir können über Matroide mit einem der einfachsten Algorithmen optimieren.

Satz 2.9 (Gewichtsgreedy für Matroide [Law76]). *Sei* $\mathcal{M} = (E, \mathcal{I})$ *ein Matroid und* $c : E \to \mathbb{R}_+$ *eine Gewichtsfunktion. Dann bestimmt der Greedy–Algorithmus eine Basis von* $\mathcal{M}$ *mit kostenminimalem Gewicht.*

2.3. Mathematische Programmierung

Wir betrachten lineare und ganzzahlige Programmierung nur am Rande dieser Arbeit. Aus diesem Grund stehen an dieser Stelle nur die wirklich notwendigen Aussagen, auf die wir uns stützen werden. Eine ausführliche Einführung in diesen Bereich findet man in [Sch98] und [KV02].

Problem 2.10 (Lineare Programmierung)

Gegeben: Eine Matrix $A \in \mathbb{R}^{m \times n}$ und Vektoren $b \in \mathbb{R}^m, c \in \mathbb{R}^n$.

Gesucht: Ein Lösungsvektor $x \in \mathbb{R}^n$, so daß $Ax \geq b$ gilt und cx minimal ist, oder entscheide, daß $\{x \in \mathbb{R}^n | Ax \geq b\}$ leer oder unbeschränkt bzgl. cx ist.

Proposition 2.11 (Khachiyan 1979 [Kha79]). *Das Problem der Linearen Programmierung läßt sich mittels der* Ellipsoidmethode *mit polynomiellem Zeitaufwand lösen.*

Problem 2.12 (Ganzzahlige Programmierung)
Gegeben: Eine Matrix $A \in \mathbb{R}^{m \times n}$ und Vektoren $b \in \mathbb{R}^m, c \in \mathbb{R}^n$.

Gesucht: Ein Lösungsvektor $x \in \mathbb{Z}^n$, so daß $Ax \geq b$ gilt und cx minimal ist.

Das Problem der Ganzzahligen Programmierung ist $\mathcal{NP}$–vollständig und viele der schwierigen Probleme lassen sich als ganzzahliges Programm beschreiben.

Proposition 2.13 (Dantzig 1951 [Dan51]). *Falls eine Optimallösung des Ganzzahligen–Programmierung–Problems existiert, findet das* Simplexverfahren *eine solche und terminiert nach spätestens* $\binom{m}{n}$ *Iterationen.*

Gemischt–ganzzahlige Programme sind lineare Programme, bei denen nur für einen Teil der Variablen Ganzzahligkeit gefordert ist.

Teil I.

Theorie

Kapitel 3.

(d, r)–Arboreszenzen

In diesem Kapitel betrachten wir eine graphentheoretische Problemstellung, die als Teilproblem in das später beschriebene Transportproblem eingeht. Wir untersuchen zunächst den einfacheren, ungerichteten Fall in Abschnitt 3.1. Erst danach wenden wir uns (d, r)–Arboreszenzen in Abschnitt 3.2 zu. Leider werden wir kein polynomielles Verfahren angeben können, um kostenminimale (d, r)–Arboreszenzen zu konstruieren. In Abschnitt 3.3 untersuchen wir das Problem hinsichtlich einer Approximation und präsentieren zumindest ein voll polynomielles Approximationsschema sowie einen pseudopolynomiellen Algorithmus.

3.1. d–Bäume

Für diesen Abschnitt sei $G = (V, E^r \dot\cup E^b)$ ein ungerichteter Graph, dessen Kantenmenge disjunkt in zwei Teilmengen partitioniert ist, sagen wir in rote Kanten E^r und blaue Kanten E^b. Jeder Kante $e \in E^r \dot\cup E^b$ ist ein Gewicht $c : (E^r \dot\cup E^b) \to \mathbb{R}$ zugeordnet. Wir beschäftigen uns mit dem Problem, einen kostenminimalen aufspannenden Baum auf G zu bestimmen, der maximal d blaue Kanten enthält, wobei d eine natürliche Zahl ist.

Definition 3.1. *Sei* $G = (V, E^r \dot\cup E^b)$ *ein ungerichteter Multigraph,* $c : (E^r \dot\cup E^b) \to \mathbb{R}$ *eine Gewichtsfunktion auf den Kanten. Ein* d–Baum *ist ein aufspannender Baum* T *auf* G *mit* $|T \cap E^b| \leq d$.

Problem 3.2 (kostenminimaler d–Baum)
Gegeben: Ein ungerichteter Multigraph $G = (V, E^r \dot\cup E^b)$, $c : (E^r \dot\cup E^b) \to \mathbb{R}$, $d \in \mathbb{N}$.
Gesucht: Ein d–Baum T mit $\sum_{e \in T} c(e)$ minimal.

Wir interessieren uns für spezielle Bäume auf Graphen. Matroide sind eine sehr gut untersuchte Verallgemeinerung dieser Problemstellung, und die Matroidtheorie stellt uns Eigenschaften zur Verfügung, die die Argumentation bei der Beweisführung einfach macht.

Die Familie der d–Bäume liegt im Schnitt zweier Matroide $\mathcal{M}_1$ und $\mathcal{M}_2$. Dabei ist $\mathcal{M}_1$ ein graphisches Matroid auf der Kantenmenge $E := (E^r \dot\cup E^b)$. $\mathcal{M}_2$ ist ein Partitionsmatroid, definiert durch die Färbung der Kanten. Im Schnitt $\mathcal{M}_1 \cap \mathcal{M}_2$ liegen die Bäume mit einer beschränkten Anzahl blauer Kanten. Da ein polynomieller Algorithmus für das gewichtete Matroidschnittproblem existiert [Law76], hätten wir ein Lösungverfahren für das Problem, einen kostenminimalen d–Baum zu bestimmen, zur Hand. Allerdings nutzt dieses allgemeine Verfahren nicht die spezielle Struktur von $\mathcal{M}_1$ und $\mathcal{M}_2$ aus und hat eine dementsprechend schlechte Laufzeit.

Wir werden im folgenden einen Algorithmus angeben, der sehr einfach zu implementieren ist und erwartet, daß $\mathcal{M}_1 = (E, \mathcal{I}_1)$ ein Matroid und $\mathcal{M}_2 = (E, \mathcal{I}_2)$ ein spezielles Partitionsmatroid ist, wobei die Grundmenge $E = E^r \dot\cup E^b$ in zwei Teilmengen partitioniert ist und

$$\mathcal{I}_2 := \{E' \subseteq E : |E' \cap E^b| \leq d\}$$

die unabhängigen Teilmengen von $\mathcal{M}_2$ beschreibt.

Algorithmus 2 berechnet zunächst eine kostenminimale Basis T für $\mathcal{M}_1$ mit möglichst wenig blauen Elementen. Danach fügen wir sukzessive dasjenige blaue Element ein, das die beste Verbesserung der Gesamtkosten erzielt. Sei $e \in E^b \setminus T$ ein blaues Element, dann existiert in $T + e$ ein eindeutiger Kreis, den wir durch das Entfernen eines Elements aus T zerstören können. Den zweiten Schritt wiederholen wir solange, bis maximal d blaue Elemente eingefügt wurden oder das Einfügen eines weiteren blauen

Elements keine Verbesserung bringt. Wir wenden also zweimal hintereinander die Idee des Greedy an. Zuerst bei der Konstruktion der Basis T von $\mathcal{M}_1$ und das zweite mal beim Einfügen der blauen Elemente.

Algorithmus 2 d–Baum–Greedy$(\mathcal{M}_1, \mathcal{M}_2, c, d)$

Initialisiere $c'(e) := \left\{ \begin{array}{ll} c(e), & e \in E^r \\ M + c(e), & e \in E^b \end{array} \right\}$
mit $M := r(\mathcal{M}_1) \max_{e \in E}\{c(e)\}$.
Berechne eine kostenminimale Basis T von $\mathcal{M}_1$.
$i = r(\mathcal{M}_1) - |T \cap E^r|$
if $d < i$ **then**
  Stop. Keine Lösung möglich.
else
  $T_i := T$
end if
while $i < d$ **do**
  $(e, f) := \operatorname{argmax}_{f \in E^b \setminus T_i}\{c(e) - c(f) > 0 : e \in C(T_i, f)\}$
  if (e, f) nicht existiert **then**
    Stop. T_i ist optimal.
  end if
  $T_{i+1} := T_i - e + f$, $i := i + 1$
end while

Satz 3.3 (Kloock et al. 2003 [KRS03]). *Algorithmus 2 berechnet einen kostenminimalen* d*–Baum.*

Beweis. Mit $\mathcal{B} = \mathcal{B}(\mathcal{M}_1)$ bezeichnen wir die Basen von $\mathcal{M}_1$. Dann ist

$$\mathcal{B}^i = \{B \in \mathcal{B} : |B \cap E^b| \leq i\} \text{ mit } 0 \leq i \leq |E^b|$$

die Menge der Basen von $\mathcal{M}_1$ mit höchstens i blauen Elementen. Für $B \in \mathcal{B}, e \in E \setminus B$ ist $C(B, e) = \{f \in B : B - f + e \in \mathcal{B}\}$ der Fundamentalkreis in $B + e$.

Falls der Algorithmus in der ersten if–Abfrage terminiert, so hat der gewöhnliche Greedy zur Bestimmung einer Basis von $\mathcal{M}_1$ keine Basis mit

höchstens d blauen Elementen bestimmen können. Somit ist der Schnitt $\mathcal{M}_1 \cap \mathcal{M}_2$ leer. Andernfalls ist T_i eine Basis von $\mathcal{M}_1$ und enthält i blaue Elemente. T_i ist aber auch optimal in $\mathcal{B}^i$, weil $\mathcal{B}^i \subseteq \mathcal{B}$. Wir zeigen nun induktiv über i, daß T_i optimal im jeweiligen $\mathcal{B}^i$ ist.

Wir nehmen an, daß T_i aus T_{i-1} entsprechend der in Algorithmus 2 beschriebenen Schritte entstanden ist. Es gilt, daß $e \in E^r$ ist, da wir uns sonst durch Austausch zweier blauer Kanten verbessern könnten, im Widerspruch zur Induktionsannahme, daß T_{i-1} optimal ist. Nun ist $e \in E^r$ aus T_{i-1} entfernt und durch $f \in E^b \setminus T_{i-1}$ ersetzt worden. T_{i-1} ist optimal in $\mathcal{B}^{i-1}$. Angenommen, es gäbe ein billigeres $T^* \in \mathcal{B}^i$, d.h. $c(T^*) < c(T_i)$. Unter diesen sei T^* so gewählt, daß $|T^* \cap T_{i-1} \cap E^b|$ maximal ist.

Sicherlich gilt, daß $|T_i \cap E^b| = i$ und $c(T_i) < c(T_{i-1})$. Dann muß auch $|T^* \cap E^b| = i$ gelten, denn ansonsten wäre $T^* \in \mathcal{B}^{i-1}$ und $c(T^*) < c(T_{i-1})$. Daher sind wir uns sicher, daß $(T^* \setminus T_{i-1}) \cap E^b \neq \emptyset$. Sei dann $f' \in (T^* \setminus T_{i-1}) \cap E^b$ ein solches blaues Element. Mit Proposition 2.8 existiert dann ein $e' \in T_{i-1} \setminus T^*$, so daß $T^* - f' + e' \in \mathcal{B}$ und $T_{i-1} - e' + f' \in \mathcal{B}$. Wir unterscheiden zwei Fälle.

(i) $e' \in (T_{i-1} \cap E^r)$: Es gilt, daß $T^* - f' + e' \in \mathcal{B}^{i-1}$. Somit ist:

$$\begin{aligned} c(T^*) - c(f') + c(e') &= c(T^* - f' + e') \\ &\geq c(T_{i-1}) = c(T_i) - c(f) + c(e) \\ &> c(T^*) - c(f) + c(e) \end{aligned}$$

im Widerspruch zur Wahl von e und f, denn $f' \in E^b \setminus T_{i-1}$ und $e' \in C(T_{i-1}, f')$.

(ii) $e' \in (T_{i-1} \cap E^b)$: Wir wissen, daß $c(T^*) \leq c(T^* - f' + e')$ ist. Da jedoch

$$|(T^* - f' + e') \cap T_{i-1} \cap E^b| = |(T^* \cap T_{i-1} \cap E^b) + e'| > |T^* \cap T_{i-1} \cap E^b|$$

ist, muß nach Wahl von T^* sogar $c(T^*) < c(T^* - f' + e')$ gelten. Wegen $c(T^* - f' + e') = c(T^*) - c(f') + c(e')$ ist also $c(f') < c(e')$. Da $e' \in C(T_{i-1}, f')$ und $f' \in (E^b \setminus T_{i-1})$, folgt daß $T_{i-1} - e' + f' \in \mathcal{B}^{i-1}$ und $c(T_{i-1} - e' + f') < c(T_{i-1})$ im Widerspruch zur Optimalität von T_{i-1}.

Wenn der Algorithmus terminiert, kann es zwei Gründe dafür geben. Entweder ist $i = d$, dann ist T_d wegen der gerade geführten Argumentation optimal. Falls es kein Paar (e, f) gibt, so daß $c(e) - c(f) > 0$ und $e \in C(T_i, f)$, und es existiert trotzdem ein billigeres $T^* \in \mathcal{B}^d$, so muß T^* mehr als i Elemente aus E^b enthalten, nämlich $i + j$ für ein $1 \leq j \leq d - i$. Durch induktive Anwendung der gerade geführten Argumentation über j wird T^* auf die gleiche Weise zum Widerspruch geführt. □

Wenn wir unterstellen, daß $\mathcal{M}_1$ ein graphisches Matroid eines rot–blau gefärbten Graphen G ist und $\mathcal{M}_2$ durch diese Partitionierung definiert ist, kann eine kostenminimale Basis von $\mathcal{M}_1$ in $\mathcal{O}(m \log^* n)$ Schritten bestimmt werden, indem man das Verfahren von Prim benutzt und als Datenstruktur Fibonacci–Heaps zugrunde legt [FT87]. Die Laufzeit des zweiten Teiles kann grob durch dm^2 Schritte abgeschätzt werden, so daß wir mit $d \leq r(\mathcal{M}_1) < m$ eine Gesamtlaufzeit von $\mathcal{O}(m^3 + m \log^* n)$ angeben können. Gabow und Tarjan haben einen weit aufwendiger zu analysierenden *Divide–And–Conquer* Ansatz zur Bestimmung eines kostenminimalen d–Baumes entwickelt, der eine deutlich bessere Laufzeit besitzt.

Satz 3.4 (Gabow, Tarjan 1984 [GT84]). *Sei $G = (V, E = (E^r \dot\cup E^b)$ ein ungerichteter Multigraph, $c : (E^r \dot\cup E^b) \to \mathbb{R}$ eine Kantenbewertung und $d \in \mathbb{N}$. Es existiert ein Algorithmus zur Konstruktion eines kostenminimalen d–Baumes mit einer Laufzeit von $\mathcal{O}(|E| \log |V| + |V| \log |V|)$ Schritten.*

3.2. Eigenschaften der (d, r)–Arboreszenz

Im vorhergehenden Abschnitt haben wir gefärbte aufspannende Bäume in ungerichteten Graphen betrachtet. Im folgenden werden wir gerichtete Graphen zugrunde legen und gefärbte aufspannende Arboreszenzen untersuchen. Wir werden das Problem aus zwei Perspektiven betrachten, zunächst aus der Sicht der Matroidtheorie und danach aus der Sicht der Linearen Programmierung.

Im Schnitt von Matroiden

Aufspannende Bäume auf ungerichteten Graphen sind sehr wohl untersuchte Strukturen. Aus der Sicht der Matroidtheorie sind dies genau die graphischen Matroide und einer der einfachsten Optimierungsalgorithmen, der *Greedy*–Algorithmus, findet immer einen aufspannenden Baum. Auch wenn wir jeder Kante des Baumes ein Gewicht zuweisen, arbeitet der Greedy zuverlässig.

Arboreszenzen sind Verallgemeinerungen von aufspannenden Bäumen und auf gerichteten Graphen definiert. Sie lassen sich als Schnitt zweier Matroide definieren, nämlich als Schnitt des graphischen Matroids mit einem Partitionsmatroid, in dem pro Knoten maximal eine eingehende Kante gewählt werden darf. Für das Optimierungsproblem, eine maximale Basis im Schnitt zweier Matroide zu finden, existiert ein polynomielles Lösungsverfahren (vgl. Abschnitt 3.1).

Wir betrachten nun (d, r)–Arboreszenzen, die eine Verallgemeinerung von d–Bäumen auf gerichteten Graphen sind. Die Kantenmenge E ist disjunkt in rote und blaue Kanten, zudem erlauben wir wie vorher parallele Kanten.

Definition 3.5. *Sei* $G = (V, E^r \dot\cup E^b)$ *ein gerichteter Multigraph mit Knoten* $r \in V$*,* $c : (E^r \dot\cup E^b) \to \mathbb{R}$ *eine Gewichtsfunktion auf den Kanten. Eine* (d, r)–Arboreszenz *ist eine in* r *gewurzelte Arboreszenz* T *auf* G *mit* $|T \cap E^b| \leq d$.

Problem 3.6 (kostenminimale (d, r)–Arboreszenz)

Gegeben: Ein gerichteter Multigraph $G = (V, E^r \dot\cup E^b)$, $r \in V$ Wurzel, $c : (E^r \dot\cup E^b) \to \mathbb{R}$, $d \in \mathbb{N}$.

Gesucht: Eine (d, r)–Arboreszenz T mit $\sum_{e \in T} c(e)$ minimal.

(d, r)–Arboreszenzen können somit als Schnitt dreier Matroide modelliert werden, indem wir ein weiteres Partitionsmatroid über die Färbung der Kanten hinzunehmen. Für den Schnitt von drei Matroiden ist kein polynomielles Verfahren bekannt. Vielmehr ist das allgemeine Problem, eine maximale unabhängige Menge im Schnitt dreier beliebiger Matroide zu bestimmen, $\mathcal{NP}$–vollständig. Der Beweis basiert auf einer einfachen

Reduktion von HAMILTON–PFAD [PS98]. Ein Hamilton–Pfad für einen Graphen $G = (V, E)$ ist ein Pfad, der alle Knoten des Graphen genau einmal besucht. Für die Reduktion betrachtet man den gerichteten Graphen, auf dem der Hamilton–Pfad bestimmt werden soll und bildet das zugehörige graphische Matroid. Die fehlenden Matroide sind jeweils Partitionsmatroide über die Köpfe bzw. Schwänze adjazenter Kanten. Im Schnitt dieser drei Matroide liegen dann alle kreisfreien Kantenmengen, die pro Knoten maximal eine ein– und eine ausgehende Kante besitzen. Die von uns betrachteten gefärbten Arboreszenzen liegen freilich im Schnitt dreier spezieller Matroide, die den vorigen zwar ähnlich sind, aber nicht genau entsprechen. Deswegen kann nicht direkt eine Aussage über den Komplexitätsstatus gezogen werden.

Trotzdem können wir aus dieser Beobachtung lernen, daß der Greedy–Algorithmus für das kostenminimale (d, r)–Arboreszenz Problem nicht beliebig schlechte Lösungen liefert, sondern zumindest eine 3–Approximation:

Satz 3.7 (Hausmann et al. 1980 [HJK80]). *Sei $(E, \mathcal{I})$ ein Unabhängigkeitssystem, das durch den Schnitt von p Matroiden beschrieben wird und $c : E \to \mathbb{R}_+$ eine Gewichtsfunktion. Dann liefert der Greedy–Algorithmus eine p–Approximation für das Minimierungsproblem über $(E, \mathcal{I})$ mit c.*

Arboreszenzen als Lineares Programm

Das Problem, eine minimale r–Arboreszenz zu bestimmen, läßt sich als Lineares Programm beschreiben. Dazu benutzen wir Lemma 2.4: Jede r–Arboreszenz ist äquivalent zu einem minimalen Graphen, in dem über jeden r–Schnitt mindestens eine Kante geht. Sei A eine Matrix, deren Spalten den Kanten von G und deren Zeilen den Inzidenzvektoren aller r–Schnitte entsprechen. Die Matrix A ist total unimodular, wodurch die Ecken des Polyeders $\{Ax \geq \mathbb{1}, x \geq 0\}$ als ganzzahlig angenommen werden können. Wegen der Polynomialität der Ellipsoidmethode für Lineare Programme [KV02] kann das Optimierungsproblem

$$\min\{cx : Ax \geq \mathbb{1}, x \geq 0\}$$

in polynomieller Zeit gelöst werden und beschreibt das ganzzahlige kostenminimale r–Arboreszenzproblem [Ful74].

Allerdings geht diese schöne Eigenschaft verloren, wenn das Lineare Programm auf triviale Weise ergänzt wird. Eine solche Erweiterung besteht z.B. im Hinzufügen einer weiteren Zeile mit dem Inzidenzvektor über die Kantenfärbung zu A und Erweiterung der rechten Seite um den Eintrag $-d$, so daß

$$\min\{cx : Bx \geq b, x \geq 0\} \tag{3.1}$$

das kostenminimale (d, r)–Arboreszenzproblem beschreibt, wobei

$$B := (A, \alpha_i)^T, b = (1, \ldots, 1, -d)^T \text{ und } \alpha_i = -1 \Leftrightarrow e_i \in E^b$$

für alle $e_i \in E^r \dot{\cup} E^b$.

Leider hat dieses System schon für einfache Beispiele nichtganzzahlige Optimallösungen. Der Graph in Abbildung 3.1 mit Wurzel r besitzt eine optimale $(1, r)$–Arboreszenz mit Kosten 10, die Optimallösung des beschriebenen Linearen Programms 3.1 ist jedoch $x_a = x_b = x_c = x_d = 0.5$ und Zielfunktionswert 7.5.

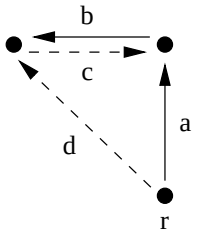


Abbildung 3.1.: *Beispielgraph mit roten (durchgezogene Linie) und blauen Kanten (gestrichelt). Die Kosten der roten Kanten sind* $c(a) = 10$ *und* $c(b) = 5$*, die Kosten der blauen Kanten* 0*.*

Bislang ist der Komplexitätsstatus des Problems, in einem Graphen eine kostenminimale (d, r)–Arboreszenz zu bestimmen, ungeklärt. Jedoch ist das Problem in der Literatur nicht unbeachtet geblieben. In den folgenden Abschnitten beschreiben wir spezielle Strukturen, durch deren Annahme kombinatorische Lösungsansätze möglich geworden sind. Zunächst betrachten wir in Abschnitt 3.2.1 die ungewichtete Variante des Problems, indem die Antwort eines Algorithmus eine beliebige (d, r)–Arboreszenz ist, falls eine solche in G existiert. Abschnitt 3.2.2 behandelt den Fall, daß alle blaue Kanten von der Wurzel des zugrunde liegenden Graphen ausgehen. Danach betrachten wir in Abschnitt 3.2.3 eine Variante

des r–Arboreszenz–Problems, in der die Kosten nicht minimiert werden sollen, sondern eine Arboreszenz mit genau Kosten $C \in \mathbb{N}$ gesucht wird.

3.2.1. Existenz einer (d, r)–Arboreszenz

Die Frage, ob in einem Graphen G überhaupt eine (d, r)–Arboreszenz existiert, ist leicht zu beantworten, indem wir die blauen Kanten mit hohen Strafkosten, etwa der Summe aller Kanten, belegen und das kostenminimale r–Arboreszenzproblem lösen. Auf diese Weise erhalten wir eine Arboreszenz T mit kleinstmöglicher Anzahl an blauen Kanten. Falls $|E^b \cap T| > d$ existiert offensichtlich keine (d, r)–Arboreszenz.

Alternativ dazu findet man in [BP87] einen kombinatorischen Ansatz, der auf die Modifikation der Kantengewichte verzichtet. Der dazugehörige Algorithmus startet mit einer r–Arboreszenz T, die nur rote Kanten enthält und ersetzt sukzessive jeweils eine rote Kante $e^r \in (E^r \cap T)$ durch eine blaue Kante $e^b \in (E^b \setminus T)$, wobei e^r und e^b denselben Kopf besitzen. Unter solchen Tauschen werden nur solche betrachtet, deren gemeinsamer Kopf in T die größte Distanz zur Wurzel hat. Falls durch den Tausch ein Kreis entsteht, wird der zugrunde liegende Graph geschrumpft und das Verfahren erneut auf den geschrumpften Graph angewendet. Entweder findet man auf diese Weise eine (d, r)–Arboreszenz oder einen Beweis, daß es keine solche gibt.

3.2.2. Wurzelbeschränkte Graphen

Gegeben einen gerichteter Graph $G = (V, E^r \dot\cup E^b)$ mit Kantenbewertung $c : E^r \dot\cup E^b \to \mathbb{R}$, dessen blaue Kanten als Schwanz die Wurzel $r \in V$ besitzen. In diesem Fall existiert ein polynomieller Algorithmus zur Konstruktion einer kostenminimalen (d, r)–Arboreszenz. Das Verfahren berechnet zunächst eine kostenminimale r–Arboreszenz T' mit möglichst vielen blauen Kanten und tauscht anschließend sukzessiv rote Kanten gegen blaue, bis nur noch d blaue Kanten in T' enthalten sind. Falls durch das Einfügen ein Kreis entsteht, wird analog zum Branching–Algorithmus von Edmonds [Edm67] der Kreis geschrumpft. Da durch die Voraussetzung

keine blaue Kante auf einem Kreis liegen kann, sind die Kosten nacheinander ausgeführter Tauschoperationen monoton wachsend.

Satz 3.8 (Gabow, Tarjan 1984 [GT84]). *Sei* $G = (V, E^r \dot\cup E^b)$ *ein gerichteter Graph mit Knoten* $r \in V$, $c : E^r \dot\cup E^b \to \mathbb{R}$ *ein Kantenbewertung und* $d \in \mathbb{N}$. *Falls alle blauen Kanten adjazent zu* r *sind, existiert ein Algorithmus zur Konstruktion einer kostenminimalen* (d, r)*–Arboreszenz auf* G *mit Laufzeit* $\mathcal{O}(m \log^* n)$.

3.2.3. Exakte r–Arboreszenz

Wir vernachlässigen in diesem Abschnitt die Färbung der Kanten und betrachten gewurzelte Arboreszenzen. Zu einem gegebenen Graphen G mit gewichteten Kanten wollen wir nun das Entscheidungsproblem betrachten, ob G eine Arboreszenz enthält, deren Gesamtgewicht genau einer vorgegebenen Zahl C entspricht.

Problem 3.9 (Exakt–C–Arboreszenz–Entscheidung)

Gegeben: Ein gerichteter Graph $G = (V, E)$, $r \in V$ Wurzel, $c : E \to \mathbb{N}$, $C \in \mathbb{N}$.

Gesucht: Eine Antwort, ob G eine r–Arboreszenz T mit Kosten $\sum_{e \in T} c(e) = C$ enthält.

Das Problem läßt sich wegen einer einfachen Reduktion von SUBSET–SUM, das als $\mathcal{NP}$–vollständig bekannt ist [Kar72], zu den schwierigen Problem zählen. Obwohl das Minimierungsproblem relativ einfach zu berechnen ist, können wir es scheinbar nicht benutzen, um die exakte Variante zu lösen. Es gehört jedoch zu der Menge der Zahlprobleme, da ein pseudopolyonieller Algorithmus bekannt ist.

Satz 3.10 (Barahona, Pulleyblank 1987 [BP87]). *Sei* $G = (V, E)$ *ein gerichteter Graph mit* $r \in V$ *und* $C \in \mathbb{N}$. *Es existiert ein Algorithmus mit Laufzeit* $\mathcal{O}((n^3 + p^2)p^2 \log p)$ *für Exakt–C–Arboreszenz–Entscheidung, wobei* $p = n \cdot \max_{e \in E}\{c(e)\}$.

Der von den beiden Autoren entwickelte Algorithmus berechnet über die Determinante einer speziellen Adjazenzmatrix die Anzahl der r–Arboreszenzen in G mit Gewicht genau C.

Die Entscheidungsvariante des exakten Problems liefert uns nur eine binäre Antwort und keinen Beweis der Existenz bzw. Nicht–Existenz der gesuchten Arboreszenz. Zu der gleichen Fragestellung betrachten wir deswegen das folgende Problem, das im Falle einer positiven Antwort zusätzlich einen Beweis in Form einer Arboreszenz, die die geforderten Eigenschaften aufweist, erwartet.

Problem 3.11 (Exakt–C–Arboreszenz–Beweis)

Gegeben: Ein gerichteter Graph $G = (V, E)$, $r \in V$ Wurzel, $c : E \to \mathbb{N}$, $C \in \mathbb{N}$.

Gesucht: Eine r–Arboreszenz T für G mit Kosten $\sum_{e \in T} c(e) = C$, falls diese existiert.

Offensichtlich kann man die Entscheidungsvariante lösen, wenn man die Beweisvariante berechnen kann. Tatsächlich sind beide Versionen unter polynomieller Reduktion äquivalent. Da dies scheinbar nicht direkt ersichtlich ist [YT01], geben wir einen kurzen Beweis an.

Lemma 3.12. *Es gilt:*

$$\textit{Exakt–C–Arboresz.–Entscheidung} =_p \textit{Exakt–C–Arboreszenz–Beweis}$$

Beweis. Es gilt sicherlich, daß Exakt–C–Arboreszenz–Entscheidung $\leq_p$ Exakt–C–Arboreszenz–Beweis. Deswegen beweisen wir nur die andere Richtung. $A_E(G, r, c, C)$ sei dann ein Algorithmus, der entscheidet, ob $G = (V, E)$ mit $r \in V$ eine r–Arboreszenz mit Kosten genau C für die Kostenfunktion c enthält. Wir konstruieren einen Algorithmus $A_B(G, r, c, C)$, der iterativ A_E benutzt und die Beweis–Variante löst.

Genau dann, wenn eine r–Arboreszenz in G mit den gesuchten Kosten existiert, wird der innere Teil des Algorithmus ausgeführt. Andernfalls liefert die erste Abfrage von $A_E(G, r, c, C)$ den Wert *falsch* und der Algorithmus A_B stoppt.

Für jede Kante e wird überprüft, ob ihr Entfernen aus G die Existenz einer solchen Arboreszenz unmöglich macht. Falls G ohne e weiterhin eine Arboreszenz mit Kosten C enthält, betrachten wir im nächsten Schritt

Algorithmus 3 $A_B(G, r, c, C)$

```
if A_E(G, r, c, C) wahr then        ▷ Existiert eine solche Arboreszenz?
    Initialisiere G' = G, C' = C
    for all e ∈ E do
        if A_E(G' − e, r, c, C') wahr then      ▷ e kann verworfen werden
            G' = G − e
        else                                    ▷ e ist Teil der Arboreszenz
            Markiere e
            Kontrahiere e in G'
            Entferne alle f ∈ E mit kopf(f) = kopf(e)
            C' = C' − c(e)
        end if
    end for
else
    Stopp.
end if
```

den reduzierten Graphen $G - e$. Andernfalls existiert in $G - e$ keine r–Arboreszenz mit Kosten C, jedoch in G. Somit muß e in der gesuchten Arboreszenz enthalten sein und wir markieren e.

Sei G' ein Graph, der eine r–Arboreszenz T mit Kosten C enthält und $e \in T$. Sei G'' der Graph, der aus G' entsteht, wenn wir e kontrahieren und alle Kanten f mit gleichem Kopf wie e entfernen. Kontrahiere in T die Kante e und nenne die resultierende Kantenmenge T'. T' enthält einen gerichteten Pfad von der Wurzel r zu jedem Knoten auf G'' und besitzt den Kostenwert $c(T') = c(T) - c(e)$.

Umgekehrt enthalte G'' eine Arboreszenz T' mit Kosten $C - c(e)$. Expandiere T' um e, indem wir die Kante selbst und den Kopf von e einfügen. Die daraus entstehende Arboreszenz ist mit den gleichen Argumenten eine r–Arboreszenz mit Kosten $c(T') + c(e) = C$ für G'.

Sei anfangs $G' = G$ wie im Algorithmus. Dann erhalten wir die Behauptung per Induktion. Jede Kante wird genau einmal betrachtet, dann wird sie entweder verworfen oder kontrahiert. Die Gesamtlaufzeit beträgt somit $\mathcal{O}(m(n^3 + p^2)p^2 \log p)$ Schritte. □

3.3. Approximation

Das Problem, eine kostenminimale (d, r)–Arboreszenz zu bestimmen, kann allgemeiner als ein Optimierungsproblem mit mehreren Zielfunktionen betrachtet werden. Wir werden uns in diesem Abschnitt zunächst dieser allgemeineren Betrachtung zuwenden und Aussagen über Eigenschaften des Lösungsraums treffen. Diese Feststellungen werden uns helfen, ein vollpolynomielles Approximationsschema für das Ausgangsproblem zu konstruieren. Weitere Beispiele für solche Multikriteria–Probleme sind z.B.

- Kürzeste–Wege–Problem mit Ressourcen
- Kostenminimaler–Fluß–Problem in mehreren Zielfunktionen
- Maximalgewichtetes Matching

Falls es nicht anders gekennzeichnet ist, basieren die Definitionen und Beweise dieses Abschnittes auf dem *Extended Abstract* von Papadimitriou und Yannakakis [PY00], deren Beweisideen hier ausgearbeitet werden. Das abschließende Korollar 3.21 kombiniert diese Erkenntnisse zu einer Aussage über das gefärbte Arboreszenzproblem.

3.3.1. Approximative Paretokurven

Sei x eine Instanz eines Maximimierungsproblems in $k \in \mathbb{N}$ Zielfunktionen und s eine Lösung für x. Dann wird s per $f_i(s) \in \mathbb{R}_+$ für alle $i = 1, \ldots, k$ eine positive rationale Zahl zugeordnet. Wir werden den an dieser Stelle betrachteten Algorithmen nur eine polynomielle Laufzeit erlauben und diese mit einem Polynom $p(|x|)$ abschätzen.

Bemerkung 3.13. *Wir gehen außerdem davon aus, daß $f_i(s) > 0$ ist und können somit annehmen, daß die Zielfunktionswerte im Intervall $\frac{1}{2^{p(|x|)}} \leq f_i(s) \leq 2^{p(|x|)}$ liegen, da alleine die Ausgabe einer kleineren bzw. größeren Zahl längere Zeit beanspruchen würde.*

Da wir es mit mehr als einer Zielfunktion zu tun haben, ist nicht unmittelbar klar, welche Lösungen wir als optimal betrachten wollen. Aus diesem Grund betrachtet man die Lösungen, die im paarweisen Vergleich nicht dominiert werden. Diese Menge wird als *Paretokurve* bezeichnet und mit $P(x)$ abgekürzt.

Definition 3.14 (Paretokurve). *Sei* $F(x)$ *die Menge aller* zulässigen Lösungen *für eine Instanz* x *eines solchen Optimierungsproblems. Dann enthält* $P(x)$ *die Menge der Vektoren* $v \in \mathbb{R}^k$ *mit den Eigenschaften*

1. *Es existiert eine Lösung* $s \in F(x)$ *mit* $f_i(s) = v_i$ *für alle* $i = 1, \ldots, k$
2. *Es existiert keine zulässige Lösung* s'*, so daß* $f_i(s') \geq v_i$ *für alle* i *und* $f_i(s') > v_i$ *für mindestens ein* $i \in \{1, \ldots, k\}$*.*

Die Paretokurve ist als Menge der Zielfunktionsvektoren definiert. Wenn wir abkürzend von einer *Lösung* $s \in P(x)$ sprechen, dann meinen wir die zu einem bestimmten Zielfunktionswert zugehörige Lösung. Z.B. merken wir uns für jeden Zielfunktionswert $v \in P(x)$ eine Lösung $s \in F(x)$ mit $f(s) = v$. Leider ist die Kardinalität von $P(x)$ für viele Probleme nicht durch ein Polynom in $|x|$ zu beschränken, wodurch es unpraktikabel wird, alle undominierten Lösungen zu betrachten. Aus diesem Grund weicht man den Optimalitätsbegriff auf und betrachtet eine Menge von Näherungslösungen, von der wir zeigen werden, daß sie in handhabbarer Größe konstruiert werden kann.

Definition 3.15 ($(1+\varepsilon)$–approximative Paretokurve). *Sei* x *eine Instanz sowie* $\varepsilon > 0$ *gegeben. Dann ist eine Menge* $P_\varepsilon(x)$ *von* $(1+\varepsilon)$–approximativen Paretolösungen *eine Menge von Vektoren* $v \in \mathbb{R}^k$ *mit*

1. *Es existiert eine Lösung* $s \in F(x)$ *mit* $f_i(s) = v_i$ *für alle* i
2. *Für alle* $s' \in F(x)$ *existiert ein* $s \in P_\varepsilon(x)$*, so daß für alle* $i \in \{1, \ldots, k\}$ *gilt:*
$$f_i(s)(1+\varepsilon) \geq f_i(s')$$

Anders ausgedrückt finden wir für jede Lösung s' eine Lösung $s \in P_\varepsilon$, die bis auf einen Faktor so gut ist wie s selbst. Die Kardinalität von $P_\varepsilon(x)$ hängt stark von der Konstruktion ab, da $P_\varepsilon(x) = P(x)$ offensichtlich alle Bedingungen erfüllt. Der folgende Satz macht jedoch Hoffnung, eine approximative Paretokurve in handhabbarer Größe bestimmen zu können.

Satz 3.16. *Für jedes Optimierungsproblem x und jedes $\varepsilon > 0$ existiert eine Menge $P_\varepsilon(x)$ mit $|P_\varepsilon(x)|$ polynomiell in $|x|$ und $\frac{1}{\varepsilon}$, jedoch exponentiell in der Anzahl der Kostenfunktionen.*

Beweis. Wir betrachten den k–dimensionalen Raum aller Zielfunktionswerte. Aufgrund der Bemerkung 3.13 liegen die Werte in jeder Dimension zwischen $\frac{1}{2^{p(|x|)}}$ und $2^{p(|x|)}$. Wir unterteilen diesen Hyperwürfel nun in Hyperquader, und zwar so, daß sich die Werte auf jeweils einer Koordinate nicht um mehr als den Faktor $(1+\varepsilon)$ unterscheiden. Dazu beginnen wir im Punkt $(\frac{1}{2^{p(|x|)}}, \ldots, \frac{1}{2^{p(|x|)}})^T$ und lassen das Hyperquader in jeder Dimension maximal wachsen, so daß es in jeder Dimension alle Punkte bis $(\frac{1+\varepsilon}{2^{p(|x|)}}, \ldots, \frac{1+\varepsilon}{2^{p(|x|)}})^T$ umfaßt. Wir untersuchen zunächst, wieviele Einteilungen pro Dimension nötig sind, damit das Verhältnis der kleineren Koordinate zur größeren genau $(1+\varepsilon)$ beträgt. Wenn i die Anzahl der Einteilungen zählt, dann ist

$$\begin{aligned}
(1+\varepsilon)^i \frac{1}{2^{p(|x|)}} &= 2^{p(|x|)} \\
\Leftrightarrow \quad i &= \log_{1+\varepsilon} 2^{2p(|x|)} = 2p(|x|)\log_{1+\varepsilon} 2 \\
&= 2p(|x|)\frac{\log_2 2}{\log_2 1+\varepsilon} = \frac{2p(|x|)}{\log_2 1+\varepsilon}
\end{aligned}$$

Für $0 \le \varepsilon \le 1$ gilt: $\varepsilon \le \log_2(1+\varepsilon)$ (vgl. Abb. 3.2). Für größere ε können wir i unabhängig von ε abschätzen, da $i \le 2p(|x|)$ viele Hyperquader ausreichen. Somit können wir die Anzahl der Hyperquader zur Partitionierung des k–dimensionalen Hyperwürfels für kleine Werte von ε durch $\mathcal{O}(\frac{(2p(|x|))^k}{\varepsilon^k})$ begrenzen. Für konstantes k ist dies polynomiell in $|x|$ und $\frac{1}{\varepsilon}$. Wir betrachten nun alle Punkte in $P(x)$ und fügen für jeden Hyperquader, der einen solchen Punkt enthält, genau einen zu $P_\varepsilon(x)$ hinzu.

