FernUniversität Hagen

Fachbereich Mathematik

Min Cost Flow in balancierten Netzwerken mit konvexer Kostenfunktion

Diplomarbeit

von

Henning Soller

Themensteller:	Prof. Dr. Hochstättler
Betreuer:	Prof. Dr. Hochstättler
Abgabetermin:	03. September 2007

Hiermit erkläre ich, dass ich die Diplomarbeit selbstständig angefertigt und nur die angegebenen Quellen verwendet habe.

Dossenheim, den 23. August 2007

Danksagung

An dieser Stelle möchte ich einigen Personen danken, die mich bei der Fertigstellung dieser Arbeit unterstützt, bzw. diese erst ermöglicht haben.
Zu allererst möchte ich hierbei Prof. Dr. Winfried Hochstättler für das interessante Thema und die Betreuung meiner Diplomarbeit danken. Ebenso möchte ich David Reeb, Ingo Deppner und Fatih Argin für das Korrekturlesen und die anregenden Diskussionen über meine Diplomarbeit danken.
Eine Diplomarbeit ist aber immer nur ein Punkt in einer langen Entwicklung. So geht ein ganz besonderer Dank an meine Eltern, die mir mein Studium erst ermöglicht haben und mich in meinen Plänen stets unterstützt haben. Ebenso möchte ich Ralf Eggers und Prof. Dr. Rolf Rannacher danken, die mein Interesse an der Mathematik in der Schule und später an der Universität erst geweckt haben und so maßgeblichen Anteil an meiner Entscheidung für ein Studium der Mathematik hatten.
Abschließend möchte ich auch Prof. Michael Raymer und Prof. Dr. Donald Kossmann danken, die mir während des Studiums an der University of Oregon und an der Universität Heidelberg die Mitarbeit in ihren Forschungsgruppen ermöglicht haben. Genauso möchte ich auch den übrigen Professoren an der Universität Kaiserslautern, der Universität Heidelberg, der Fernuniversität Hagen und der University of Oregon danken, die mich während meines Studiums begleitet haben.

Inhaltsverzeichnis

Abstract

Standard matching problems can be stated in terms of skew symmetric networks. On skew symmetric networks matching problems can be solved using network flow techniques. We consider the problem of minimizing a separable convex objective function over a skew-symmetric network with a balanced flow. We call this problem the Convex Balanced Min Cost Flow (Convex BMCF) problem.

We start with 2 examples of Convex BMCF problems. The first problem is a problem from condensed matter physics: We want to simulate a so called super-rough phase using methods from graph-theory. This problem has previously been studied by Blasum, Hochstättler, Rieger and Moll [15]. The second problem is a typical example for the minconvex-problems previously studied by Apollonio and Sebö ([5], [6]) and Berger and Hochstättler [9].

We review the results for skew-symmetric networks by Jungnickel and Fremuth-Paeger ([22], [23], [24], [25], [26]) and Kocay and Stone ([41], [42]). Using these results we present several algorithms to solve the Convex BMCF problem. We present the first complete version of the Primal-Dual algorithm previously studied by Fremuth-Paeger and Jungnickel [27]. However, we only consider the case of positive costs. We also show how to apply this algorithm to the Convex BMCF problem. Then we extend the Shortest Admissible Path Approach of Jungnickel and Fremuth-Paeger [23, p. 12] to a complete algorithm for linear as well as convex cost problems on skew symmetric networks. In the same manner we show how to adapt the Capacity Scaling algorithm by Ahuja and Orlin [3, pp. 108-113] to skew symmetric networks and balanced flows. The capacity scaling algorithm is weakly polynomial. Another possibility for a weakly polynomial algorithm is the Balanced Out-of-Kilter algorithm. This algorithm is based on Fulkerson's Out-of-Kilter algorithm [28] and Minoux's adaptation of the algorithm for convex costs [49]. We show that augmentation on valid paths is not always necessary and introduce the idea of slightly different networks. Using the same ideas for the Balanced Capacity Scaling we obtain an Enhanced Capacity Scaling algorithm. The Enhanced Capacity Scaling algorithm as well as the Balanced Out-of-Kilter algorithm are the fastest algorithms presented here with a complexity of roughly $O(m^2 \log_2 U)$.

Finally we show how to solve the problem from condensed matter physics using the new idea of anti-balanced flows on skew-symmetric networks. Using the Balanced Successive Shortest Path algorithm we also obtain a new complexity limit for the minconvex problem of $O(km\log(n))$. This improves the complexity bound of Berger [8] by a factor of $\frac{m}{k}$ in the case of separable convex costs with positive slope.

In the appendix of this thesis we consider dual approaches for the Convex BMCF problem. The Balanced Relaxation algorithm, based on the Relaxation algorithm by Bertsekas [13], does not determine a balanced flow as the resulting flow will not necessarily be integral. This way we only determine fractional matchings. As the algorithm is also slow this algorithm is probably of limited use. A better ansatz seems to be the Cancel and Tighten method by Karzanov and McCormick [38]. We review their results and end with some ideas on how to implement a balanced version of this algorithm.

Parts of this thesis have been published in:

H. Soller 'Balanced Min Cost Flow on skew symmetric networks with convex costs', submitted to Open Journal of Discrete Mathematics, 2013

Kapitel 1

Ein physikalisches Hors d'œuvre

'Some people call these problems hard...
I call them interesting' — John Toner

Der Blick in die Natur ist oft faszinierend. Warum richtet sich eine Kompassnadel immer nach Norden aus? Wie fliegt der Kranich im Schwarm? Wer malt die Bilder in den Sand (Abb. **??**)? Der geneigte Leser mag über diese Fragen verwundert sein und nach einem einführenden Beispiel zu Matchings suchen. Ein solches werden wir im 2. Kapitel angeben. Was aber haben nun Kompassnadeln, Vogelschwärme und Sandhaufen gemeinsam? Alle diese Probleme sind nach obigem Zitat, je nach Betrachtungsweise, entweder hart oder interessant, denn alle lassen sich mit Methoden aus der Renormierungsgruppentheorie lösen.
Frei nach der Aufforderung von David Mermin („*Shut up and calculate!*“) greifen wir uns das Problem der Sandhaufen heraus. Sand ist nicht immer flach, sondern es ergeben sich oft raue, interessant geformte Oberflächen wie in dem obigen Bild. Ein möglicher Grund für die „Wellen“ im Sand könnte sein, dass der darunterliegende Sand verschmutzt ist und der Sand daher keine ebene Grundlage hat.
Ein ganz ähnliches Problem ist das Kristallwachstum. Kristallzüchtung erfolgt heutzutage häufig (insbesondere bei hoher benötigter Qualität) mit sogenannten „epitaktischen Verfahren“. Dabei wird das neue Kristallmaterial in einer sauberen Umgebung Atomschicht auf Atomschicht auf eine Oberfläche aufgedampft. Natürlich ist für die technische Anwendung der Kristalle (z.B. Silizium) wichtig, wie sich eine raue Oberfläche auf die Form des Kristallwachstums auswirkt.
Das Problem von Sandkörnern oder Atomen auf einer rauen Oberfläche wollen wir uns vereinfacht vorstellen wie Quader, die man in Türmen auf einer ungleichmäßigen Fläche aufstapelt [65]. Dies ist sicherlich bereits eine starke Approximation, insbesondere für den Kristall erscheint sie aber angesichts der gleichmäßigen Struktur durchaus brauchbar. Dies sei in der folgenden Abbildung verdeutlicht (Abb. 1.1). Auf die angegebenen Bezeichnungen gehen wir später noch ein.
In einem Kristall gibt es chemische Bindungskräfte und auch die Sandkörner üben wechselseitig aufeinander eine Reibungskraft aus.
So liegt es nahe zu fragen, wie die Austauschwechselwirkung aussieht. Den Schmutz bzw. die raue Oberfläche wollen wir durch Dichtefluktuationen $\delta\rho(\vec{x}, n)$ beschreiben, wobei $\vec{x}$ die x,y-Koordinaten auf einer Oberfläche beschreibe und $z = a \cdot n(x, y)$ die Richtung des

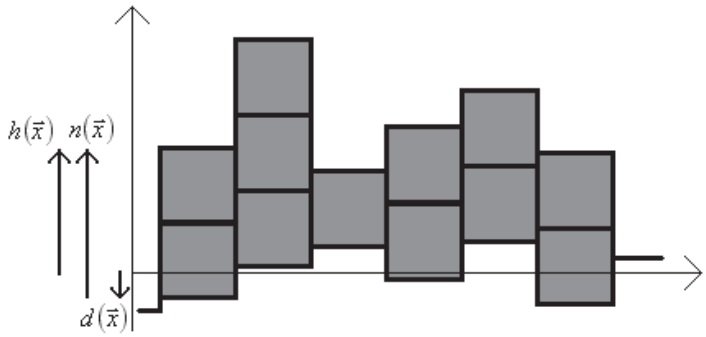

Abbildung 1.1: Illustration des Modells für Sandhaufen und Kristalle

Kristallwachstums bzw. der Gravitationskraft sei. Hierbei ist a die Gitterkonstante des Kristalls bzw. die Größe eines Sandkorns. n ist also dimensionslos.
Dann erlaubt die Symmetrie des Systems (wir wollen auf die Argumente hier nicht näher eingehen) in der freien Energie nur einen Term, der proportional ist (mit Proportionalitätskonstante $\bar{K}$) zu einem Oberflächenintegral über die Dichte:

$$E_\sigma = \bar{K} \int d\sigma \left[\rho_0 + \delta\rho(\vec{x}, z = a \cdot n(x, y))\right] \tag{1.1}$$

Hierbei sei ρ_0 die Dichte des unverschmutzten Materials.
Wir können nun die Energie Taylor-entwickeln in ∇n und $\delta\rho$. Hierbei gilt für das Oberflächenelement $d\sigma = \sqrt{1 + a^2(\nabla n)^2} d^2x$.
Wir bezeichnen diese Taylornäherung, da sie nur die „relevanten" Teile der Energie enthält hier als *Hamiltonian*:

$$H = \int d^2x \left[\frac{1}{2} v(\nabla n)^2 + \bar{K} \cdot \delta\rho(\vec{x}, z(\vec{x}))\right] \tag{1.2}$$

Hierbei haben wir wieder eine neue Proportionalitätskonstante v eingeführt.
Wir betrachten nun die statistischen Oberflächenfluktuationen von $n(\vec{x})$ unter dem Einfluss der eingefrorenen (d.h. unveränderlichen) Dichtefluktuationen $\delta\rho$.
Diese Dichtefluktuationen können wir nun noch Fourier-transformieren und erhalten:

$$\delta\rho(\vec{x}, z) = \Re\left(\sum_\mu \rho_\mu e^{i2\pi\vec{G}_\mu \begin{pmatrix} \vec{x} \\ z \end{pmatrix}} e^{i2\pi\phi_\mu(\vec{x},z)}\right) \tag{1.3}$$

Hierbei beschreibt der Term $\phi_\mu(\vec{x}, z)$ die Rauigkeit des Untergrunds. Da wir angenommen haben, dass die Dichtefluktuationen eingefroren sind, muss $e^{i\phi_\mu(\vec{x},h)}$ exponentiell abfallen auf einer Länge, die der Größenordnung der Korrelationslänge der Störungen im Kristall bzw. der Ausdehnung eines Sandhaufens entspricht.
Wir machen hier die Näherung $\phi_\mu(\vec{x}, z) \approx \phi_\mu(\vec{x}, z = 0)$. Man kann zeigen, dass diese Näherung für kleine Proben gerechtfertigt ist [66, S. 632].
Ferner nehmen wir von der Fouriertransformation hier lediglich den ersten Term mit. Man

kann zeigen, dass die übrigen Terme irrelevant sind. Wir erhalten damit den sogenannten *Sine-Gordon Hamiltonian* mit einer weiteren Proportionalitätskonstante γ_0:

$$H = \int d^2x \left\{ \frac{1}{2} v(\nabla n)^2 + \gamma_0 \cos(2\pi G_0 z(\vec{x}) + 2\pi\phi_1(\vec{x})) \right\} \tag{1.4}$$

$$= \int d^2x \left\{ \frac{1}{2} v(\nabla n)^2 + \gamma_0 \cos(2\pi G_0 a(n(\vec{x}) + d(\vec{x}))) \right\} \tag{1.5}$$