Somit enthält $P_\varepsilon(x)$ maximal so viele Punkte wie es Hyperquader gibt. Außerdem dominiert bis auf den Faktor $(1+\varepsilon)$ in jedem Hyperquader die gemerkte Lösung alle anderen Werte $v \in P$, die im gleichen Hyperquader liegen. Zusammen mit den Eigenschaften von $P(x)$ folgt, daß die von uns konstruierte Menge $P_\varepsilon(x)$ eine $(1+\varepsilon)$–approximative Paretomenge ist. □

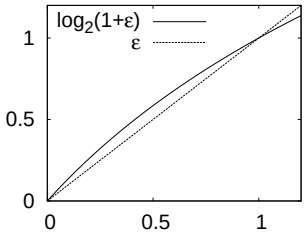


Abbildung 3.2.: *Die Funktionen* $\log_2(1+\varepsilon)$ *und* ε

Der Beweis der Existenz benötigt zur Konstruktion die Menge $P(x)$, die uns allgemein nicht vorliegt. Wir wollen nun versuchen, $P_\varepsilon(x)$ unabhängig von $P(x)$ aufzubauen. Dazu betrachten wir das folgende Problem.

Problem 3.17 (Gap)

Gegeben: Instanz x eines Maximierungsproblems, $b = (b_1, \ldots, b_k) \in \mathbb{R}_+^k$ sowie $\varepsilon > 0$.

Gesucht: Lösung $s \in F(x)$ mit $f(s) \geq b$ oder antworte, daß es keine Lösung s' gibt mit $f(s') \geq b(1+\varepsilon)$.

Satz 3.18. *Es existiert ein Algorithmus zur Konstruktion einer* $(1+\varepsilon)$*–approximativen Menge* $P_\varepsilon(x)$*, deren Größe polynomiell in* $|x|$ *und* $\frac{1}{\varepsilon}$ *beschränkt ist, genau dann wenn das Gap–Problem gelöst werden kann.*

Beweis. $\Rightarrow$: Falls uns eine Menge $P_\varepsilon(x)$ vorliegt und es ein $s \in P_\varepsilon(x)$ gibt, so daß $f(s) \geq b$, können wir das Gap–Problem positiv beantworten. Andernfalls versichert uns die Definition 3.15, daß es keine zulässige Lösung s' in $F(x)$ mit $f(s') \geq b(1+\varepsilon)$ geben kann.

$\Leftarrow$: Sei $A(x, b, \varepsilon)$ der Algorithmus, der das Gap–Problem löst. Wie im Beweis zu Satz 3.16 teilen wir den Hyperwürfel aller möglichen Zielfunktionswerte in kleinere Hyperquader auf, jedoch wählen wir eine feinere Unterteilung mit $\varepsilon' := \sqrt{1+\varepsilon} - 1$. Für jede Ecke $b \in \mathbb{R}_+^k$ eines solchen Hyperquaders starten wir den Algorithmus $A(x, b, \varepsilon')$. Von allen Antworten müssen wir uns nur diejenigen Lösungen in $P_\varepsilon(x)$ merken, die nicht dominiert werden.

Wir zeigen nun, daß $P_\varepsilon(x)$ eine $(1+\varepsilon)$–approximative Paretomenge ist. Angenommen, es existiert ein $s' \in F(x)$, jedoch kein $s \in P_\varepsilon(x)$ mit $f(s)(1+\varepsilon) \geq f(s')$. Sei b' diejenige Ecke des Hyperquaders, in dem $f(s')$ liegt und für die $b' \leq f(s')$ gilt. Der Algorithmus $A(x, b', \varepsilon')$ lieferte uns keine Lösung, ansonsten wäre die Annahme widerlegt. Dann wissen wir aber, daß es keine Lösung $s'' \in F(x)$ gibt mit $f(s'') \geq b'(1+\varepsilon')$. Insbesondere gilt dann $f(s') < b'(1+\varepsilon')$. Sei $b = \frac{b'}{1+\varepsilon'}$. Möglicherweise ist b selbst keine Ecke eines Hyperquaders (z.B. im Falle $b' = (2^{-p(|x|)}, \ldots, 2^{-p(|x|)})$). Wir nutzen diesen Vektor trotzdem als Eingabe für $A(x, b, \epsilon')$. Für b muß A eine Antwort s mit $f(s) \geq b$ geliefert haben, da $b(1+\varepsilon') = b' \leq f(s')$. Es gilt

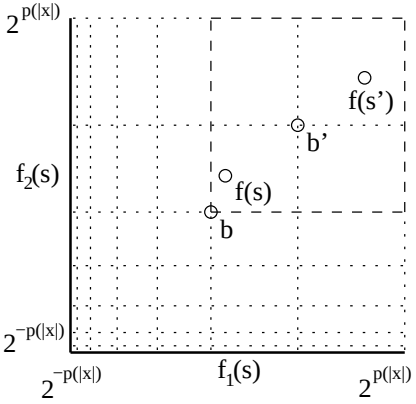


Abbildung 3.3.: *Die Einteilung des Hyperwürfels aller Werte für* $f_1(s), f_2(s)$

$$f(s') < b'(1+\varepsilon') = b(1+\varepsilon')^2 \leq f(s)(1+\varepsilon')^2 = f(s)(1+\varepsilon)$$

im Widerspruch zur Annahme, daß keine Lösung mit dieser Eigenschaft in $P_\varepsilon(x)$ existiert. Abbildung 3.3 verdeutlicht diese Situation. Die Einteilung der Hyperquader nach $\varepsilon' = \sqrt{1+\varepsilon} - 1$ ist durch gepunktete Linien gekennzeichnet, während durch gestrichelte Linien hervorgehoben ist, welchen Bereich $b(1+\varepsilon)$ umfaßt. □

Der Beweis liefert konstruktiv einen Algorithmus, der ein Lösungsverfahren A für das Gap–Problem nutzt, um eine $(1+\varepsilon)$–approximative Paretokurve zu konstruieren. Dafür nutzen wir Satz 3.16 aus und können so die Anzahl der Aufrufe von A polynomiell beschränken.

3.3.2. Der linear diskrete Fall

Wir betrachten in diesem Abschnitt nur Optimierungsprobleme, deren Lösungen n–dimensionale Vektoren $s = (s_1, \ldots, s_n) \in \mathbb{N}^n$ sind. Jede

Zielfunktion ist durch einen nichtnegativen und reellen Vektor definiert: $f_i(s) = \nu_i s, \nu_i \in \mathbb{R}^n_+$. Wir nehmen zusätzlich an, daß der größte in s vorkommende Eintrag

$$m := \max_j \{s_j\}$$

polynomiell in n beschränkt ist. Dies schränkt die Lösungen allgemein ein, denn man kann sich Probleminstanzen überlegen, so daß die Optimallösung in einem Vektor angenommen wird, dessen Einträge sich nicht auf diese Weise beschränken lassen. Dennoch findet man sehr viele kombinatorische Optimierungsprobleme, die dieser Bedingung genügen. Häufig bestehen Lösungen sogar nur aus $\{0, 1\}$–Werten, z.B. wenn die Lösung einem Inzidenzvektoren entspricht wie es beim Matching–, Kürzeste–Wege– oder dem Minimal–Aufspannenden–Baum–Problem der Fall ist.

Zu einem gegebenen Maximierungsproblem D sei die *exakte Variante von* D wie folgt definiert.

Problem 3.19 (Exakt-D)

Gegeben: Instanz x eines Maximierungsproblems D sowie eine Zahl $b \in \mathbb{Z}_+$.

Gesucht: Lösung s von D mit Kosten genau b, d.h. $f(s) = b$.

Für Probleme mit den oben genannten Eigenschaften wollen wir nun eine weitere Charakterisierung angeben, unter welchen Umständen eine $(1 + \varepsilon)$–approximative Paretokurve in polynomieller Zeit und Größe konstruiert werden kann.

Satz 3.20. *Es existiert ein polynomieller Algorithmus, um eine* $(1 + \varepsilon)$*–approximative Paretokurve* $P_\varepsilon(x)$ *zu einer gegebenen Instanz* x *eines Maximierungsproblems* D *zu konstruieren, falls die exakte Variante von* D *für jede Kostenfunktion* $f : F(x) \to \mathbb{N}$ *in pseudopolynomieller Zeit gelöst werden kann.*

Beweis. Wir werden eine Instanz des Gap–Problems lösen, indem wir die exakte Fragestellung betrachten. Zusammen mit Satz 3.18 ergibt sich dann ein Verfahren zur Konstruktion eines geeigneten $P_\varepsilon(x)$. Dazu definieren wir zunächst $r := \lceil \frac{nm}{\varepsilon} \rceil$. Für jede Zielfunktion $f_i(s) = \nu_i s$ werden wir

eine neue skalierte und ganzzahlige Zielfunktion $g_i(s) = \nu_i' s$ betrachten, wobei die neuen Koeffizienten wie folgt berechnet werden:

$$\nu_{ij}' = \min\left\{\left\lfloor \frac{\nu_{ij} r}{b_i} \right\rfloor, r\right\}$$

Das Gap–Problem besteht darin, zu entscheiden, ob es zu einem gegebenen $b = (b_1, \ldots, b_k)$ sowie $\varepsilon > 0$ eine Lösung s gibt, so daß $f(s) \geq b$ oder daß es kein s' gibt mit $f(s') \geq b(1+\varepsilon)$. Wir wollen nun zeigen, daß es reicht, bestimmen zu können, ob es eine Lösung gibt, für die $g_i(s) \geq r$ gilt. Dazu können wir uns auf die folgenden Aussagen beschränken.

(i) $g_i(s) \geq r \Rightarrow f_i(s) \geq b_i$

(ii) $f_i(s) \geq (1+\varepsilon) b_i \Rightarrow g_i(s) \geq r$

Einerseits sollte jede Lösung, die wir mittels des Tests $g_i(s) \geq r$ erhalten, auch $f_i(s) \geq b_i$ erfüllen. Falls es andererseits keine Lösung s' gibt, die um den Faktor $(1+\varepsilon)$ besser ist als b, so müssen wir dies auch mittels g_i erfahren können. Positiv ausgedrückt bedeutet das, daß jede Lösung s mit $g_i(s) \geq r$ auch $f_i(s) \geq (1+\varepsilon) b_i$ erfüllen muß. Wir betrachten nun $g_i(s)$ genauer und schreiben

$$g_i(s) = \sum_{j=1}^{n} \min\left\{\left\lfloor \frac{\nu_{ij} r}{b_i} \right\rfloor, r\right\} s_j = \sum_{j_1 \in J_1} \left\lfloor \frac{\nu_{ij_1} r}{b_i} \right\rfloor s_{j_1} + \sum_{j_2 \in J_2} r s_{j_2},$$

indem wir die Indexmenge der $j = \{1, \ldots, n\}$ in zwei Teilmengen J_1 und J_2 partitionieren. Falls $s_{j_2} > 0$ für ein $j_2 \in J_2$, so gilt sofort $g_i(s) \geq r$. Also nehmen wir an, daß $s_{j_2} = 0$ für alle $j_2 \in J_2$. Dann können wir den rechten Term ohne Einschränkung weglassen.

Mit diesen Vorüberlegungen beweisen wir nun (ii). Sei s eine zulässige Lösung und es gelte $f_i(s) \geq b_i(1+\varepsilon)$ für alle i. Nach Voraussetzung gilt

nun

$$
\begin{aligned}
& f_i(s) = \sum_{j=1}^{n} v_{ij} s_j \geq b_i(1+\varepsilon) && \Leftrightarrow && \sum_{j=1}^{n} \frac{v_{ij}}{b_i} s_j \geq 1+\varepsilon \\
\Leftrightarrow \quad & \sum_{j=1}^{n} \frac{v_{ij} r}{b_i} s_j \geq (1+\varepsilon) r && \Leftrightarrow && \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} s_j \right) - \varepsilon r \geq r \\
\Rightarrow \quad & \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} s_j \right) - \varepsilon \frac{nm}{\varepsilon} \geq r && \Leftrightarrow && \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} s_j - m \right) \geq r \\
\Rightarrow \quad & \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} s_j - s_j \right) \geq r && \Leftrightarrow && \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} - 1 \right) s_j \geq r \\
\Rightarrow \quad & \sum_{j=1}^{n} \left\lfloor \frac{v_{ij} r}{b_i} \right\rfloor s_j \geq r && \Rightarrow && g_i(s) \geq r
\end{aligned}
$$

Zu (i) : Jede Lösung, für die $g_i(s) \geq r$ gilt, muß auch $f_i(s) \geq b_i$ erfüllen. Dann rechnen wir

$$
\begin{aligned}
g_i(s) = v_i' s = \sum_{j=1}^{n} v_{ij}' s_j = \sum_{j=1}^{n} \min \left\{ \left\lfloor \frac{v_{ij} r}{b_i} \right\rfloor, r \right\} s_j &\geq r \\
\Leftrightarrow \sum_{j=1}^{n} \min \left\{ \left\lfloor \frac{v_{ij} r}{b_i} \right\rfloor \cdot \frac{b_i}{r}, b_i \right\} s_j &\geq b_i \\
\Rightarrow \sum_{j=1}^{n} \left(\frac{v_{ij} r}{b_i} \cdot \frac{b_i}{r} \right) s_j &\geq b_i \\
\Leftrightarrow \sum_{j=1}^{n} v_{ij} s_j \geq b_i \Leftrightarrow f_i(s) &\geq b_i
\end{aligned}
$$

Um das Gap–Problem zu lösen, reicht es also aus, eine Lösung s zu bestimmen, für die $g_i(s) \geq r$ gilt. Falls es keine solche Lösung gibt, wissen wir, daß es auch keine Lösung mit $f_i(s) \geq (1+\varepsilon) b_i$ gibt. Der Maximalwert, den ein g_i annehmen kann, ist rnm. Sei dann $M := rnm + 1$. Da r und m polynomiell in der Eingabelänge und $\frac{1}{\varepsilon}$ sind, gilt dies auch für M. Eine Ungleichung $g_i(s) \geq r$ können wir durch $M - r + 1$ viele Gleichungen

ersetzen:

$$g_i(s) \geq r \Leftrightarrow g_i(s) = r \vee g_i(s) = r+1 \vee \cdots \vee g_i(s) = M$$

k Gleichungen können wir durch eine ersetzen, indem die i–te Gleichung mit M^{i-1} multipliziert wird. Denn:

$$g_i(s) = l_i \text{ für alle } i \Leftrightarrow \sum_{i=1}^{k} M^{i-1} g_i(s) = \sum_{i=1}^{k} M^{i-1} l_i$$

Wird uns zu einer Probleminstanz ein Vektor $b = (b_1 \ldots, b_k)$ sowie ein $\varepsilon > 0$ gegeben, dann starten wir den Algorithmus, der Exakt–D löst, $\mathcal{O}(n^{2k})$ mal mit der Zahl $\sum_{i=1}^{k} M^{i-1} l_i$, wobei $l_i \in \{r, \ldots, M\}, i = 1, \ldots, k$. Liefert uns eine dieser Abfragen eine Lösung s, dann ist s eine zulässige Lösung für das Ausgangsproblem D mit $f_i(s) \geq b_i$ für alle i. Andernfalls gibt es keine Lösung s, so daß $f_i(s) \geq (1+\varepsilon)b_i$. Die Laufzeit des Exakt-D Algorithmus ist pseudopolynomiell in der Eingabelänge beschränkt. Da der Wert der größten Zahl polynomiell in n und $\frac{1}{\varepsilon}$ beschränkt ist, können wir eine polynomielle Gesamtlaufzeit des Verfahrens annehmen. □

3.3.3. Approximation von (d, r)–Arboreszenzen

Abschließend wenden wir uns wieder dem Problem zu, dem dieses Kapitel gewidmet ist. Um die gerade getroffenen Aussagen auf das gefärbte Arboreszenzproblem anwenden zu können, müssen wir nur zeigen, wie es als Optimierungsproblem in mehreren Zielfunktionen betrachtet werden kann.

Korollar 3.21. *Es existiert ein FPTAS für das kostenminimale* (d, r)*–Arboreszenz–Problem.*

Beweis. Sei $G = (V, E^r \dot{\cup} E^b)$ der rot und blau gefärbte Graph und $c : E^r \dot{\cup} E^b \to \mathbb{N}$ die Kostenfunktion auf den Kanten. Eine Lösung können wir als Inzidenzvektor s annehmen. Sei

$$M := 2 \sum_{e \in E^r \dot{\cup} E^b} |c(e)|.$$

Wähle dann die Koeffizienten der Kostenfunktionen $f_1(s) = \nu_1 s$ und $f_2(s) = \nu_2 s$ wie folgt:

$$\nu_{1,e} = M - c(e)$$
$$\nu_{2,e} = \begin{cases} 0 & \text{falls } e \in E^b \\ 1 & \text{falls } e \in E^r \end{cases}$$

ν_1 macht aus dem Minimierungsproblem ein Maximierungsproblem. Da die Anzahl der Kanten einer Lösung konstant ist, können wir auf jeden Kostenwert einer Kante eine Konstante addieren, wobei jede vorher optimale Lösung weiterhin optimal bleibt. ν_2 zählt die Anzahl der roten Kanten. Wir fragen nach höchstens d blauen Kanten. Das ist äquivalent zu der Frage, ob es mindestens $|V| - 1 - d$ rote Kanten gibt. Mit diesen Überlegungen folgt die Behauptung aus Satz 3.20 und Lemma 3.12. □

Sei ε' die minimale Differenz der Kosten zweier beliebiger Kanten eines gerichteten Graphen $G = (V, E)$. Für eine Kostenfunktion c sei also

$$\varepsilon' := \min_{e,f \in E} \{|c(e) - c(f)|\}.$$

Zwei Arboreszenzen auf G unterscheiden sich dann hinsichtlich ihrer Summe über die Kantenkosten mindestens um ε'. Wir können das FPTAS für gefärbte Arboreszenzen benutzen, um einen pseudopolynomiellen Algorithmus für das gleiche Problem zu konstruieren, indem wir den Genauigkeitsfaktor $\varepsilon < \varepsilon'$ wählen. Das FPTAS findet dann die optimale (d, r)–Arboreszenz. Die Laufzeit ist jedoch abhängig von ε' und somit im allgemeinen nicht polynomiell in der Eingabelänge beschränkt.

Alternativ geben wir nun einen pseudopolynomiellen Algorithmus an, der direkt mit dem Wissen aus Satz 3.10 und Lemma 3.12 konstruiert werden kann. Der Algorithmus A_{pseudo} berechnet eine kostenminimale r–Arboreszenz mit genau d blauen Kanten. Falls es eine billigere r–Arboreszenz mit weniger als d blauen Kanten geben sollte, so können wir diese ebenfalls bestimmen, indem wir den Algorithmus mit $d' = 0, \ldots, d$ starten. Dies benötigt zusätzlich $\mathcal{O}(n)$ Iterationen. Sei dazu $A_B(G, r, c, C)$ das in Algorithmus 3 beschriebene Verfahren, das eine r–Arboreszenz für G mit Kosten genau C in pseudopolynomieller Zeit berechnet, falls sie existiert.

Algorithmus 4 $A_{\text{pseudo}}(G = (V, E^r \dot\cup E^b), c, d, r)$

Sei $E := E^r \dot\cup E^b$.
Berechne $M := \sum_{e \in E} c(e) + 1$.
for all $e \in E$ **do**
  Definiere $c'(e) := \begin{cases} c(e) & \text{falls } e \in E^r \\ M + c(e) & \text{falls } e \in E^b \end{cases}$
end for
Berechne $a := (|V| - 1) \cdot \min_{e \in E}\{c(e)\}$ und $b := (|V| - 1) \cdot \max_{e \in E}\{c(e)\}$.
for all $i = a, \ldots, b$ **do**
  $T = A_B(G, r, c, dM + i)$
  if $T \neq \emptyset$ **then**
    T ist optimale r–Arboreszenz mit $|T \cap E^b| = d$. **Stopp.**
  end if
end for

Lemma 3.22 (Krumke, Räbiger 2005 [KR05]). *Algorithmus 4* A_{pseudo} *berechnet eine kostenminimale* r*–Arboreszenz mit* d *blauen Kanten und besitzt eine pseudopolynomielle Laufzeit.*

Beweis. Zunächst wird M als obere Schranke für den maximalen Wert einer Arboreszenz berechnet. Die Kostenfunktion c' addiert auf die Kosten jeder blauen Kante den Wert M. Für eine Teilmenge von Kanten $E' \subseteq E$ können wir dann anhand der Kostensumme $\sum_{e \in E'} c'(e)$ ablesen, wie viele blaue Kanten in E' enthalten sind. Auf der anderen Seite enthält jede zulässige r–Arboreszenz T genau d blaue Kanten. Wenden wir c auf T an, gilt $c(T) = c'(T) - dM$. Die Grenzen a und b sind untere bzw. obere Kostenschranken für r–Arboreszenzen bzgl. c. Sei T eine optimale r–Arboreszenz mit genau d blauen Kanten in c' und T' eine billigere r–Arboreszenz bzgl. c. Dann ist T' auch bzgl. c' günstiger, da

$$c(T') > c(T) \Leftrightarrow c'(T') - dM > c'(T) - dM,$$

Widerspruch.

Wir berechnen M in $\mathcal{O}(m)$ Schritten. Danach wird in weiteren m Schritten die Kostenfunktion c' konstruiert. Die Grenzen a und b werden eben-

falls in m Schritten bestimmt. Das Durchsuchen des Intervalls $[a, b]$ dauert maximal $b \leq m \cdot \max\{c(e)\}$ Schritte, da $a \geq 0$. Bei jedem Durchlauf wird Algorithmus A_B aufgerufen, der selbst ein pseudopolyonielles Verfahren ist. □

Wir haben ein pseudopolynomielles Verfahren für das kostenminimale (d, r)–Arboreszenz–Problem angegeben. Unter der Annahme, daß $\mathcal{P} \neq \mathcal{NP}$ gilt, kann das Problem nicht stark $\mathcal{NP}$–vollständig, aber immerhin noch schwach $\mathcal{NP}$–vollständig sein.

Kapitel 4.

Transportplanungs–Probleme

4.1. Klassische Transportprobleme

Zur Einleitung stellen wir einige klassische Transportprobleme vor, die in der Literatur gut untersucht worden sind. Die Reihenfolge ist so gewählt, daß die nachfolgenden Probleme jeweils Verallgemeinerungen der vorhergehenden sind. Wir beginnen mit dem Hamilton–Kreis–Problem, betrachten danach das Traveling–Salesperson–Problem und schließlich das Pickup–And–Delivery–Problem. Nachdem wir schrittweise allgemeinere Probleme kennengelernt haben, werden wir uns im allgemeinsten Fall auf spezielle Graphen zurückziehen, mit der Hoffnung, Strukturen identifizieren zu können, auf denen das Problem effizient lösbar ist.

Hamilton–Kreis

Das Hamilton'sche–Kreis–Problem ist wohl unter den Transport– und Tourenoptimierungsproblemen das fundamentalste, weil es ohne Kantengewichte auskommt und eine scheinbar einfache Aufgabe stellt. Wir betrachten einen ungerichteten Graphen und fragen, ob es möglich ist, alle Knoten genau einmal zu besuchen, indem wir einen geschlossenen Kantenzug entlang der Kanten wählen.

Problem 4.1 (Hamilton Kreis)
Gegeben: Ein ungerichteter Graph $G = (V, E)$.

Gesucht: Enthält G einen Hamilton–Kreis?

Das Hamilton–Kreis–Problem ist $\mathcal{NP}$–vollständig. Der dazugehörige Beweis reduziert das Vertex–Cover–Problem [GJ79]. In dem eng verwandten Hamilton–Pfad–Problem, das ebenfalls $\mathcal{NP}$–vollständig ist, wird kein geschlossener Kantenzug, sondern ein Pfad gesucht, der alle Knoten genau einmal besucht.

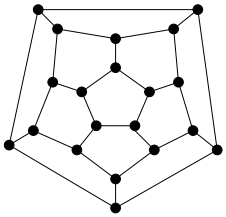


Abbildung 4.1.: *Dodekaeder für das Icosian Game.*

Das Problem wurde nach dem irischen Mathematiker Sir William Rowan Hamilton (1805–1865) benannt, der das Spiel mit dem Namen *Icosian Game* im Jahre 1856 erfand. Ziel ist es, in einem Dodekaeder entlang der Kanten zu wandern und alle Ecken genau einmal zu besuchen, wobei man schließlich automatisch zur Startecke zurückkehrt. Zwei Jahre später verkaufte er das Spiel, das es später auch in einer Reisevariante mit leichteren Regeln gab, an einen Londoner Händler für 25 Pfund. Hamilton entwickelte ebenfalls den *Icosian Calculus*, um eine Lösung für das Problem auf dem Dodekaeder zu berechnen [Ham58].

Der Handlungsreisende

Wir betrachten wieder einen ungerichteten Graphen $G = (V, E)$ und führen zusätzlich Kosten $c : E \to \mathbb{Z}_+$ auf den Kanten ein. Wir gehen nun jedoch davon aus, daß G vollständig ist. Die Knoten des Graphen repräsentieren Städte und die Kanten Wege zwischen diesen Städten. Ein Handlungsreisender steht vor dem Problem, alle Städte genau einmal zu besuchen. Dabei möchte er möglichst wenig Gesamtstrecke zurücklegen. Formal betrachten wir dann das folgende Problem:

Problem 4.2 (Traveling Salesperson (TSP))

Gegeben: Ein ungerichteter vollständiger Graph $G = (V, E)$, eine Kostenfunktion $c : E \to \mathbb{Z}_+$ und eine Konstante $C \in \mathbb{Z}_+$.

Gesucht: Enthält G einen Hamilton–Kreis $E' \subseteq E$ mit Kosten $\sum_{e \in E'} c(e) \leq C$?

Man sieht sofort, daß das Hamilton–Kreis–Problem in TSP enthalten ist. Tatsächlich läßt sich über diese Reduktion zeigen, daß das Problem des Handlungsreisenden $\mathcal{NP}$–vollständig ist. Es bleibt $\mathcal{NP}$–vollständig, falls c den diskretisierten euklidischen Abstand zwischen zwei Städten mißt oder c der Manhattan–Metrik entspricht [GJ79].

Das Problem gehört zu den sehr intensiv untersuchten Tourenplanungsproblemen [LLKS85]. Es gibt unzählige Erweiterungen des Problems, um reale Problemstellungen zu modellieren, wie z.B. Zeitfenster, Reihenfolgebedingungen und Kapazitätsbeschränkungen, die das Problem meist nicht einfacher machen.

Pickup–And–Delivery

Um den Bogen von TSP zum Pickup–And–Delivery–Problem zu schlagen, plazieren wir einige Objekte auf den Knoten des TSP–Graphen und ordnen jedem Objekt einen Auftrag zu. Ein Roboter (oder ein Fahrzeug) soll die Aufgabe lösen, m heterogene Objekte, die sich auf n verschiedenen Knoten $s_i \in \mathcal{S}$ eines Graphen $G = (\mathcal{S}, E)$ befinden, von ihren Ursprungsknoten auf vorgegebene Zielknoten zu bewegen. Ein Objekt entspricht einem *Auftrag* r, der durch ein geordnetes Paar von Knoten $r = (s_i, s_j)$ beschrieben wird, wobei $s_i \in \mathcal{S}$ der Ausgangsknoten und $s_j \in \mathcal{S}$ der Zielknoten ist. Die Menge der Aufträge bezeichnen wir mit $\mathcal{R}, |\mathcal{R}| = m$. Der Roboter darf in der Grundversion nur ein Objekt auf einmal bewegen.

Problem 4.3 (Pickup–And–Delivery (PDP))

Gegeben: Gerichteter Graph $G = (\mathcal{S}, A)$, eine nichtnegative Kostenfunktion $l : A \to \mathbb{R}_+$ auf den Kanten, ein Startknoten $s_0 \in \mathcal{S}$ sowie eine Menge $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ von Aufträgen.

Gesucht: Eine kostenminimale Tour über G, die in s_0 startet und endet, so daß alle Aufträge aus $\mathcal{R}$ transportiert werden.

In der Literatur wird zwischen einer *präemptiven* und einer *nicht–präemptiven* Version des Problems unterschieden. In der zuletzt genannten darf ein einmal aufgenommenes Objekt erst wieder am Zielknoten abgelegt werden. In der präemptiven Variante darf der Roboter Objekte

vorzeitig ablegen und zunächst andere Objekte transportieren. Irgendwann muß er allerdings das Objekt wieder aufnehmen und weitertransportieren, um die Aufgabenstellung zu lösen. Diesen Vorgang werden wir *Umladen* nennen und einen Knoten, an dem umgeladen wird, *Umladeknoten*.

Beide Varianten sind auf allgemeinen Graphen $\mathcal{NP}$–vollständig, da sie jeweils das TSP enthalten: Kopiere jeden Knoten v des Graphen und füge die Kopie mit Distanz 0 zum Originalknoten in den Graphen ein. Für jede solche Kopie v' erzeugen wir einen Auftrag (v, v'). Eine kostenminimale Lösung für das Transportproblem ist zugleich eine Lösung für das TSP mit gleichen Kosten.

Einen Überblick über Verallgemeinerungen des PDP findet man in [SS95]. Wie bei dem TSP sind diverse Nebenbedingungen denkbar, die reale Anforderungen in das Modell einbeziehen. Falls $\mathcal{NP} \neq \mathcal{P}$ gilt, ist es nicht möglich, im komplexitätstheoretischem Sinne effiziente Verfahren zu entwickeln, die das Problem optimal lösen. Das Modell ist auf der anderen Seite für reale Anwendungen von Tourenplanungsproblemen interessant und die meisten Anwender sind mit einer relativ guten Lösung, die in akzeptabler Zeit zur Verfügung steht, zufrieden. Dies motiviert heuristische Lösungsverfahren, die auf Metaheuristiken oder dem Lösen gemischt–ganzzahliger Programme basieren [Sol94], [Mue00], [Oer00].

Wir sind bisher davon ausgegangen, daß die zu transportierenden Objekte heterogen sind, also nicht untereinander ausgetauscht werden können. Wenn wir das jedoch zulassen, sprechen wir von *homogenen* Gütern und jeder Knoten des Graphen besitzt dann eine gewisse Nachfrage und ein gewisses Angebot nach diesem Gut. Solche Probleme werden meist unter dem Stichwort *Vehicle–Routing–Problem* geführt. Ein oder mehrere Fahrzeuge mit eingeschränkter Kapazität sollen mit möglichst kurzer Gesamtdistanz alle Nachfragen und Angebote befriedigen. Da dies wiederum ein ganz eigener Themenbereich ist, verweisen wir auf [TV01].

4.2. Pickup–And–Delivery auf speziellen Graphenklassen

Wir interessieren uns in dieser Arbeit für spezielle Graphenklassen und fragen jeweils, ob das Problem mit polynomiellem Zeitaufwand lösbar ist oder ob es zu den schwierigen Problemen gezählt werden muß. Möglicherweise existieren dann Algorithmen, die zwar nicht die optimale Lösung, aber eine gute und vielleicht sogar beweisbare Näherung liefern können. In diesem Abschnitt geben wir zunächst einen Überblick über eng verwandte Probleme und Resultate, danach richten wir unsere Aufmerksamkeit auf die Problemstellung, die im Zentrum dieser Arbeit steht.

4.2.1. Bisherige Arbeiten

Da das Problem auf allgemeinen Graphen sehr schwierig zu lösen ist, haben sich zahlreiche Autoren mit speziellen Graphenklassen beschäftigt, d.h. daß der Graph G und die dazugehörige Distanzfunktion $l : A \rightarrow \mathbb{R}_+$ beispielsweise einen Pfad, Kreis oder Baum beschreibt. Wir gehen immer davon aus, daß G ungerichtet ist und somit die Distanzen zwischen benachbarten Knoten symmetrisch sind. Allgemein scheint die präemptive Variante, in der an jedem Knoten umgeladen werden darf, auf den von uns betrachteten Graphenklassen einfacher als die nicht–präemptive Version zu sein.

Atallah und Kosaraju beschreiben in [AK88] ein Modell für den Pfad und den Kreis, mit dem das Problem in polynomieller Zeit sowohl im präemptiven als auch im nicht–präemptiven Fall gelöst werden kann. In der Anwendung, die zur Motivation herangezogen wird, ist ein Roboterarm an einem Ende auf einer Drehachse montiert. Um ihn herum befinden sich im Kreis angeordnete Stationen, die er mit seinem Greifarm erreichen kann. Auf einigen Stationen liegen Objekte, die zu anderen Stationen befördert werden sollen. Diese Arbeit stützt sich auf einige Resultate des genannten Artikels. Wir werden uns auch sprachlich an den darin enthaltenen Definitionen orientieren.

Die von den beiden Autoren vorgestellten Algorithmen benötigen im präemptiven Fall auf dem Pfad und auch auf dem Kreis $O(m+n)$ Zeitschritte. Im nicht–präemptiven Fall werden $O(m+n\log n)$ Schritte benötigt [Fre93]. Zusammen mit Guan konnte Frederickson ebenfalls zwei Algorithmen für den präemptiven Fall angeben, falls der zugrunde liegende Graph ein Baum ist. Die Laufzeit des ersten Algorithmus ist abhängig von der Anzahl der Zusammenhangskomponenten q im Graphen $(\mathcal{S},\mathcal{R})$, wobei $\mathcal{S}$ die Knoten des Baumes und $\mathcal{R}$ die Aufträge sind. Sie beträgt $O(m+qn)$. Der zweite Algorithmus benötigt $O(m+n\log n)$ Zeitschritte [FG92].

Falls der zugrunde liegende Graph ein Baum ist, ist das Problem schwierig, wenn die Objekte zwischenzeitlich nicht abgelegt werden dürfen. In einem weiteren Artikel zeigen die beiden Autoren, daß das nicht–präemptive PDP auf Bäumen $\mathcal{NP}$–vollständig ist [FG93]. Es werden verschiedene Approximationsalgorithmen präsentiert, von denen der bezüglich seiner Approximationsgüte Beste eine Güte von $5/4$ erreicht. Durch Kombination verschiedener Algorithmen kann ein Approximationsverhältnis von 1.21363 erzielt werden.

Hauptmeier et al. präsentieren in [HKRW01] einen schärferen Beweis der $\mathcal{NP}$–Vollständigkeit. Sie zeigen, daß das Problem schon auf Kammgraphen schwierig ist. Ein Kammgraph besteht aus einem Pfad, an den zusätzliche Knoten von Grad eins angehangen werden. Kammgraphen sind also spezielle Bäume, da sie keine Kreise enthalten. In Kapitel 6 werden wir detaillierter auf das Transportieren auf Bäumen eingehen.

Muslea untersuchte in seiner Master–Thesis [Mus96] besondere Auftragsstrukturen, die alle nicht–präemptiv zu erledigen sind. Wenn die Aufträge alle in der Wurzel eines zugrunde liegenden Baumes beginnen oder zu einem gemeinsamen Zielknoten hinführen, gibt der Autor Algorithmen mit linearer bzw. quadratischer Laufzeit an. In [Mus97] untersucht Muslea das Problem, das nicht nur ein Fahrzeug die Aufträge bedienen darf, sondern daß allgemein d Fahrzeuge verplant werden können. Jedoch sind alle zuvor genannten Varianten schon für $d \geq 2$ $\mathcal{NP}$–vollständig, sogar wenn der zugrunde liegende Baum ungewichtet und binär ist. Für $d = 2$ werden Approximationsalgorithmen mit Güte 2 angegeben. Wenn nicht anders vermerkt, betrachten wir in dieser Arbeit immer den Fall $d = 1$.

Hauptmeier et al. betrachten ebenfalls eine besondere Auftragsstruktur und gehen davon aus, daß die Aufträge partiell geordnet sind. Für zwei Aufträge $r_1, r_2 \in \mathcal{R}$ kann eine Ordnung $r_1 \prec r_2$ definiert sein. Der Auftrag r_2 darf erst dann transportiert werden, nachdem r_1 erledigt ist. Für den Fall des Pfades wird ein polynomieller Algorithmus angegeben, der die optimale Lösung berechnet. Für den Fall, daß der zugrunde liegende Graph ein Baum ist, wird ein Approximationsverfahren mit Güte $5/3$ vorgestellt [HKRW01].

Guan hat die Problemstellung verallgemeinert und führt eine Kapazität $c \in \mathbb{N}$ als Eingabegröße in einer erweiterten Fragestellung ein. Er stellt jedoch fest, daß das Problem ohne Umladen sofort $\mathcal{NP}$–vollständig wird, falls mehr als ein Objekt gleichzeitig transportiert werden darf. Auch hier zeigt sich, daß das präemptive Problem einfacher ist, denn er konnte einen Algorithmus mit Laufzeit $\mathcal{O}(m+n)$ für den Pfad präsentieren. Auf Bäumen und sogar schon auf Sternen ist es jedoch $\mathcal{NP}$–vollständig. Hingegen ist der Komplexitätsstatus für diese Variante auf dem Kreis bislang offen [Gua98].

Charikar und Raghavachari untersuchten approximative Verfahren für die nicht–präemptive Variante auf höhenbalancierten Bäumen und dem Pfad mit Fahrzeugkapazität $c \in \mathbb{N}$. Der Algorithmus für höhenbalancierte Bäume erreicht eine Güte von $\mathcal{O}(\sqrt{k})$, das Verfahren für den Pfad liefert eine 2–Approximation. Im Falle von Präemption und allgemeiner Kapazität c können sie einen Algorithmus für Bäume mit Güte 2 angeben [CR98].

In [CR98] wird außerdem das Verhältnis von nicht–präemptiven Lösungen zu präemptiven Lösungen auf den gleichen Eingabeinstanzen untersucht. Hierbei wird wieder eine allgemeine Kapazität $c \in \mathbb{N}$ unterstellt. Es werden Instanzen angegeben, für die das Verhältnis der Kosten einer optimalen nicht–präemptiven Lösung zu den Kosten einer optimalen präemptiven Lösung $\Omega(c^{2/3})$ beträgt.

Häufig ist es interessant, neben einer pessimistischen Abschätzung (unter Annahme des schlimmstmöglichen Falls) die Laufzeit der Algorithmen im praktischen Einsatz zu untersuchen. Wenn keine Daten aus realen Anwendungen zur Verfügung stehen, behilft man sich häufig mit Zufallsin-

stanzen. Coja–Oghlan et al. haben die aus Zufallsinstanzen entstehenden Strukturen betrachtet und in [COKN03] festgestellt, daß sie mit hoher Wahrscheinlichkeit einfach und effizient zu lösen sind. Mit diesem Thema werden wir uns im Kapitel 7 beschäftigen.

4.2.2. Unser Beitrag

Wir wollen die Konzepte präemptiv bzw. nicht–präemptiv verallgemeinern und dem Roboter eine Zahl $k \in \mathbb{N}, k \leq n$ mitteilen, die die maximale Anzahl an verwendeten Umladestationen limitiert. Die potentiellen Umladestationen können *exogen* durch eine Menge $B' \subseteq \mathcal{S}, |B'| = k$ gegeben sein, und der Roboter muß eine Teilmenge $B \subseteq B'$ bestimmen, die als Umladeknoten benutzt werden sollen. Oder der Roboter muß *endogen* entscheiden, welche k Knoten aus $\mathcal{S}$ als Umlademöglichkeit genutzt werden sollen, damit die Gesamtkosten minimiert werden.

In beiden Fällen führen wir positive *Umladekosten* $\Delta \in \mathbb{R}_+$ ein, die pro Umladevorgang bezahlt werden müssen. Die im Kapitel 5 vorgestellten Ergebnisse lassen sich auch auf negative Umladekosten erweitern. Falls $\Delta \leq 0$, ist es in jedem Fall günstig, alle $k \leq n$ Umladeknoten auszunutzen, auch wenn dies die Strecke, die der Roboter zurücklegt, nicht verkürzen sollte. Jedoch werden die Gesamtkosten jeder Lösung, die wir später genauer definieren werden, immer um $k\Delta$ reduziert. Wir wollen auf diesen vergleichsweise einfachen Fall nicht weiter eingehen und nehmen somit $\Delta > 0$ an, wodurch Fallunterscheidungen in einigen Beweisen vermieden werden.

Die von uns betrachteten Varianten des PDP werden im folgenden formal definiert. Während wir an dieser Stelle allgemein von Umladevorgängen sprechen, werden wir im folgenden Kapitel sehen, wie wir dies graphentheoretisch interpretieren können.

Problem 4.4 (Exogenes semi–präemptives PDP)

Gegeben: Gerichteter Graph $G = (\mathcal{S}, A)$, eine nichtnegative Kostenfunktion $l : A \to \mathbb{R}_+$, ein Startknoten $s_0 \in \mathcal{S}$, eine Menge $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ von Aufträgen, eine Menge potentieller Umladeknoten $B' \subseteq \mathcal{S}$ und Umladekosten $\Delta > 0$.

Gesucht: Eine kostenminimale Tour über G, die in s_0 startet und endet, so daß alle Aufträge aus $\mathcal{R}$ transportiert werden. Umladevorgänge dürfen nur an den Knoten B' stattfinden.

Problem 4.5 (Endogenes semi–präemptives PDP)
Gegeben: Gerichteter Graph $G = (\mathcal{S}, A)$, eine nichtnegative Kostenfunktion $l : A \to \mathbb{R}_+$, ein Startknoten $s_0 \in \mathcal{S}$, eine Menge $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ von Aufträgen, eine Zahl $k \leq n$ und Umladekosten $\Delta > 0$.
Gesucht: Eine kostenminimale Tour über G, die in s_0 startet und endet, so daß alle Aufträge aus $\mathcal{R}$ transportiert werden. Es dürfen höchstens k Umladevorgänge stattfinden.

Die Eingaben für die exogene bzw. endogene Problemstellung werden wir mit $I = (\mathcal{S}, l, \Delta, \mathcal{R}, B')$ bzw. $I = (\mathcal{S}, l, \Delta, \mathcal{R}, k)$ abkürzen, wobei wir davon ausgehen, daß der Startknoten $s_0 \in \mathcal{S}$ besonders gekennzeichnet ist und nicht explizit in unserer abkürzenden Schreibweise aufgelistet werden muß.

Wir werden in Kapitel 5 sehen, daß die exogene Version in unserer Modellierung einfacher zu lösen ist als die endogene Variante. Trotzdem verhalten sie sich hinsichtlich einer groben Unterscheidung in Probleme aus $\mathcal{P}$ und $\mathcal{NP}$–vollständige Probleme ähnlich in dem Sinne, daß sie auf den gleichen Graphenklassen in polynomieller Zeit lösbar bzw. $\mathcal{NP}$–vollständig sind. Da beide Varianten Verallgemeinerungen der nicht–präemptiven Problemstellung sind (wähle $k = n$ bzw. $B' = V$), sind sie auf Bäumen und für $c \geq 2$ schwierig zu lösen.

Kapitel 6 ist dem Fall gewidmet, daß der zugrunde liegende Graph ein Baum ist. Wir werden einen Approximationsalgorithmus für das exogene Transportproblem angeben, der Lösungen mit Kosten produziert, die höchstens um den Faktor $4/3$ von den Kosten einer Optimallösung entfernt sind. Zudem können wir ein Beispiel angeben, für das die angegebene Güte angenommen wird. Danach greifen wir auf das in Kapitel 3 beschriebene Näherungsverfahren für kostenminimale (d, r)–Arboreszenzen zurück und benutzen es, um einen $(4/3+\varepsilon)$–approximativen Algorithmus für das endogene Transportproblem auf Bäumen zu konstruieren.

	$c = 1$	$c \geq 2$
präemptives PDP		
Pfad	$\mathcal{O}(m + n)$ [AK88]	$\mathcal{O}(m + n)$ [Gua98]
Kreis	$\mathcal{O}(m + n)$ [AK88]	offen
Baum	$\mathcal{O}(m + n \log n)$ [FG92]	$\mathcal{NP}$–vollst. [Gua98]
nicht–präemptives PDP		
Pfad	$\mathcal{O}(m + n \log n)$ [AK88]	$\mathcal{NP}$–vollst. [Gua98]
Kreis	$\mathcal{O}(m + n \log n)$ [Fre93]	$\mathcal{NP}$–vollst. [Gua98]
Baum	$\mathcal{NP}$–vollst. [FG93]	$\mathcal{NP}$–vollst. [Gua98]
semi–präemptives PDP (exogen)		
Pfad	$\mathcal{O}(m + n^2)$	$\mathcal{NP}$–vollst.
Kreis	$\mathcal{O}(m^2 + mn^2)$	$\mathcal{NP}$–vollst.
Baum	$\mathcal{NP}$–vollst.	$\mathcal{NP}$–vollst.
semi–präemptives PDP (endogen)		
Pfad	$\mathcal{O}(n^3 m \log n)$	$\mathcal{NP}$–vollst.
Kreis	$\mathcal{O}(n^3 m^2 \log n)$	$\mathcal{NP}$–vollst.
Baum	$\mathcal{NP}$–vollst.	$\mathcal{NP}$–vollst.

Tabelle 4.1.: *Komplexitätsergebnisse für das Pickup–And–Delivery–Problem, abhängig von den zugrundeliegenden Graphenklassen und Umladerestriktionen*

Nachdem wir im Stande sind, für beide Probleme auf Bäumen Näherungslösungen zu generieren, benutzen wir ein Resultat von Charikar et al., mit dem Optimallösungen unserer Probleme auf gewichteten Graphen, für die wir nur die Dreiecksungleichung fordern, durch Lösen weniger Instanzen auf Bäumen abgeschätzt werden können.

Die Tabelle 4.1 faßt die Komplexitäts–Resultate für den präemptiven, nicht–präemptiven sowie den semi–präemptiven Fall auf den jeweiligen Graphenklassen zusammen.

Kapitel 5.

Semi–Präemptives Transportieren auf Pfaden und Kreisen

5.1. Problembeschreibung

Wir betrachten das semi–präemptive Pickup–And–Delivery–Problem auf Kreisen und Pfaden. Als Eingabe erhalten wir eine Menge von n Knoten $\mathcal{S}$, die entweder auf einem Kreis in der Ebene oder auf einem Pfad angeordnet sind. Ein ausgezeichneter Knoten $s_0 \in \mathcal{S}$ ist zugleich Start– und Endknoten. Im Falle des Kreises hat jeder Knoten genau zwei Nachbarn. Falls der zugrunde liegende Graph ein Pfad ist, haben die beiden Endknoten einen und die inneren Knoten zwei Nachbarn. Zwei benachbarten Knoten s_i und s_j ist eine positive Distanz $l(s_i, s_j) = l(s_j, s_i) \in \mathbb{R}_+$ zugewiesen. Weiterer Bestandteil der Eingabe ist eine Menge $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, durch die m Aufträge beschrieben sind, eine natürliche Zahl $k \leq n$ als Obergrenze der Anzahl benutzter Umladeknoten sowie die Kosten pro Umladevorgang $\Delta \in \mathbb{R}_+$.

Ein Knoten kann Ausgangs– und Zielort für mehrere Aufträge sein, er muß aber von mindestens einem Objekt als solches benutzt werden. Wir können diese Annahme ohne Einschränkung treffen, denn andernfalls löschen wir den betreffenden Knoten und passen die Distanz seiner Nachbarn zueinander entsprechend an. Der Roboter startet an einem

Startknoten $s_0 \in \mathcal{S}$ und soll, nachdem er alle Objekte vorschriftsmäßig transportiert hat, wieder zum Startknoten zurückkehren.

Wir interessieren uns für den kürzesten Bewegungsablauf, den der Roboter absolvieren muß, um alle Objekte zu bewegen und wieder zum Startknoten zurückzukehren. In der Sprache der Graphentheorie suchen wir einen kostenminimalen eulerschen Graphen, der mindestens alle Kanten aus $\mathcal{R}$ enthält. Ein Umladevorgang kann als Zerschneiden von Auftragskanten betrachtet werden. Jedoch dürfen wir eine solche Kante nur an Umladeknoten zerschneiden und verursachen dadurch zusätzliche Kosten in Höhe von Δ.

Falls der zugrunde liegende Graph ein Pfad ist, können wir die beiden Endknoten zu Nachbarn erklären und durch einen hinreichend großen Distanzwert verbinden. Durch diesen Trick könnten wir den Pfad als Spezialfall des Kreises betrachten. Auf die Unterschiede werden wir an geeigneter Stelle hinweisen.

5.2. Zirkeltraining

Bis auf Lemma 5.5 basieren alle Definitionen, Sätze und Beweise dieses Abschnittes auf der Arbeit von Atallah und Kosaraju [AK88]. Da sie für das Verständnis der darauf aufbauenden Erkenntnisse notwendig sind, werden sie hier teilweise auch detailliert wiedergegeben.

Beginnend am Startknoten s_0 werden alle anderen Knoten im Uhrzeigersinn numeriert, so daß $\mathcal{S} = \{s_0, \ldots, s_{n-1}\}$ und s_i, s_{i+1} benachbart sind, $i = 0, \ldots, n-1$. Zur Vereinfachung der Schreibweise werden wir oft s_{i+1} schreiben und meinen damit $s_{i+1 \bmod n}$, somit ist $s_n = s_0$. Bezugnehmend auf den Kreis in der Ebene nennen wir den Kreisabschnitt zwischen den Stationen s_i und s_{i+1} mit Distanz $l(s_i, s_{i+1})$ *Intervall* i. Da ein Objekt nur entlang benachbarter Knoten verschoben werden darf, muß für jede Lösung des Problems entschieden werden, ob ein Objekt *im Uhrzeigersinn* oder *gegen den Uhrzeigersinn* entlang des Kreises transportiert werden soll. Die *Drehrichtung* eines Auftrages gibt an, über welche der beiden genannten Möglichkeiten das betreffende Objekt bewegt

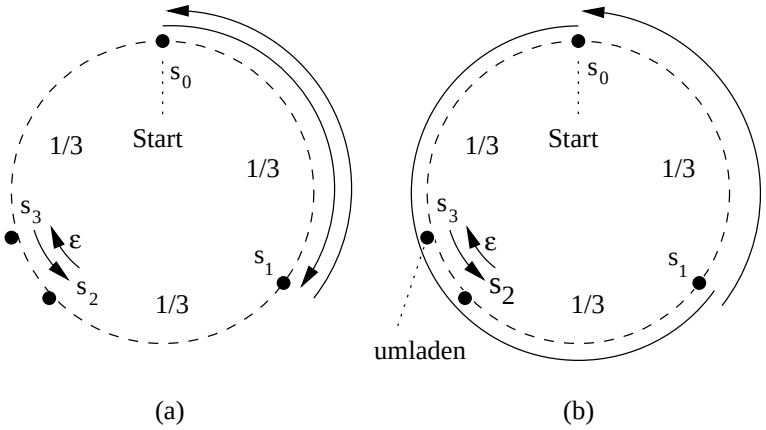


Abbildung 5.1.: *Situation auf dem Kreis, bei der es günstiger ist, einen Auftrag über den langen Weg zu transportieren. (a) Eingabeinstanz mit Auftragskanten über den kürzeren Weg eingezeichnet. (b) Lösung mit Umladen. [AK88]*

wird. Wenn die Drehrichtung festgelegt ist, kann jedem Objekt ein eindeutiger einfacher Pfad entlang des Kreises zugeordnet werden, der nur benachbarte Knoten benutzt. Die Strecke, die bei dem Transport eines Objektes benötigt wird, hängt somit von der Drehrichtung ab und entspricht der Summe der Distanzen entlang dieses Pfades. Wir gehen außerdem davon aus, daß eine der beiden Möglichkeiten länger ist als die andere. Falls dem nicht so sein sollte, können wir die Distanzen perturbieren und somit Eindeutigkeit herstellen. Wir werden bezüglich eines Auftrages häufig von dem *langen* oder *längeren* Pfad entlang des Kreises sprechen. Sei $e = (s_a, s_b)$ eine Kante entlang des Kreises. Dann meinen wir mit $e^c = (s_a, s_b)$ die Kante, die der entgegengesetzten Drehrichtung von e entspricht.

Abbildung 5.1 demonstriert, daß es Eingaben gibt, bei der eine optimale Lösung den längeren Pfad bevorzugt. Auf der linken Seite ist der Kreis, bestehend aus vier Knoten und Abständen $l(s_0, s_1) = l(s_1, s_3) = l(s_3, s_0) = \frac{1}{3}$ sowie $l(s_2, s_3) = \varepsilon$. Zudem sind vier Aufträge gege-

ben und in der Darstellung (a) über den kürzeren Pfad eingezeichnet: $\mathcal{R} = \{(s_0, s_1), (s_1, s_0), (s_2, s_3), (s_3, s_2)\}$. Um den Graphen zu einer Lösung zu erweitern, müssen augmentierende Kanten eingefügt werden, so daß er eulersch wird. Eine minimale Erweiterung für (a) beinhaltet z.B. die Kanten $(s_1, s_2), (s_2, s_1)$. Die Gesamtkosten betragen dann $\frac{4}{3} + 2\varepsilon$. Wählt man jedoch wie in (b) illustriert für den Auftrag (s_0, s_1) den längeren Weg, also in diesem Fall den gegen den Uhrzeigersinn, dann kann der Transport des Objektes am Knoten s_3 unterbrochen werden. Bevor der Weg zum Knoten s_1 fortgesetzt wird, werden erst die Aufträge (s_3, s_2) und (s_2, s_3) erfüllt. In diesem Fall kommt man ohne zusätzliche Kanten aus, und die zurückgelegte Distanz beträgt $1 + 2\varepsilon$.

Wir wollen einen *Transportgraphen* $G_T = (\mathcal{S}, A, B)$ als Lösung für das semi–präemptive PDP konstruieren, wobei $(\mathcal{S}, A)$ ein gerichteter Multigraph ist, der eine Eulertour enthält. Die Eulertour muß für alle Kanten $(s_a, s_c) \in \mathcal{R}$ einen gerichteten Pfad enthalten, der nur Kanten aus B sowie s_a und s_c als Start– und Endknoten enthält. Wird auf diesem Pfad ein Knoten aus B benutzt, so nennen wir diesen Vorgang *Umladen*. Bei der Konstruktion eines Transportgraphen muß jeder Kante eine Drehrichtung zugewiesen werden. Dann kann einer Kante $(s_a, s_c) \in \mathcal{R}$ mit festgelegter Drehrichtung ein eindeutiger Pfad entlang des Kreises zugeordnet werden. Falls dieser Pfad einen Knoten s_b enthält, so sagen wir auch, daß (s_a, s_c) den Knoten s_b *kreuzt* oder *überdeckt*. In diesem Fall kann eine Kante $(s_a, s_c) \in \mathcal{R}$ durch die Kanten $(s_a, s_b), (s_b, s_c)$ mit $s_b \in B$ ersetzt werden.

Wir werden die Kantenmenge A in drei Teilmengen $A = A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C$ partitionieren. Die Kanten in $A_{\mathcal{R}}$ dienen dem Transport der Aufträge. Wir wählen zunächst $A_{\mathcal{R}} = \mathcal{R}$, d.h. wir beginnen mit der Konstruktion von G_T, indem wir vorerst jeden Aufträge als einzelne Kante einfügen. Dazu muß jedoch die Richtung (im oder gegen den Uhrzeigersinn) für jeden Auftrag festgelegt werden. Wir werden in Lemma 5.10 sehen, daß maximal eine Kante über den längeren Weg transportiert werden muß. Im folgenden gehen wir davon aus, daß wir die Drehrichtung jedes Auftrages fest gewählt haben. Im Algorithmus, den wir am Ende des Kapitels angeben, werden alle möglichen Drehrichtungen jedes Auftrages enumeriert. Die Kantenmengen A_ψ und $A_{\mathcal{R}}$ werden nun im folgenden erklärt. Dabei

werden wir feststellen, daß das Eulerkriterium einige Kanten erzwingt. Diese werden wir A_ψ nennen. Andererseits werden wir weiterer Kanten benötigen, um Zusammenhang herzustellen. Diese Kanten werden wir mit A_C bezeichnen. Die *Kosten eines Transportgraphen* G_T definieren wir wie folgt:

$$c(G_T) = l(A_\psi) + l(A_\mathcal{R}) + l(A_C) + |B|\Delta$$

Da G_T eulersch ist, müssen nach Lemma 2.1 alle Knoten geraden Grad besitzen. Eine alternative Charakterisierung dieser Eigenschaft erhalten wir mit Einführung der folgenden Definition und des darauffolgenden Lemmas.

Definition 5.1 (Fluß). *Der* Fluß $\phi(i)$ über das Intervall i *ist die Anzahl der Kanten, die* i *im Uhrzeigersinn kreuzen, minus der Anzahl der Kanten, die* i *gegen den Uhrzeigersinn kreuzen.*

Es sei bemerkt, daß $\phi(i)$ auch Kanten zählt, die weder in s_i noch s_{i+1} enden. Man kann sich überlegen, daß die $\phi(i), i = 0, \ldots, n-1$ alle zusammen in $\mathcal{O}(m)$ Zeitschritten berechnet werden können: Man zählt zunächst, wieviele Kanten das Intervall 0 kreuzen. Für die Werte des Intervalls $i+1$ schreibt man den Wert $\phi(i)$ mittels einer geeigneten Datenstruktur fort, indem man nur in s_i bzw. s_{i+1} endende und startende Kanten betrachtet. Genau dann, wenn die Flüsse über alle Intervalle gleich einer konstanten Zahl $\psi \in \mathbb{Z}$ sind, sind auch alle Knoten gradbalanciert:

Lemma 5.2. $\forall_{s_i \in \mathcal{S}} : \delta^-(s_i) = \delta^+(s_i) \Leftrightarrow \forall_{i=0,\ldots,n-1} : \phi(i) = \psi$

Beweis. Es genügt zu zeigen, daß die Gradbalance gilt, genau dann, wenn für beliebiges i $\phi(i) = \phi(i+1)$ gilt. Wenn alle Knoten gradbalanciert sind, enden in s_i genauso viele Kanten wie starten. Zwei Kanten, die in entgegengesetzter Richtung in s_i enden und starten, erhöhen in der Summe weder $\phi(i)$ noch $\phi(i+1)$. Zwei Kanten, die in gleicher Richtung verlaufen und in s_i enden und starten, erhöhen bzw. erniedrigen $\phi(i)$ und $\phi(i+1)$ gleichermaßen. Kanten, die über i hinweg gehen, aber nicht in s_i enden oder starten, wirken sich ebenso auf $\phi(i)$ und $\phi(i+1)$ aus.

Die Rückrichtung gilt analog: Weisen alle Intervalle den gleichen Flußwert ψ auf, dann kann die Gradbalance von s_i nur verändert werden, wenn

der Knoten Kopf oder Schwanz einer Kante ist. Dann muß aber eine weitere Kante in s_i starten bzw. enden, ansonsten gölte $\phi(i+1) \neq \phi(i)$. □

Kennen wir einen optimalen Flußwert ψ, also einen Flußwert, mit dem eine optimale Lösung angenommen wird, dann gibt es einige Kanten, die im dazugehörigen Transportgraphen enthalten sein müssen. Da die Distanz zweier nicht–benachbarter Knoten auf dem Kreis gleich der Summe der Einzeldistanzen über den (nach Festlegung der Drehrichtung) eindeutigen Pfad ist, können wir uns anstelle von Kanten zwischen nicht–benachbarten Knoten auf Kanten zwischen benachbarten Knoten zurückziehen ohne die Allgemeinheit einzuschränken. Lemma 5.2 gibt uns für festes ψ neben der Richtung auch die Anzahl der notwendigen augmentierenden Kanten über alle Intervalle: Der Graph $(\mathcal{S}, \mathcal{R})$ ist kostenminimal gradbalanciert, wenn wir

$$\text{für alle } i : |\psi - \phi(i)| \text{ viele Kanten } \left\{ \begin{array}{ll} (s_i, s_{i+1}) & \text{falls } \phi(i) \leq \psi \\ (s_{i+1}, s_i) & \text{falls } \phi(i) > \psi \end{array} \right\} \quad (5.1)$$

einfügen. Wir werden sehen, daß in einem optimalen Transportgraphen nicht zu viele Werte für ψ angenommen werden können. Für fixes ψ sei A_ψ die Menge der nach obiger Vorschrift ermittelten Kanten.

Definition 5.3. *Seien ψ und ein $r \in \mathcal{R}$ gegeben. Dann bezeichnen wir mit*

$$\mathtt{bal}(\psi, r) = (\mathcal{S}, A_{\mathcal{R}} \dot\cup A_\psi)$$

den gradbalancierten Graphen*, in dem einzig der Auftrag r über die längere Drehrichtung transportiert wird und alle Intervalle den Flußwert ψ besitzen.*

Abbildung 5.2 verdeutlicht die Konstruktion von $\mathtt{bal}(\psi, r)$ an einem Beispiel mit $\psi = 0$ und $r = \emptyset$, d.h. alle Objekte werden über den kurzen Weg transportiert.

Bemerkung 5.4. *A_ψ ist unabhängig davon, ob eine Auftragskante wegen eines Umladevorgangs in mehrere Teilstücke zerlegt wird, da $\phi(i)$ davon unberührt ist.*

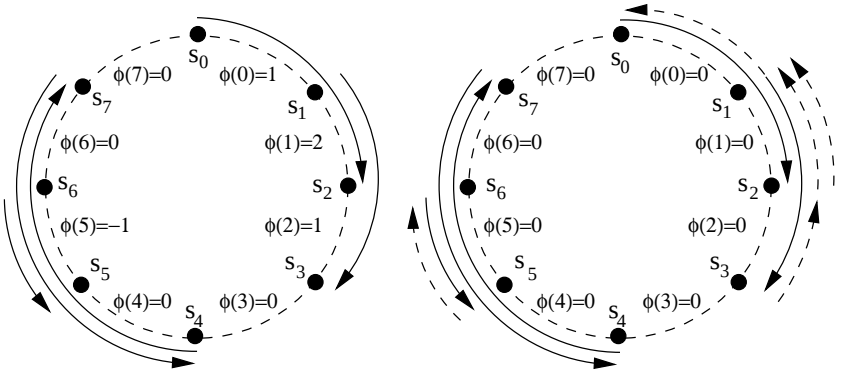


Abbildung 5.2.: *(a) Beispielgraph mit allen Aufträgen über den kurzen Weg gewählt und Flußwerten* ϕ*. (b) Der dazugehörige Graph* $\mathrm{bal}(0, \emptyset)$ *mit augmentierenden Kanten, so daß* $\phi(i) = 0$ *für alle Intervalle* i*.*

Wir haben eingangs gesagt, daß die Kanten des von uns zu konstruierenden Transportgraphen in drei Teilmengen disjunkt aufgeteilt sind. Für die folgenden Lemmata ist es hilfreich, wenn wir dies allgemein für jeden Transportgraphen annehmen können.

Lemma 5.5. *Die Kantenmenge* A *jedes Transportgraphen* $G_T = (\mathcal{S}, A, B)$ *kann beschrieben werden durch* $A = A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C$*.*

Beweis. Zähle für ein beliebiges Intervall i den Flußwert $\phi(i)$. Nach Lemma 5.2 gilt dieser Wert ebenso für alle anderen Intervalle. Dann ist $\psi := \phi(i)$. Sei $\mathcal{R}$ die Eingabemenge der Aufträge. Für jedes Paar $(s_a, s_b) \in \mathcal{R}, s_a, s_b \in \mathcal{S}$ können wir leicht (bezüglich ihrer Kantenkardinalität) kürzeste (s_a, s_b)–Pfade in A bestimmen, wobei wir alle Kanten, die in $\mathcal{S} \setminus (B \cup \{s_a, s_b\})$ anfangen oder enden, ignorieren. Diese Knoten dürfen nach Definition des Transportgraphen nicht auf einem Pfad liegen, den wir einem Auftrag zuordnen. Die Vereinigung dieser Pfade merken wir uns als $A_{\mathcal{R}}$. Falls die verbleibenden Kanten $A \setminus A_{\mathcal{R}}$ nicht ausschließlich zwischen

benachbarten Knoten verlaufen, so zerschneiden wir diese zu nicht weiter verkürzbaren Kanten A'. Da für alle Intervalle $\phi(i) = \psi$ gilt, können wir nun A_ψ konstruieren, indem wir entsprechende Kanten aus A' gemäß der Vorschrift 5.1 wählen. Schließlich setzen wir $A_C = A' \setminus A_\psi$. Der Graph $(\mathcal{S}, A_\mathcal{R} \dot\cup A_\psi \dot\cup A_C)$ ist eulersch, da G_T eulersch ist und das Zerschneiden von Kanten an Knoten auf ihrem eindeutigen Pfad verändert nicht die Gradbalance eines Knotens. Die Kosten sind identisch, da wir nur Kanten in kleinere Kanten zerteilt haben, wobei die Summe der Distanzen erhalten bleibt. □

Lemma 5.6. *Sei* $G_T = (\mathcal{S}, A_\mathcal{R} \dot\cup A_\psi \dot\cup A_C, B)$ *ein kostenminimaler Transportgraph. Dann existiert ein Intervall* i*, so daß* A_ψ *höchstens eine Kante zwischen* s_i *und* s_{i+1} *enthält.*

Beweis. Angenommen, jede solche Gesamtkosten–minimale Augmentierung fügt mindestens zwei Kanten über jedes Intervall ein. Dann betrachte einen ungerichteten Kreis $K \subseteq A_\psi$ mit n Kanten, der alle Knoten besucht. Angenommen, die Gesamtkosten der Kanten in K, die im Uhrzeigersinn verlaufen, sind mindestens so hoch wie die Kosten der Kanten gegen den Uhrzeigersinn. Dann entferne diese und verdopple diejenigen gegen den Uhrzeigersinn. Die Optimalität, Gradbalance sowie Zusammenhangskomponenten bleiben erhalten. Jedoch reduziert sich die Anzahl der Kanten über mindestens einem Intervall. Im Umkehrfall verfahre analog. □

Das vorige Lemma hilft uns, mögliche Werte für den Flußwert ψ zu begrenzen.

Korollar 5.7. *Sei* $G_T = (\mathcal{S}, A_\mathcal{R} \dot\cup A_\psi \dot\cup A_C, B)$ *ein optimaler Transportgraph. Dann gilt:*

$$\psi \in [-m-1, m+1]$$

Beweis. Nach Lemma 5.6 gibt es ein Intervall i, in das maximal eine zusätzliche Kante eingefügt wird. Über jedes Intervall können höchstens m Auftragskanten $A_\mathcal{R}$ hinweg gehen. □

Gemäß Lemma 2.1 ist ein Graph genau dann eulersch, wenn alle Knoten gradbalanciert sind und der Graph zusammenhängend ist. Für festes ψ können wir Gradbalance in jedem Knoten herstellen, indem Lemma 5.2 und die darauffolgenden Bemerkungen angewendet werden. Der daraus resultierende Graph ist $\mathrm{bal}(\psi, r) := (\mathcal{S}, \mathcal{R} \cup A_\psi)$. Um Zusammenhang herzustellen, gibt es zwei Möglichkeiten: Wir können entweder weitere augmentierende Kanten einfügen oder einen Umladevorgang nutzen. Mit Verweis auf Lemma 2.3 können wir folgendes beobachten.

Bemerkung 5.8.

1. *Alle Aufträge einer Zusammenhangskomponente von* $\mathrm{bal}(\psi, r)$ *können ohne weitere augmentierende Kanten oder Umladen transportiert werden, wobei jeder Knoten der Zusammenhangskomponente als Startknoten gewählt werden kann.*

2. *Wenn zwei Zusammenhangskomponenten durch Einfügen von weiteren augmentierenden Kanten verschmolzen werden sollen, dann können wir uns auf benachbarte Zusammenhangskomponenten beschränken, also auf Paare von Zusammenhangskomponenten, bei denen mindestens zwei Knoten benachbart sind.*

Eine Konsequenz davon ist, daß wir jede Zusammenhangskomponente nur einmal besuchen und wieder verlassen müssen.

Sei $a \in A$ eine Kante in G_T. Dann bezeichnen wir mit $Z(e) \subseteq \mathcal{S}$ die starke Zusammenhangskomponente von $\mathrm{bal}(\psi, r)$, in der sich sowohl der Kopf als auch der Schwanz von a befindet. Wir sagen, e *überdeckt* eine Knotenmenge $V \subseteq \mathcal{S}$, falls für alle Knoten $v \in V$ gilt, daß e über v hinweg geht oder v Kopf bzw. Schwanz von e ist. Eine Knotenmenge $V \subseteq \mathcal{S}$ überdeckt eine zweite Menge $W \subseteq \mathcal{S}$, wenn die Vereinigung der Kanten, die in V starten oder enden, W überdeckt. Wenn $Z(e)$ eine Menge W überdeckt, dann kann an allen Knoten aus W eine Kante aus $Z(e)$ zerschnitten und somit die zugehörigen Zusammenhangskomponenten verschmolzen werden.

Lemma 5.9. *Sei* $G_T = (\mathcal{S}, A, B)$ *ein optimaler Transportgraph und* $e \in A$ *ein langer Auftrag. Dann überdeckt* $Z(e)$ *ganz* $\mathcal{S}$.

Beweis. Angenommen, $Z(e)$ überdeckt nicht den ganzen Kreis $\mathcal{S}$. Dann ist der auf die Kanten von $Z(e)$ eingeschränkte Fluß Null und jedes von $Z(e)$ überdeckte Intervall enthält mindestens zwei gegenläufige Kanten. Wenn wir e durch e^c ersetzen, ändert sich nichts an der Gradbalance der Knoten oder den Zusammenhangskomponenten. Die Menge der Knoten, die $Z(e)$ in G_T überdeckte, ist eine Teilmenge der Knoten, die $Z(e^c)$ in $G_T - e + e^c$ überdeckt. Außerdem ist $c(G_T) > c(G_T - e + e^c)$, Widerspruch. □

Lemma 5.10. *In einem optimalen Transportgraphen G_T wird maximal ein Objekt entlang seines langen Weges transportiert.*

Beweis. Sei ψ der in G_T vorliegende Flußwert und Z_i sowie Z_j zwei Zusammenhangskomponenten von $\mathtt{bal}(\psi, r)$, die jeweils einen langen Auftrag $e_i \in Z_i$ und $e_j \in Z_j$ enthalten. Nach Lemma 5.9 überdeckt sowohl Z_i als auch Z_j jeweils den ganzen Kreisumfang.

(i) $Z_i \neq Z_j$. Es existiert in G_T ein Pfad von s_0 nach Z_i sowie ein Pfad von s_0 nach Z_j. Zu einer der beiden Zusammenhangskomponenten muß jedoch ein Pfad von s_0 aus existieren, der keinen Knoten der anderen Zusammenhangskomponente benutzt, andernfalls wäre G_T nicht optimal. Ohne Einschränkung existiert somit ein Pfad von s_0 nach Z_i ohne einen Knoten aus Z_j zu benutzen. Ersetze e_j durch e_j^c. Da Z_i ganz $\mathcal{S}$ überdeckt, können alle Umladevorgänge, die über e_j durchgeführt wurden, zu gleichen Kosten über Kanten aus Z_i ersetzt werden. Der daraus resultierende Graph ist jedoch billiger, da $l(e_j^c) < l(e_j)$, Widerspruch.

(ii) $Z_i = Z_j$. Wir unterscheiden zwei weitere Fälle.

(ii.a) Jedes Intervall, das e_i überdeckt, wird auch von einer weiteren Kante aus Z_i überdeckt. Dann ist in $G_T - e_i + e_i^c$ weiterhin $Z(e_i^c) = Z_i$ und überdeckt den gesamten Kreis. Widerspruch zur Optimalität.

(ii.b) Mindestens ein Intervall wird nur von e_i und keiner weiteren Kante aus Z_i überdeckt. Dann ist der Flußwert, der nur Kanten aus Z_i berücksichtigt, $+1$ oder -1. Dadurch muß aber jedes Intervall, das sowohl von e_i als auch e_j überdeckt wird, von mindestens einer weiteren Kante aus Z_i überdeckt werden. Dann können aber beide langen Kanten durch

ihre komplementären Kanten ersetzt werden und in $G_T - e_i + e_i^c - e_j + e_j^c$ ist $Z(e_i^c) = Z(e_j^c) = Z_i$ und die induzierten Kanten überdecken ganz $\mathcal{S}$. Widerspruch. □

Um einen optimalen Transportgraphen zu bestimmen, werden wir iterativ einen Auftrag $r \in \mathcal{R} \cup \{\emptyset\}$ so orientieren, daß er über den längeren Weg entlang des Kreises transportiert wird. Lemma 5.10 besagt, daß in jedem optimalen Transportgraphen maximal ein Auftrag lang gewählt wird. Dadurch erhalten wir $m + 1$ Möglichkeiten. Zusätzlich werden alle Werte für $\psi \in [-m - 1, m + 1]$ enumeriert (Korollar 5.7). In den folgenden Abschnitten gehen wir somit davon aus, daß r und ψ fixiert sind, wodurch A_ψ und somit alle Zusammenhangskomponenten von $bal(\psi, r)$, die wir mit $Z_i, i = 0, \ldots, q$, abkürzen, determiniert sind. Analog zum Startknoten s_0 soll Z_0 diejenige Zusammenhangskomponente bezeichnen, die s_0 enthält. Zudem bezeichnen wir bei gegebener Eingabe, festem Flußwert ψ sowie einem langen Auftrag r mit

$$\gamma := l(A_{\mathcal{R}}) + l(A_\psi)$$

die konstanten Kosten für den Transport der Aufträge und zur Herstellung der Gradbalance. Die Summe der Kantenkosten in $bal(\psi, r)$ ist genau γ.

Das verbleibende Problem ist, zu entscheiden, wie die Zusammenhangskomponenten verschmolzen werden können. Um das zu erreichen, gibt es zwei Möglichkeiten:

- Wir fügen weiterer augmentierender Kanten zwischen benachbarten Knoten aus verschiedenen Zusammenhangskomponenten ein. Um den Flußwert ψ zu gewährleisten, müssen immer Paare von anti–parallelen Kanten eingefügt werden.

- Wir laden um bzw. zerschneiden Kanten. Alleine die Kanten aus $A_{\mathcal{R}}$ kommen dafür in Frage, da die restlichen Kanten nur zwischen benachbarten Knoten verlaufen und somit nicht weiter verkürzt werden können.

5.3. Exogen definierte Umladeknoten

Sei $I = (\mathcal{S}, l, \Delta, \mathcal{R}, B')$ die Eingabe für die exogene Variante des semi–präemptiven Pickup–And–Delivery–Problems. Insbesondere heißt das, daß alle möglichen Umladeknoten in B' gegeben sind. Wir werden die möglichen Werte für ψ und r enumerieren, dann ist das Tripel (I, ψ, r) die aktuelle Konfiguration mit festem ψ und r.

5.3.1. Arboreszenzen

Wir konstruieren einen gerichteten Hilfsgraphen $H = (V, E = E^r \dot{\cup} E^b)$, dessen Kantenmenge in rote ($e \in E^r$) und blaue Kanten ($e \in E^b$) unterteilt ist. In beiden Fällen sind die Kanten durch eine Funktion $c : E^r \cup E^b \to \mathbb{R}_+$ gewichtet. Für jede starke Zusammenhangskomponente Z_i in $bal(\psi, r)$ erzeugen wir einen Superknoten $v_i \in V$, so daß v_0 der Zusammenhangskomponente Z_0 entspricht, in der sich der Startknoten s_0 befindet. Wir starten mit $E^r = E^b = \emptyset$ und bauen die Kantenmenge des Multigraphen wie folgt auf:

- Füge eine rote Kante (v_i, v_j) mit Kosten $c(v_i, v_j) = 2l(s_a, s_b)$ zu E^r hinzu, falls es benachbarte Knoten $s_a, s_b \in \mathcal{S}$ gibt, die in verschiedenen Zusammenhangskomponenten $s_a \in Z_i, s_b \in Z_j$ sind. Falls es mehrere Kandidaten gibt, wähle s_a und s_b so, daß $l(s_a, s_b)$ minimal ist. Eine Kante dieses Typs entspricht dem Einfügen zweier anti–paralleler Kanten zwischen den Zusammenhangskomponenten i und j in $bal(\psi, r)$.

- Füge eine Kante (v_i, v_j) mit Kosten $c(v_i, v_j) = \Delta$ zu E^b hinzu, falls eine Kante $(s_a, s_c) \in A_{\mathcal{R}}$ mit $s_a, s_c \in Z_i$ existiert, die einen Knoten $s_b \in B' \cap Z_j$ kreuzt, $i \neq j$. Falls es mehrere solche Kanten gibt, so wählen wir eine beliebige.