Hierbei gilt $d(\vec{x}) = \frac{\phi_1(\vec{x})}{a \cdot G_0}$. Tsai und Shapir haben nun den Term $n(\vec{x}) + d(\vec{x})$ folgendermaßen interpretiert [67]: Da wir ja nur ganzzahlige Mengen an Sandkörnern und Atomen haben gilt $n(\vec{x}) \in \mathbb{N}$. $d(\vec{x})$ beschreibt die Rauigkeit des Untergrunds. D.h. wie in dem Schaubild (Abb. 1.1) gezeichnet, wird die Gesamthöhe des Atoms bzw. Sandkorns durch $h(\vec{x}) = n(\vec{x}) + d(\vec{x})$ angegeben. Hierbei können wir $d(\vec{x}) \in [-0,5; 0,5]$ wählen, da die Rauigkeit sonst wie ein weiteres Sandkorn wirkt.
Ferner sehen wir, dass der Hamiltonian eine Symmetrie gewonnen hat, die der Ausgangshamiltonian nicht hatte: $n(\vec{x}) \to n(\vec{x}) + m$ mit $m \in \mathbb{N}$. Wir können also eine neue Lage von Atomen oder Sandkörnern aufschichten, ohne die Energie zu verändern.
Diese Symmetrie wollen wir nun verwenden, um einen noch einfacheren Hamiltonian zu finden. Dazu kehren wir noch einmal zu unserer Ausgangsüberlegung zurück. Wir wollten eine Beschreibung der Austauschwechselwirkung erhalten. Wir nehmen dazu an, dass diese durch eine Funktion V beschrieben werde und abhängig sei von der Höhe h des Sandhaufens bzw. des Kristallturms. Nach obiger Argumentation gilt $h_i = n_i + d_i$, wenn wir die Haufen bzw. Türme durchnummerieren mit $i \in \mathbb{N}$.
Um die Symmetrie $h_i \to h_i + m \ \forall m \in \mathbb{N}$ zu erfüllen, können wir am einfachsten einen Wechselwirkungshamiltonian wählen, der nur von der Differenz der h_i's abhängig ist, also:

$$H = -\sum_{(ij)} V(h_i - h_j) \tag{1.6}$$

Für das Sandhaufen-Problem ist offensichtlich, dass die Wechselwirkung nur von den nächsten Nachbarn kommen kann. Bei der Wechselwirkung von Kristallatomen ist dies weniger eindeutig. Trotzdem macht man meist die Annahme, nur die nächsten Nachbarn (n.N.) trügen zur Wechselwirkung bei.
Die Potentialfunktion kann im Prinzip beliebig gewählt werden. Eine glatte Oberfläche mit $h_i = h_j$ (mit i und j nächsten Nachbarn) erscheint vorteilhaft. Wir können also V so wählen, dass für $h_i = h_j$ gilt $V(h_i - h_j) = 0$. Als einfachsten Ansatz können wir nun $V(h_i - h_j) = \left(\frac{J}{b}\right)^2 (h_i - h_j)^2$ nehmen mit einer Proportionalitätskonstanten $\frac{J}{b}$. In [17, S. 582-584] wird dieser Ansatz noch etwas weitergehender motiviert.
Man erhält damit das *„Solid-on-Solid" (SOS) Modell* auf ungeordneten Oberflächen:

$$H = \left(\frac{J}{b}\right)^2 \sum_{(ij) n.N.} (h_i - h_j)^2 \tag{1.7}$$

Die 2 betrachteten Hamiltonians ((1.5) und (1.7)) sehen nun völlig unterschiedlich aus. Interessant ist aber, dass sie beide die gleichen Symmetrien aufweisen.

In der obig erwähnten Renormierungsgruppentheorie interessiert man sich bei den dargestellten Problemen für sogenannte Phasenübergänge wie z.B. von Wasser zu Eis (oder von wirr umherfliegenden Vögeln zum Schwarm). Da die Symmetrien der beiden Hamiltonians gleich sind, kann man zeigen, dass beide Hamiltonians die selbe Universalitätsklasse, d.h. die selben Phasenübergänge, haben [34, S. 137].
Für den Sine-Gordon Hamiltonian fanden Toner und diVincenzo einen interessanten Phasenübergang bei niedrigen Temperaturen von einer rauen zu einer sog. *superrauen Phase*. In dieser Phase werden die Fluktuationen in den Höhen bei tiefen Temperaturen entgegen der Intuition nicht kleiner, sondern größer. Wir können uns also vorstellen, dass Kristalle dann sehr unschöne Oberflächen bekommen und Sandhaufen bei geringem Wind (entspricht der Temperatur) besonders hoch werden.
Nun können wir versuchen diese supperraue Phase zu beobachten. Zum Einen kann man dies mittels Röntgenstrahluntersuchungen von Proben tun. Ein anderer Weg wäre die Simulation des Systems. Hierbei müssen wir die superraue Phase besonders gut bei tiefen Temperaturen beobachten können [58, S. 1]. Der geneigte Physiker mag nun sofort an Monte-Carlo Simulationen denken. Diese sind aber bei tiefen Temperaturen schwer zu beherrschen bzw. wenig effizient [58, S. 1].
Statt dessen können wir uns hier an eine Erfindung von Leonhard Euler im Jahre 1736 erinnern: Die Graphentheorie [35, S. 1].
Wir wollen also versuchen unser Problem als Graphenproblem darzustellen. Dazu nehmen wir uns den Hamiltonian und versuchen den Zustand niedrigster Energie zu finden. Dies entspricht der Berechnung des exakten Grundzustandes des Systems bei Temperatur 0 Kelvin. Wir können also untersuchen, ob das System bei $T = 0K$ immer noch in der superrauen Phase ist.
Notieren wir an dieser Stelle die Energieminimierung explizit:

$$\min H(n) = \left(\frac{J}{b}\right)^2 \sum_{(ij)n.N.} ((d_i + n_i) - (d_j + n_j))^2 \tag{1.8}$$

Den Faktor $\frac{J}{b}$ können wir hier auch fortfallen lassen, da er ohnehin nur eine Proportionalitätskonstante ist. Wir definieren nun $d_{ij} = d_i - d_j$ und $x_{ij} = n_i - n_j$ und erhalten damit den reskalierten Hamiltonian: $H(x) = \sum_{(ij)n.N.}(d_{ij} + x_{ij})^2$.
Wir sehen sofort $(d_{ij} + x_{ij})^2 = (d_{ji} + x_{ji})^2$, denn $(h_i - h_j)^2 = (h_j - h_i)^2$.
Die veränderlichen Variablen sind allein die x_{ij}'s. Ferner ist i immer genau dann ein nächster Nachbar von j, wenn j ein nächster Nachbar von i ist.
Die Verbindung zu Netzwerken ist nun anschaulich klar, auch wenn wir die gleich verwendeten Begriffe erst später einführen werden. Die Sandkörner werden zu Knoten im Netzwerk und die nächsten Nachbarn durch Kanten verbunden. Um nicht 2 Kanten (ij) und (ji) einführen zu müssen, verdoppeln wir die Knoten und haben dann nur noch einfache Kanten zwischen diesen Knoten. Wir versehen die neuen Knoten mit einem Strich. Knoten i und Knoten i' beschreiben also dasselbe Sandkorn. Die Struktur des entstehenden Netzwerks wird in der folgenden Abbildung veranschaulicht, in der wir zunächst die Struktur einer einzelnen Sandkorn- bzw. Atomlage gezeichnet haben und daneben die Übertragung in ein Netzwerk. Hierbei wird die Symmetrie der nächsten Nachbarn auch im Netzwerk deutlich.

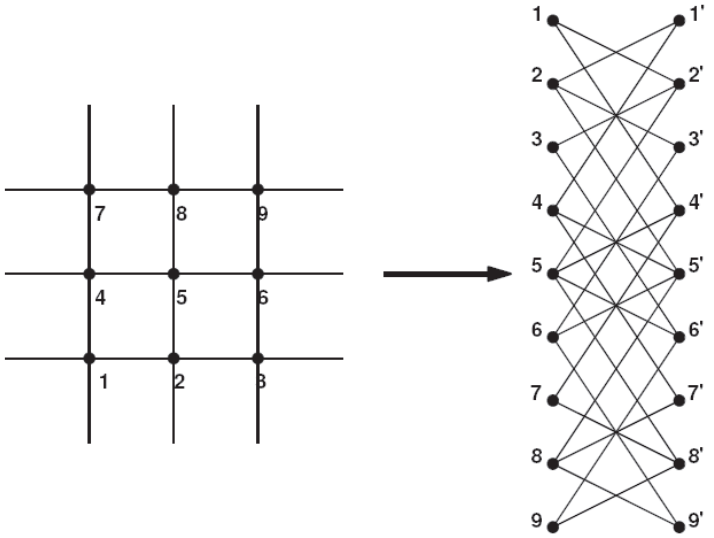

Abbildung 1.2: Illustration des Übergangs zu einem Netzwerkproblem

Man wird zunächst einwenden, dass das Netzwerk ja gar nicht notwendig verbunden ist. Wir werden deshalb zwei Knoten s und t einführen, die jeweils zu allen Knoten ohne Strich bzw. allen mit Strich verbunden sind. Aufgrund der auffälligen Symmetrie bezeichnen wir dann solche Netzwerke als schiefsymmetrische Netzwerke. Die einzelnen Terme in der Summe des Hamiltonians können wir offenbar mit den einzelnen Kanten assoziieren und wir werden sie als „Kosten" auf den Kanten bezeichnen.
Das Ziel soll nun sein die obig dargestellte Netzwerkstruktur zu nutzen, um $H(x)$ zu minimieren.
Bevor wir nun allerdings die bereits benutzten Begrifflichkeiten wie Knoten und Kanten korrekt einführen, wollen wir noch ein Matching-Problem vorstellen, welches wir mit unseren Algorithmen für schiefsymmetrische Netzwerke lösen können.

Kapitel 2

Optimales Rudern

Ein Lehrer steht vor seiner Klasse mit 20 Schülern. Schon einige Male ist die Klasse zum Rudern gefahren und hat dabei das Rudern im Zweier geübt (also ein Ruder backbord und ein Ruder steuerbord). Heute will er von den Schülern beim Rudern Zeiten nehmen. Damit sich keine Boote gegenseitig behindern, will er nur ein Boot auf der Strecke fahren lassen. Der Lehrer glaubt, dass innerhalb der 2 Unterrichtsstunden 8 Zeitfahrten möglich sind.
Bei der Einteilung der Paare für die Boote muss der Lehrer zudem darauf achten, dass Schüler zusammen rudern, deren Rhythmus in etwa gleich ist. Wer schon einmal das Riemenrudern im Zweier versucht hat, wird wissen, wie schwer es ist, einen gleichen Schlagrhythmus zu finden, um das Boot geradeaus zu steuern.
Der Lehrer will nun natürlich von möglichst vielen Schülern eine Zeitnahme machen. Es soll also möglichst kein Schüler mehrmals rudern. Dies kann aber vorkommen, weil z.B. 5 Schüler den gleichen Schlagrhythmus haben und mit keinem anderen rudern können. Wählen wir also ein Paar von Schülern aus, so können wir die Schüler damit für ein weiteres Zeitfahren nicht automatisch ausschliessen.
Wie kann der Lehrer eine möglichst gute Paarung von Schülern finden?
Auch hier hilft uns die Graphentheorie. Versuchen wir das Problem zunächst in einem Graphen zu visualisieren. Dabei identifizieren wir jeden Schüler mit einem Knoten des Graphen. Jedes Paar von Schülern mit in etwa gleichem Schlagrhythmus verbinden wir nun durch eine Kante. Wir erhalten hierbei z.B. folgenden Graphen (Abb. 2.1). Jede der Kanten steht dabei für ein mögliches Paar von Ruderern. Damit entspricht jeder Paarung von Schülern ein Untergraph des in (Abb. 2.1) dargestellten Graphen. Hierbei besteht der Untergraph gerade aus den der Paarung entsprechenden Kanten sowie den Anfangs- und Endpunkten dieser Kanten. Ein, allerdings nicht optimales, Beispiel für eine Paarung ist in (Abb. 2.2) angegeben. Allgemein bezeichnet man Probleme, bei denen eine optimale Paarung auf einem Graphen bestimmt werden soll nach dem englischen Wort für „zusammenführen“ (to match) als Matching-Probleme.
Die Abbildung (Abb. 2.1) sieht offensichtlich ziemlich wirr aus. Man erkennt dabei Knoten, die nur mit einem anderen Knoten verbunden sind, also Schüler, die einen sehr speziellen Ruderstil haben und andere, die mit 3 oder 4 Knoten verbunden sind. Es erscheint allerdings praktisch unmöglich eine auch nur ansatzweise optimale Paarung durch „geschultes Auge zu sehen“. Gleichzeitig gibt es exponentiell viele Möglichkeiten, wie man