Kanten aus E^r und aus E^b können parallel zwischen den gleichen Knotenpaaren verlaufen. Der gefärbte Hilfsgraph H ist also ein Multigraph.

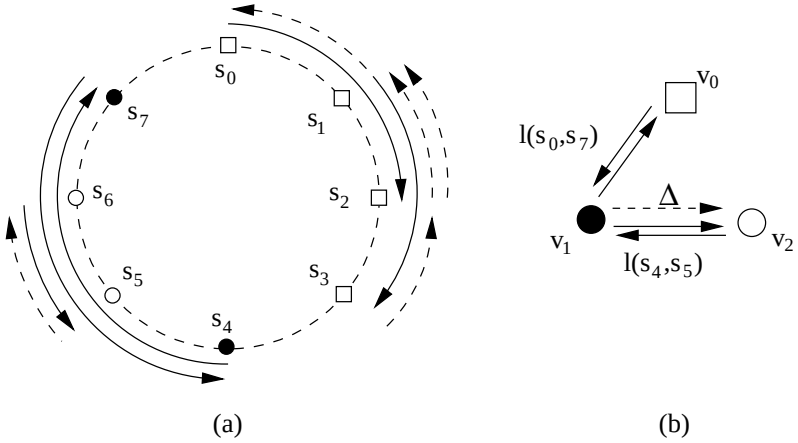


Abbildung 5.3.: *(a) Balancierter Graph* $\mathrm{bal}(0,\emptyset)$ *und (b) der zugehörige Hilfsgraph* H

In einer für H geeigneten Datenstruktur merken wir uns zusätzlich auf den Kanten, von welchem Typ sie sind sowie die betroffenen Knoten in $\mathrm{bal}(\psi, r)$. In Abbildung 5.3 wird das bisherige Beispiel fortgeführt. Auf der linken Seite der Darstellung ist noch einmal $\mathrm{bal}(\psi, r)$ mit $\psi = 0$ und $r = \emptyset$ aufgezeichnet, wobei die Knoten einer Zusammenhangskomponente ein eigenes Symbol erhalten haben. Rechts daneben ist der daraus konstruierte Hilfsgraph, wobei die Superknoten in H das Symbol ihrer zugehörigen Zusammenhangskomponente besitzen. Die durchgezogenen Kanten in H entsprechen den roten und die gestrichelten den blauen Kanten. Im Beispiel ist unterstellt, daß $l(s_0, s_7) \leq l(s_3, s_4)$ und $l(s_4, s_5) \leq l(s_6, s_7)$ ist.

Bemerkung 5.11. *Im exogenen Fall können wir beobachten, daß es im Hilfsgraphen* H *maximal* $|B'| \leq k$ *verschiedene Knoten gibt, die Kopf einer blauen Kante sind.*

Satz 5.12. *Sei* (I, ψ, r) *gegeben und* $H = (V, E)$ *wie vorher konstruiert. Es existiert ein Teilgraph* $T \subseteq E$ *mit Kosten* $c(T)$*, der eine* v_0*–Arboreszenz für* H *enthält g.d.w. es einen Transportgraphen* G_T *mit*

Kosten $l(G_T) = c(T) + \gamma$ *für* I *mit Flußwert* ψ *und langem Auftrag* r *gibt.*

Beweis. Sei T mit Kosten $c(T)$ gegeben und T enthalte eine ν_0–Arboreszenz für $H = (V, E = E^r \dot\cup E^b)$. Wir werden T benutzen, um einen Transportgraphen $G_T = (\mathcal{S}, A_\mathcal{R} \dot\cup A_\psi \dot\cup A_C, B)$ zu konstruieren, der den Auftrag r über den langen Weg entlang des Kreises transportiert und Flußwert ψ besitzt. Die Mengen $\mathcal{S}$ und $\mathcal{R}$ erhalten wir per Eingabe direkt aus I. Initialisiere mit $A_\mathcal{R} = \mathcal{R}$ und berechne die Menge A_ψ nach der sich aus Lemma 5.2 ergebenden Vorschrift. Wir starten in ν_0 und durchlaufen alle Knoten von T mittels Tiefensuche. Dann sei ν_i der aktuelle Knoten und ν_j der aktuelle Sohn von ν_i in T. $(\nu_i, \nu_j) \in T \subseteq E^r \dot\cup E^b$ entspricht dann entweder einer Nachbarschaftsbeziehung zweier Knoten s_a, s_b aus verschiedenen Zusammenhangskomponenten Z_i und Z_j in $bal(\psi, r)$ oder einem Auftrag $(s_a, s_c) \in \mathcal{R}$, der am Umladeknoten $s_b \in B'$ zerschnitten werden kann.

(i) Im ersten Fall ($(\nu_i, \nu_j) \in E^r$) fügen wir in A_C zwei anti–parallele Kanten (s_a, s_b) und (s_b, s_a) ein. Die Kosten dafür betragen

$$l(s_a, s_b) + l(s_b, s_a) = 2l(s_a, s_b) = c(\nu_i, \nu_j)$$

und der Flußwert ψ wird dadurch nicht verändert.

(ii) Im zweiten Fall ($(\nu_i, \nu_j) \in E^b$) ist $(s_a, s_c) \in A_\mathcal{R}$ oder (s_a, s_c) wurde schon in mehrere Teilstücke zerschnitten. Sei dann (s_p, s_q) dasjenige Teilstück, das s_b kreuzt. Entferne (s_p, s_q) aus $A_\mathcal{R}$ und ersetzen die Kante durch zwei Kanten $(s_p, s_b), (s_b, s_q)$ mit Kosten $l(s_p, s_b) = \Delta$ und

$$l(s_b, s_c) = \begin{cases} l(s_a, s_c) & \text{falls } q = c \\ \Delta & \text{sonst} \end{cases}$$

Beim Zerschneiden einer Kante erhalten somit alle Teilstücke den Kostenwert Δ, nur das letzte Teilstück erhält den ursprünglichen Kostenwert. Der Flußwert bleibt dabei wie im Fall (i) unberührt. Füge $s_b \in B'$ zu B hinzu. Die Kosten der Kante in H entsprechen den zusätzlichen Umladekosten Δ.

Nachdem dies für alle Kanten aus T erledigt ist, ist G_T ein zulässiger Transportgraph: Alle Knoten $s_i \in \mathcal{S}$ sind von s_0 aus erreichbar, da alle Knoten $v_i \in V$, denen die Zusammenhangskomponenten Z_i entsprechen, von v_0 aus erreichbar sind. Für jede Kante in E wurden zwei Zusammenhangskomponenten in $\mathrm{bal}(\psi, r)$ verschmolzen. Somit ist G_T zusammenhängend und alle Knoten besitzen weiterhin gleichen Eingangs– wie Ausgangsgrad. Für jeden Auftrag $r \in \mathcal{R}$ gibt es einen eindeutigen Pfad in $A_{\mathcal{R}}$, da zunächst für jeden Auftrag eine Kante in $A_{\mathcal{R}}$ eingefügt und später in kleinere Einheiten zerteilt wurde. Nach Konstruktion von H gilt $B \subseteq B'$. Die Kosten des Transportgraphen betragen wie behauptet:

$$l(G_T) = \sum_{e \in T} c(e) + l(A_{\mathcal{R}}) + l(A_\psi) = c(T) + \gamma$$

Sei nun umgekehrt $G_T = (\mathcal{S}, A, B)$ ein Transportgraph für I mit Flußwert ψ und $r \in \mathcal{R}$ wird als einziges Objekt über den längeren Weg transportiert. Nach Lemma 5.5 können wir $A = A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C$ annehmen und die Kosten entsprechend schreiben als:

$$l(G_T) = l(A_{\mathcal{R}}) + l(A_\psi) + l(A_C) + \Delta|B|$$

Betrachte für diese Eingabe $\mathrm{bal}(\psi, r)$ und den daraus konstruierten Graphen H. Seien Z_i die Zusammenhangskomponenten von $\mathrm{bal}(\psi, r)$ und v_i die dazu korrespondierenden Knoten in H. Wir betrachten im folgenden nur Kanten (s_a, s_b) zwischen verschiedenen Zusammenhangskomponenten aus $\mathrm{bal}(\psi, r)$. Für jede solche Kante seien $s_a \in Z_i$ und $s_b \in Z_j$. Benutze Tiefensuche, um ausgehend von $i = 0$ alle Zusammenhangskomponenten von $\mathrm{bal}(\psi, r)$ zu besuchen.

(i) Falls $(s_a, s_b) \in A_C$, existiert nach Konstruktion von H eine rote Kante $(v_i, v_j) \in E^r$ mit Kosten $2l(s_a, s_b)$. Füge diese zu T hinzu. Markiere (s_a, s_b) sowie (s_b, s_a) als benutzt.

(ii) Falls $(s_a, s_b) \in A_{\mathcal{R}}$ und $s_b \notin B$, dann existiert ein Auftrag $(s_c, s_b) \in \mathcal{R}$. In diesem Fall tun wir nichts, da s_c und s_b in der gleichen Zusammenhangskomponente liegen müssen. Andernfalls ist $s_b \in B$ ein Umladeknoten und es existiert ein Auftrag $(s_c, s_d) \in \mathcal{R}$ mit $s_c, s_d \in Z_z$ und (s_a, s_b) liegt auf dem Pfad von s_c nach s_d entlang des Kreises. Füge

eine blaue Kante $(v_z, v_j) \in E^b$ zu T mit Kosten Δ hinzu. Markiere (s_a, s_b) als benutzt. G_T kann mehrmals am gleichen Knoten umladen. Für jedes $s_b \in B$ werden wir jedoch nur eine solche Kante in T einfügen.

(iii) Der Fall $(s_a, s_b) \in A_\psi$ ist ausgeschlossen, da wir nur solche Kanten betrachten, die zwischen verschiedenen Zusammenhangskomponenten verlaufen. Die Kanten aus A_ψ definieren aber gerade die Zusammenhangskomponenten.

Wir begannen in Z_0 und besuchten per Tiefensuche alle Zusammenhangskomponenten Z_i von $bal(\psi, r)$, und in jedem Schritt haben wir eine Kante mit Kopf v_i in T eingefügt. Somit existiert am Ende in T ein gerichteter Pfad von v_0 zu jedem anderen Knoten v_i. Daraus folgt, daß T eine v_0–Arboreszenz enthalten muß. Die Kosten von T sind $c(T) = l(A_C) + \Delta|B|$. □

5.3.2. Flußwerte

In [AK88] wurde für die präemptive Variante des Problems auf dem Kreis ein Ergebnis vorgestellt, das die Laufzeit der Algorithmen für den nicht–präemptiven Fall verbessert. Leider ist der dazugehörige Beweis nicht auf die semi–präemptive Version anwendbar. Wir geben im folgenden einen eigenen Beweis an, der sowohl für den exogenen als auch für den endogenen Fall gilt und können dadurch die Laufzeit unserer Algorithmen um den Faktor m verbessern. Da wir eine Verallgemeinerung betrachten, gilt dieser Beweis ebenfalls für die präemptive Version des Problems.

Lemma 5.13. *Sei G_T ein optimaler Transportgraph und e ein langer Auftrag in G_T. Dann gibt es mindestens ein Intervall des Kreises, das nur von e und keinem weiteren Auftrag überdeckt wird.*

Beweis. Nehmen wir das Gegenteil an und vermuten, daß jedes von e überdeckte Intervall auch von mindestens einer weiteren Kante überdeckt wird. Dann gibt es in $G_T - e$ Kanten $f_1, \ldots, f_p$, deren Vereinigung e überdeckt.

Seien die f_i so gewählt, daß p minimal ist, insbesondere heißt das, daß kein f_i ein anderes vollständig überdeckt. Dann können wir die f_i im Uhrzeigersinn numerieren, so daß f_i entweder einen Endknoten mit f_{i+1} gemein hat oder f_i und f_{i+1} sich gegenseitig überlappen, d.h. f_i überdeckt einen Endknoten von f_{i+1} und umgekehrt ($1 \leq i < p$). Mit Lemma 5.10 wissen wir, daß $p \geq 2$. Ein Auftrag wird nur über den langen Weg transportiert, wenn die Kante in G_T an einem Umladeknoten zerschnitten wird (andernfalls wäre G_T nicht optimal). Sei Z_q diejenige Zusammenhangskomponente, in der sich der dazugehörige Umladeknoten befindet und $Z_e = Z(e)$ die Zusammenhangskomponente, in der sich e befindet. Abbildung 5.4 illustriert die Lagen von e, e^c und der f_i auf dem Kreis, ohne auf die Orientierung der Kanten einzugehen.

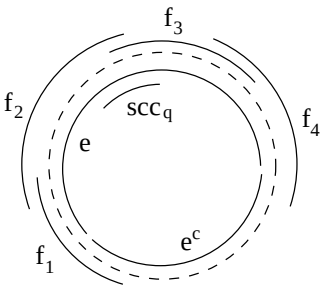


Abbildung 5.4.: *Die Lage von e und der f_i auf dem Kreis*

Betrachte den zu G_T zugehörigen gradbalancierten Graphen $bal(\psi, e)$. Diesen können wir mit Verweis auf Lemma 5.5 leicht konstruieren, indem wir alle Paare von anti–parallelen Kanten, die als einzige ein Intervall überdecken, entfernen. Seien dann $Z_1, \ldots, Z_p$ die zu $f_1, \ldots, f_p$ gehörigen Zusammenhangskomponenten in $bal(\psi, e)$. Eines der f_i kreuzt einen Knoten aus Z_q. Sei dies $f_x, x \leq p$. Aus Satz 5.12 (bzw. Satz 5.19 im endogenen Fall) wissen wir, daß G_T einer v_0–Arboreszenz $T = (V, E^r \dot{\cup} E^b)$ mit Kosten $c(T) = l(G_T) - \gamma = l(G_T) - l(bal(\psi, e))$ entspricht. Dann entsprechen den Zusammenhangskomponenten $Z_1, \ldots, Z_p, Z_q$ sowie Z_e die Knoten $v_1, \ldots, v_p, v_q, v_e \in V$, die durch T aufgespannt werden.

Da in G_T die Kante e an einem Knoten aus Z_q zerschnitten wurde, enthält T eine blaue Kante $(v_e, v_q) \in E^b$. Das Ziel des nun folgenden Vorgehens wird sein, in G_T die Kante e durch e^c zu ersetzen und den resultierenden Graphen zu reparieren. Da $l(e) > l(e^c)$ ist, ergäbe sich ein Widerspruch zur Annahme und die Behauptung des Lemmas wäre gezeigt. Wir arbeiten jedoch nicht direkt auf G_T, sondern modifizieren die dazugehörige Arboreszenz T.

Sei T' die Teilarboreszenz von T mit v_q als Wurzel, d.h. alle von T' induzierten Knoten liegen in T unterhalb von v_e und v_q. Sei $t \leq p$ der kleinste Index, so daß $v_t \in T'$.

(i) Falls t nicht existiert, so sind alle $v_i, i = 1, \ldots, p$ von v_0 aus erreichbar, ohne über v_q zu gehen. Dann enthält aber $T - (v_e, v_q) + (v_x, v_q)$ keinen Kreis und ist eine v_0–Arboreszenz. Der Graph $bal(\psi, e) - e + e^c = bal'(\psi', \emptyset)$ ist weiterhin gradbalanciert und enthält dieselben Zusammenhangskomponenten. Somit hat der zu bal' gehörige Hilfsgraph H' dieselbe Knotenmenge wie H. Die Kanten in H' sind bis auf Kanten mit Schwanz v_e dieselben wie in H. Insbesondere ist $T' = T - (v_e, v_q) + (v_x, v_q)$ dann v_0–Arboreszenz für H' mit Kosten $c(T') = c(T)$ und der gleichen Anzahl an blauen Kanten. Jedoch gilt $l(bal') < l(bal)$, d.h. der zu T' zugehörige Transportgraph G'_T ist billiger als G_T und beide benutzen die gleiche Anzahl an Umladeknoten. Widerspruch zur Annahme, daß G_T optimal ist.

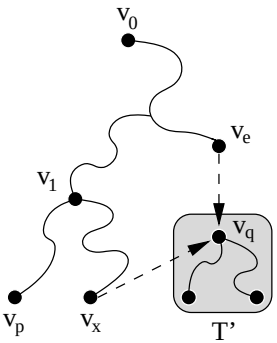


Abbildung 5.5.: *Keines der* $v_i, i = 1, \ldots, p$ *liegt in* T'

(ii) Es existiert ein Knoten v_t, der von v_0 aus nur über v_q erreichbar ist. Entweder ist $t = 1$, dann kreuzt e^c einen Endknoten von f_t und es existiert eine Kante $(v_e, v_t) \in E^b$ oder $t > 1$, dann existiert eine Kante $(v_{t-1}, v_t) \in E^b$, da sich f_{t-1} und f_t überlappen. Sei g eine solche Kante. In beiden Fällen liegt der Schwanz von g nicht in T' und es existiert ein gerichteter (v_0, v_t)–Pfad in $T + g$. Sei $i = t$ und $j \leq q$ der nächst größere Index, so daß $v_j \in T'$. Bestimme den eindeutigen (v_i, v_j)–Pfad P in T' ohne auf die Richtung der Kanten zu achten. Wir unterscheiden zwei Möglichkeiten:

(a) P enthält nur rote Kanten. Für jede rote Kante existiert eine anti–parallele Kante mit gleichen Kosten. Orientiere alle Kanten in P, so daß ein gerichteter (v_i, v_j)–Pfad entsteht. Dadurch kann (kurzzeitig)

$\delta^-(v) = 2$ für ein v auf P gelten. In diesem Fall gibt es noch weitere $v_l \in T', l \leq q$, zu denen es (noch) keinen gerichteten (v_0, v_l)–Pfad gibt.

(b) P enthält mindestens eine blaue Kante. Dann sei w die erste blaue Kante, die in P auftritt. Orientiere alle Kanten des Teilpfades von v_i aus, bis ein Endknoten von w erreicht wird. Aus den Vorüberlegungen wissen wir, daß sich f_{j-1} und f_j überlappen. Somit existiert $(v_{j-1}, v_j) \in E^b$. Füge diese Kante zu T hinzu und entferne w.

Abbildung 5.6 illustriert den Fall (ii).

Wähle danach $i = j$ und wiederhole die beiden Schritte bis $j = q$ ist und nenne die entstandene Kantenmenge T^*. Per Induktion gilt jeweils, daß es einen gerichteten (v_0, v_i)–Pfad gibt. In jedem Schritt stellen wir sicher, daß es danach einen gerichteten (v_0, v_j)–Pfad in T^* gibt. Somit gilt nach Beendigung des Verfahrens, daß für alle $i = 1, \ldots, q$ ein gerichteter (v_0, v_i)–Pfad in $T^* + g - (v_e, v_q)$ existiert. Alle Knoten in T', die nicht auf einem in (a) und (b) betrachteten Pfad P lagen, sind weiterhin von v_0 aus über einen gerichteten Pfad erreichbar, da höchstens Kanten entfernt wurden, die aus ihnen hinausgingen und in einem Knoten $v \in P$ endeten. Außerdem gilt $c(T^* + g - (v_e, v_q)) = c(T)$ und $T^* + g - (v_e, v_q)$ enthält genauso viele blaue Kanten wie T. Sei wie in (i) $bal'(\psi', \emptyset) = bal(\psi, e) - e + e^c$, dann gilt $l(bal') < l(bal)$. Ist H' der aus bal' konstruierte Hilfsgraph, dann ist $T^* + g - (v_e, v_q)$ eine v_0–Arboreszenz für H'. Der daraus konstruierte Transportgraph G'_T ist billiger als G_T und benutzt genauso viele Umladeknoten. Widerspruch. □

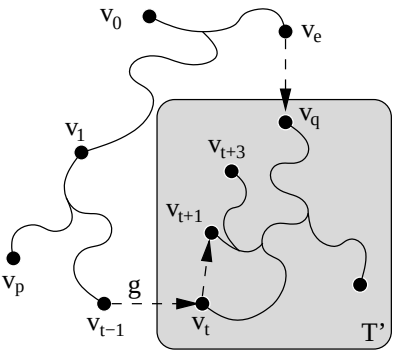


Abbildung 5.6.: *Mindestens eines der* $v_i, i = 1, \ldots, p$ *liegt in* T'

Korollar 5.14. *Sei* G_T *ein optimaler Transportgraph mit Flußwert* ψ. *Falls* G_T *einen langen Auftrag* r *enthält, so gilt* $|\psi| = 1$.

Beweis. Nach Lemma 5.13 gibt es ein Intervall, das nur von r und keiner anderen Kante überdeckt wird. Dann ist der Flußwert über dieses Intervall entweder $+1$ oder -1. □

5.3.3. Kombinatorische Lösungsverfahren

Die bisherigen Ergebnisse lassen sich zu einem Algorithmus zusammenfassen, der das exogene semi–präemptive Transportproblem auf dem Kreis löst. Diesen Algorithmus werden wir in vereinfachter Form auch für den Fall anwenden, daß der zugrunde liegende Graph ein Pfad ist.

Algorithmus 5 Exogen$(\mathcal{S}, l, \Delta, \mathcal{R}, B')$

```
T* = ∅
for all r ∈ R ∪ {∅} do                          ▷ wähle einen Auftrag lang
    for all ψ ∈ [−m − 1, m + 1] do               ▷ enumeriere Flußwerte
        if r ≠ ∅ and |ψ| ≠ 1 then
            next for
        end if
        Konstruiere bal(ψ, r)                                ▷ Korollar 5.7
        Konstruiere Hilfsgraph H = (V, E)
        Bestimme eine kostenminimale v0–Arboreszenz T
        if c(T) < c(T*) then
            T* = T
        end if
    end for
end for
Konstruiere Transportgraph G_T aus T*          ▷ s. Beweis zu Satz 5.12
```

Satz 5.15. *Der Algorithmus 5* Exogen$(\mathcal{S}, l, \Delta, \mathcal{R}, B')$ *berechnet einen optimalen Transportgraphen für das exogene und semi–präemptive PDP auf dem Kreis in* $\mathcal{O}(m^2 + mn^2)$ *Zeitschritten.*

Beweis. Die Korrektheit des Algorithmus folgt aus Korollar 5.7, Lemma 5.10, Satz 5.12 und Korollar 5.14: Wir enumerieren alle möglichen Werte für ψ eines optimalen Transportgraphen und probieren ebenfalls alle Möglichkeiten aus, einen Auftrag r über den langen Weg entlang des Kreises zu transportieren. Für ein festes Paar (ψ, r) haben wir in Satz 5.12 gezeigt, daß wir dieses Teilproblem optimal durch die Konstruktion einer kostenminimalen Arboreszenz lösen können.

Die äußere Schleife benötigt $m + 1$ Schritte. Falls $r = \emptyset$ ist, werden $2m + 3$ Durchläufe der inneren Schleife durchgeführt. Falls ein Auftrag r über den langen Weg transportiert wird, wird der innere Teil lediglich 2 mal aufgerufen (siehe Korollar 5.14). Also werden beide for–Schleifen $2m + 3 + 2m \in \mathcal{O}(m)$ mal passiert. Im Kern der beiden Schleifen werden die folgenden Operationen benutzt:

- Konstruiere $bal(\psi, r)$: Die Werte für ϕ brauchen nur einmalig berechnet werden. Mit den Bemerkungen zur Definition 5.1 ist dies in $\mathcal{O}(m)$ Schritten möglich. Die Zusammenhangskomponenten in $bal(\psi, r)$ lassen sich damit für festes ψ und r leicht mit Aufwand $\mathcal{O}(n)$ errechnen.
- Der Hilfsgraph H besteht aus höchstens $\frac{n}{2}$ Knoten und die Kantenmenge kann in $\mathcal{O}(n^2)$ Schritten aufgebaut werden.
- Eine kostenminimale v_0–Arboreszenz läßt sich in $\mathcal{O}(m + n \log n)$ Schritten bestimmen (s. Satz 2.6).

Somit erhalten wir die behauptete Gesamtlaufzeit von $\mathcal{O}(m(n+n^2+m+n\log n)) = \mathcal{O}(m^2 + mn^2)$. □

Wenn der zugrunde liegende Graph kein Kreis, sondern ein Pfad ist, können wir aufgrund der besonderen Struktur auf die Enumeration möglicher Werte für ψ und r verzichten:

Korollar 5.16. *Das exogene und semi–präemptive PDP auf dem Pfad kann in $\mathcal{O}(m + n^2)$ Schritten optimal gelöst werden.*

Beweis. Wir benutzen einen Teil des Algorithmus 5. Offensichtlich gibt es auf dem Pfad für jeden Auftrag $r \in \mathcal{R}$ nur eine Möglichkeit, einen Auftrag ohne Umwege zu transportieren. Zusätzlich kann man sich leicht überlegen, daß auf jedem Intervall die Anzahl der Kanten von links nach rechts gleich der Anzahl der Kanten von rechts nach links sein muß, oder andernfalls der Graph nicht gradbalanciert sein kann. Daraus folgt, daß der Flußwert ψ eines Transportgraphen gleich Null sein muß. Somit reicht es, daß der für den Pfad modifizierte Algorithmus nur den Schleifenkern von Algorithmus 5 durchführt. Zusammen mit den Bemerkungen im Beweis zu Satz 5.15 folgt die Behauptung. □

Bemerkung 5.17. *Für den Spezialfall, daß der zugrunde liegende Graph ein Pfad ist, können wir $\psi = 0$ annehmen. Außerdem können wir die Anzahl der Zusammenhangskomponenten in $\mathrm{bal}(0, r)$ verringern. Angenommen, für ein Intervall i ist der Fluß $\phi(i) = 0$. Es kann sein, daß es Aufträge gibt, die dieses Intervall kreuzen. Dann gibt es jedoch genau so viele Aufträge über dieses Intervall in der entgegen gesetzten Richtung. Andererseits ist es möglich, daß es keinen Auftrag gibt, der i kreuzt. In diesem Fall können wir trotzdem zwei anti–parallele Kanten einfügen, da sonst einige Knoten des Graphen nicht vom Startknoten aus erreichbar sind. Für die Laufzeit, die sich immer auf den schlimmsten Fall bezieht, hat diese Beobachtung keine Bedeutung.*

5.4. Endogen zu bestimmende Umladeknoten

Sei nun $I = (\mathcal{S}, l, \Delta, \mathcal{R}, k)$ die Eingabe für die endogene Variante des semi–präemptiven Pickup–And–Delivery–Problems. Im Gegensatz zur exogenen Version ist uns nun nur eine natürliche Zahl $k \leq n$ gegeben, die uns die Anzahl der verwendeten Umladeknoten mit $|B| \leq k$ beschränkt. Jedoch kann nun frei entschieden werden, welche der Knoten aus V in B aufgenommen werden. Dann ist wiederum das Tripel (I, ψ, r) die aktuelle Konfiguration mit festem ψ und r.

5.4.1. Gefärbte Arboreszenzen

Wir konstruieren analog zum exogenen Fall einen gerichteten Hilfsgraphen, auf dem wir eine Arboreszenz bestimmen wollen. Jedoch kennen wir in dieser endogenen Variante nicht die Umladeknoten. Sie zu bestimmen ist nun Teil der Aufgabenstellung. Sei also $H = (V, E = E^r \dot{\cup} E^b)$ ein gefärbter Hilfsgraph, der wie folgt konstruiert wird. Wir starten mit $E^r = E^b = \emptyset$ und bauen die Kantenmenge des Multigraphen wie folgt auf:

- Füge eine gerichtete rote Kante (v_i, v_j) mit Kosten $c(v_i, v_j) = 2l(s_a, s_b)$ zu E^r hinzu, falls es zwei benachbarte Knoten $s_a, s_b \in \mathcal{S}$ gibt, die in verschiedenen Zusammenhangskomponenten $s_a \in Z_i, s_b \in Z_j, i \neq j$ liegen. Falls es mehrere Kandidaten s_a, s_b gibt, so wähle diejenigen mit minimaler Distanz. Eine Kante dieses Typs entspricht dem Einfügen zweier anti–paralleler Kanten zwischen den Zusammenhangskomponenten i und j in $bal(\psi, r)$.
- Füge eine gerichtete blaue Kante (v_i, v_j) mit Kosten $c(v_i, v_j) = \Delta$ zu E^b hinzu, falls es eine Kante $(s_a, s_c) \in A$ mit $s_a, s_c \in Z_i$ gibt, die einen Knoten $s_b \in Z_j, i \neq j$ kreuzt. Es gilt, daß $(s_a, s_c) \in \mathcal{R}$ ein Auftrag ist, da nur diese Kanten andere Knoten kreuzen können. Falls es mehrere solche Kanten gibt, so wählen wir eine beliebige.

Der Unterschied bei der Konstruktion des Hilfsgraphen im exogenen Fall liegt nur darin, daß wir die Kanten zusätzlich mit Farben kennzeichnen.

Bemerkung 5.18. *Analog zum exogenen Fall entsprechen die blau gefärbten Kanten Umladevorgängen auf dem Kreis, aber jetzt ist die Anzahl der Knoten, die Kopf einer blauen Kanten sind, nicht mehr durch* k *beschränkt.*

Wenn wir eine v_0–Arboreszenz für H bestimmen würden, wäre im allgemeinen nicht sichergestellt, daß darin maximal k Umladekanten benutzt würden. Deswegen betrachten wir v_0–Arboreszenzen, bei denen wir maximal k blaue Kanten benutzen dürfen. Das Problem wurde bereits in Definition 3.5 als (d, r)–Arboreszenz–Problem eingeführt. In unserem Fall ist $d = k$ und $r = v_0$.

Die roten Kanten in H entsprechen dem Einfügen zweier anti–paralleler Kanten zwischen den Knoten zweier verschiedener Zusammenhangskomponenten. Die blauen Kanten entsprechen einem Umladevorgang und beziehen sich immer auf eine Beziehung zwischen Auftrag und Umladeknoten. Durch das Beschränken der blauen Kanten in T wird die Anzahl der Umladevorgänge beschränkt. Wir zeigen nun das folgende: Vorausgesetzt, wir könnten eine optimale (d, r)–Arboreszenz für H bestimmen, so könnten wir auch das endogene Transportproblem auf dem Kreis lösen.

Satz 5.19. *Sei* (I, ψ, r) *gegeben und* $H = (V, E)$ *wie vorher konstruiert. Es existiert ein Teilgraph* $T \subseteq E^r \dot{\cup} E^b$ *mit Kosten* $c(T)$*, der eine* (k, v_0)*–Arboreszenz für* H *enthält g.d.w. es einen Transportgraphen* G_T *mit Kosten* $l(G_T) = c(T) + \gamma$ *für* I *mit Flußwert* ψ *und langem Auftrag* r *gibt.*

Beweis. Der Beweis kann analog zum Beweis des Satzes 5.12 geführt werden. Allerdings gilt Bemerkung 5.11 nicht. In jeder v_0–Arboreszenz gibt es jedoch für jeden Knoten $v \in V, v \neq v_0$ genau eine Kante, deren Kopf v ist. Es reicht also aus, statt der Knoten die Anzahl der blauen Kanten durch k zu beschränken. Dies ist gerade die zusätzliche Bedingung einer (k, v_0)–Arboreszenz. □

Unangenehmerweise ist kein Algorithmus bekannt, der eine kostenminimale (d, r)–Arboreszenz für allgemeine Graphen berechnet. Wir suchen jedoch eine solche Arboreszenz für den Hilfsgraphen H, der eine spezielle Struktur hat. Wir werden im folgenden sehen, daß wir unter Ausnutzung spezieller Eigenschaften des Transportproblems auf dem Kreis eine kostenminimale Arboreszenz für H bestimmen können. Trotzdem bleibt das allgemeine (d, r)–Arboreszenzproblem offen. Aufgrund der engen Verwandtschaft mit dem Pickup–And–Delivery Problem werden wir in Kapitel 3 weitere Eigenschaften untersuchen und einige bekannte Resultate hinsichtlich der Lösbarkeit auf anderen Graphenklassen präsentieren. Dort werden wir uns auch der folgenden Frage zuwenden: Wenn wir schon keinen Algorithmus zur exakten Lösung des Problems vorweisen können, inwieweit können wir eine approximative Lösung geben?

5.4.2. Gefärbte Bäume

Wenden wir uns jetzt wieder dem Problem zu, eine kostenminimale (d, r)–Arboreszenz für den Hilfsgraphen H zu finden. Wir fordern im folgenden einige Eigenschaften, die H aufweisen sollte und werden dann den Hilfsgraphen in seiner ungerichteten Version betrachten. Für diesen ungerichteten, aber immer noch gefärbten Graphen, werden wir einen aufspannenden Baum mit höchstens k blauen Kanten bestimmen. Außerdem werden wir feststellen, daß wir aus diesem aufspannenden Baum eine Arboreszenz konstruieren können.

Sei $H = (V, E^r \dot\cup E^b)$ ein gerichteter Multigraph und $c : E^r \dot\cup E^b \to \mathbb{R}$ eine Gewichtsfunktion auf den Kanten. Wir betrachten die folgenden Eigenschaften.

(H1) $\forall_{(v_i,v_j)\in E^r} : c(v_i, v_j) = c(v_j, v_i)$ und
$\forall_{(v_i,v_j)\in E^b} : c(v_i, v_j) = \Delta \in \mathbb{R}$

(H2) $(v_i, v_j) \in E^r \Leftrightarrow (v_j, v_i) \in E^r$

(H3) $\forall_{(v_j,v_i)\in E^b} \forall_{(v_0,v_i)\text{–Pfade P in } H-v_j} \exists_{v_x\in P} : (v_x, v_j) \in E^b$

Die Eigenschaft H1 fordert von der Kostenfunktion Symmetrie auf den roten Kanten und Konstanz auf den blauen Kanten. Die zweite Bedingung erwartet, daß der rote Teilgraph (V, E^r) selbst symmetrisch ist. Die dritte ist etwas schwächer. Für jede blaue Kante $(v_j, v_i) \in E^b$ betrachten wir einen gerichteten Pfad P von der Wurzel v_0 zum Kopf v_i, der nicht v_j besucht. Damit H3 erfüllt ist, muß P einen Knoten v_x enthalten, der Schwanz einer blauen Kante $(v_x, v_j) \in E^b$ ist. Diese letzte Eigenschaft wirkt zunächst etwas ungewöhnlich, wir werden jedoch bald sehen, daß es Instanzen des Transportproblems auf dem Kreis gibt, deren jeweiliger Hilfsgraph alle drei Eigenschaften besitzt. Schauen wir uns zunächst einmal an, wofür diese Eigenschaft nütze ist.

Zu einem gerichteten Hilfsgraphen $H = (V, E^r \dot\cup E^b)$ ist der zugrunde liegende ungerichtete Graph $H' = (V, E^{r'} \dot\cup E^{b'})$ wie folgt definiert:

- Konstruiere die Kantenmengen, indem die Orientierung vergessen wird. Doppelte rote Kanten oder doppelte blaue Kanten werden weggelassen. Jedoch können parallele Kanten mit unterschiedlichen Farben existieren. Für jede rote ungerichtete Kante $\{v_i, v_j\}$ in H' existiert nach Konstruktion von H also immer auch die gerichtete Kante (v_i, v_j) in H. Gibt man uns eine ungerichtete blaue Kante $\{v_i, v_j\}$, so wissen wir nur, daß mindestens eine der beiden gerichteten blauen Kanten (v_i, v_j) oder (v_j, v_i) in H existiert.

- Wir benutzen folgende Kostenfunktion c' auf H':

$$c'(v_i, v_j) := \begin{cases} c(v_i, v_j) & \text{falls } \{v_i, v_j\} \in E^r \\ \Delta & \text{falls } \{v_i, v_j\} \in E^b \end{cases}$$

Auf H' können wir einen kostenminimalen d–Baum in $\mathcal{O}(m \log n)$ bestimmen, indem wir den Algorithmus von GABOW und TARJAN benutzen (s. Satz 3.4). Die Eigenschaften H1 bis H3 garantieren, daß wir eine gefärbte Arboreszenz mit gleichen Kosten konstruieren können.

Lemma 5.20. *Sei H ein wie bisher konstruierter gerichteter Hilfsgraph und H' der zugrunde liegende ungerichtete Graph. Weiterhin ist $d \in \mathbb{N}$ und T' ein kostenminimaler d–Baum für H'. Dann existiert eine (d, v_0)–Arboreszenz T für H mit Kosten $c(T) = c'(T')$.*

Beweis. Wähle v_0 als Wurzel und benutze Tiefensuche, um alle Knoten von T' zu besuchen. Sei v_i der aktuelle Knoten und v_j der aktuelle Sohn von v_i. $\{v_i, v_j\} \in T'$ ist entweder rot oder blau gefärbt.

(i) Falls $\{v_i, v_j\} \in E^{r'}$ rot ist, muß wegen der Eigenschaft H2 eine rote Kante $(v_i, v_j) \in E^r$ existieren. Füge (v_i, v_j) zu T hinzu.

(ii) Falls $\{v_i, v_j\} \in E^{b'}$ blau ist und eine blaue gerichtete Kante $(v_i, v_j) \in E^b$ existiert, so fügen wir (v_i, v_j) zu T hinzu und sind ebenfalls fertig. Andernfalls gilt $\{v_i, v_j\} \in E^{b'}$, aber $(v_i, v_j) \neq E^b$. Dann sei P der bislang konstruierte (v_0, v_i)–Pfad in T. Eigenschaft H3 versichert uns der Existenz eines Knotens v_x auf P mit der Eigenschaft, daß eine Kante $(v_x, v_j) \in E^b$ existiert. Diese Kante fügen wir T hinzu.

Da alle roten Kanten symmetrische Kosten und alle blaue Kanten gleiche Kosten haben (H1), folgt $c(T) = c'(T')$. Da wir für jede ungerichtete blaue Kante in T' eine gerichtete blaue Kante in T eingefügt haben, gilt $d \geq |T' \cap E^{b'}| = |T \cap E^{b}|$. Mit einer geeigneten Datenstruktur kann T aus T' in $\mathcal{O}(m)$ Zeitschritten konstruiert werden, indem man sich in jedem Knoten merkt, ob er Kopf einer blauen Kante ist. □

5.4.3. Spezielle Instanzen

Wir werden nun ganz spezielle Instanzen des endogenen semi–präemptiven Transportproblems auf dem Kreis identifizieren, deren zugrunde liegender Hilfsgraph H die Forderungen H1, H2 und H3 befriedigt. Nachdem wir gesehen haben, wie solche Probleme gelöst werden können, werden wir in Abschnitt 5.4.4 einen Algorithmus für allgemeine Instanzen auf dem Kreis präsentieren. Zur Erinnerung: Z_0 ist die Zusammenhangskomponente von $\mathtt{bal}(\psi, r)$, in der sich der Startknoten befindet. Wir konzentrieren unsere Überlegungen zunächst auf Instanzen (I, ψ, r) mit der folgenden Eigenschaft:

(P1) $\nexists (s_a, s_c) \in \mathcal{R} : (s_a, s_c)$ kreuzt alle Knoten der Zusammenhangskomponente Z_0

Wir sollten bemerken, daß sich die Eigenschaft P1 auf festes ψ und r bezieht. Verallgemeinert man die Forderung auf die Eingabeinstanz I, dann muß sichergestellt werden, daß kein Auftrag ganz Z_0 kreuzt, egal wie man die Drehrichtung wählt. Sicherlich lassen sich leicht Instanzen I konstruieren, die P1 erfüllen. Die Forderung ist aber nicht zu abwegig, wenn wir das Problem auf dem Pfad betrachten. Verlangen wir, daß der Startknoten an einem der beiden Enden des Pfades liegen muß [Räb04a], ist P1 immer erfüllt, weil keine Kante jemals s_0 kreuzen wird.

Das folgende Lemma charakterisiert den Zusammenhang von blauen Kanten im Hilfsgraphen und der Lage von Aufträgen auf dem Kreis. Die Aussage ist trivial, wird aber zum Verständnis der darauffolgenden Argumentationen vorausgesetzt.

Lemma 5.21. *Sei* $\mathrm{bal}(\psi, r)$ *ein balancierter Graph zu* (I, ψ, r) *und* $H = (V, E^r \dot\cup E^b)$ *der dazugehörige gerichtete Hilfsgraph. Die folgenden Aussagen sind äquivalent:*

(i) Alle Aufträge $(s_a, s_c) \in \mathcal{R}, s_a, s_c \in Z_i$*, die einen Knoten* $s_b \in Z_j, j \neq i$ *kreuzen, überdecken ganz* Z_j*.*

(ii) $(v_i, v_j) \in E^b \Rightarrow (v_j, v_i) \notin E^b$

Beweis. $\Rightarrow$: Per Definition existiert $(v_i, v_j) \in E^b$ und somit ein Auftrag $(s_a, s_c) \in \mathcal{R}$, $s_a, s_c \in Z_i$, der einen Knoten $s_y \in Z_j$ kreuzt. Angenommen, es existiert ebenfalls $(v_j, v_i) \in E^b$. Dann muß nach Konstruktion von H auch ein Auftrag $(s_b, s_d) \in \mathcal{R}, s_b, s_d \in Z_j$ existieren, der einen Knoten $s_x \in Z_i$ kreuzt. (s_a, s_c) kreuzt ganz Z_j, dann verläuft die Kante (s_a, s_c) über s_b und s_d. Da $Z(s_a) = Z(s_x)$ gibt es in $\mathrm{bal}(\psi, r)$ einen Pfad von s_a nach s_x. Eine Kante dieses Pfades kreuzt aber entweder s_b oder s_d, aber nicht beide. Widerspruch.

$\Leftarrow$: Sei $(v_i, v_j) \in E^b$ und $(v_j, v_i) \notin E^b$. Dann existiert ein Auftrag $(s_a, s_c) \in \mathcal{R}, s_a, s_c \in Z_i$, der einen Knoten $s_b \in Z_j$ kreuzt. Sollte es einen Auftrag geben, der nicht alle Knoten aus Z_j kreuzt, greift die gleiche Argumentation wie vorher: Dieser Knoten muß auf einem Pfad innerhalb der Komponente Z_j liegen und somit von einer Kante aus Z_j gekreuzt werden, Widerspruch. □

Lemma 5.22. *Sei* (I, ψ, r) *eine Instanz des Transportproblems mit festem* ψ *und* r*, und* H *sei der wie vorher angegeben konstruierte Hilfsgraph zu* $\mathrm{bal}(\psi, r)$*. Falls* (I, ψ, r) *die Eigenschaft P1 erfüllt, dann erfüllt* H *die Eigenschaften H1, H2 und H3.*

Beweis. Die Eigenschaften H1 und H2 folgen direkt aus der Konstruktion von H. Es verbleibt Eigenschaft H3 zu zeigen. Sei $(v_j, v_i) \in E^b$ eine beliebige blaue Kante in H. Falls $(v_i, v_j) \in E^b$ existiert, wären wir fertig. Somit nehmen wir an, daß $(v_i, v_j) \notin E^b$. Dann ist P irgendein gerichteter (v_0, v_i)–Pfad in $H - v_j$. Wir werden zeigen, daß P einen Knoten v_x mit $(v_x, v_j) \in E^b$ enthält.

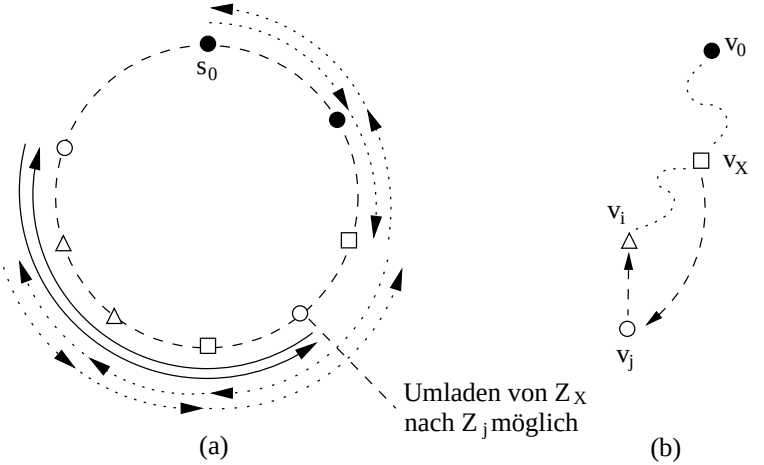


Abbildung 5.7.: *(a) Die Situation auf dem Kreis, passend zu (b)* (v_0, v_i)*–Pfad in* H *plus zusätzlicher Kante* (v_x, v_j) *für Instanzen mit Eigenschaft P1 bzw. H1, H2 und H3.*

Im Beweis zu Satz 5.19 wird zu einem gegebenen Teilgraphen von H ein Transportgraph konstruiert, indem man beginnend bei der Wurzel alle Knoten des Hilfsgraphen mittels Tiefensuche besucht und Kanten zwischen verschiedenen Zusammenhangskomponenten von $\mathtt{bal}(\psi, r)$ einfügt. Wir wollen das gleiche Verfahren für den von P induzierten Teilgraphen von H anwenden. P spannt zwar nicht ganz V auf, enthält aber die Wurzel v_0. Der so konstruierte Teil–Transportgraph $G_T^P = (\mathcal{S}, A^P, B^P)$ enthält eine große Zusammenhangskomponente, in der sich der Startknoten s_0 und alle Knoten der Zusammenhangskomponenten Z_p mit $v_p \in P$ befinden, insbesondere die Knoten in Z_i. Außerdem ist Z_j in G_T^P isoliert, weil v_j nicht auf P liegt. $(v_j, v_i) \in E^b$ existiert, weil es einen Auftrag $(s_a, s_c) \in \mathcal{R}$ gibt mit $s_a, s_c \in Z_j$, der einen Knoten $s_b \in Z_i$ kreuzt. Wir unterscheiden zwei Möglichkeiten:

(i) Entweder wird s_0 ebenfalls von (s_a, s_c) gekreuzt. Da der Auftrag wegen Eigenschaft P1 aber nicht ganz Z_0 kreuzt, gibt es wegen Lemma 5.21 einen Auftrag, der in Z_0 startet und endet und somit einen Knoten aus

Z_j kreuzt. Damit existiert aber eine Kante $(v_0, v_j) \in E^b$. Mit $x = 0$ ist H3 erfüllt.

(ii) Oder (s_a, s_c) kreuzt nicht s_0. In A^P existiert dann aber ein Pfad von s_0 nach s_b entlang des Kreises, weil $s_b \in Z_i$ und v_i auf P liegt. Da $(v_i, v_j) \notin E^b$, kreuzt (s_a, s_c) alle Knoten von Z_i (Lemma 5.21). Dann muß eine Kante des Pfades von s_0 nach s_b aber einen der beiden Endknoten s_a oder s_c kreuzen. Sei Z_x die Zusammenhangskomponente dieser Kante. Dann existiert eine blaue Kante $(v_x, v_j) \in E^b$. Abbildung 5.7 illustriert diese Situation. □

Für eine Eingabeinstanz, die P1 erfüllt, können wir jetzt einen Algorithmus angeben, der das zugehörige Problem optimal löst.

Algorithmus 6 k-RobotLight(I, ψ, r)

Initialisiere $A_{\mathcal{R}} := \mathcal{R}$
Berechne $bal(\psi, r)$ ▷ Korollar 5.7
Konstruiere den gerichteten Hilfsgraphen $H = (V, E = E^r \dot\cup E^b)$ mit $c : E \to \mathbb{R}_+$
Konstruiere den ungerichteten Hilfsgraphen $H' = (V, E' = E^{r\prime} \dot\cup E^{b\prime})$ mit $c' : E' \to \mathbb{R}_+$
Berechne k–Baum T' für (H', c') ▷ Satz 3.4
Konstruiere (k, v_0)–Arboreszenz T ▷ Lemma 5.20
Konstruiere Transportgraphen $G_T = (\mathcal{S}, A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C, B)$ ▷ Satz 5.19

Satz 5.23. *Algorithmus 6* k-RobotLight *berechnet zu gegebenem* (I, ψ, r) *mit Eigenschaft P1 einen optimalen Transportgraphen in* $\mathcal{O}(m \log n)$ *Schritten.*

Beweis. (I, ψ, r) erfüllt P1. Nach Lemma 5.22 erfüllt H dann die Eigenschaften H1, H2 und H3. Mit dem Beweis zu Lemma 5.20 können wir aus einem k–Baum T' für H' eine (d, v_0)–Arboreszenz mit gleichen Kosten berechnen. Die Kosten des Baumes sind sicherlich eine untere Schranke für die Kosten der Arboreszenz. Da T' bezüglich c' minimal ist, folgt die Optimalität für T bezüglich c. Schließlich benutzen wir das Verfahren aus dem Beweis zu Satz 5.19, um den Transportgraphen zu konstruieren.

Die Laufzeit des Algorithmus ist hauptsächlich von der Zeit zur Berechnung des kostenminimalen d–Baumes abhängig. □

5.4.4. Kombinatorisches Lösungsverfahren

Den Algorithmus k-RobotLight des vorhergehenden Abschnittes können wir nur für Instanzen nutzen, die die Eigenschaft P1 erfüllen. Wir werden nun sukzessive ein dynamisches Programm entwickeln, das allgemeine Instanzen des endogenen Transportproblems auf dem Kreis lösen kann. Dabei werden wir Abschnitte auf dem Kreis identifizieren, auf die k-RobotLight angewandt werden kann. Dieses Teilproblem werden wir iterativ mit maximal $k' = 0, \ldots, k$ vielen Umladeknoten lösen. Danach betrachten wir einen erweiterten Kreisabschnitt und lösen das erweiterte Teilproblem erneut, wobei wir nur noch $k'' = k - k'$ viele Umladeknoten benutzen dürfen. Die dadurch entstehenden Paare von Lösungen werden schließlich optimal kombiniert.

Definition 5.24. *Wir nennen eine Teilmenge der Knoten* $X \subseteq \mathcal{S}$ konsekutiv, *falls zwischen allen* $s_a, s_b \in X$ *ein Pfad von* s_a *nach* s_b *existiert, der nur benachbarte Knoten aus* X *benutzt.*

Wir versuchen nun eine konsekutive Menge von Knoten zu bestimmen, die P1 erfüllen und für die wir eine Teillösung generieren können. Auf jeden Fall sollte der Startknoten in dieser Menge enthalten sein, da wir sonst keinen zulässigen Transport generieren können. P1 verlangt, daß kein Auftrag die Zusammenhangskomponente Z_0 vollständig kreuzt. Das bedeutet, daß im dazugehörigen Hilfsgraphen $H = (V, E^r \dot{\cup} E^b)$ keine blaue Kante nach v_0 führt.

Definition 5.25. *Sei* $X \subseteq \mathcal{S}$ *konsekutiv und* $s_0 \in X$. *Dann definieren wir für festes* (I, ψ, r)

$$C_X := \{s \in \mathcal{S} | \exists (v_{Z(s)}, v_{Z(t)})\text{–}\textit{Pfad in } (V, E^b), \\ \nexists (v_{Z(t)}, v_{Z(s)})\text{–}\textit{Pfad in } (V, E^b) \textit{ mit } t \in X\}$$

die Menge der Knoten, von denen man X *aus über Umladevorgänge erreichen kann, jedoch nicht umgekehrt.*

Wir werden mit $X := \{s_0\}$ beginnen. Für eine gegebene Menge X ist C_X diejenige Knotenmenge, deren Aufträge X komplett kreuzt. Da wir eine Umgebung um s_0 suchen, in der es keine solchen Aufträge gibt, enthält C_X genau die Knoten, die nicht in der Umgebung vorkommen sollen.

Definition 5.26. *Mit*

$$S_X := \mathit{argmax}\{|X'| : X' \subseteq \mathcal{S} \setminus C_X, s_0 \in X', X' \textit{ konsekutiv}\}$$

bezeichnen wir die maximale konsekutive Knotenmenge um den Startknoten, die keinen Knoten aus C_X *enthält. Für eine solche Umgebung* S_X *ist dann*

$$lr_X := \begin{cases} i | s_{i+1} \in S_X, s_i \notin S_X & \textit{falls } S_X \neq \mathcal{S} \\ 0 & \textit{falls } S_X = \mathcal{S} \end{cases}$$

$$rr_X := \begin{cases} i | s_{i-1} \in S_X, s_i \notin S_X & \textit{falls } S_X \neq \mathcal{S} \\ 0 & \textit{falls } S_X = \mathcal{S} \end{cases}$$

der linke *bzw.* rechte Rand von S_X *auf dem Kreis.*

Falls C_X leer ist, enthält $S_X = \mathcal{S}$ den ganzen Kreis und der Rand von S_X ist genau der Startknoten, d.h. $s_{lr_X} = s_{rr_X} = s_0$. Anfangs werden wir $C_{\{s_0\}}, S_{\{s_0\}}, lr_{\{s_0\}}$ und $rr_{\{s_0\}}$ berechnen. Dann sind die Knoten $s_i \in S_{\{s_0\}}$ eine Umgebung um den Startknoten, die P1 erfüllen. Wenn wir also zu einer Eingabe I die Aufträge auf $S_{\{s_0\}}$ beschränken, so daß $\mathcal{R}_{S_{\{s_0\}}} := \mathcal{R} \cap (S_{\{s_0\}} \times S_{\{s_0\}})$, dann erfüllt die modifizierte Eingabe $I' = (\mathcal{S}, l, \Delta, \mathcal{R}_{S_{\{s_0\}}}, k, \psi, r)$ die Eigenschaft P1 und wir können `k-RobotLight` anwenden. Im allgemeinen impliziert die Definition von S_X folgendes.

Bemerkung 5.27. *Sei* $X \subseteq \mathcal{S}$ *konsekutiv und* $s_0 \in X$. *Falls* $(\mathcal{S}, l, \Delta, \mathcal{R}_X, k, \psi, r)$ *die Eigenschaft P1 besitzt, dann erfüllt auch* $(\mathcal{S}, l, \Delta, \mathcal{R}_{S_X}, k, \psi, r)$ *Eigenschaft P1.*

Das Beispiel in Abbildung 5.8 illustriert, daß ein Randknoten $s_{lr_{\{s_0\}}}$ nicht notwendigerweise in einer Zusammenhangskomponente liegen muß, von der ein Auftrag den Startknoten kreuzt. Trotzdem existiert in H mit $s_{lr_{\{s_0\}}} \in Z_i$ ein blauer (v_i, v_0)–Pfad und kein blauer (v_0, v_i)–Pfad.

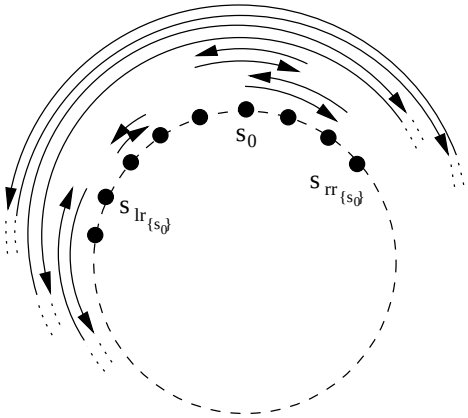


Abbildung 5.8.: *Ein gradbalancierter Graph* $\mathrm{bal}(0, \emptyset)$ *mit Rändern und inneren Knoten*

Bemerkung 5.28. *Wir beobachten, daß es keinen Auftrag* $(s_i, s_j) \in \mathcal{R}$ *mit* $s_i \in S_X, s_j \in \mathcal{S} \setminus S_X$ *geben kann. (Andernfalls existiert eine der Kanten* $(v_{Z(s_i)}, v_{Z(s_{lr_X})}) \in E^b$ *oder* $(v_{Z(s_i)}, v_{Z(s_{rr_X})}) \in E^b$*.)*

Lemma 5.29. *Sei* (I, ψ, r) *gegeben,* $X \subseteq \mathcal{S}$ *konsekutiv und* $s_0 \in X$*. Falls* $S_X \neq \mathcal{S}$*, enthält jeder zulässige Transportgraph* G_T *ein Paar anti–paralleler Kanten zwischen*

(a) s_{lr_X} *und* s_{lr_X+1}

(b) s_{rr_X} *und* s_{rr_X-1}

(c) beiden Möglichkeiten aus a) und b)

Beweis. Wir haben in Bemerkung 5.28 beobachtet, daß es keinen Auftrag in S_X gibt, der die Randknoten s_{lr_X} und s_{rr_X} kreuzt. Somit können die Zusammenhangskomponenten aus S_X mit $\mathcal{S} \setminus S_X$ nur verbunden werden, wenn augmentierende Kanten zwischen den Randknoten und ihren Nachbarn aus S_X eingefügt werden. Um den Flußwert ψ zu gewährleisten, müssen wir immer Paare von anti–parallelen Kanten einfügen. □

Angenommen, wir hätten das Teilproblem der Umgebung S_X gelöst, z.B. indem wir augmentierende Kanten zwischen s_{lr_X} und s_{lr_X-1} eingefügt haben. Dann wollen wir die erweiterte Umgebung $S'_X = S_{X\cup\{s_{lr_X}\}}$ betrachten, für die P1 gilt. Wir werden feststellen, daß die Grenzen der erweiterten Umgebungen unabhängig davon sind, welchen Randknoten wir mit dem inneren Teil der alten Umgebung verbunden haben. Das ist eine wichtige Beobachtung, denn ohne diese Eigenschaft müßten wir — der bisherigen Idee folgend — zu viele Teilmengen enumerieren, um das Optimum zu bestimmen.

Lemma 5.30. *Sei $X \subseteq \mathcal{S}$ konsekutiv, $s_0 \in X$ und $S_X \neq \mathcal{S}$. Dann gilt:*

$$S_{S_{X\cup\{s_{lr_X}\}}} = S_{S_{X\cup\{s_{rr_X}\}}} = S_{S_{X\cup\{s_{lr_X},s_{rr_X}\}}}$$

Beweis. Per Definition sind $s_{lr_X}, s_{rr_X} \in C_X$, d.h. im blauen Teilgraphen (V, E^b) existiert ein kürzester $(v_{Z(s_{lr_X})}, v_{Z(s_{i'})})$–Pfad für einen beliebigen Knoten $s_{i'} \in S_X$. Sei $(v_a, v_b) \in E^b$ die letzte Kante eines solchen Pfades. Nach Konstruktion von H existiert dann ein Auftrag $(s_i, s_{i'}) \in \mathcal{R}$, der in $s_i \in Z_a \subseteq \mathcal{S}$ startet und einen Knoten $s_j \in Z_b \subseteq S_X$ kreuzt. $(s_i, s_{i'})$ kreuzt alle Knoten aus S_X, ansonsten existiert mit Lemma 5.21 eine blaue Kante $(v_b, v_a) \in E^b$ im Widerspruch zu $s_j \in S_X$. Also sind entweder s_{lr_X} und s_{rr_X} in der gleichen Zusammenhangskomponente oder $(s_i, s_{i'})$ kreuzt ebenfalls s_{rr_X}. In beiden Fällen existiert ein gerichteter $(v_{Z(s_{lr_X})}, v_{Z(s_{rr_X})})$–Pfad in (V, E^b). Die Existenz eines gerichteten $(v_{Z(s_{lr_X})}, v_{Z(s_{rr_X})})$–Pfades wird auf analoge Weise bewiesen. Also existiert in (V, E^b) ein $(v_{Z(s_{rr_X})}, v')$–Pfad für einen beliebigen Knoten $v' \in V$, genau dann wenn es einen $(v_{Z(s_{lr_X})}, v')$–Pfad gibt. □

Zu gegebener konsekutiver Knotenmenge $X \subseteq \mathcal{S}$ mit $s_0 \in X$ betrachten wir die folgenden modifizierten Eingaben für das Transportproblem. Um die Lesbarkeit zu erhöhen, nutzen wir die Schreibweise für Intervalle aus der Mengenlehre, um zu kennzeichnen, ob die Ränder dazugehören oder nicht. Abbildung 5.9 veranschaulicht die jeweiligen modifizierten Eingaben. Dazu sei angemerkt, daß diese Teilprobleme im Falle $S_X \neq \mathcal{S}$ als Problemstellungen auf dem Pfad betrachtet werden können.

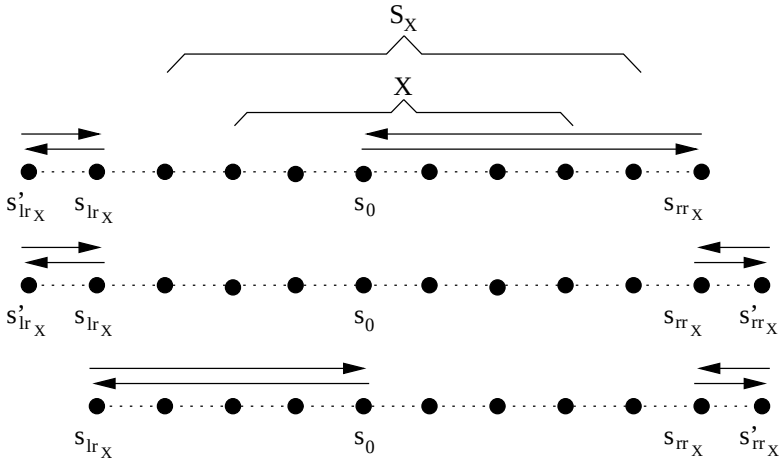


Abbildung 5.9.: *Die zu* $(I^{[X]}, k, \psi, r)$, $(I^{[X]}, k, \psi, r)$ *und* $(I^{(X)}, k, \psi, r)$ *gehörigen Graphen. Innerhalb von* S_X *wurden keine Kanten eingezeichnet.*

Definition 5.31.

- $(I^{[X]}, k, \psi, r) := (\mathcal{S}', l', \Delta, \mathcal{R}', k, \psi, r)$ *mit*

  $$\begin{aligned} \mathcal{S}' :=&\quad S_X \cup \{s'_{lr_X}, s_{lr_X}, s_{rr_X}\} \\ \mathcal{R}' :=&\quad (\mathcal{R} \cap (S_X \times S_X)) \cup \{(s'_{lr_X}, s_{lr_X}), (s_{lr_X}, s'_{lr_X}), (s_0, s_{rr_X}), \\ &\quad (s_{rr_X}, s_0)\} \end{aligned}$$

- $(I^{[X]}, k, \psi, r) := (\mathcal{S}', l', \Delta, \mathcal{R}', k, \psi, r)$ *mit*

  $$\begin{aligned} \mathcal{S}' :=&\quad S_X \cup \{s'_{lr_X}, s_{lr_X}, s_{rr_X}, s'_{rr_X}\} \\ \mathcal{R}' :=&\quad (\mathcal{R} \cap (S_X \times S_X)) \cup \{(s'_{lr_X}, s_{lr_X}), (s_{lr_X}, s'_{lr_X}), (s'_{lr_X}, s_{lr_X}), \\ &\quad (s_{lr_X}, s'_{lr_X})\} \end{aligned}$$

- $(I^{[X]}, k, \psi, r) := (\mathcal{S}', l', \Delta, \mathcal{R}', k, \psi, r)$ *mit*

  $$\begin{aligned} \mathcal{S}' :=&\quad S_X \cup \{s_{lr_X}, s_{rr_X}, s'_{rr_X}\} \\ \mathcal{R}' :=&\quad (\mathcal{R} \cap (S_X \times S_X)) \cup \{(s'_{rr_X}, s_{rr_X}), (s_{rr_X}, s'_{rr_X}), (s_0, s_{lr_X}), \\ &\quad (s_{lr_X}, s_0)\} \end{aligned}$$

In allen drei Fällen ist die Distanzfunktion l' definiert als l beschränkt auf X und mit Distanz Null zwischen den zusätzlichen Knoten und ihrer Originale, z.B. $l'(s'_{lr_X}, s_{lr_X}) := 0$.

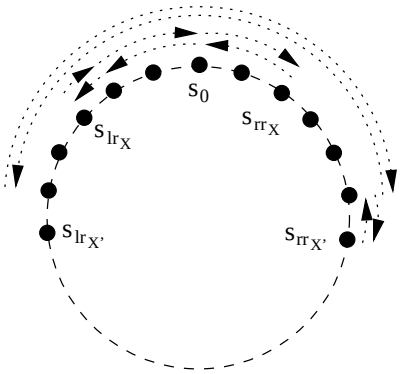


Abbildung 5.10.: *Die Idee des dynamischen Programms. In diesem Beispiel repräsentiert eine gepunktete Linie einen Kantenzug.*

In $(I^{[X)}, k, \psi, r)$ gehen wir davon aus, daß in einer Optimallösung S_X und $\mathcal{S} \setminus S_X$ über anti–parallele Kanten zwischen dem linken Rand s_{lr_X} von S_X und seinem benachbarten Knoten s_{lr_X+1} innerhalb von S_X verbunden sind. Für $(I^{[X]}, k, \psi, r)$ nehmen wir dies über beide Ränder an und in $(I^{(X]}, k, \psi, r)$ über den rechten Rand. Im folgenden bezieht sich die Erklärung auf den ersten Fall $(I^{[X)}, k, \psi, r)$. Sei $G_{[X)}$ ein optimaler Transportgraph dafür. Da s_{lr_X} in S_X isoliert ist, erzwingen wir durch das Einfügen von künstlichen Aufträgen zu einer Kopie s'_{lr_X}, daß der Knoten in einer Lösung $G_{[X)}$ besucht wird. Andererseits muß es möglich sein, daß in einer solchen Lösung einige Knoten über s_{rr_X} erreicht werden. Zu diesem Zweck werden künstliche Aufträge zwischen s_0 und s_{rr_X} erzeugt. Wenn später diese künstlichen Aufträge entfernt werden, verbleibt ein Pfad von s_0 nach s_{lr_X}, so daß beide Knoten in derselben Zusammenhangskomponente sind. Die Knoten s_0 und s_{rr_X} sind nicht notwendigerweise in der gleichen Zusammenhangskomponente, was genau dem betrachteten Fall entspricht.

Wir werden iterativ annehmen, daß entweder $G_{[X)}$, $G_{[X]}$ oder $G_{(X]}$ als Teil der allgemeinen Lösung dient. In einem weiteren Schritt identifizieren wir eine erweiterte Umgebung S'_X um für alle drei Fälle bereits gelöste Umgebung $S_X \subset S'_X$. Beide Umgebungen haben jeweils drei mögliche Lösungstypen, die kombiniert werden. Wir nehmen beispielhaft an, daß in einer Optimallösung für I der Flußwert ψ angenommen wird und $r \in$

$\mathcal{R}$ über den langen Weg transportiert wird. Weiterhin werden in S_X k' viele Umladeknoten benutzt (d.h. $|B \cap S_X| \leq k'$) und in S'_X höchstens k'' viele. Außerdem ist S_X mit $\mathcal{S} \setminus S_X$ über Kanten zwischen den Knoten s_{lr_X}, s_{lr_X+1} verbunden und S'_X mit $\mathcal{S} \setminus S'_X$ über Kanten zwischen den Knoten $s_{rr_{X'}}, s_{rr_{X'}-1}$. Abbildung 5.10 skizziert diese Situation.

Gegeben einen Transportgraphen $G^{k'}_{[X)}$ als Lösung für $(I^{[X)}, k', \psi, r)$, fixieren wir die nicht–künstlichen Kanten und Umladeknoten und konstruieren eine neue Eingabe $(I^{(X']}_{[X)}, k'', \psi, r)$, die die gleiche wie $(I^{(X']}, k'', \psi, r)$ ist, jedoch ohne Wahlmöglichkeiten innerhalb von S_X. Dies ist das allgemeine Vorgehen der Prozedur `Fix`.

Algorithmus 7 Fix$(G^{k'}_{[X)}, I^{[X')}, \psi, r)$

$G^{k'}_{[X)} = (\mathcal{S}'_X, A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C, B_X)$, $I^{[X')} = (\mathcal{S}'_{X'}, l', \Delta, \mathcal{R}_{X'})$
for all $e \in A_C$ **do**
  Erzeuge einen künstlichen Auftrag e
  Füge e zu $\mathcal{R}_C$ **hinzu**
end for
Sei $(s_a, s_c) \in \mathcal{R}_{X'}$ ein Auftrag, der s_0 kreuzt.
Sortiere $s_{b_{i=1,\dots,q}} \in B_X$ nach ihrem Auftreten auf dem eindeutigen (s_a, s_c)–Pfad
Definiere $\mathcal{R}' := (\mathcal{R}_{X'} \setminus \{(s_a, s_c)\}) \cup \{(s_a, s_{b_1}), (s_{b_1}, s_{b_2}), \dots, (s_{b_q}, s_c)\} \cup \mathcal{R}_C$
Gib $(\mathcal{S}', l', \Delta, \mathcal{R}')$ **zurück**

Der Algorithmus 7 erhält als Eingabe einen Transportgraphen $G^{k'}_{[X)}$ des Teilproblems $(I^{[X')}, k', \psi, r)$ und fixiert die zum Transport der Objekte innerhalb von S_X notwendigen Kanten aus A_C und Umladeknoten B_X. Die Kanten aus A_C werden als künstliche Aufträge $\mathcal{R}_C$ deklariert. Eine Auftragskante (s_a, s_c) aus S_X, die an Umladeknoten zerschnitten wurde, wird als Folge von mehreren künstlichen Aufträgen zerlegt. Durch das Einführen von künstlichen Aufträgen werden diese Kanten erzwungen.

Bei einem Umladevorgang wird eine Kante aus S_X zerschnitten. In der Prozedur `Fix` wird statt der ursprünglichen Kante ein Auftrag $(s_a, s_c) \in \mathcal{R}$ an denselben Umladeknoten zerschnitten. Solange $S_X \subsetneq \mathcal{S}$ gilt kön-

nen wir garantieren, daß ein solcher Auftrag existiert. Weiter können wir für die Laufzeitabschätzung annehmen, daß die Umladeknoten s_{b_i} mit Indizes versehen sind, so daß wir uns in `Fix` eine Sortierung sparen können. Somit hängt die Laufzeit des Algorithmus nur von der Kardinalität von S_X ab und kann mit $\mathcal{O}(|S_X|)$ abgeschätzt werden. Die Algorithmen, um Teillösungen der anderen Typen $G^{k'}_{[X]}$ und $G^{k'}_{(X]}$ zu fixieren, arbeiten analog.

Im folgenden Algorithmus `k-Robot` betrachten wir iterativ Teilprobleme auf dem Kreis, die jeweils P1 erfüllen. Wegen der Vielzahl der unterschiedlichen Teillösungen und –probleme ist in Tabelle 5.1 eine Übersicht der verwendeten Schreibweisen zusammengestellt, die als Referenz beim Lesen des Algorithmus zu verstehen ist.

Symbol	*Bemerkung*
z	bester bisher gefundener Kostenwert einer zul. Lösung
r	lang gewählter Auftrag
ψ	Flußwert
$X \subseteq \mathcal{S}$	bisher gelöster konsekutiver Abschnitt auf dem Kreis
$X' \subseteq \mathcal{S}$	aktuell zu lösender Abschnitt auf dem Kreis
$I^{[X')}$	mod. Eingabe, s. d. der linke Rand mit s_0 zusammen hängt (Def. 5.31)
$I^{[X']}$	mod. Eingabe, s. d. beide Ränder mit s_0 zusammenhängen
$I^{(X']}$	mod. Eingabe, s. d. der rechte Rand mit s_0 zus.hängt
$G^{k'}_{[X')}$	Transportgraph zu $I^{[X')}$, $I^{[X')}_{(X)}$, $I^{[X')}_{[X]}$ oder $I^{[X')}_{(X]}$ mit max. k' Umladeknoten, so daß s_0 und der linke Rand von X' zusammenhängend sind
$I^{[X')}_{(X]}$	Durch Alg. `Fix` entstandene mod. Eingabe: in X wurde die Lösung $G^{k'}_{(X]}$ fixiert, der Abschnitt in X' wurde so mod., daß der linke Rand und s_0 zusammenhängen

***Tabelle 5.1.:** Übersicht der Algorithmussymbole*

Satz 5.32. *Algorithmus* `k-Robot` *berechnet zu gegebenen Input* I *einen optimalen Transportgraphen.*

Algorithmus 8 k-Robot$(I = (\mathcal{S}, l, \Delta, \mathcal{R}, k))$

Initialisiere $z := \infty$
for all $r \in \mathcal{R} \cup \{\emptyset\}$ **do**
  for all $\psi \in [-m-1, \ldots, m+1]$ **do**
    if $r \neq \emptyset$ **and** $|\psi| \neq 1$ **then next for**
    if $S_{\{s_0\}} \neq \mathcal{S}$ **then**
      for all $k' = 0, \ldots, k$ **do**
        $G^{k'}_{[\{s_0\})} = \texttt{k-RobotLight}(I^{[\{s_0\})}, k', \psi, r)$
        $G^{k'}_{[\{s_0\}]} = \texttt{k-RobotLight}(I^{[\{s_0\}]}, k', \psi, r)$
        $G^{k'}_{(\{s_0\}]} = \texttt{k-RobotLight}(I^{(\{s_0\}]}, k', \psi, r)$
      end for
    end if
    $X = s_{\{s_0\}}, X' = S_{X \cup \{s_{ll_X}, s_{rl_X}\}}$
    while $X \neq \mathcal{S}$ **do**
      for all $k' = 0, \ldots, k$ **do**
        Konstruiere $I^{[X')}_{[X)}, I^{[X')}_{[X]}, I^{[X')}_{(X]}$, $I^{[X']}_{[X)}, I^{[X']}_{[X]}, I^{[X']}_{(X]}$ und $I^{(X')}_{[X)}, I^{(X')}_{[X]}, I^{(X')}_{(X]}$ mittels `Fix`.
        for all $k'' = 0, \ldots, k - k'$ **do**
          $G^{k''}_{[X')} = \text{argmin}\{l(G') | G' \in \{$
            $\texttt{k-RobotLight}(I^{[X')}_{[X)}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{[X')}_{[X]}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{[X')}_{(X]}, k'', \psi, r)\}\}$
          $G^{k''}_{[X']} = \text{argmin}\{l(G') | G' \in \{$
            $\texttt{k-RobotLight}(I^{[X']}_{[X)}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{[X']}_{[X]}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{[X']}_{(X]}, k'', \psi, r)\}\}$
          $G^{k''}_{(X']} = \text{argmin}\{l(G') | G' \in \{$
            $\texttt{k-RobotLight}(I^{(X']}_{[X)}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{(X']}_{[X]}, k'', \psi, r)$,
            $\texttt{k-RobotLight}(I^{(X']}_{(X]}, k'', \psi, r)\}\}$
        end for
      end for
      $X = X', X' = S_{X \cup \{s_{ll_X}, s_{rl_X}\}}$
    end while
    if $l(G^k_{[X]}) < z$ **then** $G^* = G^k_{[X]}, z = l(G^k_{[X]})$
  end for
end for
Gib G^* **zurück**

Beweis. Nach Korollar 5.7 existiert ein optimaler Transportgraph G_T^* mit Flußwert $\psi \in [-m-1, m+1]$ und aus Lemma 5.10 ist bekannt, daß höchstens ein Auftrag $r \in \mathcal{R}$ lang gewählt wird. Algorithmus 8 enumeriert alle möglichen Paare (ψ, r) aus den angegebenen Intervallen. Im folgenden gehen wir davon aus, daß der Algorithmus ein solches optimales Paar gefunden hat und wir beweisen die Behauptung per Induktion über die Anzahl t der While–Schleifen Durchläufe.

$t = 0$*:* In diesem Fall ist $X = \mathcal{S}_{\{s_0\}}$ und (I, ψ, r) erfüllt dann Eigenschaft P1. $G^k_{[X]}$ ist die Lösung des Problems $(I^{[X]}, k, \psi, r)$, das (I, ψ, r) mit zusätzliche künstlichen Aufträgen der Länge Null ist.

$t \to t+1$*:* Von den Berechnungen in Iteration t haben wir uns $3(k+1)$ viele Teillösungen gemerkt, nämlich $G^{k'}_{(X]}, G^{k'}_{[X]}$ und $G^{k'}_{(X)}$, für alle $k' = 0, \ldots, k$. In Durchlauf $t+1$ werden $9(k+1)$ Probleme der Typen $I^{[X']}_{[X)}, \ldots, I^{(X']}_{(X]}$ konstruiert, die jeweils optimal mit k-RobotLight gelöst werden können, da $X' := S_{X \cup \{s_{lr_X}, s_{rr_X}\}}$ gewählt wird und mit den Überlegungen in Lemma 5.30 sowie Bemerkung 5.27 gilt, daß I eingeschränkt auf X' die Eigenschaft P1 besitzt. Sei $G'_T = (\mathcal{S}, A', B')$ eine optimale Lösung für das Gesamtproblem (I, ψ, r). Dann betrachten wir die Abschnitte X und X' in G'_T. Aus Lemma 5.29 wissen wir, daß in X und X' jeweils der linke Rand, der rechte Rand oder beide Ränder mit s_0 zusammenhängen muß. Wir beschränken uns im Rest des Beweises auf einen dieser Fälle und überlassen es dem Leser, die restlichen Fälle auf die gleiche Weise zu beweisen. Wir nehmen somit an, daß

$$(s_{lr_{X'}}, s_{lr_{X'}+1}), (s_{lr_{X'}+1}, s_{lr_{X'}}) \notin A',$$

$$(s_{rr_{X'}}, s_{rr_{X'}-1}), (s_{rr_{X'}-1}, s_{rr_{X'}}) \in A',$$

$$(s_{lr_X}, s_{lr_X+1}), (s_{lr_X+1}, s_{lr_X}) \in A' \text{ und}$$

$$(s_{rr_X}, s_{rr_X-1}), (s_{rr_X-1}, s_{rr_X}) \notin A'.$$

Sei außerdem

$$|B' \cap X| \leq k', |B' \cap X'| \leq k'' + k',$$

d.h. innerhalb von X werden maximal k' Umladeknoten und in X' zusätzlich k'' Umladeknoten benutzt. Sicherlich können wir G'_T benutzen, um

einen Transportgraphen

$$G_T^* = (X' \cup \{s_{lr_{X'}}, s_{rr_{X'}}, s'_{rr_{X'}}\}, A^*, B^*)$$

für das Problem $(I^{(X')}, k'' + k', \psi, r)$ zu konstruieren, indem wir $\mathcal{S} \setminus (X' \cup \{s_{lr_{X'}}, s_{rr_{X'}}\})$ und alle dazu inzidenten Kanten entfernen und zusätzliche künstliche Knoten und Kanten für künstliche Auftrage einfügen. Wir werden zeigen, daß die Lösung, die wir für das Problem $(I_{[X)}^{(X')}, k'', \psi, r)$ konstruieren, ebenfalls eine Lösung für das Problem $(I^{(X')}, k'' + k', \psi, r)$ ist und daß die Kosten dieser Lösung gleich denen von G_T^* sind. Per Induktion erhalten wir dann mit $X = \mathcal{S}$ eine optimale Lösung für das Problem $(I^{[X]}, k, \psi, r)$, was äquivalent zum Problem (I, ψ, r) ist.