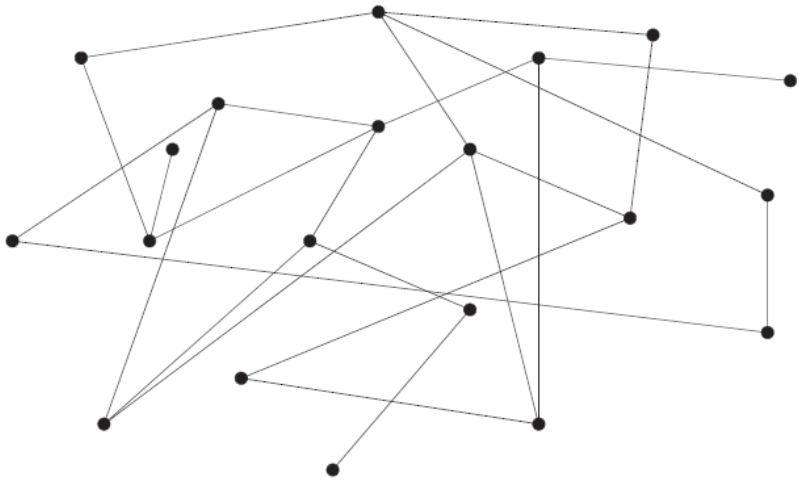

Abbildung 2.1: Graph, der alle möglichen Ruderpaarungen für 20 Schüler enthält

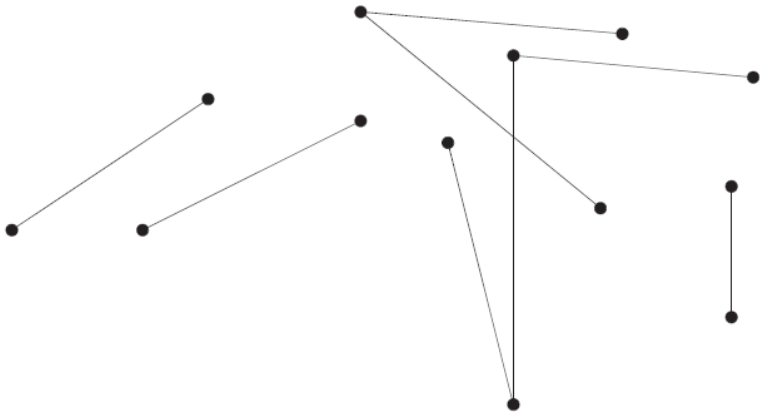

Abbildung 2.2: Untergraph, der eine mögliche Paarung der Schüler repräsentiert

die Paarungen vornehmen könnte, sodass auch der Einzelvergleich aller Möglichkeiten nach einem bestimmten Schema wenig sinnhaft erscheint.
Die Suche nach effizienten Lösungsalgorithmen scheint also selbst bei derart kleinen Problemen dringend geboten.
Wir werden später sehen, dass es sich bei diesem Matching-Problem um ein sogenanntes Minsquare-Problem handelt, das wir ebenfalls mit Methoden auf schiefsymmetrischen Netzwerken lösen können.

Kapitel 3

Schiefsymmetrische Netzwerke

Nachdem wir nun zunächst etwas salopp den Begriff des schiefsymmetrischen Netzwerks eingeführt haben, wollen wir in diesem Abschnitt einige grundlegende Charakteristika schiefsymmetrischer Netzwerke festhalten und auch die Verbindung zu den bereits erwähnten Matching Problemen andeuten.

3.1 Schiefsymmetrisches und balanciertes Netzwerk

In der Literatur gibt es sowohl den Begriff des schiefsymmetrischen Netzwerks (z.B. [30]), als auch den Begriff des balancierten Netzwerks (z.B. [22]), die beide praktisch synonym gebraucht werden. Wir wollen hier den Begriff des schiefsymmetrischen Netzwerks gebrauchen, da sich der Begriff des „balanced network“ bereits in der Physik der Neuronen [61], der Informatik [44] und der Elektrotechnik [16] eingebürgert zu haben scheint. Auch in der Wirtschaftswissenschaft scheint die „balanced scorecard“ zum Gebrauch dieses Terminus anzuregen (z.B. bei der „Balanced Network Systems GmbH“).
Es soll dabei aber nicht verkannt werden, dass Kocay und Stone den Begriff des balancierten Netzwerks erstmals eingeführt haben [41] und auch die maßgebliche Arbeit von Jungnickel und Fremuth-Paeger [22] den Begriff des balancierten Netzwerks verwendet. Auf die Arbeiten von Kocay und Stone sowie Jungnickel und Fremuth-Paeger wollen wir auch unsere Einführung hier abstützen.

3.2 Definition schiefsymmetrischer Netzwerke

Wir müssen zunächst ein wenig grundlegende Notation einführen. Ein *Graph* $G = (V, E)$ ist ein Paar von disjunkten, endlichen Mengen, wobei die Elemente von E die *Kanten* und die Elemente von V die *Knoten* repräsentieren. E ist hierbei mit $p_{ij} \in \mathbb{N} \cup \{0\}$ von der Form:

$$E = \{(ij)^k | i, j \in V; k \leq p_{ij}, k \in \mathbb{N}\} \tag{3.1}$$

p_{ij} gibt also gerade die Anzahl der Kanten zwischen i und j an. Hierbei orientiert sich die Notation für Kanten an der im Buch von Ahuja und Orlin [3]. Diese Notation wird meist

im Kontext von Min Cost Flow Algorithmen verwendet, weshalb sie hier abweichend von der Notation in [22] verwandt werden soll.
Ein Knoten i heißt mit einer Kante *inzident*, wenn i Start- oder Endknoten der Kante ist. Gilt für 2 Knoten $i, j \in V$, dass $(ij)^k \in E$ für mindestens ein $k \in \{1, \cdots, p_{ij}\}$, so heißen sie *benachbart* oder *adjazent*.
Wir wollen im Folgenden stets einfache Graphen betrachten. Hierbei gilt $p_{ij} \leq 1 \; \forall i, j \in V$.

Definition 3.1 (einfach)**.** *Ein Graph heißt einfach, wenn es zwischen 2 Knoten nicht mehr als eine Kante gibt.*

In einem einfachen Graph bezeichnen wir daher die Kanten auch mit (ij) für $i, j \in V$ statt mit $(ij)^1$.
Als Beispiel betrachten wir folgenden Graphen (Abb. 3.1). Die Buchstaben B, H, S und M mögen dabei stellvertretend für Bremen, Hagen, Siegen und Mannheim stehen. Damit gilt in diesem Beispiel $V = \{B, H, S, M\}$ und $E = \{(BS), (BH), (BM), (HM), (SM)\}$.

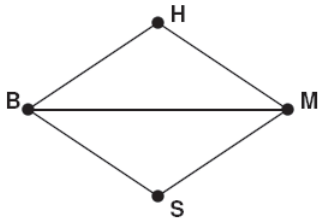

Abbildung 3.1: Beispiel für einen einfachen Graphen

Auf solchen Graphen können wir nun Spaziergänge, Wege und Pfade definieren: Ein *Spaziergang* auf G ist ein nichtleerer Teilgraph $S = (V, W)$ von G der Form $W = \{(i_1 i_2), (i_2 i_3) \cdots, (i_{k-1} i_k)\}$. Sind auf einem Spaziergang S die Kanten $(i_l, i_{l+1}) \in W$ paarweise verschieden, so bezeichnen wir den Spaziergang als *Weg*. Sind hierbei sogar die i_l, $l \in \{1, \cdots, k\}$ paarweise verschieden, so nennen wir den Spaziergang S einen *Pfad*. Ein Pfad kann also keinen Knoten doppelt enthalten. Die Länge eines Pfades ist damit auf die Anzahl n von Knoten im Netzwerk beschränkt [35, S. 5].
Wir bezeichnen Spaziergänge als *geschlossen*, falls $i_1 = i_k$ gilt. Das Analogon für Pfade nennen wir einen Kreis. Ein *Kreis* ist hierbei ein Teilgraph $C = (V, W)$, so dass die Menge W wieder die Eigenschaft $i_1 = i_k$ hat und die i_l, $l \in \{1, \cdots, k-1\}$ wieder paarweise verschieden sind. Es wiederholt sich also nur der Knoten $i_1 = i_k$ [43, S. 16]. Man konstruiert das schiefsymmetrische Netzwerk zu einem einfachen Graphen, um Matchingprobleme auf diesem Graphen zu lösen. Bevor wir nun allerdings das Netzwerk konstruieren können, müssen wir den Begriff des Netzwerks korrekt einführen. Der erste Schritt zu einem Netzwerk ist dabei ein sogenannter Digraph [43, S. 26].

Definition 3.2 (Digraph)**.** *Ein gerichteter Graph oder auch Digraph (directed graph) ist ein Paar von disjunkten endlichen Mengen V (Knoten) und E (Kanten), wobei die Elemente von E gerichtete Paare (ij) von Elementen $i, j \in V$, $i \neq j$ sind. Man bezeichnet die Kanten E manchmal auch als Bögen (arcs) um den Unterschied zum ungerichteten Graph zu verdeutlichen.*

Die Sprechweise für Knoten und Kanten bleibt dabei die selbe. Zwei Kanten (ij), (ji) mit $i, j \in V$ heißen *antiparallel*.
Für die Konstruktion eines schiefsymmetrischen Netzwerks benötigen wir nun noch eine Menge V', die disjunkt zu V ist. Ferner existiere eine bijektive Abbildung $\pi : V \to V'$. Wir definieren nun $i' := \pi(i)$ und nennen i' den *komplementären Knoten* zu i. Die Elemente von V nennen wir die *äußeren* und die von V' die *inneren Knoten*. **Bemerkung:** Wenn man will, kann man die Menge V' explizit konstruieren. Eine Idee ist folgende Konstruktion:

$$V' := \{(i, 0) | i \in V\} \tag{3.2}$$

Man kann die Bijektion π dann einfach notieren: $\pi(i) := (i, 0)$.
Neben den genannten Knotenmengen wählen wir nun eine Quelle s (source) und eine Senke t (target), wobei weder s noch t aus V oder V' seien. Mit den Kantenmengen $A_G := \{(ij') : (ij) \in E\}, A_s := \{(sj) : j \in V\}$ und $A_t := \{(i't) : i \in V\}$ definieren wir den *schiefsymmetrischen Digraphen* $D_G := (V \cup V' \cup \{s, t\}, A_G \cup A_s \cup A_t)$.
Die Richtung einer Kante im schiefsymmetrischen Digraphen ist dabei folgendermaßen vorgegeben: Alle Kanten (ij') mit $i \in V$, $j' \in V'$ sind *Vorwärtskanten*. Ferner sind alle Kanten (si), $i \in V$ Vorwärtskanten, genauso wie alle Kanten $(j't)$, $j' \in V'$. Sprechen wir von Kanten (ji'), so meinen wir, dass wir auf der Kante $(ij') \in A_G$ rückwärts gehen und sprechen daher von *Rückwärtskanten*. Für die Rückwärtskante schreiben wir auch $\overline{(ij')} = j'i$. Die Knoten bzw. Kanten eines Digraphen (bzw. später eines Netzwerks) bezeichnen wir mit $V(D_G)$ bzw. $A(D_G)$.
Der schiefsymmetrische Digraph hat hierbei noch eine interessante Eigenschaft. Wir geben zunächst folgende Definition an [43, S. 70]:

Definition 3.3 (bipartit)**.** *Ein Graph $G = (V, E)$ heißt bipartit, falls eine Partition $V = S \biguplus T$ der Knotenmenge V existiert, so dass die Kantenmengen $E|S$ und $E|T$ leer sind. Mit anderen Worten: Es gibt keine Kanten $(ij) \in E$ mit $i, j \in S$ oder $i, j \in T$.*

Mit der Schreibweise $E|S$ meinen wir hierbei alle Kanten mit Anfangs- und Endpunkt in S. Zur Illustration dieser Definition betrachten wir noch den bipartiten Graphen in der (Abb. 3.2). Die Partition der Knotenmenge ist dabei durch durch $S = \{1, 4\}$ und $T = \{2, 3\}$ gegeben.