Sicherlich ist

$$G_{(X']}^{k''} = \texttt{k-RobotLight}(I_{[X)}^{[X')}, k'', \psi, r)$$

eine zulässige Lösung für $(I^{(X')}, k''+k', \psi, r)$. Der Algorithmus wählt zwar das Minimum von drei Alternativen aus, aber wir werden zeigen, daß die Behauptung schon für diese Wahl gilt. Falls eine der beiden anderen Möglichkeiten des Algorithmus noch billiger als die von uns gezeigte ist, so gilt die Behauptung weiterhin. Sei dazu $G_{[X)}^{k'}$ der von `Fix` verwendete Transportgraph zur Konstruktion von $I_{[X)}^{[X')}$. Angenommen $l(G_T^*) < l(G_{(X']}^{k''})$. Mit Lemma 5.5 nehmen wir an, daß $A^* = A_{\mathcal{R}}^* \dot\cup A_\psi^* \dot\cup A_C^*$. Sei $Y := X' \setminus X$, dann können wir A_C^* bezugnehmend auf die Zusammenhangskomponenten aus $db(\psi, r)$ weiter zerlegen in

$$A_C^* = (A_C^* \cap (X \times X)) \dot\cup (A_C^* \cap (Y \times Y)) \dot\cup \{(s_{lr_X+1}, s_{lr_X}), (s_{lr_X}, s_{lr_X+1})\}.$$

Dann partitionieren wir ebenfalls die Kantenmenge A des Transportgraphen

$$G_{(X']}^{k''} = (X' \cup \{s_{lr_{X'}}, s_{rr_{X'}}, s'_{rr_{X'}}\}, A, B)$$

entsprechend, so daß

$$A_C = A_C^X \dot\cup A_C^Y \dot\cup \{(s_{lr_X+1}, s_{lr_X}), (s_{lr_X}, s_{lr_X+1})\}$$

und wir stellen fest, daß A_C^X die Kanten aus $G_{[X)}^{k'}$ sind. Abbildung 5.11 illustriert genau diese Situation. Weil für festes ψ und r die Kosten eines

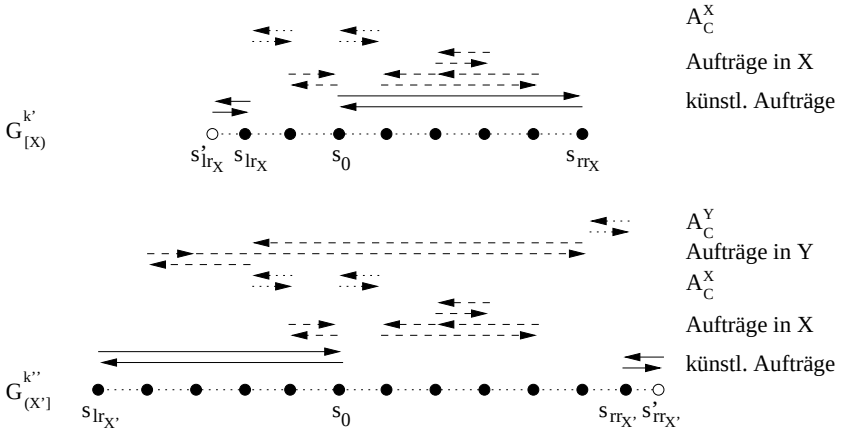


Abbildung 5.11.: *$G^{k'}_{[X)}$ ist ein optimaler Transportgraph für $I^{[X)}$. Die Aufträge in X (gestrichelt) definieren die Zusammenhangskomponenten von X. Zusammenhang wird durch Einfügen der Kanten A_C^X (gepunktet) erzeugt. Die künstlichen Aufträge (durchgehend) erzwingen, daß s_{lr_X} mit s_0 verbunden ist. Die Kanten aus A_C^X werden in $G^{k''}_{(X']}$ übernommen und durch Kanten A_C^Y ergänzt.*

Transportgraphen nur von A_C und B abhängen, folgt:

$$\begin{gathered} l(G_T^*) < l(G^{k''}_{(X']}) \Leftrightarrow \\ l(A_C^* \cap (X \times X)) + |B^* \cap X|\Delta + l(A_C^* \cap (Y \times Y)) + |B^* \cap Y|\Delta \\ < l(A_C^X) + k'\Delta + l(A_C^Y) + k''\Delta \end{gathered} \tag{5.2}$$

Konstruiere $G_X = (X \cup \{s'_{lr_X}, s_{lr_X}, s_{rr_X}\}, A_X, B_X)$: Initialisiere zunächst $B_X := B^* \cap X$ und $A_X := (A^* \cap (X \times X))$. Füge anti–parallele Kanten $(s'_{lr_X}, s_{lr_X}), (s_{lr_X}, s'_{lr_X})$ zwischen dem linken Rand und dem künstlichen Knoten A_X hinzu. Sei $B_{X^r} \subseteq B_X$ die Teilmenge der Umladeknoten, die im Uhrzeigersinn von s_0 liegen und seien diese Knoten in dieser Drehrichtung entsprechend indiziert. Dann erweitern wir die Menge A_X durch die Kanten $(s_0, s_{b_1}), (s_{b_1}, s_{b_2}), \ldots, (s_{b_{|B_{X^r}|}}, s_{rr_X}), (s_{rr_X}, s_0)$. Sei P ein gerichteter (s_0, s_{lr_X})–Pfad in A^*. P existiert, ansonsten wäre G_T^* nicht zusammenhängend. Für jeden Knoten $s_b \in B_{X^l} := B_X \setminus B_{X^r}$ existiert genau eine Kante

(s_a, s_c) in P, die s_b kreuzt. Ersetze $(s_a, s_c) \in A_X$ durch (s_a, s_b) und (s_b, s_c). Nun ist G_X ein Transportgraph für $(I^{[X)}, k'\psi, r)$. Da $G^{k'}_{[X)}$ per Induktion eine optimale Lösung für $(I^{[X)}, k', \psi, r)$ ist, muß $l(G_X) \geq l(G^{k'}_{[X)})$ gelten. Da sich G_X und $G^{k'}_{[X)}$ hinsichtlich der Kosten nur in der Anzahl der Umladestationen und den Kanten zwischen Zusammenhangskomponenten unterscheiden können, gilt:

$$l(A^*_C \cap (X \times X)) + |B^* \cap X|\Delta \geq l(A^X_C) + k'\Delta \tag{5.3}$$

Wenden wir dieses Ergebnis auf die Aussage 5.2 an, erhalten wir

$$l(G^*_T) < l(G^{k''}_{[X']}) \Leftrightarrow l(A^*_C \cap (Y \times Y)) + |B^* \cap Y|\Delta < l(A^Y_C) + k''\Delta \tag{5.4}$$

Wir konstruieren $G_Y = (X' \cup \{s_{lr_X}, s_{rr_X}, s_{rr'_X}\}, A_Y, B_Y)$, indem wir Komponenten von $G^{k'}_{[X)}$ innerhalb von X und Komponenten von G^*_T innerhalb von Y benutzen. Sei $B_Y := (B^* \cap Y) \cup (B \cap X)$, und sei anfangs $A_Y = (A^* \cap (Y \times Y)) \cup (A \cap (X \times X)) \cup \{(s_{lr_X}, s_{lr_{X'}+1}), (s_{lr_{X'}+1}, s_{lr_{X'}})\}$. Weiterhin sei $(s_a, s_c) \in (A^* \cap (Y \times Y))$ die Kante, die die Knoten aus X komplett kreuzt und in der Prozedur `Fix` zerschnitten wurde. Sortiere alle $b_i \in (B \cap X)$ entsprechend ihres Vorkommens auf dem eindeutigen (s_a, s_c)–Pfad entlang des Kreises. Dann ersetze (s_a, s_c) durch $(s_a, s_{b_1}), (s_{b_1}, s_{b_2}), \ldots, (s_{b_{|B\cap X|}}, s_c)$. Der so entstandene Graph G_Y ist ein Transportgraph für das Problem $(I^{(X']}_{[X)}, k'' + k', \psi, r)$ und die Kosten betragen

$$\begin{aligned} l(G_Y) &= l(A^*_C \cap (Y \times Y)) + |B^* \cap Y|\Delta + l(A_C \cap (X \times X)) + |B \cap X|\Delta \\ &= l(A^*_C \cap (Y \times Y)) + |B^* \cap Y|\Delta + l(A^X_C) + k'\Delta \\ &\underset{5.4}{<} l(A^Y_C) + k''\Delta + l(A^X_C) + k'\Delta \\ &= l(G^{k''}_{[X']}) \end{aligned}$$

im Widerspruch zur Optimalität von $G^{k''}_{[X']}$ als Lösung des Teilproblems $(I^{(X']}_{[X)}, k'' + k', \psi, r)$.

Am Ende der While–Schleife wird das neue X' um mindestens zwei Knoten erweitert. Lemma 5.30 versichert uns, daß $X' = S_{X \cup \{s_{lr_X}, s_{rr_X}\}}$

gewählt werden darf, unabhängig davon, ob in X der linke oder der rechte Rand mit s_0 verbunden wurde.

Falls $r = \emptyset$ werden $2m + 3$ viele Werte für $\psi \in [-m - 1, m + 1]$ ausprobiert. Falls $r \in \mathcal{R}$ werden nur die Werte $\psi \in \{-1, +1\}$ enumeriert (s. Korollar 5.14). Somit werden die beiden äußeren for–Schleifen $2m + 3 + 2m \in \mathcal{O}(m)$ mal durchlaufen. Daneben hängt die Laufzeit des Algorithmus hauptsächlich von den drei verschachtelten Schleifen ab, welche jeweils $\mathcal{O}(n)$ häufig ausgeführt werden, und von der Laufzeit, die `k-RobotLight` benötigt. Somit können wir die Laufzeit mit $\mathcal{O}(n^3 m^2 \log n)$ begrenzen. □

Analog zum exogenen Fall können wir uns auf wenige Möglichkeiten für Werte von ψ und r beschränken, wenn der zugrunde liegende Graph ein Pfad ist.

Korollar 5.33. *Das endogene und semi–präemptive Transportproblem auf dem Pfad kann in $\mathcal{O}(n^3 m \log n)$ Schritten optimal gelöst werden.*

Beweis. Die Argumentation beruht wie im Beweis zu Korollar 5.16 auf der Beobachtung, daß stets $\psi = 0$ gilt und keine Drehrichtung für die Aufträge festgelegt werden muß. □

Die Bemerkung 5.17 zur Reduktion der Anzahl der Zusammenhangskomponenten gilt auch im endogenen Fall.

In Teil II werden die Details und Rechenergebnisse einer Implementierung des dynamischen Programms erläutert.

Kapitel 6.

Semi–Präemptives Transportieren auf Bäumen

Wir haben in Kapitel 5 gesehen, daß das semi–präemptive Transportproblem auf Pfaden im komplexitätstheoretischen Sinne einfach zu lösen ist. Die Erweiterung des Problems auf einen Kreis konnte durch intensives Ausprobieren verschiedener Parameter gelöst werden. Ein Kreis macht das Problem also nicht strukturell schwieriger. Wir werden nun sehen, daß, wenn der zugrunde liegende Graph, auf dem sich der Roboter bewegen darf, ein zusammenhängender kreisfreier Graph ist, das Problem zu den schwierigen Problemen gezählt werden darf.

Auf dem Pfad nutzen wir die Eigenschaft aus, daß jedem Auftrag ein eindeutiger Pfad zugeordnet werden kann. Die Verallgemeinerung auf dem Kreis bietet jedem Auftrag zwei mögliche Richtungen des Transports (mit und gegen den Uhrzeigersinn). Der Baum weist ebenso die schöne Eigenschaft auf, daß es zwischen je zwei Knoten einen eindeutigen Pfad gibt. Damit werden uns in Abschnitt 6.1 beschäftigen. Trotz dieser Gemeinsamkeit ist das Problem auf dem Baum schwierig, sogar dann schon, wenn wir $k = 0$ wählen und somit die nicht–präemptive Variante betrachten. In Abschnitt 6.2 geben wir eine Reduktion vom Steinerbaum–Problem an. Abschließend untersuchen wir in Abschnitt 6.3, inwieweit man sich der Optimallösung durch Heuristiken annähern kann.

6.1. Eigenschaften

Sei $I = (\mathcal{S}, l, \Delta, \mathcal{R}, B')$ und außerdem eine natürliche Zahl k in der endogenen Variante die Eingabe für das semi–präemptive Transportproblem. Die Kostenfunktion $l : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_+$ beschreibt den zugrunde liegenden Baum $G = (\mathcal{S}, A_G)$. $s_0 \in \mathcal{S}$ ist der ausgezeichnete Startknoten und wir werden s_0 häufig als Wurzel des Baums betrachten. Die Menge der Aufträge $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ muß im allgemeinen nicht alle Knoten verwenden. Im Gegensatz zur Variante auf dem Pfad können wir nicht einfach unbenutzte Knoten löschen. Allerdings können wir Blätter von G, die weder Ausgangs– noch Endknoten eines Auftrages sind, entfernen, da in keiner Optimallösung Aufträge über sie transportiert werden. Ähnliches gilt für Knoten mit Grad zwei, die von keinem Auftrag betroffen sind. Diese können gelöscht und die Distanz auf der neuen Kante zwischen den bisherigen Nachbarn angepaßt werden, falls kein Auftrag auf ihnen beginnt oder endet. Somit nehmen wir an, daß alle Knoten mit Grad eins oder zwei Start oder Ziel mindestens eines Auftrags sind.

Wie vorher suchen wir einen kostenminimalen Transportgraphen $G_T = (\mathcal{S}, A, B)$, der eine Eulertour und für alle Aufträge $r \in \mathcal{R}$ einen gerichteten Pfad enthält. Wir partitionieren die Kantenmenge A des Transportgraphen mit den Argumenten aus Lemma 5.5 in Teilmengen $A = A_{\mathcal{R}} \dot\cup A_\psi \dot\cup A_C$. Wie im Fall des Pfades muß jedesmal, wenn in G_T eine Kante des Baumes in der einen Richtung durchlaufen wird, sie auch in der entgegengesetzten Richtung benutzt werden, oder G_T ist nicht eulersch. Der Beweis dazu ist analog zu Lemma 5.2 mit Flußwert $\psi = 0$. Statt A_ψ schreiben wir zur Erinnerung nun A_0. Wegen der Eindeutigkeit der Pfade können wir Definition 5.1 des Flusses aus dem vorherigen Kapitel anpassen, um zu zählen, wie viele Aufträge über eine Kante des Baumes hinweg gehen.

Definition 6.1 (Fluß). *Der* Fluß $\phi(s_i, s_j)$ über die Baumkante $(s_i, s_j) \in A_G$ *ist die Anzahl der Aufträge, die* (s_i, s_j) *auf ihrem eindeutigen Pfad in der Richtung von* s_i *nach* s_j *kreuzen minus der Anzahl der Aufträge, die* (s_i, s_j) *von* s_j *nach* s_i *kreuzen.*

Wir wollen uns erneut auf gradbalancierte Instanzen zurückziehen und

können die minimale Anzahl der augmentierenden Kanten zwischen s_i und s_j berechnen. Wir fügen für alle $(s_i, s_j) \in A_G$

$$|\phi(s_i, s_j)| \text{ viele Kanten } \left\{ \begin{array}{ll} (s_i, s_j) & \text{falls } \phi(s_i, s_j) \leq 0 \\ (s_j, s_i) & \text{falls } \phi(s_i, s_j) > 0 \end{array} \right\} \tag{6.1}$$

in A_0 ein. Wegen der Eindeutigkeit der Pfade zwischen je zwei Knoten gilt die Bemerkung 5.17, mit der wir die Anzahl der Zusammenhangskomponenten reduzieren können, auch für Bäume. Auf diese Weise entsteht ein gradbalancierter Graph $\mathtt{bal} = (\mathcal{S}, A_{\mathcal{R}} \dot\cup A_0)$, der für jede Kante des zugrunde liegenden Baumes G mindestens zwei Kanten enthält, die diese überdecken.

Bemerkung 6.2. *Wir können die Anzahl der Kanten, die wir zur Konstruktion von* bal *in* A_0 *einfügen, durch* $\mathcal{O}(nm)$ *beschränken, wenn wir die angegebene Anweisung 6.1 benutzen. In [FG93] geben Frederickson und Guan eine Methode zur Konstruktion von* bal *an, die aufeinanderfolgende Kanten zählt und zu längeren Kanten vereinigt. Sowohl die Laufzeit des Verfahrens als auch die Anzahl der Kanten in* A_0 *kann dadurch auf* $\mathcal{O}(n + m)$ *begrenzt werden.*

6.2. Komplexität

In diesem Abschnitt werden wir zeigen, daß das semi–präemptive Transportproblem schwierig im Sinne der Komplexitätstheorie ist, falls der zugrunde liegende Graph ein Baum ist. Das Problem zählt sogar schon zu den schwierigen Problemen, wenn kein Umladevorgang stattfindet. Dieser Spezialfall ist durch die Wahl von $k = 0$ im endogenen und durch $B' = \emptyset$ im exogenen Fall enthalten. Der Beweis wird ein anderes Problem, von dem bekannt ist, daß es $\mathcal{NP}$–vollständig ist, auf das nicht–präemptive Transportproblem reduzieren.

Problem 6.3 (Steinerbaum)

Gegeben: Ein ungerichteter Graph $G = (V, E)$, eine Menge von *Terminalen* $R \subseteq V$, $d \in \mathbb{N}$. Die Knoten $V \setminus R$, die keine Terminale sind, nennen wir *Steinerknoten*.

Gesucht: Ein Teilbaum von G, so daß alle Terminale R überdeckt sind, und der maximal d Kanten benutzt.

Satz 6.4 (Garey, Johnson 1979 [GJ79]). *Das Steinerbaum–Problem ist $\mathcal{NP}$–vollständig. Es bleibt $\mathcal{NP}$–vollständig, wenn* G *bipartit ist.*

Das Entscheidungsproblem des Pickup–And–Delivery–Problems ist das folgende: Gegeben eine Instanz $I = (\mathcal{S}, l, \Delta, \mathcal{R}, B')$ sowie eine Zahl $C \in \mathbb{R}$, gibt es einen Transportgraphen mit Kosten höchstens C? Wegen der Bedeutung der Aussage für das von uns betrachtete Problem werden wir die Beweisidee des folgenden Satzes wiedergeben.

Satz 6.5 (Frederickson, Guan 1993 [FG93]). *Das nicht–präemptive Transportproblem auf Bäumen ist $\mathcal{NP}$–vollständig.*

Beweis. Das nicht–präemptive PDP liegt in der Klasse $\mathcal{NP}$, da wir einen Transportgraphen raten und in polynomieller Zeit überprüfen können, ob alle Aufträge transportiert werden können. Außerdem können wir leicht die Gesamtkosten als Summe der Kantenkosten berechnen und kontrollieren, ob diese kleiner als C sind.

Wir reduzieren im folgenden das Steinerbaum–Problem auf bipartiten Graphen auf das nicht–präemptive Transportproblem. Sei $G = (V, E)$ der Graph für das bipartite Steinerbaum–Problem, $R \subseteq V$ sei die Menge der zu überdeckenden Terminale und $d \in \mathbb{N}$ eine Schranke. Ohne Einschränkung können wir annehmen, daß G zusammenhängend ist. Wir beschreiben jetzt, wie eine Instanz für das nicht–präemptive Transportproblem konstruiert wird. Die Abbildungen 6.1 und 6.2 demonstrieren die Reduktion an einem Beispiel. Abbildung 6.1 enthält einen bipartiten Graphen mit $R = \{v_1, v_2, v_3\}$ und eingezeichnetem aufspannenden Baum (durchgezogene Linien).

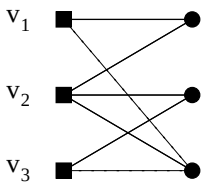


Abbildung 6.1.: *Beispielgraph für das bipartite Steinerbaum–Problem*

Der Baum $G' = (\mathcal{S}, A)$ wird durch folgende Maßnahmen erschaffen. Jedem Knoten $v \in V$ weisen wir eine natürliche Zahl $\#(v)$ zu, und v wird in G′ insgesamt $\#(v) + 1$ Repräsentanten haben, die wir $v^1, \ldots, v^{\#(v)}, v$

taufen. Sei zunächst $T \subseteq E$ ein beliebiger aufspannender Baum von G, dann beginnen wir mit den Mengen $\mathcal{S} = V$ und $A = T$, wobei wir A erweitern werden. Wir initialisieren alle $v \in V$ mit dem Wert $\#(v) = 0$. Wir betrachten nun eine beliebige Kante $e = (u, v) \in E \setminus T$, also eine, die nicht im ausgewählten Baum liegt. Weil G bipartit ist, liegt genau einer der Knoten u und v in R. Sei ohne Einschränkung der Allgemeinheit $v \in R$. Erhöhe den Wert $\#(v)$ um eins und füge einen neuen Knoten $v^{\#(v)}$ zu $\mathcal{S}$ hinzu. Zudem erweitern wir A um die Kante $(u, v^{\#(v)})$. Die Kosten solcher Kanten sollen 1 betragen. Dieses Vorgehen wiederholen wir für jede Kante $(u, v) \in E \setminus T$, die nicht in dem aufspannenden Baum T liegt.

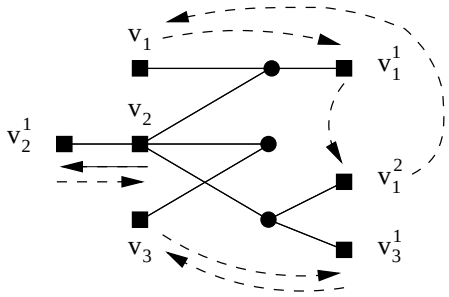


Abbildung 6.2.: *Durch die Reduktion entstandenes Transportproblem zu Beispiel 6.1*

Für jeden Knoten v, dessen Zähler $\#(v)$ nach diesem ersten Schritt Null ist, setzen wir $\#(v) = 1$ und vervollständigen G′ durch Hinzunahme eines Knotens $v^{\#(v)}$ sowie einer Kante $(v, v^{\#(v)})$. Diese Kanten werden mit Kosten 0 ausgestattet. Die Menge $\mathcal{R}$ der Aufträge setzt sich wie folgt zusammen. Jedem Knoten $v \in V$ ordnen wir $\#(v) + 1$ Aufträge $(v, v^1), (v^1, v^2), \ldots, (v^{\#(v)-1}, v^{\#(v)}), (v^{\#(v)}, v)$ zu. Schließlich wählen wir einen beliebigen Knoten aus R als Startknoten und setzen $C = 2d + l(bal)$, wobei $l(bal)$ die Summe der Gewichte im balancierten Graphen ist.

Zu Beginn der Konstruktion besteht G′ aus dem Baum T. Danach werden Pfade der Länge eins an Knoten des Baumes angehangen, wodurch die Baumeigenschaft von G′ erhalten bleibt. G′ besitzt nach Konstruktion maximal $|E| + |R| + 1$ Knoten und höchstens $|E| + |R|$ Kanten. Der balancierte Graph bal kann wie im vorigen Abschnitt angegeben in polynomieller Zeit berechnet werden. Insgesamt benötigt die Erschaffung der Transport–Instanz also polynomiell viele Zeitschritte. Es verbleibt zu zeigen, daß G genau einen Steinerbaum mit höchstens d vielen Kanten

besitzt, wenn es einen Transportgraphen für G' und $\mathcal{R}$ gibt, dessen Gesamtkosten maximal C betragen.

Sei S ein Teilbaum von G, der mindestens die Terminale R aufspannt und Kosten $d' \leq d$ besitzt. Weiter sei bal der balancierte Graph der Eingabe G' und $\mathcal{R}$. Für jede Kante $(u, v) \in S$ fügen wir zwei anti–parallele Kanten (u, v) und (v, u) zu bal hinzu. Weil der Graph vorher gradbalanciert war, ist er es nach Hinzufügen der augmentierenden Kanten weiterhin. Zudem ist bal danach eulersch, weil alle Knoten, in denen Aufträge starteten und endeten, einen korrespondierenden Knoten $v \in R$ besitzen. Die Kosten dieser Erweiterung betragen $2d'$ und die Gesamtkosten des so entstandenen Transportgraphen bal' belaufen sich auf $2d' + l(bal) \leq 2d + l(bal) = C$.

Andererseits sei $G_T = (\mathcal{S}, A)$ ein nicht–präemptiver Transportgraph für G' und $\mathcal{R}$ mit Kosten $C' \leq C$. Sei $A = A_{\mathcal{R}} \dot\cup A_\Psi \dot\cup A_C$ wie in Lemma 5.5 partitioniert, so daß A_C genau die anti–parallelen Kanten zwischen den Zusammenhangskomponenten von bal enthält. Zu jedem Knoten in R gibt es eine Zusammenhangskomponente in bal, die auf der Eulertour liegt. Wir entwerfen einen Steinerbaum S für G, indem wir für jedes Paar von anti–parallelen Kanten in A_C eine Kante zwischen den entsprechenden Knoten aus G wählen. Da die Eulertour alle Zusammenhangskomponenten von bal mindestens einmal besucht, muß S die Menge R aufspannen und somit einen Steinerbaum enthalten. Die Kosten von S sind $\frac{C' - l(bal)}{2} \leq \frac{C - l(bal)}{2} = d$. □

Das nicht–präemptive PDP ist sogar schon schwierig, wenn der zugrunde liegende Baum eine besondere Struktur hat: Hauptmeier et al. haben in [HKRW01] die $\mathcal{NP}$–Vollständigkeit für Kammgraphen nachgewiesen. Kammgraphen bestehen aus einem Pfad und einzelnen an den Knoten des Pfades angehangenen „Zinken“, so daß der Grad jedes Knotens zwischen eins und drei liegt.

Im Gegensatz zum nicht–präemptiven Transportproblem ist für die präemptive Version auf Bäumen ein polynomieller Algorithmus bekannt [FG92]. In diesem Fall darf an jedem Knoten kostenlos umgeladen werden. Das bedeutet, daß wir alle Kanten und insbesondere die Aufträge in

minimale Teilstücke zerschneiden dürfen, ohne die Zulässigkeit zu verlieren. In unserer Modellierung enthält der daraus konstruierte Graph $\mathtt{bal}$ genau eine Zusammenhangskomponente und somit eine Eulertour.

6.3. Approximation

Das semi–präemptive Transportproblem gehört zu denjenigen Problemen, für die kein effizienter Algorithmus bekannt ist, um es optimal zu lösen. Im vorigen Abschnitt wurde gezeigt, daß es zu jenen Problem gehört, die wir im Sinne der Komplexitätstheorie als schwierig bezeichnen wollen.

Wir werden uns in diesem Abschnitt damit beschäftigen, welche Qualität wir von Lösungen erwarten können, die von Algorithmen mit polynomieller Laufzeit geschaffen wurden. Dabei werden wir uns weiterhin auf den Fall beschränken, daß der zugrunde liegende Graph ein Baum ist.

6.3.1. Hilfsgraph

Das nicht–präemptive Transportproblem ist unter polynomieller Reduktion äquivalent zum Steinerbaum–Problem. Ein klassisches Verfahren, um Steinerbaum–Probleme zu approximieren, ist die Abschätzung über minimal aufspannende Bäume. Wendet man diese Idee auf die nicht–präemptive Variante an, kann gezeigt werden, daß die dadurch entstehenden Lösungen höchstens um den Faktor $4/3$ schlechter sind als der Wert einer Optimallösung. Generell können alle Approximationsalgorithmen für das Steinerbaum–Problem zur Berechnung von Näherungslösungen für das nicht–präemptive Transportproblem herangezogen werden [FG93].

Leider können wir die Modellierung aus der nicht–präemptiven Variante nicht auf die semi–präemptive Version anwenden. Durch wenige Modifikationen ist es jedoch möglich, zumindest die Idee der Approximation über einen aufspannenden Baum für das exogene semi–präemptive Transportproblem zu übertragen.

Zu diesem Zweck betrachten wir den gerichteten Hilfsgraphen $H = (V, E^r \dot\cup E^b)$, der genauso konstruiert wird wie im Falle des Kreises: Für

jede Zusammenhangskomponente in bal erzeugen wir einen Knoten in H. Da bal gradbalanciert ist, sind alle Zusammenhangskomponenten automatisch stark zusammenhängend. Wir können beobachten, daß es in bal Zusammenhangskomponenten gibt, die Auftragskanten aus $A_{\mathcal{R}}$ enthalten und Zusammenhangskomponenten ohne solche Kanten. Zusammenhangskomponenten des zweiten Typs nennen wir *trivial*, falls sie nicht den Startknoten enthalten. In einem zulässigen Transportgraphen müssen alle Auftragskanten in einer Eulertour enthalten sein. Andererseits müssen triviale Zusammenhangskomponenten nicht notwendigerweise auf einer Eulertour liegen. Dementsprechend gibt es in H Knoten, die den trivialen Zusammenhangskomponenten entsprechen. Diese Knoten werden wir ebenfalls *trivial* nennen. Es sei bemerkt, daß in den Hilfsgraphen, die auf Kreisen oder Pfaden basieren, keine trivialen Knoten vorkommen. Die in rot und blau eingefärbten Kantenmengen E^r und E^b werden auf die gleiche Art konstruiert wie in Abschnitt 5.3.1 angegeben. Eine rote Kante entspricht somit einer Nachbarschaft zweier Zusammenhangskomponenten, und eine blaue Kante symbolisiert einen möglichen Umladevorgang. Die Kosten einer roten Kante entsprechen dem doppelten minimalen Abstand der zugehörigen Zusammenhangskomponenten in bal und die Kosten der blauen Kanten sind konstant Δ. Wir erlauben parallele Kanten, H ist also ein Multigraph.

6.3.2. Steinerarboreszenzen

Problem 6.6 (r–**Steinerarboreszenz**)

Gegeben: Ein gerichteter Graph $G = (V, E)$, eine Kostenfunktion $c : E \to \mathbb{R}_+$, eine Menge von *Terminalen* $R \subseteq V$, eine Wurzel $r \in V$.

Gesucht: Eine Teilmenge $S \subseteq E$, so daß für alle Terminale $t \in R$ ein gerichteter (r, t)–Pfad existiert, und die $\sum_{e \in S} c(e)$ minimiert.

Wegen der Minimierungsforderung ist jede optimale r–Steinerarboreszenz zyklenfrei. Außerdem kann jede Steinerbaum Instanz leicht in eine r–Steinerarboreszenz Instanz überführt werden. Durch diese Verallgemeinerung vererbt sich die $\mathcal{NP}$–Vollständigkeit [HRW92].

Satz 6.7. *Sei* I *die Eingabe für das exogene Transportproblem auf einem Baum. Die Kosten eines optimalen Transportgraphen für* I *betragen* $l(\mathtt{bal}) + c(S)$*, wobei* bal *der gradbalancierte Graph zu* I *und* S *eine kostenminimale* ν_0*–Steinerarboreszenz auf dem Hilfsgraphen* H *ist.*

Beweis. Die Beweisführung ist ähnlich der zu Satz 5.12. Gegeben eine Instanz I mit Startknoten $s_0 \in \mathcal{S}$, dann konstruieren wir nach Vorschrift 6.1 den gradbalancierten Graphen bal. Zu bal konstruieren wir den rot–blau gefärbten Hilfsgraphen $H = (V, E = E^r \dot\cup E^b)$, dessen Knotenmenge V eine Teilmenge $V' \subseteq V$ von trivialen Knoten enthält. Setze $R := V'$ und löse das ν_0–Steinerarboreszenz–Problem für H, wobei ν_0 die Zusammenhangskomponente aus bal repräsentiert, die den Startknoten s_0 enthält. Sei dann S eine kostenminimale Lösung. S spannt die nicht–trivialen Knoten $V \setminus R$ auf.

Konstruiere einen Transportgraphen $G_T = (\mathcal{S}, A_\mathcal{R} \dot\cup A_0 \dot\cup A_C, B)$ wie folgt, wobei $A_\mathcal{R}$ und A_0 der Kantenmenge von bal entsprechen, und initialisiere $A_C = \emptyset, B = \emptyset$. Für jede von S benutzte rote Kante $(\nu_i, \nu_j) \in E^r$ existieren nach Konstruktion von H zwei benachbarte Knoten $s_a \in Z_i$ und $s_b \in Z_j, i \neq j$, wobei $Z_i, Z_j \subseteq \mathcal{S}$ starke Zusammenhangskomponenten in bal sind. Füge für jede solche rote benutzte Kante zwei anti–parallele Kanten (s_a, s_b) und (s_b, s_a) zu A_C hinzu. Das Hinzufügen der beiden augmentierenden Kanten addiert $l(s_a, s_b) + l(s_b, s_a) = 2l(s_a, s_b) = c(\nu_i, \nu_j)$ auf die Kosten von G_T.

Für jede von S benutzte blaue Kante $(\nu_i, \nu_j) \in E^b$ existieren in B′ ein Umladeknoten $s_b \in \mathcal{S}$ und ein Auftrag $(s_a, s_c) \in \mathcal{R}$, der s_b kreuzt. Anfangs ist $A_\mathcal{R} = \mathcal{R}$. Im Laufe des Vorgehens können Kanten aus $A_\mathcal{R}$ zerschnitten werden. Wir wissen, daß $s_b \notin B$, denn ansonsten gibt es in S zwei blaue Kanten mit Kopf ν_j. Dann ist S aber keine Arboreszenz. Da $s_b \notin B$, wurde bisher keine Kante aus $A_\mathcal{R}$ an s_b zerschnitten. Somit existiert in $A_\mathcal{R}$ eine Kante, die s_b kreuzt. Füge s_b zu B hinzu und ersetze in $A_\mathcal{R}$ die Kante (s_a, s_c) durch (s_a, s_b) sowie (s_b, s_c). Durch das Hinzufügen von s_b zu B erhöhen sich die Kosten von G_T um $\Delta = c(\nu_i, \nu_j)$.

Beide Modifikationen erhalten die Gradbalance. Da in S für alle nicht–trivialen Knoten $\nu \in V \setminus V'$ ein gerichteter (ν_0, ν)–Pfad existiert, liegen

alle nicht–trivialen Zusammenhangskomponenten von $\mathtt{bal}$ und somit auch alle Kanten aus $A_{\mathcal{R}}$ zusammen mit s_0 auf einer Eulertour im Transportgraphen G_T. □

Aus Satz 6.7 geht hervor, daß eine Approximation des Steinerarboreszenz–Problems mit endlicher Güte eine endliche Approximation für das exogene Transportproblem auf Bäumen liefert. Dazu werden wir die Idee der Qualitätsabschätzung über einen aufspannenden Baum auf gerichtete Graphen übertragen und auf das Transportproblem anwenden.

Wir wissen, daß das Problem, eine minimale Steinerarboreszenz zu finden, schwierig zu approximieren ist: Unter der Voraussetzung, daß $\mathcal{NP} \not\subseteq \mathrm{DTIME}(\mathcal{O}(n^{\log\log n}))$ gilt, gibt es kein polynomielles Verfahren, das eine Güte besser als $\ln n$ garantiert. D.h., falls es für die Probleme der Klasse $\mathcal{NP}$ keine exakten Algorithmen mit schwach superpolynomieller Laufzeit gibt, liegt r–Steinerarboreszenz nicht in $\mathcal{APX}$. Der Beweis dazu basiert auf der Reduktion von Set Cover, für das diese Aussage bekannt ist [Fei98].

Andererseits liefert eine sehr einfache Methode schon eine n–Approximation: Für jedes Terminal $t \in V$ bestimmen wir den kürzesten Pfad von der Wurzel zu t, unabhängig davon, ob er über Steinerknoten geht, und bilden so eine Lösung. Charikar et al. haben 1999 den ersten nicht–trivialen Approximationsalgorithmus für allgemeine Steinerarboreszenz Instanzen mit einer Güte von $\mathcal{O}(n^{\varepsilon})$ für jede Konstante $\varepsilon > 0$ vorgestellt [CCC+99].

6.3.3. Exogenes Transportieren auf Bäumen

Die bekannten Resultate über Näherungslösungen für Steinerarboreszenzen, die im letzten Abschnitt vorgestellt wurden, geben wenig Hoffnung, eine $(1+\varepsilon)$–Approximation mit einer Konstante $\varepsilon > 0$ für das exogene Transportproblem auf Bäumen zu finden, da Satz 6.7 die Zielfunktionswerte aneinander koppelt. Der Hilfsgraph H, für den wir eine Steinerarboreszenz bestimmen wollen, hat jedoch eine spezielle Struktur. Zum einen

ist der Teilgraph von H, der nur rote Kanten enthält, symmetrisch, d.h. für jede Kante (u, v) existiert auch eine Rückkante (v, u). Allein der blaue Teilgraph kann asymmetrisch sein. Aber auch hier können wir Nützliches beobachten. Blaue Kanten in H entsprechen einer Umlademöglichkeit in $\mathtt{bal}$. Diese besteht, falls eine Kante aus $A_{\mathcal{R}}$, also ein Auftrag, über Knoten einer anderen Zusammenhangskomponente hinweg geht. Kanten aus $A_{\mathcal{R}}$ können aber nach Definition nur in nicht–trivialen Zusammenhangskomponenten enthalten sein. Folgerichtig existiert in H keine blaue Kante, deren Schwanz ein trivialer Knoten ist. Diese beiden Beobachtungen wollen wir festhalten, da sie für die Qualitätsabschätzung des nun folgenden Verfahrens wichtig sind.

Bemerkung 6.8. *Sei* $H = (V, E = E^r \dot{\cup} E^b)$ *der wie angegeben konstruierte Hilfsgraph zu* $\mathtt{bal}$. *Mit den vorigen Kommentaren gilt:*

(i) $(u, v) \in E^r \Leftrightarrow (v, u) \in E^r$ *und* $c(u, v) = c(v, u)$.

(ii) $(u, v) \in E^b \Rightarrow u$ *ist nicht–trivial.*

Wir betrachten den *pfadreduzierten Hilfsgraphen* $H' = (V \setminus V', E')$, dessen Knotenmenge aus den nicht–trivialen Knoten von H besteht bzw. dessen Knotenmenge die nicht–trivialen Zusammenhangskomponenten von $\mathtt{bal}$ repräsentiert. Eine gerichtete Kante $(u, v) \in E'$ zwischen je zwei Knoten entspricht dem kürzesten Pfad von u nach v in H. Die Kosten einer Kante in H' entsprechen den Kosten des kürzesten Weges in H. Falls auf einem kürzesten Weg wenigstens eine blaue Kante aus H enthalten ist, so färben wir die Kante in H' ebenfalls blau, und ansonsten rot.

Satz 6.9. *Sei* T *eine kostenminimale* v_0*–Arboreszenz in* H' *und* S *eine kostenminimale* v_0*–Steinerarboreszenz in* H *mit* $v_0 \in V \setminus V'$. *Es gilt:*

$$c(T) \leq 2c(S)$$

Beweis. Es genügt zu zeigen, daß in H' eine v_0–Arboreszenz mit Kosten höchstens $2c(S)$ existiert. Sei S eine optimale v_0–Steinerarboreszenz für H. Wir beginnen in der Wurzel v_0 und durchlaufen S mittels Tiefensuche. Jedesmal, wenn wir einen Knoten besuchen, notieren wir dies. Sei $P = (v_0, \ldots, v_p)$ die dadurch beschriebene Folge der durch S induzierten

Knoten. In P tritt ein Knoten mehrmals auf, außer er ist ein Blatt in S. Durch P wird ein Kantenzug in H beschrieben, wobei einige Kanten entgegen ihrer Orientierung auftreten.

P enthält mindestens alle nicht–trivialen Knoten. Wir betrachten P und markieren alle $v \in V \setminus V'$. Dann sei $Q \subseteq P$ die Teilfolge von P, in der die so markierten nicht–trivialen Knoten entsprechend ihrer Reihenfolge in P aufgelistet sind, wobei ein Knoten mehrfach auftreten kann. Zwei in Q aufeinanderfolgenden Knoten entspricht eine Folge von Knoten in P und somit ein Kantenzug in H. Darin wird jede Kante aus S zweimal durchlaufen, nämlich einmal vorwärts und einmal rückwärts entgegen ihrer Orientierung. Somit sind die Kosten des durch P beschriebenen Kantenzuges genau $2c(S)$.

Da $v_0 \in V \setminus V'$, ist die Wurzel v_0 in Q enthalten. Falls $|V \setminus V'| = 1$ ist, sind wir fertig, also nehmen wir an, daß $|V \setminus V'| \geq 2$. Sei v_i ein beliebiger Knoten in Q, aber nicht der letzte, und v_j der Nachfolger von v_i in Q. Es gilt $v_i \neq v_j$, ansonsten enthält S eine Schleife. Wir betrachten nun nur die Paare v_i, v_j, bei denen v_j das erstemal während der Tiefensuche besucht wird, d.h. falls kein $k < j$ existiert, mit $v_k \in Q, v_k = v_j$.

Der Knoten v_j wird also während der Tiefensuche das erstemal besucht. Sei $P' := (v_i, u_1, u_2, \ldots, u_q, v_j)$ die entsprechende Teilfolge in P. In P' existieren genau zwei nicht–triviale Knoten, nämlich v_i und v_j, und die Knoten $u_1, \ldots, u_q$ sind allesamt trivial. Andernfalls ist v_j nicht Nachfolger von v_i in Q. Möglicherweise werden in dem durch P' induzierten Kantenzug einige Kanten entgegen ihrer Orientierung benutzt. Es kann jedoch keine blaue Kante sein, da der Kopfknoten einer solchen Kante mit Bemerkung 6.8 nicht–trivial sein muß. Falls eine rote Kante in H rückwärts benutzt wird, existiert eine anti–parallele Kante mit gleichen Kosten. Wir können also einen gerichteten Kantenzug von v_i nach v_j in H bestimmen, der die gleichen Kosten wie der durch P' induzierte besitzt. Wir werden die Kosten dieses Kantenzuges mit $c(P')$ bezeichnen. Die Kanten in H' entsprechen kürzesten Pfaden in H. Somit existiert eine Kante $(v_i, v_j) \in E'$ mit $c(v_i, v_j) \leq c(P')$. Diese fügen wir zu T hinzu.

Zu jedem Knoten, der das erstemal bei der Tiefensuche durchlaufen wird, haben wir einen Pfad von seinem Vorgänger v_i konstruiert. Jeder

nicht–triviale Knoten wird in P durchlaufen, insbesondere auch das erstemal. Per Induktion existiert dann in T ein gerichteter Pfad von der Wurzel v_0 zu jedem anderen Knoten $v \in V \setminus V'$. T enthält somit eine v_0–Arboreszenz für H' und die Summe der in T enthaltenen Kantenkosten ist $c(T) \leq 2c(S)$. □

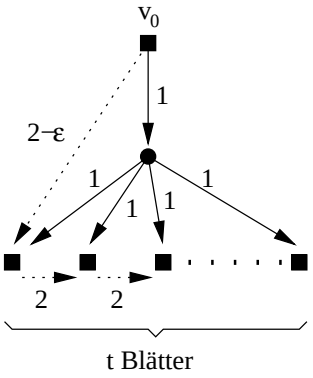


Abbildung 6.3.: *Beispiel zum Beweis zu Satz 6.9*

In der nebenstehenden Abbildung 6.3 ist die Konstruktion einer v_0–Arboreszenz T aus einer Steinerarboreszenz S verdeutlicht. Die Abbildung zeigt den Hilfsgraphen $H = (V, E^r \dot{\cup} E^b)$, dessen nicht–trivialen Knoten $v \in V \setminus V'$ als Quadrate und die trivialen Knoten $v \in V'$ als Kreise dargestellt sind. Die v_0–Steinerarboreszenz S ist durch durchgezogene Kanten gekennzeichnet. Die gestrichelten Kanten sind zusätzliche in H existierende Kanten, die jedoch nicht in S ausgewählt wurden, weil sie zu einer teureren Steinerarboreszenz führen würden. Alle von S ausgewählten Kanten besitzen Gewicht 1, während Kanten von v_0 zu den t Blättern $2 - \varepsilon$, für einen kleinen Wert $\varepsilon > 0$, kosten. Kanten zwischen Blättern haben Gewicht 2. In dieser Illustration können wir davon ausgehen, daß alle Kanten rot gefärbt sind. Das Beispiel beweist zudem, daß die Abschätzung scharf ist. Der Wert der Steinerarboreszenz beträgt $c(S) = 1 + t$ und die Kosten der Arboreszenz betragen $c(T) = 2t - \varepsilon$. Wenn wir die Anzahl t der Blätter gegen unendlich laufen lassen, geht die Approximationsgüte

$$\frac{c(T)}{c(S)} = \frac{2t - \varepsilon}{1 + t} \underset{t \to \infty}{=} 2$$

gegen den Wert 2.

Der Algorithmus 9 `Arboreszenz-Heuristik` faßt die bisherigen Überlegungen zusammen.

Algorithmus 9 Arboreszenz–Heuristik(I)

Konstruiere die Hilfsgraphen H und H′ aus I.
Bestimme eine kostenminimale v_0–Arboreszenz T auf H′. ▷ Satz 2.6
Konstruiere aus T eine Steinerarboreszenz S für H. ▷ Satz 6.9
Konstruiere einen Transportgraphen G_T aus S. ▷ Satz 6.7

Satz 6.10. *Sei* I *eine Eingabe für das exogene semi–präemptive Transportproblem auf einem Baum und* G_T^* *ein optimaler Transportgraph für* I. *Dann liefert Algorithmus 9 einen Transportgraphen* G_T *mit Kosten* $l(G_T) \leq \frac{4}{3} l(G_T^*)$.

Beweis. Der gradbalancierte Baum $\mathtt{bal}$, der durch Anwendung der Vorschrift 6.1 sowie der Bemerkung 5.17 entstanden ist, enthält für jede Baumkante mindestens zwei anti–parallele Kanten, entweder Auftragskanten $e \in A_{\mathcal{R}}$ oder augmentierende Kanten $e \in A_0$, somit gilt $l(\mathtt{bal}) \geq l(A_G)$.

Eine rote Kante einer kostenminimalen v_0–Arboreszenz T^* für H′ entspricht nach Satz 6.7 dem Einfügen zweier anti–parallelen Kanten zwischen zwei Zusammenhangskomponenten aus $\mathtt{bal}$. Um die Zusammenhangskomponenten miteinander zu verbinden und eine Tour zu konstruieren, reicht es, über jede Kante zwischen den Zusammenhangskomponenten zwei anti–parallele Kanten einzufügen. Die Kanten einer kostenminimalen v_0–Arboreszenz T^* für H′ entsprechen also im schlimmsten Fall zwei anti–parallelen Kanten des zugrunde liegenden Baumes, d.h. $2l(A_G) \geq c(T^*)$. Zusammenfassend gilt also:

$$l(\mathtt{bal}) \geq 2l(A_G) \geq c(T^*) \tag{6.2}$$

Jetzt schätzen wir die Kosten einer Lösung G_T, die wir aus der v_0–Arboreszenz T^* auf H′ gewinnen können, im Verhältnis zu den Kosten

einer Optimallösung G_T^* ab:

$$\begin{aligned}\frac{l(G_T)}{l(G_T^*)} &\underset{\text{Satz 6.7}}{=} \frac{l(bal)+c(T^*)}{l(bal)+c(S)} = 1+\frac{c(T^*)-c(S)}{l(bal)+c(S)} \underset{6.2}{\leq} 1+\frac{c(T^*)-c(S)}{c(T^*)+c(S)} \\ &\underset{\text{Satz 6.9}}{\leq} 1+\frac{c(T^*)-\frac{1}{2}c(T^*)}{c(T^*)+\frac{1}{2}c(T^*)} \\ &= \frac{4}{3}\end{aligned}$$

□

6.3.4. Endogenes Transportieren auf Bäumen

Die Ergebnisse der exogenen Variante lassen sich auf die endogene Version übertragen. Wir werden in diesem Abschnitt auf die Unterschiede hinweisen, die sich in der endogenen Betrachtungsweise ergeben.

Aus einer endogenen Eingabeinstanz können wir, genauso wie wir es im exogenen Fall getan haben, mit Hilfe der Anweisung 6.1 einen gradbalancierten Graphen bal herstellen und uns auf starke Zusammenhangskomponenten beziehen. Aus bal konstruieren wir schließlich den gefärbten Hilfsgraphen H wie er in Abschnitt 6.3.1 definiert ist.

Wie im exogenen Fall interessieren wir uns für einen kostenminimalen Teilgraphen, der alle nicht–trivialen Knoten aus H aufspannt. Der Unterschied zu vorher liegt nun darin, daß die Anzahl der Umladevorgängen durch eine Zahl $k \leq n$ beschränkt ist. Es genügt somit nicht, nur eine r–Steinerarboreszenz für H zu konstruieren, da diese nicht die Anzahl der Umladevorgänge beschränkt.

Eine intuitive Forderung wäre also, daß die gesuchte Steinerarboreszenz maximal k blaue Kanten benutzen darf. Dies wird auch das Vorgehen in diesem Abschnitt sein. Eine solche gefärbte Steinerarboreszenz werden wir — wiederum durch Ausnutzen der speziellen Struktur unseres Hilfsgraphen H — durch eine (k, v_0)–Arboreszenz approximieren.

Problem 6.11 ((d,r)–Steinerarboreszenz)

Gegeben: Ein gerichteter Graph $G = (V, E = (E^r \dot\cup E^b))$, dessen Kantenmenge E in die Mengen $E^r \dot\cup E^b$ partitioniert ist, eine Kostenfunktion $c : E \to \mathbb{R}_+$, eine Menge von *Terminalen* $R \subseteq V$, eine Wurzel $r \in V$ sowie eine Zahl $d \in \mathbb{N}$.

Gesucht: Ein Teilmenge $S \subseteq E$, so daß für alle Terminale $t \in R$ ein gerichteter (r,t)–Pfad existiert, $|S \cap E^b| \leq d$ gilt, und der $\sum_{e \in S} c(e)$ minimiert.

Mit $d = |E^b|$ enthält das (d,r)–Steinerarboreszenz–Problem das r–Arboreszenz–Problem, denn somit können alle blauen Kanten frei gewählt werden. Dadurch ist das gefärbte Arboreszenz–Problem nicht nur $\mathcal{NP}$–vollständig, sondern mindestens ebenso schwierig zu approximieren.

Falls wir jedoch eine (d,r)–Steinerarboreszenz für den Hilfsgraphen H zur Verfügung hätten, könnten wir sie nutzen, um einen Transportgraphen für das endogene Transportproblem auf Bäumen zu konstruieren.

Satz 6.12. *Sei* I *die Eingabe für das endogene Transportproblem auf einem Baum. Die Kosten eines optimalen Transportgraphen für* I *betragen* $l(bal)+c(S)$*, wobei* bal *der gradbalancierte Graph zu* I *und* S *eine kostenminimale* (k,v_0)*–Steinerarboreszenz auf dem Hilfsgraphen* H *sind.*

Beweis. Der Beweis wird analog zum Beweis des Satzes 6.7 geführt. Der einzige Unterschied besteht darin, daß wir sicherstellen müssen, daß nicht mehr als k Umladevorgänge stattfinden. Dies wird durch die (k,v_0)–Steinerarboreszenz geleistet, da diese höchstens k blaue Kanten enthält. □

Jede Approximation des (d,r)–Steinerarboreszenz–Problems reicht uns also, um das endogene Transportproblem auf Bäumen zu approximieren. Im exogenen Fall haben wir eine r–Arboreszenz dazu genutzt, um die eine r–Steinerarboreszenz auf dem Hilfsgraphen näherungsweise zu lösen. Dazu haben wir die besondere Struktur, die sich aus dem Transportproblem für H ergibt, ausgenutzt. Die Bemerkung 6.8 gilt genauso für H im endogenen Fall, da sich die Hilfsgraphen entsprechen.

Aus H werden wir erneut einen pfadreduzierten Hilfsgraphen $H'' = (V \setminus V', E'')$ ableiten, dessen Kantenmenge jedoch nur eine Teilmenge der Kanten aus $H' = (V', E')$ ist: Die Knotenmenge von H'' besteht wie vorher aus den nicht–trivialen Knoten, und eine Kante aus H'' entspricht einem kürzesten Weg in H zwischen zwei nicht–trivialen Knoten, der maximal eine blaue Kante benutzt. Eine Kante in H'' ist rot, falls alle Kanten des kürzesten Weges rot sind. Sie ist blau, falls dieser *genau eine* blaue Kante enthält.

Bemerkung 6.13. *Ein kürzester Weg von $u \in V$ nach $v \in V$, der maximal eine blaue Kante enthält, kann in polynomieller Zeit bestimmt werden, indem wir folgendes Verfahren anwenden:*

Betrachte den roten Teilgraphen $H^r = (V, E^r)$ und füge diesem eine blaue Kante $e \in E^b$ zu H^r hinzu. Für eine Kante $e \in E^b$ nennen wir den so entstandenen Graphen $H^e = H^r + e$. Ein kürzester (u, v)–Weg in H^e ist ein kürzester (u, v)–Weg in H, der nur rote Kanten und eventuell e benutzt. Für jede blaue Kante $e \in E^b$ wird H^e erzeugt und der kürzeste (u, v)–Weg bestimmt. Unter all diesen wählen wir denjenigen mit den billigsten Kosten aus. Dieser ist ein (u, v)–Weg mit höchstens einer blauen Kante und billigst unter solchen.

Wir wollen nun eine gefärbte Arboreszenz in H'' nutzen, um eine gefärbte Steinerarboreszenz für H zu approximieren.

Satz 6.14. *Sei T eine kostenminimale (k, v_0)–Arboreszenz in H'' und S eine kostenminimale (k, v_0)–Steinerarboreszenz in H mit $v_0 \in V \setminus V'$. Es gilt:*

$$c(T) \leq 2c(S)$$

Beweis. Analog zum Beweis zu Satz 6.9 müssen wir die Existenz einer (k, v_0)–Arboreszenz in H'' mit Kosten höchstens $2c(S)$ beweisen.

Wir werden den genannten Beweis Schritt für Schritt nachvollziehen und jeweils auf die Besonderheiten der aktuellen Situation eingehen. Wir durchmustern erneut S mittels Tiefensuche. Sei also $P = (v_0, \ldots, v_p)$ der dadurch beschriebene Weg. Wie vorher argumentieren wir, daß in P alle Kanten aus S genau zweimal auftreten, einmal in ihrer Orientierung und

einmal gegen ihre Orientierung. Wir haben festgestellt, daß die Kosten des durch P beschriebenen Kantenzuges genau $2c(S)$ betragen.

Sei dann $Q = (v_0, \ldots, v_q)$ die Teilfolge von P, die nur die nicht–trivialen Knoten aus P enthält. Markiere zur Induktionsverankerung v_0. Wir werden nun nacheinander die Knoten aus Q markieren und Kanten zu einem Teilgraphen T von H'' hinzufügen, wobei eine Marke bedeutet, daß der betroffene Knoten von v_0 aus über einen gerichteten Pfad in T erreichbar ist. Nachdem alle nicht–trivialen Knoten markiert sind, werden diese von T aufgespannt. Somit wird T die gesuchte (k, v_0)–Arboreszenz bilden.

Sei dann v_j der erste unmarkierte Knoten in Q und v_i der markierte Vorgänger von v_j. Beide Knoten sind nicht–trivial. Außerdem ist $P^{i,j} = (v_i, u_1, \ldots, u_p, v_j)$ die Teilfolge in P, die dem gewählten Knotenpaar v_i, v_j entspricht. Wir wissen, daß alle $u_i, i = 1, \ldots, p$ triviale Knoten sind, andernfalls wäre v_j nicht der direkte Nachfolger von v_i in Q gewesen. Eine solche Teilfolge $P^{i,j}$ beschreibt einen Kantenzug in S, wobei einige Kanten entgegen ihrer Orientierung benutzt werden. Mit $c(P^{i,j})$ ist die Summe der Kantenkosten gemeint, die in $P^{i,j}$ benutzt werden.

Wegen Bemerkung 6.8 müssen alle blauen Kanten, die in $P^{i,j}$ (entweder in ihrer oder gegen ihre Orientierung) benutzt werden, einen nicht–trivialen Knoten als Schwanz besitzen. Dadurch kommen nur zwei Kanten in Frage, die blau gefärbt sein könnten: (v_i, u_1) und (v_j, u_p).

Wir wollen nun zeigen,

(i) daß $P^{i,j}$ ein gerichteter (v_i, v_j)–Pfad in S und somit auch in H ist,

(ii) daß $P^{i,j}$ maximal eine blaue Kante induziert.

Dann existiert in H'' eine Kante (v_i, v_j), die höchstens so teuer wie $P^{i,j}$ ist. Zu (i): Da v_i markiert ist, existiert in T ein gerichteter (v_0, v_i)–Pfad. Kanten aus T entsprechen aber per Induktionsvoraussetzung gerichteten Pfaden in S. Also existiert in S ebenfalls ein gerichteter (v_0, v_i)–Pfad, der zusätzlich triviale Knoten benutzt. Entweder ist v_i Wurzel oder es existiert eine Kante mit Kopf v_i, die jedoch nicht durch $P^{i,j}$ induziert wird, weil v_i schon markiert ist. In beiden Fällen muß $P^{i,j}$ die Kante (v_i, u_1) in ihrer Orientierung durchlaufen, oder ansonsten enthält S zwei

Kanten mit Kopf v_i im Widerspruch zur Arboreszenz–Eigenschaft. Dann müssen in der Tiefensuche auch die nachfolgenden Kanten entsprechend ihrer Orientierung benutzt werden. Die Orientierung kann sich erst umdrehen, nachdem ein Blatt erreicht wurde. Blätter können aber nur nicht–triviale Knoten sein, und der erste nicht–triviale Knoten in $P^{i,j}$ ist v_j. Somit beschreibt $P^{i,j}$ einen gerichteten (v_i, v_j)–Pfad.

Insbesondere wird auch (u_p, v_j) in ihrer Orientierung durchlaufen. Wir haben in Bemerkung 6.8 festgestellt, daß $(u_p, v_j) \in E^b$ nur dann gilt, falls u_p ein nicht–trivialer Knoten ist, u_p ist jedoch trivial. Somit ist auch (ii) bewiesen.

Also existiert in H'' eine Kante (v_i, v_j) mit Kosten $c(v_i, v_j) \leq c(P^{i,j})$. Diese fügen wir zu T hinzu. Da in T ein gerichteter (v_0, v_i)–Pfad existierte, existiert nun auch ein gerichteter (v_0, v_j)–Pfad über v_i. Aus diesem Grund markieren wir ebenfalls v_j.

Auf die beschriebene Weise lassen sich alle Knoten aus $V \setminus V'$ markieren, da sie von S aufgespannt werden. Somit existiert aber auch in T für jeden Knoten ein gerichteter Pfad von der Wurzel. Die Kosten einer Kante (v_i, v_j) in T sind höchstens so teuer wie der durch die Teilfolge $P^{i,j}$ beschriebene Pfad. Die betrachteten Teilfolgen $P^{i,j}$ sind disjunkt und wir betrachten keine Teilfolge zweimal. Somit enthält H'' eine Arboreszenz T mit höchstens k blauen Kanten, deren Kosten höchstens

$$c(T) = \sum_{(v_i,v_j)\in T} c(v_i, v_j) \leq \sum_{(v_i,v_j)\in T} c(P^{i,j}) \leq c(P) \leq 2c(S)$$

betragen. □

Die Abschätzung in Satz 6.14 ist scharf. Der Graph in Abbildung 6.3 dient auch hier als Beispielinstanz.

Leider haben wir keinen Algorithmus zur Hand, der das (d, r)–Arboreszenz–Problem optimal löst. In Abschnitt 3.3.3 haben wir allerdings ein voll polynomielles Approximationsschema angegeben, das eine Lösung für das Problem findet, deren Kostenwert bis auf einen Faktor $(1 + \varepsilon)$ von dem einer Optimallösung entfernt liegt. Dies nutzen wir in Algorithmus 10 `(d,r)-Arboreszenz-Heuristik`(I) aus.

Algorithmus 10 (d,r)–Arboreszenz–Heuristik(I)

Konstruiere die Hilfsgraphen H und H'' aus I.
Bestimme approximativ eine (k, v_0)–Arboreszenz T' auf H'. ▷ Korollar 3.21
Konstruiere aus T' eine Steinerarboreszenz S für H. ▷ Satz 6.14
Konstruiere einen Transportgraphen G_T aus S. ▷ Satz 6.12

Satz 6.15. *Sei* I *eine Eingabe für das endogene semi–präemptive Transportproblem auf einem Baum und* G_T^* *ein optimaler Transportgraph für* I. *Dann liefert Algorithmus 10 einen Transportgraphen* G_T *mit Kosten* $l(G_T) \leq l(G_T^*)(\frac{4}{3} + \varepsilon)$ *für beliebiges* $\varepsilon > 0$.

Beweis. Sei A_G die Kantenmenge des zugrunde liegenden Baumes, bal der gradbalancierte Hilfsgraph zu I und T^* eine kostenminimale (k, v_0)–Arboreszenz für H''. Aus dem Beweis zu Satz 6.10 wissen wir, daß

$$l(bal) \geq 2l(A_G) \geq c(T^*) \tag{6.3}$$

gilt.

Wie im Beweis zu Satz 6.10 werden die Kosten einer Lösung G_T, die aus einer durch den Algorithmus 10 produzierten (k, v_0)–Arboreszenz T auf H'' konstruiert wird, im Verhältnis zu den Kosten einer Optimallösung G_T^* abgeschätzt.

Der Algorithmus aus Korollar 3.21, den wir für eine Näherungslösung einer (k, v_0)–Arboreszenz einsetzen, erwartet einen Genauigkeitsparameter ε'. Diesen wählen wir als:

$$\varepsilon' := \frac{3}{2}\varepsilon$$

Das Verhältnis der Lösungskosten des Algorithmus 10 zur Optimallösung

beträgt dann:

$$\begin{aligned}
\frac{l(G_T)}{l(G_T^*)} &\underset{\text{Satz 6.12}}{=} \frac{l(bal)+c(T)}{l(bal)+c(S)} \\
&\underset{\text{Kor. 3.21}}{\leq} \frac{l(bal)+c(T^*)(1+\varepsilon')}{l(bal)+c(S)} = 1+\frac{c(T^*)(1+\varepsilon')-c(S)}{l(bal)+c(S)} \\
&\underset{6.3}{\leq} 1+\frac{c(T^*)(1+\varepsilon')-c(S)}{c(T^*)+c(S)} \\
&\underset{\text{Satz 6.14}}{\leq} 1+\frac{c(T^*)(1+\varepsilon')-\frac{1}{2}c(T^*)}{c(T^*)+\frac{1}{2}c(T^*)} = \frac{4}{3}+\frac{\varepsilon' c(T^*)}{\frac{3}{2}c(T^*)} \\
&= \frac{4}{3}+\varepsilon
\end{aligned}$$

□

Die endogene Variante läßt sich auf Bäumen fast genauso gut (bis auf einen Genauigkeitsfaktor $\varepsilon > 0$) wie die exogene Version näherungsweise mit einer ähnlichen Strategie lösen. Anstelle einer 4/3–Approximation können wir nur eine $(4/3+\varepsilon)$–Approximation mit einem $\varepsilon > 0$ angeben. Das ε dieser Abschätzung stammt aus Korollar 3.21, in dem wir ein FPTAS für das (d, r)–Arboreszenz–Problem angegeben haben: Wenn wir für jenes Problem einen exakten Algorithmus mit polynomieller Laufzeit angeben könnten, würde sich die Approximationsschranke dieser Abschätzung minimal auf 4/3 verbessern.

6.4. Anwendung auf allgemeine Metriken

Wir konnten in den vorigen Abschnitten dieses Kapitels Approximationsalgorithmen für das exogene als auch für das endogene Transportproblem auf Bäumen angeben. Wir wollen nun untersuchen, inwieweit wir Lösungen für beide Versionen des Transportproblems generieren können, wenn wir über den zugrunde liegenden Graphen nur wissen, daß die darauf definierte Distanzfunktion eine Metrik ist. Da wir ungerichtete Graphen G

betrachten, beschränken wir uns nur insofern, daß wir für eine nichtnegative Gewichtsfunktion auf den Kanten von G die Dreiecksungleichung fordern. Unter Zuhilfenahme bestehender Ergebnisse werden wir eine Metrik auf allgemeinen Graphen durch Metriken auf Bäumen abschätzen.

Für zwei Knoten $s, t \in \mathcal{S}$ des Graphen $G = (\mathcal{S}, E)$ mit $(s, t) \in E$ sei eine Metrik $l_G(s, t) \in \mathbb{R}$ gegeben. Für einen Baum $T = (\mathcal{S}, E')$ auf derselben Knotenmenge sei $l_T(s, t) \in \mathcal{R}$ die Entfernung von s nach t über den eindeutigen (s, t)–Pfad in T. Die Metrik l_T soll l_G nicht unterschätzen. Wir sagen auch, daß l_T die Metrik l_G *dominiert*, wenn $l_T(s, t) \geq l_G(s, t)$ für alle $s, t \in \mathcal{S}$.

Definition 6.16 (α–probabilistische Approximation). *Eine Menge* M *von Metriken* l_T *ist eine* α–probabilistische Approximation *für die Metrik* l_G*, falls es eine Wahrscheinlichkeitsverteilung* μ *über* M *gibt, so daß für alle Knotenpaare* s, t*, für die* l_G *definiert ist, die erwartete Entfernung von* s *nach* t *bezüglich* μ *höchstens um den Faktor* α *länger ist als* $l_G(s, t)$*, d.h.*

$$E[l_\mu(s, t)] \leq \alpha l_G(s, t) \text{ für alle } s, t \in \mathcal{S}$$

Bartal liefert uns in [Bar96] und [Bar98] ein nützliches Werkzeug: Jeder metrisch gewichtete und zusammenhängende Graph G mit n Knoten kann durch eine Menge von gewichteten Bäumen α–probabilistisch approximiert werden, wobei $\alpha = \mathcal{O}(\log n \log\log n)$. Zudem kann die probabilistische Verteilung mit polynomiellem Zeitaufwand berechnet werden. Randomisierte Algorithmen liefern uns keine deterministische Aussage über Approximationsgüten. Das folgende Resultat liefert uns eine Derandomisierung des Ergebnisses von Bartal.

Satz 6.17 (Charikar et al. 1998 [CCG$^+$98]). *Jede endliche Metrik auf* n *Punkten kann* $\mathcal{O}(\log n \log\log n)$*–probabilistisch durch eine Wahrscheinlichkeitsverteilung auf* $\mathcal{O}(n \log n)$ *Baum–Metriken approximiert werden. Die Bäume und die Verteilung können in polynomieller Zeit berechnet werden.*

Da wir die exogene und endogene Problemstellung auf Bäumen mit einer Güte $\beta = 4/3$ bzw. $\beta = 4/3 + \varepsilon$ approximieren können, wenden wir die Erkenntnisse von Charikar et al. darauf an.

Korollar 6.18. *Wenn für das exogene bzw. endogene PDP auf Bäumen eine β–Approximation existiert, dann gibt es eine $\mathcal{O}(\beta \log n \log\log n)$–Approximation auf metrisch gewichteten Graphen G mit n Knoten.*

Beweis. Sei μ die Wahrscheinlichkeitsverteilung auf der Menge der Baum–Metriken M, die wir aus Satz 6.17 mit polynomiellem Zeitaufwand erhalten und l_G die Distanzfunktion auf G.

Für μ und zwei Knoten s, t aus G gilt mit $\alpha = \mathcal{O}(\log n \log\log n)$, daß $E[l_\mu(s,t)] \leq \alpha l_G(s,t)$. Sei A^* eine Menge von Kanten auf G, die einen optimalen Transport für das exogene bzw. endogene Transportproblem auf G beschreibt. Sei außerdem T ein beliebiger Baum aus M. Jeder Kante $(s,t) \in A^*$ entspricht ein eindeutiger Pfad von s nach t in T mit Distanz $l_t(s,t)$. Wir konstruieren für T einen Transportgraphen, der durch die Kantenmenge A^T beschrieben werden soll. Für jede Kante aus A^* fügen wir einen Pfad in A^T ein. Die Menge der Umladeknoten, die in A^* benutzt wird, bleibt in A^T erhalten. Die Kosten einer solchen Lösung betragen im Erwartungswert:

$$\sum_{(s,t)\in A^*} E[l_\mu(s,t)] \leq \alpha \sum_{(s,t)\in A^*} l_G(s,t)$$

Zu jedem Baum T aus M betrachten wir die Lösung A^T. Dann muß es mindestens einen Baum T^* geben, für den

$$l_{T^*}(A^{T^*}) = \sum_{(s,t)\in A^*} l_{T^*}(s,t) \leq \alpha \sum_{(s,t)\in A^*} l_G(s,t) = \alpha l_G(A^*) \qquad (6.4)$$

gilt, denn andernfalls kann der Erwartungswert nicht angenommen werden.

Wir lösen das exogene bzw. endogene PDP auf G wie folgt: Für jeden Baum T aus M konstruieren wir eine β–approximative Lösung und merken uns die beste unter ihnen. Sei A die β–approximative Lösung für T^*, und A^{T^*} eine optimale Lösung für das Problem auf T^*. Dann gilt:

$$l_{T^*}(A) \leq \beta l_{T^*}(A^{T^*}) \underset{6.4}{\leq} \alpha\beta l_G(A^*)$$

Weil die Kardinalität von M durch $\mathcal{O}(n \log n)$ begrenzt ist, benötigen wir nur polynomiellen Zeitaufwand, um alle Bäume aus M zu betrachten. □

Bemerkung 6.19. *Charikar et al. können für Metriken auf planaren Graphen sogar eine schärfere Abschätzung angeben. Für solche Graphen ist eine Approximation mit* $\alpha = \mathcal{O}(\log n)$ *möglich.*

Teil II.

Implementierung

Kapitel 7.

Details

Der praktische Teil dieser Arbeit befaßt sich mit der konkreten Implementierung der im theoretischen Teil entwickelten Algorithmen. Hinsichtlich einer pessimistischen Abschätzung, die den schlimmsten Fall betrachtet, besitzt der Algorithmus für die exogene Problemstellung eine bessere Laufzeit als die endogene Variante. Die Implementation befaßt sich ausschließlich mit der zweiten und in einer solchen Analyse schwierigeren Version.

In den folgenden Abschnitten werden zunächst Details der Umsetzung des dynamischen Programms aus Kapitel 5 dargelegt. Außerdem stellen wir in Abschnitt 7.2 eine gemischt–ganzzahlige Modellierung vor.

7.1. Dynamisches Programm

Der Algorithmus 8 (`k-Robot`) und die Unterroutinen 6 (`k-RobotLight`) sowie 7 (`Fix`) wurden wie in Kapitel 5 beschrieben in der Programmiersprache C++ implementiert und mit dem freien Compiler *g++ (GCC)* der Version 3.3.5 übersetzt.

Die genannten Algorithmen benötigen graphentheoretische Datenstrukturen zur Verwaltung des Kreisgraphen sowie der daraus abgeleiteten Hilfsgraphen. Neben diesen Datenstrukturen werden Algorithmen gebraucht, um auf Elemente zuzugreifen oder diese zu modifizieren. Diese

Details bieten bei einer praktischen Implementierung erhebliches Potential zur Effizienzsteigerung. Unser Augenmerk lag jedoch auf einer abstrakteren Ebene. Die Umsetzung der Datenstrukturen in eine Programmiersprache haben wir deswegen der frei verfügbaren Software–Bibliothek *Boost Graph Library* [SLL02] in der Version 1.32 überlassen.

In `k-RobotLight` wird auf einem ungerichteten Hilfsgraphen ein kostenminimaler d–Baum berechnet. Für die Berechnung ist ein effizientes Verfahren von Gabow und Tarjan bekannt (s. Satz 3.4), das jedoch eine sehr aufwendige Datenstruktur benutzt. Der von uns entwickelte Algorithmus 2 berechnet ebenfalls einen optimalen d–Baum, aber besitzt bezüglich der $\mathcal{O}$–Notation eine schlechtere Laufzeitschranke. Andererseits ist er wesentlicher einfacher zu implementieren. Aus diesem Grund haben wir uns für ihn und gegen das Verfahren von Gabow und Tarjan entschieden.

7.2. Gemischt–ganzzahlige Modellierung

Kombinatorische Optimierungsprobleme lassen sich häufig als gemischt–ganzzahlige Programme formulieren. Im Gegensatz zu Linearen Programmen lassen sie sich (unter der Voraussetzung, daß $\mathcal{P} \neq \mathcal{NP}$ gilt) im allgemeinen nicht effizient lösen (siehe dazu Abschnitt 2.3).

MIP 1 auf Seite 127 ist eine Modellierung des endogenen semi–präemptiven PDP für den Fall, daß der zugrunde liegende Graph ein Kreis ist. Im folgenden werden zunächst die verwendeten Variablen, die Zielfunktion und schließlich die Nebenbedingungen beschrieben.

Wie wir in den vorigen Abschnitten gesehen haben, muß für jeden Auftrag entschieden werden, in welcher Richtung er um den Kreisumfang transportiert wird. Dementsprechend führen wir eine $\{0,1\}$–Variable x_r ein, wobei

$$x_r = \left\{ \begin{array}{ll} 0 & \text{falls Auftrag } r \textit{ gegen den Uhrzeigensinn} \\ 1 & \text{falls Auftrag } r \textit{ im Uhrzeigensinn} \end{array} \right\}$$

transportiert wird.

MIP 1 Eine Formulierung für das endogene Transportproblem auf dem Kreis

$$\min \sum_{r\in\mathcal{R}} l_r^+ x_r + l_r^-(1-x_r) \; + \sum_{i=0}^{n-1} l_i(y_i^+ + y_i^-) + \Delta \sum_{i=0}^{n-1} b_i$$

unter Berücksichtigung der Nebenbedingungen

$$\sum_{i=0}^{n-1} b_i \le k \tag{7.1}$$

$$\sum_{r\in\mathcal{R}} \lambda_r^+(i)x_r - \lambda_r^-(i)(1-x_r) \; + y_i^+ - y_i^- = \psi \qquad i = 0,\dots,n-1 \tag{7.2}$$

$$\begin{aligned} |\{(s_o,s_i)\in\mathcal{R}\}| \\ +\textstyle\sum_{r\in\mathcal{R}} \lambda_r^+(i)x_r + \lambda_r^-(i)(1-x_r) \end{aligned} \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ s_o &\in \mathcal{S}, |i-o|\neq 1 \end{aligned} \tag{7.3}$$

$$\begin{aligned} |\{(s_o,s_i)\in\mathcal{R}\}| \\ +\textstyle\sum_{r\in\mathcal{R}} \lambda_r^+(i)x_r + \lambda_r^-(i)(1-x_r) \; + y_o^+ \end{aligned} \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ o &= i-1 \end{aligned} \tag{7.4}$$

$$\begin{aligned} |\{(s_o,s_i)\in\mathcal{R}\}| \\ +\textstyle\sum_{r\in\mathcal{R}} \lambda_r^+(i)x_r + \lambda_r^-(i)(1-x_r) \; + y_i^- \end{aligned} \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ o &= i+1 \bmod n \end{aligned} \tag{7.5}$$

$$|\{(s_o,s_i)\in\mathcal{R}\}| + b_i \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ s_o &\in \mathcal{S}, |i-o|\neq 1 \end{aligned} \tag{7.6}$$

$$|\{(s_o,s_i)\in\mathcal{R}\}| + b_i + y_o^+ \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ o &= i-1 \end{aligned} \tag{7.7}$$

$$|\{(s_o,s_i)\in\mathcal{R}\}| + b_i + y_i^- \ge f_{oi}^j \qquad \begin{aligned} s_i,s_j &\in \mathcal{S}\setminus\{s_0\} \\ o &= i+1 \bmod n \end{aligned} \tag{7.8}$$

$$\sum_{\substack{d\neq j,\\ o\neq d}} f_{od}^j - \sum_{\substack{d\neq j,\\ o\neq d}} f_{do}^j = 0 \qquad s_o,s_j \in \mathcal{S}\setminus\{s_0\} \tag{7.9}$$

$$\sum_{i=1}^{n-1} f_{0,i}^j \ge 1 \qquad s_j \in \mathcal{S}\setminus\{s_0\} \tag{7.10}$$

Restriktionen der Variablen:

$$\begin{aligned} x_r &\in \{0,1\} & r &\in \mathcal{R} \\ b_i &\in \{0,1\} & s_i &\in \mathcal{S} \\ y_i^+, y_i^-, \psi &\in \{-m-1,\dots,m+1\} & s_i &\in \mathcal{S} \\ f_{od}^j &\in [0,1] & s_j &\in \mathcal{S}\setminus\{s_0\}, (o,d)\in\mathcal{R} \end{aligned}$$

Die Variable $b_i \in \{0,1\}, i = 0,\ldots,n-1$ legt binär fest, an welchen Knoten des Kreises ein Umladevorgang stattfindet und somit ein Auftrag unterbrochen wird. Zwischen benachbarten Knoten auf dem Kreis können Leertransporte stattfinden, die durch die Variable y_i gezählt werden. Der Wert darf ganzzahlige Werte annehmen. Da wir den Fluß ψ über alle Intervalle entsprechend Korollar 5.7 beschränken, legen wir den Wertebereich für die Variablen ψ als auch für die y_i auf das ganzzahlige Intervall $[-m-1, m+1]$ fest. Dadurch erzwingen wir mit Hinweis auf Lemma 5.2 in allen Knoten Gradbalance. Wir gehen davon aus, daß wir es mit Instanzen zu tun haben, in denen alle Knoten Ausgangs– oder Endpunkt eines Auftrages sind. In einer Lösung müssen alle Knoten über einen Pfad, der entweder Auftragskanten oder Leerfahrten benutzt, vom Startknoten s_0 aus erreichbar sein. Um dies im Modell zu gewährleisten, soll jeweils für alle Knoten $s_j \in \mathcal{S}$ ein positiver Fluß (im Sinne von Netzwerkflüssen zwischen Quelle und Senke, s. [KV02]) vom Startknoten s_0 zu s_j fließen. Die Variable $f^j_{od} \in [0,1]$ zählt den Fluß von s_0 nach s_j über die Kante (o,d) mit $o,d \in \mathcal{S}$, die entweder eine Auftragskante oder Leerfahrt ist. Wegen der Gradbalance liegen alle Kanten auf einem Kreis, und die Eigenschaft, daß alle Knoten von s_0 aus erreichbar sind, stellt den notwendigen Zusammenhang aller Knoten her.