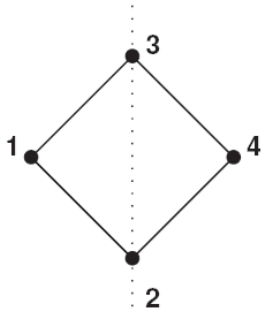

Abbildung 3.2: Beispiel für einen bipartiten Graphen

Damit ist auch offensichtlich, dass ein schiefsymmetrischer Digraph immer bipartit ist.

Die Partition kann man dabei allgemein angeben mit:

$$V \cup V' \cup \{s,t\} = (V \cup \{t\}) \uplus (V' \cup \{s\}) \tag{3.3}$$

Definieren wir nun s als inneren Knoten und t als äußeren Knoten, so gilt, dass jeder Pfad abwechselnd auf inneren und äußeren Knoten verläuft.
Betrachten wir nun ein Beispiel für den Übergang von einem Graphen zum schiefsymmetrischen Digraphen (Abb. 3.3).
In dem Bild wird auch deutlich woher die Namensgebung der Schiefsymmetrie kommt.

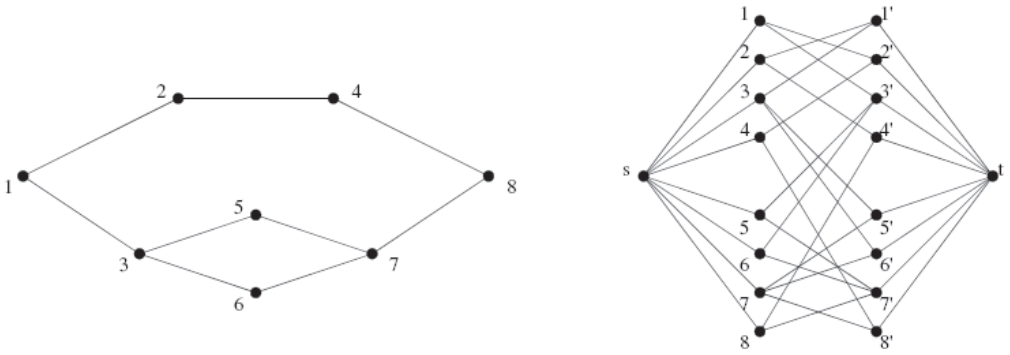

Abbildung 3.3: Beispiel für den Übergang zum schiefsymmetrischen Digraphen

Der Begriff der Komplementarität ist für die Struktur eines schiefsymmetrischen Digraphen offensichtlich von großer Bedeutung. Wir wollen den Begriff deshalb an dieser Stelle auf beliebige Knotenmengen erweitern: Wir definieren $s' := t$ und $j' := i$, wenn mit dieser Definition auch $i' = j$ gilt. Für Mengen gilt $W' := \{i' : i \in W\}$ für eine gegebene Menge $W \subseteq V(D_G)$. Gilt $W = W'$, so nennen wir W eine *selbst-komplementäre Menge*.
Nehmen wir Schleifen (also Kanten der Form (ii)) im Graphen G aus, so werden Kanten im Ausgangsgraphen durch 2 Kanten (ij') und (ji') repräsentiert. Diese Kanten bezeichnet man folglich auch als *komplementäre Kanten*. Eine Schleife wird lediglich durch eine Kante (ii') in D_G beschrieben.
Bisher haben wir nur Digraphen eingeführt. Wir haben den Begriff des Netzwerks zwar benutzt, aber noch nicht definiert. Dies wollen wir nun nachholen [43, S. 63]:

Definition 3.4 (Netzwerk)**.** *Sei $G = (V, E)$ ein Digraph und $v, w : E \to \mathbb{R}$ 2 Funktionen. Wir bezeichnen das Tripel (G, v, w) als Netzwerk und die Funktionen v und w als obere und untere Kapazitätsbeschränkung. $v(ij)$ bzw. $w(ij)$ ist dabei die Kapazitätsbeschränkung der Kante $(ij) \in E$.*

Wir betrachten hier lediglich Matching-Probleme. Hier führt man nur ganzzahlige Kapazitäten $cap : E \to \mathbb{N}$ ein. Wir werden später noch sehen, warum andere Kapazitäten keinen Sinn machen würden.
Netzwerke kann man sich hierbei wie hierarchische Computernetzwerke vorstellen. Wir

haben eine Menge von Servern und eine Menge von Terminals, wobei wir die Server-Terminal Richtung als Vorwärtsrichtung auszeichnen wollen. Die Struktur des Netzwerks wird im Digraphen kodiert und zusätzlich die Bandbreiten der Netzwerkverbindungen in der Kapazitätsbeschränkung.
Wir müssen also auf dem schiefsymmetrischen Digraphen noch Kapazitätsbeschränkungen auf den Kanten definieren, um ein schiefsymmetrisches Netzwerk zu erhalten.
Offensichtlich sollten wir die Kapazitäten nicht irgendwie wählen. Wir haben bereits angesprochen, dass die Motivation zur Untersuchung schiefsymmetrischer Netzwerke Matching-Probleme sind. Es liegt also nahe zunächst Matching-Probleme zu betrachten, um die Kapazitätsbeschränkung dann so zu wählen, dass sie eine einfache Lösung von Matchingproblemen erlaubt. Wir wollen an dieser Stelle den Begriff des Matchings sehr technisch einführen, um die Definition des schiefsymmetrischen Netzwerks zu ermöglichen. Im nächsten Abschnitt werden wir den Begriff des Matchings dann etwas anwendungsorientierter betrachten.
Wir betrachten bei Matching-Problemen nicht mehr allein einen Graphen, sondern sogenannte *Multigraphen* $G := (V, E, c, u)$ mit der oberen Kapazitätsbeschränkung $c : E \to \mathbb{N}$ und der unteren Kapazitätsbeschränkung $u : E \to \mathbb{N}$. Man bezeichnet hierbei $c(ij)$ auch als die *Multiplizität* der Kante (ij). Der Begriff der Multiplizität bezeichnet hier ausdrücklich keine Kosten, sondern vielmehr wie viel Fluss über die Kante geschickt werden darf, bzw. wie oft die Kante verwendet werden darf.
Aus solchen Graphen konstruieren wir schiefsymmetrische Netzwerke, um auf dem Multigraphen Matchingprobleme zu lösen. Es liegt also nahe für die Kanten, die nicht mit s und t inzidieren die oberen und unteren Kapazitätsbeschränkungen der Kanten im Multigraphen in das schiefsymmetrische Netzwerk zu übernehmen.
Kommen wir nun zum Begriff des Matchings auf dem Multigraphen. Jede Abbildung $\phi : E \to \mathbb{N}_0$ mit $u \leq \phi \leq c$ nennen wir einen *Untergraph* von G. Die sogenannte *Gradsequenz* von ϕ in G ist dabei gegeben durch:

$$deg_\phi(i) := \phi((ii)) + \sum_{(ij)\in E} \phi((ij)) \tag{3.4}$$

Es gilt hierbei $\phi((ii)) := 0$, falls keine Schleife (ii) auftritt. Wir bezeichnen nun $deg_\phi(i)$ als den *Grad von i im Untergraphen ϕ*. $deg_\phi(i)$ soll dabei angeben, wie viel Fluss insgesamt aus dem Knoten $i \in V$ herauslaufen darf.
Offenbar wird bei dem Grad eines Knotens eine Schleife doppelt gezählt. In den späteren Algorithmen wird die Bedeutung dieses technischen Details deutlich werden.
Betrachten wir nun den Graphen selbst, so kann man ebenfalls eine *Gradfunktion* $deg : V \to \mathbb{N}$ definieren:

$$deg(i) := c((ii)) + \sum_{(ij)\in E} c((ij)) \tag{3.5}$$

Wir bezeichnen nun allgemein jede Abbildung $\theta : V \to \mathbb{N}_0$ mit $\theta \leq deg$ als *Gradsequenz von G*.
Bevor wir fortfahren, wollen wir uns die Begriffsbildung eines Knotengrades anhand eines Beispiels verdeutlichen. Betrachten wir dazu den Multigraphen G in (Abb. 3.4).

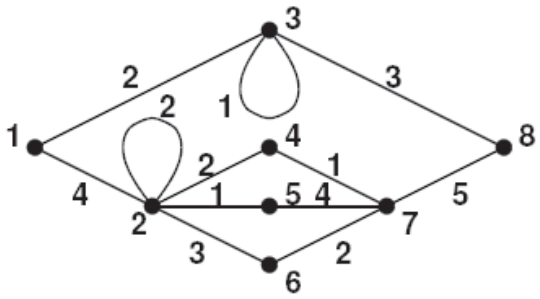

Abbildung 3.4: Beispiel für einen Multigraphen

In der Abbildung sind die Kapazitätsbeschränkungen jeweils an den Kanten angegeben. Betrachten wir den Knoten 2, so gilt:

$$\begin{aligned} deg(2) &= c((22)) + c((24)) + c((25)) + c((26)) + c((22)) + c((21)) && (3.6)\\ &= 2c((22)) + c((24)) + c((25)) + c((26)) + c((21)) && (3.7)\\ &= 4 + 2 + 1 + 3 + 4 = 14 && (3.8) \end{aligned}$$

In Bezug auf Matching-Probleme gibt der Grad eines Knotens $i \in V$ also an, mit vielen anderen Knoten man i verpaaren kann.
Völlig analog kann man den *unteren Grad* eines Knotens einführen:

$$udeg(i) := u((ii)) + \sum_{(ij)\in E} u((ij)) \tag{3.9}$$

Ein Matching-Problem charakterisieren wir nun ebenfalls durch obere und untere Grenzen für die Anzahl der möglichen Verpaarungen eines Knotens. Diese müssen nun aber offensichtlich die Grade und unteren Grade im Graphen respektieren. Wir definieren also zwei Gradsequenzen a und b auf G, wobei a die unteren und b die oberen Grenzen darstelle. Es gilt also $a(i) \leq b(i),\ a(i) \geq udeg(i)$ und $b(i) \leq deg(i)\ \forall i \in V(G)$.
Wir können damit das *Untergraph-Netzwerk* $M := (V, E, c, a, b)$ definieren. Dieses besitzt also nicht nur die Kapazitätsbeschränkung c, sondern zusätzlich noch die Beschränkungen auf den Knoten a und b. Wir bezeichnen damit jeden Untergraphen ϕ von G mit $a \leq deg_\phi \leq b$ als *Matching* von M. $deg_\phi(i)$ für einen Knoten $i \in V$ gibt damit für ein Matching ϕ von G an, mit wie vielen Knoten i verpaart worden ist. Ist G sogar bipartit, so sprechen wir von *Zuweisungen* anstelle von Matchings.
Damit ist nun auch klar, dass die Kapazitäten auf den Kanten si und $j't$ nur jeweils durch die Funktion b gegeben sein können, womit wir nun die Kapazitätsbeschränkung auf dem Netzwerk notieren können:

$$cap(ij') := c((ij))\ \forall\ \text{Kanten}\ \ (ij) \in E, i \neq j \tag{3.10}$$

$$cap(ii') := 2c((ii))\ \forall\ \text{Schleifen}\ \ (ii) \in E \tag{3.11}$$

$$cap(si),\ cap(i't) := b(i)\ \forall\ \text{Knoten}\ \ i \in V \tag{3.12}$$

Ganz analog können wir auch die untere Kapazitätsbeschränkung definieren:

$$l(ij') := u((ij))\ \forall\ \text{Kanten}\ \ (ij) \in E, i \neq j \tag{3.13}$$

$$l(ii') := 2u((ii)) \; \forall \text{ Schleifen } (ii) \in E \tag{3.14}$$

$$l(si),\ l(i't) := a(i) \; \forall \text{ Knoten } i \in V \tag{3.15}$$

Wir bezeichnen nun das Netzwerk $N_M := (D_G, cap, l)$ als das schiefsymmetrische Netzwerk assoziiert mit dem durch M beschriebenen Matching-Problem. Wir können also für jedes Matching-Problem ein schiefsymmetrisches Netzwerk konstruieren.
Die Form schiefsymmetrischer Netzwerke bleibt dabei immer gleich, so dass es sinnvoll ist allgemein von schiefsymmetrischen Netzwerken zu sprechen, ohne sich auf ein spezifisches, durch M beschriebenes Matching-Problem zu beziehen. Wir verstehen dabei unter einem schiefsymmetrischen Netzwerk folgendes:

Definition 3.5 (Schiefsymmetrisches Netzwerk)**.** *Ein schiefsymmetrisches Netzwerk ist ein Netzwerk mit den folgenden Eigenschaften:*

- *die Knoten von N bestehen aus der Quelle s, der Senke t, einer Menge von Knoten $X := \{x_1, \cdots, x_n\}$ und einer von X disjunkten Menge X' für die eine Bijektion $\pi : V \to V'$ existiert*
- *N enthält das Paar von Kanten (sx_i) und $(x_i't)$ für $1 \leq i \leq n$*
- *die sonstigen Kanten treten paarweise zwischen X und X' auf: entweder (x_ix_j') und (x_jx_i') bzw. $(x_i'x_j)$ und $(x_j'x_i)$*
- *für die Kapazitäten gilt: $cap(sx_i) = cap(x_i't),\ cap(x_ix_j') = cap(x_jx_i') \; \forall$ Kanten (ij)*

Auch für ein schiefsymmetrisches Netzwerk N bezeichnen wir die Kanten des Digraphen wieder mit $A(N)$ und die Knoten mit $V(N)$.
Auf einem Netzwerk betrachten wir nun Flüsse. Bei der Definition eines Flussnetzwerks wird nun auch klar, weshalb wir 2 ausgezeichnete Knoten s und t benötigen [43, S. 154]:

Definition 3.6 (Fluss, Flussnetzwerk)**.** *Sei $N = (G, cap, l)$ ein Netzwerk und s und t 2 Knoten aus $V(N)$. t sei mit s durch Kanten verbunden. Ein Fluss auf N ist dann eine Zuweisung $x : E \to \mathbb{R}$. Ist auf einem Netzwerk N ein Fluss definiert, so bezeichnet man N als Flussnetzwerk.*

Wir bezeichnen hier die Funktionswerte $x(ij)$ analog zur Nomenklatur im Buch von Ahuja et. al. mit x_{ij} [3, S. 5]. Den *Vektor der Flussvektoren* bezeichnen wir wie die Funktion mit x. Ob wir die Funktion oder den Vektor meinen, wird dabei jeweils aus dem Zusammenhang deutlich werden.
Die verschiedenen Funktionen x, l und cap sind jeweils spezifisch zu unterscheiden. Wir wollen daher ihre Bedeutung wieder an einem kleinen Beispiel erläutern. Wir betrachten dazu wieder unseren Beispielgraphen (Abb. 3.5).
Wir haben nun zusätzlich an den Kanten Richtungen und Flusswerte angegeben. Es gilt also $x = (2, 3, 31, 7, 5)$, bzw.

$$x(ij) = \begin{cases} 2, & (ij) = (BH) \\ 3, & (ij) = (BS) \\ 31, & (ij) = (BM) \\ 7, & (ij) = (HM) \\ 5, & (ij) = (SM) \end{cases} \tag{3.16}$$

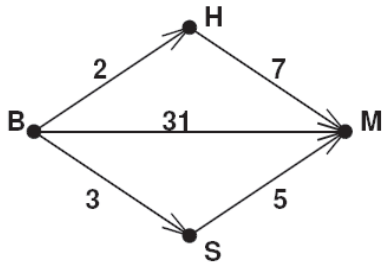

Abbildung 3.5: Ein gerichteter Graph mit Flusswerten

Wir definieren nun zusätzlich die Funktionen *cap* und l, um ein Netzwerk (G, cap, l) zu erhalten:

$$cap(ij) := \begin{cases} 3, & (ij) = (BH) \\ 6, & (ij) = (BS) \\ 32, & (ij) = (BM) \\ 10, & (ij) = (HM) \\ 7, & (ij) = (SM) \end{cases} \qquad l(ij) = \begin{cases} 0, & (ij) = (BH) \\ 2, & (ij) = (BS) \\ 10, & (ij) = (BM) \\ 1, & (ij) = (HM) \\ 3, & (ij) = (SM) \end{cases} \tag{3.17}$$

Das Netzwerk (G, cap, l) ist dann ein Flussnetzwerk, da wir auf dem Netzwerk einen Fluss x definiert haben.
Wir definieren nun auf den Knoten eines Flussnetzwerkes noch Überschüsse und Defizite. Man schreibt dazu $e(i) := \sum_{(ji)\in A(N)} x_{ji} - \sum_{(ij)\in A(N)} x_{ij}$ und bezeichnet diesen Wert als den *Überschuss* (excess) auf einem Knoten i, wobei natürlich $e(i)$ auch negative Werte annehmen kann (Defizit).
Es kann in einem Netzwerk sinnvoll sein, zu fordern, dass gewisse Knoten einen bestimmten Überschuss bzw. Defizit aufweisen sollen. Z.B. sollte eine Quelle einen Überschuss aufweisen. Ferner sehen wir in unserem Netzwerk-Beispiel, dass der Fluss wichtige Eigenschaft $l(ij) \leq x_{ij} \leq cap(ij)$ besitzt. Wir geben hier dieser wichtigen Eigenschaft noch einen Namen:

Definition 3.7 (zulässig)**.** *Wir bezeichnen einen Fluss auf einem Netzwerk als zulässig, falls er die Kapazitätsgrenzen des Netzwerks respektiert ($l(ij) \leq x_{ij} \leq cap(ij)$) und die Überschüsse bzw. Defizite, die wir auf den Knoten definiert haben, die vorgegebenen Werte haben.*

3.3 Verbindung zu Matchings

Die Motivation zur Einführung schiefsymmetrischer Netzwerke bei Kocay und Stone bestand in der einfachen Lösbarkeit von Matching Problemen auf schiefsymmetrischen Netzwerken.
Hierbei nutzen wir aus, dass das schiefsymmetrische Netzwerk eine bipartite Struktur hat. Für bipartite Matchingprobleme basieren die schnellsten bekannten Algorithmen auf Flussalgorithmen [43, S. 207].
Der von uns eingeführte Matching-Begriff ist aber sehr technisch. Wir wollen daher an

dieser Stelle zunächst eine weniger technische Definition eines Matchings angeben [35, S. 213]:

Definition 3.8 (Matching)**.** *Sei $G = (V, E)$ ein Graph. Ein Graph $M = (V, E_M)$ mit $E_M \subseteq E$ heißt ein Matching, wenn jeder Knoten in V mit höchstens einer Kante von M inzident ist. Es gilt also* $\deg_M(i) \leq 1 \; \forall i \in V$ *mit der Gradfunktion deg_M auf M. Gilt $(ij) \in E_M$, so sagen wir auch i ist gematcht zu j.*

Es sei hier darauf hingewiesen, dass der zuvor angegebene Matchingbegriff allgemeiner ist, da nur $a \leq deg_M \leq b$ verlangt wurde. Dies beinhaltet auch sogenannte Faktor-Probleme, die wir später noch ansprechen werden. Die obige Definition ist für die Anschauung an dieser Stelle aber weit brauchbarer.
Um die vorherige Definition eines Matchings zu erläutern und die Verbindung der beiden Definitionen zu zeigen, wollen wir nun an ein typisches Beispiel für ein Matching-Problem behandeln. Nehmen wir dazu einen Tierpfleger für Schwarzstörche. Dieser habe einige männliche und weibliche Störche zur Verpaarung bekommen. Da Störche Paare bilden, kann ein männlicher Storch immer nur mit einem weiblichen Storch zusammengebracht werden. Daneben soll der genetische Pool nicht verkleinert werden, weshalb nur Paare, die genetisch nicht eng verwandt sind, zusammengebracht werden sollen.
Übertragen wir dies auf ein Matching Problem, so gilt nach obiger Konvention für M für die obere Kapazitätsgrenze $b(i) = 1$, da jeder Storch verpaart werden kann, allerdings nicht mehrmals. Man muss aber einen Storch nicht verpaaren, also gilt $a(i) = 0$. Auf Kanten mit Endknoten, die ungleichgeschlechtlichen Störchen entsprechen, gilt $c(ij) = 1$, da man die Störche verpaaren kann. Man muss die Störche allerdings nicht verpaaren, sodass $u(ij) = 0$ für alle Kanten gilt. Ein Beispiel hierzu sehen wir in (Abb. 3.6). Wir haben hierbei den Störchen Namen gegeben, wobei die Buchstaben dem jeweiligen Namen entsprechen sollen.

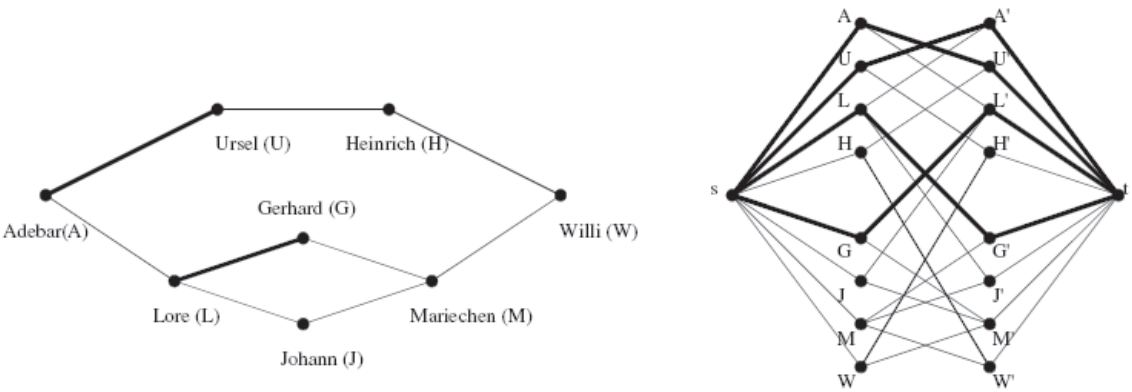

Abbildung 3.6: Beispiel für Matching mit Störchen

In unserem Beispiel gilt für alle Kanten bis auf (HW) für die Multiplizität $c(ij) = 1$. Für (HW) gilt $c(HW) = 0$.

In dem Beispiel ist außerdem bereits durch dicke Kanten ein mögliches Matching eingezeichnet. Wir sehen sofort, dass wir Johann und Mariechen noch zu einem Paar machen könnten. Egal wie wir aber versuchen dann weiter zu verbessern, werden wir nie mehr als 3 Paare bilden können. Versuchen wir möglichst viele Paare zu bilden haben wir ein sogenanntes *Maximal Matching Problem*.
Bei einem Max Matching Problem möchten wir ein Matching größtmöglicher Kardinalität finden.
Hierbei definieren wir die *Kardinalität eines Untergraphen* von G durch:

$$|\phi| := \frac{1}{2} \sum_{i \in V} deg_\phi(i) \tag{3.18}$$

Diese Definition ist sinnvoll, wenn man sich daran erinnert, dass jede Kante bei der Gradsequenz des Multigraphen genau 2 Mal gezählt wird und folglich gilt $\sum_{i \in V} deg(i) = 2|E|$. $|\phi|$ wird somit mit der Anzahl der Kanten in ϕ identifiziert. Wir können diese Notation aber auch für *nicht-ganzzahlige Matchings* verwenden (also Abbildungen $\phi : E \to \mathbb{R}$ mit $0 \leq \phi \leq c$ und $a \leq deg_\phi \leq b$).
Wir interessieren uns nun für Flüsse auf dem schiefsymmetrischen Netzwerk, die das Bild eines (evtl. nicht-ganzzahligen) Matchings ϕ von M unter folgender Abbildung sind:

$$x_{ij'} := \phi((ij)) \; \forall \text{ Kanten } \; (ij) \in E \tag{3.19}$$

$$x_{ii'} := 2\phi((ii)) \; \forall \text{ Schleifen } \; (i,i) \in E \tag{3.20}$$

$$x_{si}, \; x_{i't} := deg_\phi(i) \; \forall \text{ Knoten } \; i \in V \tag{3.21}$$

Dies ist die natürliche Einbettung von Matchings in die Flüsse auf dem Netzwerk. Wir sagen daher, dass ϕ zu einem Fluss x *korrespondiert*. Hierbei müssen wir Rückwärtskanten nicht behandeln, da mit unserer Interpretation von Rückwärtskanten eine Flusserhöhung auf einer Rückwärtskante einer Flussverringerung auf der Vorwärtskante entspricht. Insbesondere ist die „Kapazität“ der Rückwärtskante stets $cap(j'i) = x_{ij'}$, so dass ein positiver Fluss auf der Rückwärtskante nicht zulässig ist. Zulässige Flüsse sind damit immer positiv.
Wir wollen nun anhand unseres Beispiels in (Abb. 3.6) einmal explizit nachvollziehen, wie das Matching auf der linken Seite zu dem Fluss im schiefsymmetrischen Netzwerk auf der rechten Seite korrespondiert.
Hierzu geben wir zunächst noch einmal die Kapazitäten im Multigraphen an, wie wir sie obig herausgearbeitet haben:

$$c(ij) = 1 \; \forall (ij) \in E \backslash \{(HW)\} \tag{3.22}$$

$$c(ij) = 0 \Leftrightarrow (ij) = (HW) \tag{3.23}$$

$$u(ij) = 0 \; \forall (ij) \in E \tag{3.24}$$

Konstruiert man nun den schiefsymmetrischen Digraphen D_G aus dem Multigraphen auf der linken Seite, so erhält man die rechte Seite der Abbildung. Die Vorwärtsrichtung der Kanten geht dabei stets von links nach rechts.
Für die Kapazitäten des schiefsymmetrischen Netzwerks gilt:

$$cap(ij') = c(ij) = 1 \; \forall (ij') \in A(D_G) \backslash \{(HW'), (WH')\} \tag{3.25}$$

$$
\begin{aligned}
cap(HW') &= cap(WH') = c(HW) = 0 && (3.26)\\
cap(si) &= b(i) = 1,\ cap(i't) = b(i) = 1\ \forall i \in V(D_G) && (3.27)\\
l(ij') &= u(ij) = 0\ \forall (ij') \in A(D_G) && (3.28)\\
l(si) &= a(i),\ l(i't) = a(i) = 0\ \forall i \in V(D_G) && (3.29)
\end{aligned}
$$

Schleifen tauchen in dem hier betrachteten Multigraphen nicht auf.
Wir müssen nun noch den, zu dem Matching korrespondierenden, Fluss konstruieren. Nach unserer vorhergehenden Diskussion gibt $\phi((ij))$ für eine Kante $(ij) \in E$ an, wie oft der Knoten i zum Knoten j gematcht ist. Es gilt also:

$$
\begin{aligned}
\phi((ij)) &= 1,\ (ij) \in \{(AU), (GL)\} && (3.30)\\
\phi((ij)) &= 0,\ (ij) \in \{(UH), (HW), (AL), (LJ), (JM), (GM), (MW)\} && (3.31)
\end{aligned}
$$

Wir können uns daraus die Knotengrade direkt berechnen:

$$
\begin{aligned}
deg_\phi(i) &= 1, i \in \{A, U, L, G\} && (3.32)\\
deg_\phi(i) &= 0, i \in \{H, W, M, J\} && (3.33)
\end{aligned}
$$

Mit diesen Informationen können wir uns nun den Fluss berechnen, der dem dargestellten Matching entspricht:

$$
\begin{aligned}
x_{ij'} &= 1,\ (ij') \in \{(AU'), (UA'), (LG'), (GL')\} && (3.34)\\
x_{ij'} &= 0,\ (ij') \in \{(AL'), (LA'), (UH'), (HU'), (HW'),\\
&\quad (WH'), (LJ'), (JL'), (JM'), (MJ'), (GM'), (MG'), (MW'), (WM')\} && (3.35)\\
x_{si} &= 1,\ i \in \{A, U, L, G\} && (3.36)\\
x_{si} &= 0,\ i \in \{H, W, M, J\} && (3.37)\\
x_{i't} &= 1,\ i' \in \{A', U', L', G'\} && (3.38)\\
x_{i't} &= 0,\ i' \in \{H', W', M', J'\} && (3.39)
\end{aligned}
$$

Dieser Fluss ist in (Abb. 3.6) auf der rechten Seite ebenfalls mit dicken Strichen angedeutet. Das Matching kann man dabei einfach „sehen“.
Für alle Knoten $i \neq s, t$ des Netzwerks gilt mit dieser Einbettung von Matchings in Flüsse auf dem Netzwerk $e(i) = 0$. Wir bezeichnen daher einen solchen Fluss x auch als *st-Fluss* und $val(x) = e(s) = -e(t) = 2|\phi|$ als den *Flusswert* (flow value). Ein *maximaler Fluss* ist ein *st*-Fluss mit maximalem Flusswert.
Allerdings korrespondieren maximale ganzzahlige Matchings von M nicht immer mit maximalen Flüssen x auf N_M und nicht jeder Fluss x kann in ein ganzzahliges Matching ϕ mit $2|\phi| = val(x)$ transformiert werden. Wir benötigen dazu noch folgende Anforderungen:

- $x_{ij'} = x_{ji'}\ \forall$ Kanten $(ij') \in A(N_M)$
- $x_{ij'}$ ist ganzzahlig für alle Kanten $(ij') \in A(N_M)$
- $x_{ii'}$ ist gerade für alle Schleifen $(ii') \in A(N_M)$

Jeden Fluss x auf N_M, der diese Anforderungen erfüllt nennen wir *balanciert*.
Diese Bijektion zwsichen Matchings auf dem Multigraphen und balancierten Flüssen auf dem schiefsymmetrischen Netzwerk ist dabei offensichtlich. Wäre der Fluss nicht ganzzahlig und $x_{ii'}$ für Schleifen nicht gerade, so könnten wir kein ganzzahliges Matching bestimmen. Würde $x_{ij'} \neq x_{ji'}$ gelten, so wäre Knoten i zu j, aber eventuell j nicht zu i gematcht. Für jeden balancierten Fluss können wir das Matching ϕ aus der obigen bijektiven Abbildung zwischen Matchings und Flüssen auf dem schiefsymmetrischen Netzwerken direkt bestimmen.
Da alle balancierten Flüsse ganzzahlig sein müssen, wird damit auch deutlich, weshalb nicht-ganzzahlige Kapazitäten nicht betrachtet werden müssen.
Damit haben wir nun zwar herausgearbeitet, wie Flüsse und Matchings auf schiefsymmetrischen Netzwerken zusammenhängen, aber noch keine Lösung des Max Matching Problems beschrieben. Dies ist nun aber ganz einfach. Wir weisen den Kanten von V nach V' und den si und $i't$-Kanten im Netzwerk die Kapazität 1 zu und bestimmen den maximalen balancierten Fluss (Max Bal Flow). Es wird jeder Knoten i mit maximal einem Knoten j' verbunden (und genauso j mit i'). Da der Fluss maximal sein soll, werden möglichst viele Knoten „gematcht“ und man erhält die Lösung des Max Matching Problems (genauer in [22, S. 41]).
Ganz analog können wir uns auch das Faktor-Problem stellen: Gegeben sei eine Funktion $b : V \to \mathbb{N}$. Hat G dann einen Untergraph H, so dass $deg_H(i) = b(i)\ \forall\ i \in V$?
Auch dieses Problem lässt sich auf dem schiefsymmetrischen Netzwerk einfach lösen. Auf den Kanten zwischen V und V' setzen wir die Kapazitäten wieder auf 1 und setzen auf den Kanten (si) und $(i't)\ \forall\ i \in V$ die Kapazitäten auf $cap(si) = cap(i't) = b(i)$.
Wir berechnen nun wieder einen maximalen Fluss x auf N. Dann kann der erwähnte Untergraph H nur existieren, wenn für den Flusswert von x gilt: $val(x) = \sum_{i \in V} b(i) = \sum_{i' \in V'} b(i')$.
Wir können somit schiefsymmetrische Netzwerke verwenden, um Matchingprobleme mit Netzwerkflussalgorithmen zu behandeln. Diese Möglichkeit haben wir, da das behandelte Netzwerk, wie auch in den Illustrationen deutlich wird, bipartit ist. Wir können also die Netzwerkflussmethoden, die man zur Lösung von Matchingproblemen auf bipartiten Graphen benutzen kann, durch schiefsymmetrische Netzwerke auf allgemeinere Graphen anwenden.
Betrachtet man die Darstellung in [35, S. 364-367] für das bipartite Matchingproblem, so wird die Analogie zu den hier gemachten Betrachtungen sofort deutlich.
In der Tat ist der dortig präsentierte Ansatz zur Lösung des nicht-bipartiten Matchingsproblems konzeptionell fast identisch zu unserem Ansatz schiefsymmetrischer Netzwerke. Leider werden wir später auch auf die gleichen Probleme stoßen.
Wir sehen bereits, dass das erste interessante Problem Max Bal Flow ist. Wir werden zunächst die Methoden zur Lösung dieses Problems vorstellen und danach auf das schwierigere Balanced Minimum Cost Flow (Bal Min Cost Flow) und Convex Bal Min Cost Flow Problem eingehen, das wir anfangs in unserem physikalischen Hors d'œuvre vorfinden.

Kapitel 4

Maximum Balanced Flow

Im vorhergehenden Abschnitt haben wir bereits darauf hingewiesen, dass Max Bal Flow zur Lösung von Matchingproblemen verwandt werden kann. Das Maximum Balanced Flow Problem wird ausführlich in den Arbeiten von Jungnickel und Fremuth-Paeger behandelt. Wir wollen daher an dieser Stelle nur einen Einblick in die Thematik bieten. Dieser ist aber durchaus sinnvoll, da die Probleme Bal Min Cost Flow und Max Bal Flow eng verwandt sind und wir in diesem Kapitel einige auch für Bal Min Cost Flow wichtige Sätze beweisen wollen. Daneben benötigen wir teilweise die mit Max Bal Flow berechnete maximale Flussrate als Eingangsgröße für unsere Algorithmen, sodass wir zumindest die Grundlagen zur Lösung des Problems darstellen wollen.
Es gibt 2 grundlegende Ideen für Max Bal Flow. Beide versuchen auf sogenannten erweiternden Wegen den Fluss zu erhöhen, d.h. man sucht Wege von s nach t, auf denen noch freie Kapazität ist und erhöht den Fluss. Bei der ersten Idee erhöhen wir den Fluss nur um ganzzahlige Flusseinheiten und bei der 2. Idee erhöhen wir auch um nicht-ganzzahlige Flusseinheiten. Dies müssen wir dann natürlich im Nachhinein noch reparieren.
Wir wollen hier vor allem auf die erste Idee, die sogenannten Balanced Augmentation, eingehen und werden die 2. Idee hier nur kurz erwähnen.