Neben den Variablen benutzen wir zwei binäre Funktionen $\lambda^+_r(i)$ und $\lambda^-_r(i)$, die abhängig von der Drehrichtung eines Auftrages $r \in \mathcal{R}$ Auskunft geben, ob ein Knoten $s_i \in \mathcal{S}$ gekreuzt wird:

$\lambda^+_r(i) = 1 \Leftrightarrow$ Auftrag r im Uhrzeigersinn kreuzt den Knoten s_i

$\lambda^-_r(i) = 1 \Leftrightarrow$ Auftrag r gegen den Uhrzeigersinn kreuzt den Knoten s_i

Die erste Nebenbedingung reglementiert die Anzahl der verwendeten Umladeknoten, die mit $b_i \in \{0,1\}$ pro Knoten $s_i \in \mathcal{S}$ gezählt werden.

Für einen Flußwert $\psi \in \{-m-1,\ldots,m+1\}$ sollen alle Intervalle $i \in \{0,\ldots,n-1\}$ genau diesen Wert annehmen. Die Restriktion 7.2 zählt den Fluß über ein Intervall i gemäß Definition 5.1. Zur Erinnerung verweisen wir auf Lemma 5.2 und bemerken, daß diese Nebenbedingung Gradbalance in jedem Knoten des Graphen herstellt.

Wie vorher beschrieben soll der Zusammenhang des Graphen G_T durch $n-1$ Flüsse vom Startknoten s_0 zu allen anderen Knoten hergestellt werden. Ein Fluß, der von s_0 nach s_j fließt, darf nur Auftragskanten in ihrer Drehrichtung um den Kreis und Leerfahrten zwischen benachbarten Knoten benutzen. Die Variable f^j_{oi} enthält den Wert eines Flusses mit Quelle s_0 und Senke s_j zwischen den Knoten s_o und s_i. Die Nebenbedingung 7.3 stellt sicher, daß nur dann ein Fluß $f^j_{oi}, j \in \mathcal{S} \setminus \{s_0\}$ fließt, wenn es entweder einen Auftrag $(s_o, s_i) \in \mathcal{R}$ gibt oder ein Auftrag den Knoten s_i kreuzt. Im letzten Fall müssen wir noch sicherstellen, daß s_i als Umladeknoten deklariert wurde. Dies erreichen wir durch die Nebenbedingung 7.6. Falls die Knoten s_o und s_i benachbart sind, darf der Fluß von s_o nach s_i auch über Leerkanten verlaufen. Dies wird in den Zeilen 7.4 und 7.7 für $o = i-1$ bzw. in den Zeilen 7.5 und 7.8 für den Fall $o = (i+1) \bmod n$ behandelt.

Flußwert, der in einen Knoten hinein fließt, muß auch wieder heraus fließen. Davon ausgenommen sind nur die Quelle s_0 und die Senken $s_j, j = 1, \ldots, n-1$. Für jeden Fluß f^j verlangt die Zeile 7.9 die Flußerhaltungseigenschaft. Schließlich wird in Nebenbedingung 7.10 ein positiver Flußwert an der Quelle s_0 für jeden Fluß generiert.

Die angegebene Formulierung wurde im Rahmen dieser Arbeit mit Hilfe der Softwarebibliothek *Ilog Cplex Concert Technology* in der Version 7.1 implementiert [Ilo01]. Sie dient allerdings eher der Kontrolle der Ergebnisse des dynamischen Programms `k-Robot`, das wir zuvor beschrieben haben, als einer vergleichenden Laufzeitanalyse. Die Modellierung eines mathematischen Programms für das vorliegende Problem ist ein interessantes Themengebiet, das in dieser Arbeit nur angerissen wird.

Kapitel 8.

Rechenergebnisse

Wir werden sehen, daß reine Zufallsinstanzen meist leicht zu lösen sind. Deswegen wird in Abschnitt 8.1 erläutert, welche Struktur die Testinstanzen aufweisen. Schließlich werden in Abschnitt 8.2 die konkreten Laufzeitergebnisse vorgestellt.

8.1. Generierung von Instanzen

Wir haben drei Sätze von Beispielinstanzen generiert, die von beiden Modellen gelöst werden sollen. Instanzen des ersten Typs sind gleichverteilte Zufallsinstanzen. Die des zweiten und dritten Typs enthalten etwas mehr Struktur, um mehr Zusammenhangskomponenten zu erzeugen.

kreis-n.pdp

Die erste Menge von Instanzen enthält Eingaben für das endogene PDP auf einem Kreisgraphen mit n Knoten und m Aufträgen, wobei die Anfangs– und Endknoten der Aufträge zufällig und unabhängig voneinander ermittelt wurden.

Nach einigen Testläufen — und die Rechenergebnisse, die in Kapitel 8 zusammengestellt wurden, belegen dies — wurde schnell deutlich, daß solche reinen Zufallsinstanzen leicht zu lösen sind. Coja–Oghlan et al. untersuchten Zufallsinstanzen auf Kammgraphen und sie konnten sogar

beweisen, daß solche Instanzen mit sehr hoher Wahrscheinlichkeit einfach zu lösen sind.

Satz 8.1 (Coja–Oghlan et al. 2005 [COKN03]). *Wenn m Aufträge uniform zwischen n Knoten auf einem Kammgraphen verteilt werden und $m \to \infty$, dann enthält* bal *mit Wahrscheinlichkeit $1 - \varepsilon$ für beliebiges $\varepsilon > 0$ eine einzelne Zusammenhangskomponente.*

Wird eine solche zufällige Kreisinstanz nach Vorschrift 5.1 auf Seite 60 und Bemerkung 5.17 gradbalanciert, so enthält der daraus entstehende Graph bal mit einer Wahrscheinlichkeit beliebig nahe an 1 nur eine einzelne Zusammenhangskomponente. Für das dynamische Programm verbleibt dann nichts mehr zu tun, da bal schon ein optimaler Transportgraph ist. Von solchen Zufallsinstanzen erwarten wir demnach kurze Laufzeiten.

Für eine Instanz mit n Knoten, wählen wir $k = \lfloor n/4 \rfloor$. Die Anzahl m der Aufträge soll in etwa der Anzahl der Knoten entsprechen, um möglichst viele Zusammenhangskomponenten zu erzeugen. Das Programm, das die Instanzen erzeugt, löscht Knoten, die weder Start– noch Endpunkt eines Auftrages sind. Deswegen enthalten die Instanzen nur ungefähr so viele Aufträge wie Knoten. Die Distanzwerte benachbarter Knoten liegen im Intervall $[0, 1000]$. Die Umladekosten betragen $\Delta = 100$.

kreis-n-V.pdp und kreis-n-v.pdp

Der zweite Satz mit Testinstanzen wurde nicht völlig zufällig erzeugt. Bei diesen Eingaben sollten nach Durchführen des Gradbalancierungs–Schrittes möglichst viele Zusammenhangskomponenten erhalten bleiben. Dies wird dadurch erzwungen, daß alleine die Aufträge dafür sorgen, daß die Knoten schon gradbalanciert sind. Zu diesem Zweck existiert zu jedem Auftrag $(s_i, s_j) \in \mathcal{R}$ ein gegenläufiger Auftrag $(s_j, s_i) \in \mathcal{R}$, d.h. jeder Knoten ist einmal Startknoten und einmal Endknoten von Aufträgen. Eine Zusammenhangskomponente in bal enthält dadurch genau zwei Knoten.

Sei n eine gerade Zahl. Dann werden die Instanzen wie folgt generiert: Ein Kreis mit n Knoten enthält für alle $i = 0, \ldots, \frac{n}{2} - 1$ genau die Aufträge

(s_i, s_j) und (s_j, s_i), wobei $j = i + \frac{n}{2} \bmod n$. Zwei benachbarte Knoten des Kreises haben eine Distanz aus dem Intervall $[0, 1000]$. Die Umladekosten betragen $\Delta = 50$ Kosteneinheiten.

In einem ersten Lauf wurden die oben beschriebenen Instanzen mit $k = \lfloor n/2 \rfloor$ Umladevorgängen generiert und gerechnet. Wir haben die Zahl k so gewählt, daß die maximal mögliche Anzahl an Umladevorgängen stattfinden kann. Da $\Delta > 0$ ist, werden in der Optimallösung nur $k' \leq k$ viele Umladeknoten benutzt.

Zu jeder solchen Instanz `kreis-n-V.pdp` wurde eine zweite Eingabe `kreis-n-v.pdp` erzeugt, in der die Anzahl der Umladeknoten auf $\lfloor k'/2 \rfloor$ beschränkt wird. Alle anderen Parameter blieben unverändert.

kreis-n-W.pdp und kreis-n-w.pdp

Auch die Menge der dritten Testinstanzen weist eine Struktur auf, wobei diesmal zusätzlich darauf geachtet wurde, daß die Eingabe die Eigenschaft P1 (s. Definition auf Seite 81) nicht erfüllt. Die Laufzeit des dynamischen Programms `k-Robot` sollte sich dadurch erwartungsgemäß verschlechtern, da die „gutartigen“ Intervalle um den Startknoten herum enger werden und der Algorithmus dadurch mehr Iterationsschritte benötigt.

Sei n eine gerade Zahl. Dann erzeugen wir die Instanzen folgendermaßen: Ein Kreis mit n Knoten enthält für alle $i = 0, \ldots, \frac{n}{2} - 1$ genau die Aufträge $(s_{(n-i) \bmod n}, s_{i+1})$ und $(s_{i+1}, s_{(n-i) \bmod n})$. Die Distanzen zwischen benachbarten Knoten liegen im Intervall $[0, 1000]$. Wir wählen die Maximalanzahl der Umladeknoten $k = n/2$. Bei n Knoten existieren bei diesem Aufbau der Eingabeinstanz und Flußwert $\psi = 0$ genau $n/2$ Zusammenhangskomponenten in bal. Es können also maximal viele Umladeknoten benutzt werden. Die Kosten pro Umladevorgang sind mit $\Delta = 50$ gewählt.

Nachdem die Instanzen `kreis-n-W.pdp` alle gelöst wurden, wurde eine weitere Menge von Instanzen `kreis-n-w.pdp` erzeugt, in denen nur der Parameter k geändert wurde. Wenn in der berechneten Optimallösung zu `kreis-n-W.pdp` k' viele Umladeknoten vorhanden sind, wurde die neue Eingabe `kreis-n-w.pdp` mit $k = \lfloor k'/2 \rfloor$ generiert.

Alle Testinstanzen wurden mit Hilfe von Programmen in der Interpretersprache *Perl* [WCS97], Version 5.8.4, erzeugt.

8.2. Laufzeitergebnisse

Alle Testläufe wurden auf `fireball.mi.uni-koeln.de`, ein System mit Intel Pentium 4–Prozessor mit 3,06 GHz, gerechnet. Insgesamt standen 2 Gigabyte Hauptspeicher zur Verfügung, wobei das dynamische Programm auch bei einer Instanz mit 100 Knoten mit weniger als 32 Megabyte Hauptspeicher auskam. Die mathematische Formulierung konnte ab einer Größe von 90 Knoten aufgrund der Hauptspeicher–Restriktionen nicht mehr gerechnet werden. Als Betriebssystem wurde *Linux* in der Kernelversion 2.6.8 der Distribution *Debian GNU/Linux „Sarge“* zugrundegelegt. Alle Laufzeitangaben beziehen sich auf die Prozessorzeit, also die Zeit, die das Programm tatsächlich auf der CPU genutzt hat.

Die folgenden drei Tabellen enthalten die Rechenergebnisse der verschiedenen Testsätze. Die Spalten sind wie folgt beschriftet: n bezeichnet die Anzahl Knoten auf dem Kreis, m die Anzahl der Aufträge und k die maximale Anzahl der Umladeknoten, die in einer Lösung benutzt werden dürfen. Der pseudopolynomielle Algorithmus `k-Robot` enumeriert alle möglichen Flußwerte ψ und Drehrichtungen der Aufträge. Für jede Kombination der beiden Werte kann der gradbalancierte Graph bal berechnet werden. ZK gibt die größte Anzahl Zusammenhangskomponenten in einem solchen bal an. Die folgenden drei Werte beziehen sich auf eine optimale Lösung und waren für beide benutzten Algorithmen jeweils gleich. ψ^* ist der Flußwert auf dem Kreis und k' ist die Anzahl der benutzten Umladeknoten. Die darauf folgende Spalte gibt Auskunft über den Kostenwert einer Optimallösung. Die beiden letzten Spalten enthalten die Laufzeiten der beiden Implementationen in Sekunden.

Tabelle 8.1 enthält die Rechenergebnisse für die Testinstanzen des Typs `kreis-n.pdp`. Die Laufzeiten sind im Vergleich mit den anderen Instanztypen die kürzesten. Es fällt auf, daß auch größere Eingaben höchstens eine oder zwei Zusammenhangskomponenten in bal aufweisen. Mit Satz 8.1

Eingabe						opt.	MIP	k-Robot
n	m	k	ZK	ψ^*	k′	Zielfkt.	Zeit (s)	Zeit (s)
10	10	3	1	0	0	15665	0,01	0,01
20	19	5	1	0	0	72087	0,14	0,18
30	28	7	1	0	0	135242	2,22	0,62
40	38	10	1	1	0	304973	10,85	2,15
50	49	12	1	-5	0	401183	50,27	6,27
60	60	15	1	0	0	627383	61,70	12,43
70	65	17	2	-6	0	776699	288,04	23,40
80	77	20	1	1	0	1053170	755,72	38,58
90	89	22	1	-1	0	1392950		64,21
100	97	25	1	-2	0	1617340		96,97

Tabelle 8.1.: *Laufzeitergebnisse für den Datensatz* `kreis-n.pdp`

haben wir eine Begründung für diese Beobachtung geliefert: Die Gleichverteilung der Aufträge sorgt dafür, daß die meisten Knoten schon auf einer Zusammenhangskomponente liegen. Die Berechnung für das gemischt–ganzzahlige Programm konnten wegen der Speicherrestriktionen ab einer Größe von 90 Knoten nicht mehr durchgeführt werden.

In Tabelle 8.2 sind die Datensätze der Typen `kreis-n-V.pdp` und `kreis-n-v.pdp` aufgeführt und Tabelle 8.3 die der Instanzen `kreis-n-W.pdp` und `kreis-n-w.pdp`. Die Laufzeiten des Algorithmus `k-Robot` hängt unter anderem von der Maximalanzahl der Umladeknoten k ab. Wenn die Eingabe eine deutlich kleinere Zahl k besitzt, schlägt sich das in der Laufzeit des pseudopolynomiellen Verfahrens nieder. Das gemischt–ganzzahlige Programm wurde nach spätestens 18000 Sekunden (5 Stunden) abgebrochen.

Eingabe					opt.	MIP	k-Robot
n	k	ZK	ψ^*	k′	Zielfkt.	Zeit (s)	Zeit (s)
10	5	5	0	2	12875	0,39	0,09
	1		0	1	12902	0,58	0,02
12	6	6	0	5	36470	2,00	0,19
	2		0	2	37605	9,37	0,05
14	7	7	0	6	38192	13,50	0,37
	3		0	3	38944	51,96	0,11
16	8	8	0	6	38370	15,92	0,67
	3		0	3	38628	45,38	0,16
18	9	9	0	8	87825	606,94	1,25
	4		0	4	89251	2269,82	0,33
20	10	10	0	9	98372	1241,07	2,10
	4		0	4	99873	4147,79	0,45
22	11	11	0	8	77167	5410,23	3,52
	4		0	4	77346	2261,55	0,58
24	12	12	0	11	133502	3347,13	5,72
	5		0	5	135849		1,06
26	13	13	0	11	170263		8,91
	5		0	5	173165		1,33
28	14	14	0	13	193715		14,08
	6		0	6	196146		2,33
30	15	15	0	11	176651		19,75
	5		0	5	178524		2,07
40	20	20	0	19	358262		131,33
	9		0	9	361190		19,20
50	25	25	0	20	566506		510,66
	10		0	10	570064		51,50
60	30	30	0	28	878007		1795,97
	14		0	14	880172		231,54
70	35	35	0	30	1145230		1114,75
	15		0	15	1149462		557,67
80	40	40	0	37	1578711		703,02
	18		0	18	1583913		1510,33

Tabelle 8.2.: *Laufzeitergebnisse für die Datensätze* `kreis-n-V.pdp` *und* `kreis-n-v.pdp`

Eingabe					opt.	MIP	k-Robot
n	k	ZK	ψ^*	k′	Zielfkt.	Zeit (s)	Zeit (s)
10	5	5	0	2	16629	0,50	0,18
	1		0	1	17475	0,67	0,04
12	6	6	0	1	20226	1,56	0,42
	0		0	0	21373	0,23	0,03
14	7	7	0	2	24572	2,79	0,84
	1		0	1	25554	8,61	0,01
16	8	8	0	2	45881	20,46	1,24
	1		0	1	47032	45,26	0,13
18	9	9	0	3	37300	38,45	2,69
	1		0	1	38275	135,51	0,23
20	10	10	-1	5	53063	189,71	4,16
	2		1	2	54418	742,24	0,50
22	11	11	0	4	67784	884,47	6,55
	2		0	2	68999	2417,32	0,69
24	12	12	0	5	95059	1803,98	9,68
	2		0	2	97940	8510,07	0,86
26	13	13	0	6	87541	16507,40	13,72
	3		0	3	87894		1,60
28	14	14	0	5	96919		20,05
	2		0	2	98614		1,38
30	15	15	0	6	110989		28,63
	3		0	3	112486		2,67
40	20	20	0	8	193312		110,26
	4		0	4	193619		8,48
50	25	25	1	13	340718		369,41
	6		0	6	341796		32,35
60	30	30	0	11	434848		959,09
	5		0	5	436825		49,65
70	35	35	0	17	693242		1951,06
	8		0	8	696664		156,58
80	40	40	1	17	793078		478,56
	8		0	8	795300		263,11

Tabelle 8.3.: *Laufzeitergebnisse für die Datensätze* `kreis-n-W.pdp` *und* `kreis-n-w.pdp`

Literaturverzeichnis

[ACG+99] Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. March etti Spaccamela und M. Protasi: *Complexity and Approximation — Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag, 1999.

[AK88] Atallah, M.J. und S.R. Kosaraju: *Efficient solutions to some transportation problems with applications to minimizing robot arm travel.* SIAM Journal Computing, 17:849–869, 1988.

[Bar96] Bartal, Yair: *Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications.* In: *IEEE Symposium on Foundations of Computer Science*, Seiten 184–193, 1996.

[Bar98] Bartal, Yair: *On Approximating Arbitrary Metrics by Tree Metrics.* In: *Proc. of the 30th Ann. ACM Symp. on Theory of Computing*, Seiten 161–168, 1998.

[BP87] Barahona, F. und W.R. Pulleyblank: *Exact Arborescences, Matchings and Cycles.* Discrete Applied Mathematics, 16:91–99, 1987.

[CCC+99] Charikar, M., C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha und M. Li: *Approximation Algorithms for Directed Steiner Problems.* Journal of Algorithms, 33:73–91, 1999.

[CCG+98] Charikar, M., C. Chekuri, A. Goel, S. Guha und S. Plotkin: *Approximating a Finite Metric by a Small Number of Tree Metrics.*
In: *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, Seiten 379–388, Washington, DC, USA, 1998. IEEE Computer Society.

[CL65] Chu, Y.J. und T.H. Liu: *On the shortest arborescence of a directed graph.*
Science Sinica, 14:1396–1400, 1965.

[COKN03] Coja-Oghlan, A., S. O. Krumke und T. Nierhoff: *Scheduling a server on a caterpillar network - a probabilistic analysis.*
In: *Proceedings of the 6th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2003.
to appear.

[CR98] Charikar, M. und B. Raghavachari: *The Finite Capacity Dial–A–Ride Problem.*
In: *IEEE Symposium on Foundations of Computer Science*, Seiten 458–467, 1998.

[Dan51] Dantzig, G.: *Maximization of a linear function of variables subject to linear inequalities.*
In: Koopmans, T. (Herausgeber): *Activity Analysis of Production and Allocation*, Seiten 359–373. John Wiley & Sons, 1951.

[Die96] Diestel, R.: *Graphentheorie.*
Springer-Verlag, 1996.

[Edm67] Edmonds, J.: *Optimum branchings.*
Journal of Research of the National Bureau of Standards, B 71:233–240, 1967.

[Fei98] Feige, U.: *A Threshold of* $\ln n$ *for Approximating Set Cover.*
Journal of the ACM, 45:634–652, 1998.

[FG92] Frederickson, G. und D. Guan: *Preemptive Ensemble Motion Planning on A Tree.*

SIAM Journal of Computing, 21:1130–1152, 1992.

[FG93] Frederickson, G. und D. Guan: *Nonpreemptive Ensemble Motion Planning on A Tree.*
Algorithms, 15:29–60, 1993.

[Fre93] Frederickson, G.: *A note on the complexity of a simple transportation problem.*
SIAM Journal of Computing, 22:57–61, 1993.

[FT87] Fredman, M. L. und R. E. Tarjan: *Fibonacci heaps and their uses in improved network optimization problems.*
Journal of the ACM, 34:596–615, 1987.

[Ful74] Fulkerson, D.R.: *Packing rooted directed cuts in a weighted directed graph.*
Mathematical Programming, 6:1–13, 1974.

[GGST86] Gabow, H.N., Z. Galil, T. Spencer und R.E. Tarjan: *Efficient Algorithms for finding minimum spanning trees in undirected and directed graphs.*
Combinatorica, 6:109–122, 1986.

[GJ79] Garey, M. R. und D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP–Completeness.*
Freeman, New York, 1979.

[GT84] Gabow, H.N. und R.E. Tarjan: *Efficient Algorithms for a Family of Matroid Intersection Problems.*
Journal of Algorithms, 5:80–131, 1984.

[Gua98] Guan, D. J.: *Routing a vehicle of capacity greater than one.*
Discrete Applied Mathematics, 81:41–57, 1998.

[Ham58] Hamilton, W. R.: *Account of the Icosian Calculus.*
In: *Proceedings of the Royal Irish Academy*, Band 6, Seiten 415–416, 1858.

[HJK80] Hausmann, D., T. A. Jenkyns und B. Korte: *Worst case analysis of greey type algorithms for independence systems.*
Mathematical Programming Study, 12:120–131, 1980.

[HKRW01] Hauptmeier, D., S. O. Krumke, J. Rambau und H.-C. Wirth: *Euler is standing in line. Dial–a–ride problems with precendence–constraints.*
Discrete Applied Mathematics, 113:87–107, 2001.

[HRW92] Hwang, F. K., D. S. Richards und P. Winter: *The Steiner Tree Problem.*
Annals of Discrete Mathematics. North Holland, Amsterdam, 1992.

[Ilo01] Ilog: *Ilog Cplex 7.1 User's Manual*, 2001.

[Kar72] Karp, R.M.: *Reducibility among combinatorial problems.*
In: Miller, R.E. und J.W. Thatcher (Herausgeber): *Complexity of Computer Computations*, Seiten 85 – 103. Plenum Pub Corp, 1972.

[Kha79] Khachiyan, L. G.: *A Polynomial Algorithm in Linear Programming.*
Soviet Mathematics Doklady, 20:191 – 194, 1979.

[KR05] Krumke, S. O. und D. Räbiger: *Persönliche Kommunikation*, 2005.

[KRS03] Kloock, M., D. Räbiger und R. Schrader: *Persönliche Kommunikation*, 2003.

[KV02] Korte, B. und J. Vygen: *Combinatorial Optimization. Theory and Algorithms.*
Springer, Berlin, 2. Auflage, 2002.

[Law76] Lawler, E.: *Combinatorial Optimization: Networks and Matroids.*
Holt, Rinehart and Winston, New York, 1976.

[LLKS85] Lawler, E. L., J.K. Lenstra, A. H. G. Rinnooy Kan und D. B. Shmoys (Herausgeber): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.*
John Wiley, Chichester, UK, 1985.

[Mue00] Mues, C.: *Ein Pickup & Delivery–Problem mit Umladung von Gütern.*

Diplomarbeit, Universität Bonn, 2000.

[Mus96] Muslea, I.: *The k–Vehicle Routing Problem in Trees.*
Diplomarbeit, West Virginia University, 1996.

[Mus97] Muslea, I.: *The Very Offline k-Vehicle Routing Problem in Trees.*
In: *International Conference of the Chilean Computer Science Society*, Seiten 155–163, 1997.

[Oer00] Oertel, P.: *Routing with Reloads.*
Doktorarbeit, Universität zu Köln, 2000.

[OW96] Ottmann, T. und P. Widmayer: *Algorithmen und Datenstrukturen.*
Spektrum–Verlag, 3 Auflage, 1996.

[Oxl92] Oxley, J.G.: *Matroid Theory.*
Oxford University Press, 1992.

[PS98] Papadimitriou, C. H. und K. Steiglitz: *Combinatorial Optimization. Algorithms and Complexity.*
Dover, Mineola, 1998.

[PY00] Papadimitriou, C. H. und M. Yannakakis: *On the approximability of trade–offs and optimal access of web sources (Ext. Abstract).*
In: *41st Annual Symposium on Foundations of Computer Science: Proceedings: 12–14 November, 2000, Redondo Beach, California*, Seiten 86–92, 2000.

[Räb04a] Räbiger, D.: *Semi–Preemptive Routing on a Line (Ext. Abstract).*
Electronic Notes in Discrete Mathematics, 17C:241–246, 2004.

[Räb04b] Räbiger, D.: *Semi–Preemptive Routing on a Linear and Circular Track.*
Eingereicht bei Discrete Applied Mathematics, 2004.

[Sch98] Schrijver, Alexander: *Theory of Linear and Integer Programming.*
Series in Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[SLL02] Siek, J. G., L. Lee und A. Lumsdaine: *The Boost Graph Library.*
C++ In–Depth. Addison–Wesley, Boston, 2002.

[Sol94] Sol, M.: *Column Generation Techniques for Pickup and Delivery Problems.*
Doktorarbeit, Technical University Eindhoven, 1994.

[SS95] Savelsbergh, M. W. P. und M. Sol: *The general Pickup and Delivery Problem.*
Transportation Science, 29:17–29, 1995.

[TV01] Toth, P. und D. Vigo (Herausgeber): *The vehicle routing problem.*
Society for Industrial and Applied Mathematics, 2001.

[WCS97] Wall, L., T. Christiansen und R. Schwartz: *Programmieren mit Perl.*
O'Reilly, Köln, 1997.

[Weg93] Wegener, Ingo: *Theoretische Informatik.*
B. G. Teubner, 1993.

[Wel76] Welsh, D. J. A.: *Matroid Theory.*
Academic Press, 1976.

[Whi86] White, Neil L. (Herausgeber): *Theory of matroids.*
Cambridge University Press, Cambridge, 1986.

[YT01] Yamada, T. und M. Takahashi: *Heuristic and exact algorithms for the spanning tree detection problem*, 2001.
IFIP 2001, *http://www.nda.ac.jp/~yamada/paper/stdp.pdf.*

Index

G_T, *siehe* Transportgraph
H, *siehe* Hilfsgraph
$I^{(X]}$, *siehe* konsekutive Knotenmenge
$I^{(X)}$, *siehe* konsekutive Knotenmenge
$I^{[X]}$, *siehe* konsekutive Knotenmenge
Δ, *siehe* Umladekosten
ψ, *siehe* Fluß auf Kreisen
$\mathtt{bal}(\psi, r)$, *siehe* balancierter Hilfsgraph
(d,r)–Arboreszenz, 26, 77
    Approximationsalgorithmus, 41
(d,r)–Steinerarboreszenz, 114

Algorithmus
    `(d,r)-Arboreszenz-Heuristik`, 118
    `Arboreszenz-Heuristik`, 112
    `Exogen`, 74
    `Fix`, 91
    `Gewichtsgreedy`, 16
    `d-Baum-Greedy`, 23
    `k-RobotLight`, 84
    `k-Robot`, 93
    Güte, 9–10
    Laufzeit, 7–8
    pseudopolynomieller, 8
Approximationsschema, 10
APX, 10, 108
Arboreszenz, 14, 66
    Charakterisierung, 14
    gefärbte, *siehe* (d,r)–Arboreszenz
Auftrag, 47
    Drehrichtung eines, 56
    langer, 57

Baum, 13
    gefärbter, *siehe* d–Baum

d–Baum, 21, 23
DTIME, 8, 108

Exakt–C–Arboreszenz–Problem, 30, 31
Exakt-D, 38

Fluß

auf Bäumen, 100
auf Kreisen, 59
in Netzwerken, 128
FPTAS, *siehe* Approximationsschema
für (d,r)–Arboreszenz, 41
Fundamentalkreis, 23

Gap–Problem, 36
Graph
Definitionen, 10
eulerscher, 12
Multigraph, 13
zusammenhängender, 11
stark, 11, 13

Hamilton–Kreis–Problem, 45
Hilfsgraph
balancierter, 60
endogenes PDP auf Baum, 113–114
endogenes PDP auf Kreis, 77
exogenes PDP auf Baum, 105–106
exogenes PDP auf Kreis, 66–67
pfadreduzierter, 109

Instanz, 7, 33
Generierung, 131–134
Intervalle überdecken, 58

Knoten kreuzen, 58
konsekutive Knotenmenge, 85, 88–90
Rand einer, 86, 87

Matroid, 15–16
graphisches, 15, 22, 26
Partitionsmatroid, 15, 22, 26
Schnitt von p Matroiden, 27
Schnitt von drei Matroiden, 26
Schnitt von zwei Matroiden, 22

NP–vollständig
diverse Begriffe, 8–10

Paretokurve, 33–37
Pickup–And–Delivery–Problem, 47
nicht–präemptives, 47
präemptives, 47
semi–präemptives
endogenes, 53
exogenes, 52
Programm
dynamisches, 85, 93, 125
ganzzahliges, 17
gemischt–ganzzahliges, 17, 126–129
lineares, 16, 27–28
PTAS, *siehe* Approximationsschema

r–Schnitt, 14, 27
Rechenergebnisse, 134–135

Steinerarboreszenz–Problem, 106
Steinerbaum–Problem, 101

Transportgraph, 58
Transportproblem, 45
    auf speziellen Graphen, 49–54
    semi–präemptives, 55
Traveling–Salesperson–Problem, 46

Umladeknoten
    endogen definierte, 76
    exogen definierte, 66
Umladekosten, 52
Umladen, 58

Zahlproblem, 8
Zerschneiden von Kanten, 56

Kurzzusammenfassung

Das Problem, einen Roboter (oder ein Fahrzeug) so zwischen n Stationen zu steuern, daß m Objekte von ihrem Ausgangspunkt zu ihrem Ziel transportiert werden können und dabei die zurückgelegte Fahrstrecke minimiert wird, bezeichnet man als *Pickup–And–Delivery–Problem*. Diese Aufgabenstellung ist in der Vergangenheit gut untersucht worden, sogar wenn die Stationen auf eine besondere Weise angeordnet sind, z.B. in einer Reihe oder im Kreis. Üblicherweise darf der Roboter entweder *jede* oder *keine* Station als Umladestation benutzen. Man spricht in diesen Fällen von einem *präemptiven* bzw. *nicht–präemptiven* Transport.

Wir werden diese Konzepte verallgemeinern, indem nur ein Teil der Stationen für das Umladen benutzt werden darf. In Anlehnung an die beiden anderen Versionen soll diese Fragestellung *semi–präemptiv* heißen. Dabei differenzieren wir zwischen einer *exogenen* und einer *endogenen* Variante. In der ersten darf der Roboter nur an gekennzeichneten Stationen umladen. In der zweiten teilen wir ihm eine Zahl k mit, und es ist Teil der Aufgabenstellung zu entscheiden, an welchen k Stationen umgeladen wird. Wir zeigen, daß sowohl die exogene als auch die endogene Version auf einem Pfad und auf einem Kreis effizient lösbar ist und geben jeweils dynamische Lösungverfahren an. Beide Varianten sind auf Bäumen $\mathcal{NP}$–vollständig. Für den exogenen Fall geben wir einen Approximationsalgorithmus mit einer Güte von $4/3$ an. Für die endogene Aufgabenstellung präsentieren wir eine $(4/3+\varepsilon)$–Approximation für beliebiges $\varepsilon > 0$. Schließlich nutzen ein bekanntes Resultat, um eine Approximation für das exogene und endogene Transportproblem auf metrisch gewichteten Graphen angeben zu können.

Aus der angewandten Problemstellung ergibt sich ein graphentheoretisches Teilproblem, das wir näher untersuchen. In einem gerichteten Graphen $G = (V, E^r \dot\cup E^b)$ mit einer rot/blau–partitionierten Kantenmenge soll zu gegebener Kostenfunktion eine kostenminimale und aufspannende Arboreszenz bestimmt werden, die höchstens $d \in \mathbb{N}$ blaue Kanten benutzt. Wir geben ein voll polynomielles Approximationsschema an. Dieses nutzen wir, um das endogene Transportproblem auf Bäumen zu approximieren.

Abstract

We consider the problem of routing a robot (or vehicle) between n stations in order to transport m objects from their source node to their destination node such that the total travel distance is minimized. This problem is known as *Pickup and Delivery Problem* and well studied in literature, even if the stations are specially arranged, e.g. on a linear track or circle. The robot may use either *all* or *none* of the stations for reloading. These cases are said to be *preemptive* resp. *non–preemptive*.

We will generalize these concepts by restricting the nodes that can be used for reloading. According to the previous versions we will call this variation *semi–preemptive*. We will distinguish between an *exogenous* and an *endogenous* case. Problems of the first type will allow the robot to use only a subset of stations that can be used as reload nodes. The latter case consists of a number k and it will be part of the problem to decide at which stations the robot reloads, but no more than k times. We will show that both the exogenous and the endogenous problem are efficiently solvable by dynamic programming on paths and circles. Both variants are $\mathcal{NP}$–complete on trees. For the exogenous case we will give an approximation algorithm with ratio $4/3$. The endogenous case can be solved with ratio $(4/3 + \varepsilon)$, for arbitrary $\varepsilon > 0$.

In conclusion we use a known result in order to approximately solve the exogenous resp. endogenous routing problem on graphs weighted by a metric.

We will study a graph theoretical problem which arises from the mentioned transportation problem. Given a directed graph $G = (V, E^r \dot{\cup} E^b)$ with the arc set partitioned into red and blue subsets and a cost function on the arc set, find a minimal cost arborescence spanning G and using at most d blue arcs. Within this work a fully polynomial time approximation scheme (FPTAS) will be presented for this problem. The FPTAS will be used to approximately solve the endogenous transportation problem on a tree.

Ich versichere, daß ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen —, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; daß diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; daß sie — abgesehen von unten angegebenen Teilpublikationen — noch nicht veröffentlicht worden ist sowie, daß ich eine solche Veröffentlichung vor Abschluß des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen dieser Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Professor Dr. Rainer Schrader betreut worden.

Teilpublikationen:

- Dirk Räbiger, *Semi–Preemptive Routing on a Line*. Electronic Notes in Discrete Mathematics, 17C:241–246, 2004 [Räb04a].
- Dirk Räbiger, *Semi–Preemptive Routing on a Linear and Circular Track*. Eingereicht bei Discrete Applied Mathematics, 2004 [Räb04b].

Lebenslauf

Ich wurde am *23. Mai 1974* zu Christi Himmelfahrt um 17 Uhr 56 in *Köln* am Rhein geboren und später auf den Namen *Dirk Räbiger* getauft. Meine *Nationalität* ist *deutsch*. Bis zum heutigen Tag bin ich *ledig* geblieben.

In den Jahren *1980 bis 1984* besuchte ich die *Grundschule* in Köln–Nippes. Daran anschließend lernte ich am *Erich–Kästner Gymnasium* in Köln–Niehl und schloß *1993* mit dem Abitur ab.

Ich weigerte mich, Dienst an der Waffe zu leisten und half statt dessen während meines *Zivildienstes* bis *1994* behinderten Menschen bei der *Johanniter Unfallhilfe* in Köln–Deutz.

Im *Wintersemester 1994/1995* begann ich das Studium der *Politikwissenschaften*, *Germanistik* und *Pädagogik* an der *Universität zu Köln*.

Zum *Wintersemester 1995/1996* wechselte ich auf das Fach *Wirtschaftsinformatik*, das ich schließlich im *März 2000* mit Erlangen des Diploms beendete. Während des Studiums arbeitete ich als *Studentische Hilfskraft* am *Zentrum für Paralleles Rechnen* und programmierte für die Projekte *Paralor* und *Nikos*.

Ab *April 2000* erhielt ich eine Anstellung als *Wissenschaftlicher Mitarbeiter* am *Zentrum für Angewandte Informatik Köln*, die bis heute währt. Zunächst arbeitete ich in einem Projekt der Bausparkassen zur Kollektivsimulation. Später betreute ich als Assistent Vorlesungen des Lehrstuhls für Informatik, Prof. Dr. Schrader, und forschte im Rahmen der Kombinatorischen Optimierung.