4.1 Maximum Flow

Bevor wir allerdings auf die Bestimmung eines Maximum Balanced Flows eingehen, wollen wir kurz das Maximum Flow Problem auf einem schiefsymmetrischen Netzwerk betrachten. Hier lässt sich die grundlegende Idee der Flusserhöhung (Augmentation) sehr einfach erläutern. Außerdem wollen wir hier auf die Verwandtschaft mit dem Problem der Bestimmung eines Schnitts minimaler Kapazität in einem Netzwerk eingehen.
Hierbei ist ein *st-Schnitt* oder einfach *Schnitt* eines Netzwerks N eine Zweiteilung $V(N) = S \biguplus T$ mit $s \in S,\ t \in T$. Die Kapazität eines solchen Schnitts definieren wir durch:

$$cap(S,T) := \sum \{cap(ij)\ :\ (ij) \in A(N),\ i \in S,\ j \in T\} \tag{4.1}$$

Um eine bessere Vorstellung eines Schnitts zu entwickeln betrachten wir das Beispiel in (Abb. 4.1). An den Kanten ist jeweils die Kapazität notiert. Ferner sei $S = \{s, 1, 2, 3\}$ und $T = \{4, 5, 6, t\}$. Es gilt damit für die Kapazität des Schnitts (S, T):

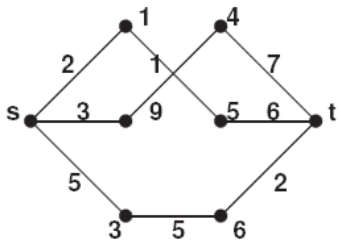

Abbildung 4.1: Netzwerk mit eingezeichneten Kapazitäten zur Illustration eines Schnitts

$$\begin{aligned} cap(S,T) &= cap(15) + cap(24) + cap(36) && (4.2)\\ &= 1 + 9 + 5 = 15 && (4.3) \end{aligned}$$

Wir können nun unsere Notation für Schnitte (S,T) auf beliebige Funktionen $f : A(N) \to \mathbb{R}_0^+$ erweitern:

$$f(S,T) := \sum \{f(ij) \ : \ (ij) \in A(N),\ i \in S,\ j \in T\} \tag{4.4}$$

Auf einem schiefsymmetrichen Netzwerk gilt $e(i) = 0\ \forall i \in V(N)\backslash\{s,t\}$ für einen zulässigen Fluss. Ferner gilt aufgrund unserer Interpretation von Rückwärtskanten $x_{ij} \geq 0,\ x_{ji} \geq 0\ \forall i \in S$. Damit folgt für einen zulässigen Fluss x auf dem schiefsymmetrischen Netzwerk N:

$$\begin{aligned} val(x) &= e(s) = \sum_{i \in S} e(i) && (4.5)\\ &= \sum_{i \in S} \left(\sum_{(ij) \in A(N)} x_{ij} - \sum_{(ji) \in A(N)} x_{ji} \right) && (4.6)\\ &= x(S,T) - x(T,S) \leq cap(S,T) && (4.7) \end{aligned}$$

Damit haben wir eine Beziehung, die an die schwache Dualität aus der linearen Programmierung erinnert. In der Tat handelt es sich hier lediglich um einen Spezialfall dieser schwachen Dualität. Die Frage ist also, ob eventuell ein Schnitt (S^*,T^*) und ein st-Fluss x^* existieren, so dass $val(x^*) = cap(S^*,T^*)$. Dies entspräche dann der starken Dualität aus der linearen Programmierung.
Den Nachweis der starken Dualität für unseren Spezialfall wollen wir daher als nächstes erbringen, indem wir den Algorithmus nach Ford und Fulkerson [21] für Maximum Flow (Max Flow) beschreiben.
Zunächst müssen wir noch einige Begriffe einführen. Wir halten uns hierbei wieder an die Darstellung im Buch von Fremuth Paeger [22, S. 21-23].
Wir benötigen zunächst den Begriff des Restnetzwerks:

Definition 4.1 (Restnetzwerk)**.** *Sei N ein Flussnetzwerk und x ein Fluss auf N. Dann ist die Restkapazität bezüglich x definiert als:*

$$rescap(ij) := \begin{cases} cap(ij) - x_{ij} & \textit{für } (ij) \in A(N) \\ x_{ij} & \textit{für } \overline{(ij)} \in A(N) \\ 0 & \textit{sonst} \end{cases} \tag{4.8}$$

Die Kanten (ij) mit $rescap(ij) > 0$ bilden mit allen Knoten, die mit einer dieser Kanten inzidieren das Restnetzwerk $N(x)$.

Wir hatten bereits erwähnt den Fluss längs erweiternder Pfade erhöhen zu wollen.

Definition 4.2 (erweiternder Pfad)**.** *Ein erweiternder Pfad ist ein st-Pfad im Restnetzwerk $N(x)$.*

Ist p ein erweiternder Pfad, so definieren wir den *elementaren Fluss mit Träger p* als:

$$x_p(ij) := \begin{cases} 1, & (ij) \in p \\ -1, & \overline{(ij)} \in p \\ 0, & \text{sonst} \end{cases} \tag{4.9}$$

Ist x ein Fluss, so ist offensichtlich $x + x_p$ ebenfalls ein Fluss auf N mit $val(x + x_p) = val(x) + 1$. Wir nennen jeden solchen Schritt zur Erhöhung des Flusswertes auf einem Netzwerk einen *Erhöhungsschritt*.
Wir wollen zunächst beweisen, dass ein Fluss maximal ist, wenn kein Erhöhungsschritt der dargestellten Art mehr möglich ist.

Satz 4.3 (Augmenting Path Theorem)**.** *Sei N ein Flussnetzwerk mit ganzzahligen Kapazitäten. Ein Fluss x ist maximal genau dann, wenn kein erweiternder Pfad mehr auf $N(x)$ existiert.*

Beweis. x kann nicht maximal sein, wenn ein erweiternder Pfad existiert, da sonst $val(x + x_p) = val(x) + 1$ gilt.
Wir müssen nun noch die Rückrichtung beweisen. Es existiere also kein erweiternder Pfad. Sei S die Menge von Knoten erreichbar von s und T die Menge von Knoten erreichbar von t. Gilt $i \in S,\ j \in T$ dann gilt wegen der Definition von T damit $rescap(ij) = 0$. Damit haben wir $x(T, S) = 0$ und $x(S, T) = cap(S, T)$ und damit aufgrund der schwachen Dualität die Maximalität von x. □

Betrachten wir nun einmal ein Beispiel für einen Erhöungsschritt. In dem Netzwerk in (Abb. 4.2) sind jeweils an den Kanten die Restkapazitäten notiert. Auf der linken Seite sieht man das Netzwerk selbst und auf der rechten Seite das Restnetzwerk.

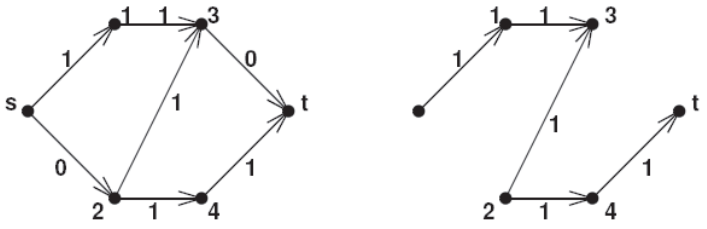

Abbildung 4.2: Netzwerk mit Restkapazitäten und Restnetzwerk

Den erweiternden Pfad p kann man aus der Darstellung des Restnetzwerks direkt ablesen:

$$p = \{(s1), (13), \overline{(23)}, (24), (4t)\} \tag{4.10}$$

Auch den elementaren Fluss mit Träger p wollen wir hier angeben:

$$x_p(ij) = \begin{cases} 1, \ (ij) \in \{(s1), (13), (24), (4t)\} \\ -1, \ (ij) = (23), \ 0, \ sonst \end{cases} \tag{4.11}$$

Erhöhen wir nun den Fluss längs p, so erhalten wir das Restnetzwerk in (Abb. 4.3). Da wir keinen erweiternden Pfad mehr finden können, ist der Fluss $x_{neu} = x + x_p$ nach dem Augmenting Path Theorem maximal, obwohl die Kante (23) noch eine Restkapazität von 2 aufweist.

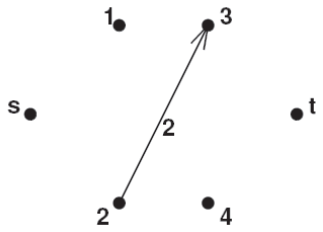

Abbildung 4.3: Restnetzwerk nach der Flusserhöhung

Der Algorithmus zur Lösung von Max Flow ist damit denkbar einfach:

```
Max Flow Algorithm
begin

    repeat

        finde einen erweiternden Pfad p auf N(x)
        erhöhe den Fluss längs p
        berechne das neue Restnetzwerk N(x + x_p)

    until: kein erweiternden Pfad in N(x) mehr zu finden

end;
```

Es bleibt damit natürlich noch das Problem erweiternde Pfade zu bestimmen. Dieses wollen wir hier aber nicht behandeln.
Vielmehr können wir direkt aus der Form des Algorithmus eine wichtige Folgerung für den Fluss ableiten.

Korollar 4.4 (Integrality Theorem)**.** *Auf jedem Flussnetzwerk N mit ganzzahligen Kapazitäten existiert ein ganzzahliger maximaler Fluss.*

Beweis. Wir können unseren Max Flow Algorithmus benutzen und den Beweis per Induktion führen.
$x \equiv 0$ ist ein ganzzahliger Fluss als Anfangsfluss.
Sei nun nach dem i-ten Erhöhungsschritt der Fluss x_i ganzzahlig, dann gilt im nächsten Schritt $val(x_{i+1}) = val(x) + 1$ und damit ist auch x_{i+1} ganzzahlig. Der Algorithmus terminiert also mit einer ganzzahligen Lösung. □

Mit dem Beweis zum Augmentation Path Theorem haben wir aber auch bereits bewiesen, dass es immer einen Schnitt mit den zuvor formulierten Eigenschaften gibt. Auch das wollen wir in einem Korollar festhalten.

Korollar 4.5 (Maximum Flow Minimum Cut Theorem)**.** *Der Flusswert des maximalen Flusses und die Kapazität eines minimalen Schnitts auf einem Netzwerk mit ganzzahligen Kapaztäten sind gleich.*

4.2 Balanced Augmentation

Nach dem Max Flow Problem behandeln wir nun das Problem auf schiefsymmetrischen Netzwerken. Die Idee des Algorithmus bleibt dabei die selbe. Wir können allerdings die Symmetrie noch ein wenig ausnutzen. Ist nämlich p ein beliebiger Pfad von i nach j in $N(x)$ und P die Menge der Kanten von p, dann erhalten wir mit $P' := \{(ij)' : (ij) \in P\}$ offensichtlich einen $j'i'$-Pfad p' auf $N(x)$, den wir als *komplementären Pfad* bezeichnen.
Erhöhen wir den Fluss längs eines st-Pfades p, so ist p' ebenfalls ein st-Pfad. Die Idee ist nun längs p und p' simultan den Fluss zu erhöhen. Dazu müssen wir aber sicher gehen, dass wir längs p und p' den Fluss auch tatsächlich simultan ganzzahlig erhöhen können. Dazu geben wir folgende Definition:

Definition 4.6 (gültig)**.** *Ein st-Pfad heißt gültig, wenn es auf p kein Paar komplementärer Kanten (ij), $(ij)'$ gibt, so dass $rescap(ij) = rescap(j'i') = 1$.*

Wir können also den Fluss längs gültiger, erweiternder Pfade p und p' erhöhen. Der Träger eines Flusses ist dann definiert durch $Supp(x) := \{(ij) \in A(N) : x_{ij} > 0\} \cup \{\overline{(ij)} \in A(N) : x_{ij} < 0\}$.
Auch diese Idee kommt aus der Matchingtheorie. Im bipartiten Matching vergrößern wir die Anzahl der gematchten Kanten auf sogenannten M-erweiternden (M für Matching) Pfaden. Solche Pfade sind alternierend zwischen gematchten und ungematchten Kanten und beginnen und enden jeweils auf einer noch nicht gematchten Kante [45, S. 188].

Definition 4.7 (M-erweiternder Pfad)**.** *Sei $G = (V, E)$ ein Graph und M ein Matching in G. Einen Pfad $\{(i_1, i_2), \cdots, (i_{2k-1}, i_{2k})\}$ nennt man M-erweiternd, wenn $deg_M(i_1) = deg_M(i_{2k}) = 0$ und $(i_{2j}, i_{2j+1}) \in M$ für alle $i = 1, \cdots, k-1$.*

Bevor wir nun auf die wichtige Balanced Decomposition eingehen wollen, müssen wir noch ein technisches Detail erwähnen.
In der Theorie der Netzwerkflüsse werden oft nicht Flüsse im eigentlichen Sinne behandelt (in unserem Fall also st-Flüsse), sondern häufig Zirkulationen. Unter einer *Zirkulation* verstehen wir dabei einen Fluss mit der Eigenschaft $e(i) = 0 \; \forall \; i \in V(N)$.
Wir können aus jedem st-Fluss allerdings ganz einfach eine Zirkulation machen, indem wir die Kante (ts) zum Flussnetzwerk hinzufügen und $x_{ts} = val(x)$ setzen.
Diese Transformation sei in der folgenden (Abb. 4.4) noch einmal erläutert. Auf der linken Seite sehen wir ein Flussnetzwerk, wobei die Flüsse an den Kanten eingetragen sind. Es gilt offensichtlich $e(s) = 3$ und $e(t) = -3$.
Auf der rechten Seite ist das durch Hinzunahme von (ts) entstandene Netzwerk eingezeichnet. In diesem gilt $e(i) = 0$ für alle Knoten i im Netzwerk.

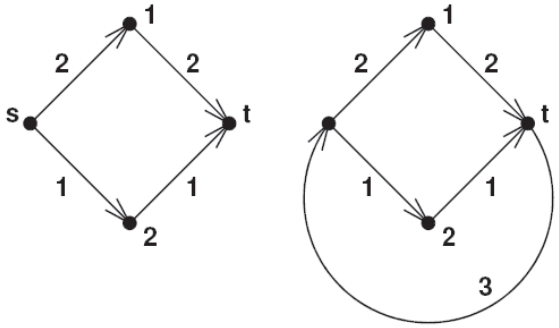

Abbildung 4.4: Illustration der Transformation eines Flusses in eine Zirkulation

Wie gesagt ist dies aber lediglich ein technisches Detail, das manchmal eine etwas schönere Formulierung erlaubt. Wir werden später bei den Algorithmen noch darauf eingehen. Wir kommen nun damit wie angekündigt zur Balanced Decomposition.

Satz 4.8 (Balanced Decomposition)**.** *Seien x und y zwei unterschiedliche balancierte Zirkulationen auf dem schiefsymmetrischen Netzwerk N, dann gibt es gültige Kreise $p_1, \cdots, p_k$ im Restnetzwerk $N(x)$, so dass*

$$y - x = \sum_{i=1}^{k} (x_{p_i} + x_{p_i'}) \tag{4.12}$$

Beweis. Wir setzen $\delta := y - x$ und betrachten die Mengen von Kanten

$$A_\delta^+ := \{(ij) \in A(N) \ : \ \delta(ij) > 0\} \tag{4.13}$$

$$A_\delta^- := \{\overline{(ij)} \ : \ (ij) \in A(N), \delta(ij) < 0\} \tag{4.14}$$

Es gilt $Supp(\delta) = A_\delta^+ \cup A_\delta^-$ und $Supp(\delta)$ ist selbst-komplementär. Man bezeichnet $Supp(\delta)$ auch als den *Träger* der Funktion δ. Ferner ist $\delta(ii')$ gerade, da ja nach Vorraussetzung $x_{ii'}$ und $y_{ii'}$ gerade sind. Setzen wir $\delta(ij) := -\delta(\overline{(ij)}) \ \forall \ (ij) \in A_\delta^-$, so ist $N(x)[Supp(\delta)]$ wieder ein schiefsymmetrisches Netzwerk. Die Schreibweise $N(x)[Supp(\delta)]$ bezeichnet hierbei das Netzwerk, welches aus allen Kanten im $Supp(\delta)$ und allen mit solchen Kanten inzidenten Knoten entsteht. Insbesondere gilt damit $\delta(ij) > 0 \ \forall (ij) \in Supp(\delta)$.
Wir zeigen die Balanced Decomposition nun per Induktion. Hierzu definieren wir zunächst ein Maß für δ:

$$\Pi(\delta) := \sum_{(ij) \in Supp(\delta)} \delta(ij) \tag{4.15}$$

Wir definieren nun zunächst $\delta_0 := \delta$, für die anfängliche Differenz. In jedem Induktionsschritt $i \in \mathbb{N}$ wollen wir nun den Fluss y_{i-1} auf maximalen Pfaden verringern. Für alle $i \in \mathbb{N}$ gilt dann $\Pi(\delta_i) \geq 0$ mit einem ganzzahligen $\Pi(\delta_i)$. Die veränderte Zirkulation y_i hat auf keinem Knoten ein Defizit oder einen Überschuss und es gilt $\Pi(\delta_i) < \Pi(\delta_{i-1})$, sofern nicht $\Pi(\delta_{i-1}) = 0$ gilt.

Wir zeigen zunächst für $\Pi(\delta_0) = \Pi(\delta)$ die Ganzzahligkeit und $\Pi(\delta) \geq 0$. Hierzu betrachten wir zunächst:

$$\Pi(\delta_0) := \sum_{(ij) \in Supp(\delta)} \delta(ij) \tag{4.16}$$

Es gilt $\delta(ij) > 0 \; \forall (ij) \in Supp(\delta)$, also folgt $\Pi(\delta_0) \geq 0$.
$\Pi(\delta_0)$ ist sogar ganzzahlig, da $\delta_0 = x - y$ und die Zirkulationen x und y beide ganzzahlig sind. Ferner hat y als Zirkulation auf keinem Knoten ein Defizit. Damit haben wir den Induktionsanfang.
Für den Induktionsschritt i betrachten wir nun einen Spaziergang p_i auf $N(x)[Supp(\delta_{i-1})]$, so dass p_i und p_i' zusammen jede Kante $(kl) \in Supp(\delta_{i-1})$ maximal $|\delta_{i-1}(kl)|$ mal überqueren. Nehmen wir nun an p_i sei ein maximaler (Anzahl der Kanten) Spaziergang mit dieser Eigenschaft. Wir wollen nun zeigen, dass p_i ein geschlossener Spaziergang ist, wobei wir natürlich die Maximalität ausnutzen.
Zunächst benötigen wir allerdings die Eigenschaft der Zirkulationen x und y_{i-1} auf keinem Knoten einen Überschuss/Defizit zu haben. Dies überträgt sich auf δ_{i-1}, so dass wir eine „Kontinuitätsgleichung“ auf jedem Knoten $k \in V(N)$ haben der Form (k ist hier natürlich fest):

$$\sum_{(kl) \in Supp(\delta_{i-1})} \delta_{i-1}(kl) = \sum_{(lk) \in Supp(\delta_{i-1})} \delta_{i-1}(lk) \tag{4.17}$$

Sei nun m der Anfangsknoten und n der Endknoten von p_i. Für $m \neq n$ gibt es mindestens eine Kante $(kl) \in Supp(\delta_{i-1})$, die noch nicht zusammen $|\delta(kl)|$ mal von p_i und p_i' überquert wurde. Wir können dann den Pfad p_i genau um diese Kante erweitern. Gibt es noch eine Schleife (kk'), die noch nicht $|\delta_{i-1}(kk')|$ mal überquert wurde, können wir den Weg ebenfalls um diese Schleife verlängern. In beiden Fällen haben wir damit den Widerspruch zur Maximalität und p_i ist damit ein geschlossener Spaziergang.
Wir wollen nun von y_{i-1} die von x verschiedenen Anteile subtrahieren. Dazu nutzen wir aus, dass sich p als geschlossener Spaziergang in endlich viele Kreise $(p_1, \cdots, p_h)$ zerlegen lässt, die alle auf $N(x)$ gültig sind. Wir erhalten damit eine neue balancierte Zirkulation mit:

$$y_i := y_{i-1} - \sum_{i=1}^{h} (x_{p_i} + x_{p_i'}) \tag{4.18}$$

y_i ist ganzzahlig, da die x_{p_i} ganzzahlige Flüsse sind und y_{i-1} ganzzahlig war.
Da die p_i's alle Kreise sind, sind auch die p_i''s alle Kreise. Da y_{i-1} gemäß unserer Annahme keinen Knoten mit Defizit hatte, hat y_i damit auch keinen.
Es gilt nun $\delta_i := y_i - x$. Mit der gleichen Konstruktion wie für δ_0 ist $N(x)[Supp(\delta_i)]$ wieder ein schiefsymmetrisches Netzwerk und $\delta_i(ij) \geq 0 \; \forall (ij) \in N(x)[Supp(\delta_i)]$. Damit gilt auch wieder $\Pi(\delta_i) \geq 0$.
Zeigen wir nun noch die letzte Behauptung. Gilt $\Pi(\delta_{i-1}) = 0 = \sum_{(kl) \in Supp(\delta_{i-1})} \delta_{i-1}(kl)$, so folgt wegen $\delta(kl) \geq 0 \; \forall (kl) \in Supp(\delta_{i-1})$ auch $\delta_{i-1}(kl) = 0 \; \forall (kl) \in Supp(\delta_{i-1})$. Damit also $\delta_{i-1} = y_{i-1} - x = 0 \Rightarrow y_{i-1} = x$. Damit muss aber, da wir ja von y_{i-1} nur die von x

verschiedenen Teile abziehen auch $\delta_i = 0$ gelten. Betrachten wir nun den Fall $\Pi(\delta_{i-1}) > 0$. Dann gilt:

$$\begin{aligned}
\Pi(\delta_{i-1}) &= \sum_{(kl)\in Supp(\delta_{i-1})} \delta_{i-1}(kl) && (4.19)\\
&= \sum_{(kl)\in Supp(\delta_i)} \delta_i(kl) + \sum_{i=1}^{h} \left(\sum_{(kl)\in p_i} (\delta_{i-1}(kl) + \delta_{i-1}((kl)')) \right) && (4.20)\\
&= \sum_{(kl)\in Supp(\delta_i)} \delta_i(kl) + 2(|p_1| + \cdots + |p_h|) && (4.21)\\
&= \Pi(\delta_i) + 2(|p_1| + \cdots + |p_h|) && (4.22)\\
&\Rightarrow \Pi(\delta_i) < \Pi(\delta_{i-1}) && (4.23)
\end{aligned}$$

Damit haben wir alle Behauptungen für den Induktionsschritt gezeigt.
Da $\Pi(\delta_i) < \Pi(\delta_{i-1})$, $\Pi(\delta_i) \geq 0$ $\forall i \in \mathbb{N}$ und $\Pi(\delta_i)$ ganzzahlig sein muss, können wir nur endlich viele Updates von y durchführen. Damit haben wir den Beweis des Satzes. □

Für den Algorithmus brauchen wir noch den folgenden Satz, der sich unmittelbar aus der Balanced Decomposition ergibt:

Satz 4.9 (Augmenting Path Theorem)**.** *Sei x ein balancierter st-Fluss auf dem schiefsymmetrischen Flussnetzwerk N. Genau dann ist x ein maximaler balancierter Fluss, wenn es keinen gültigen st-Pfad in $N(x)$ gibt.*

Beweis. „⇒“ Wir nehmen an p sei ein gültiger st-Pfad in $N(x)$. Dann erhält man einen balancierten Fluss y mit $val(y) = val(x) + 2$ mit $y := x + x_p + x_{p'}$. Damit kann x nicht maximal gewesen sein.
„⇐“ Nehmen wir an, x sei nicht maximal und es existiere kein gültiger st-Pfad. y sei aber eine maximale Lösung. Dann fügen wir die Kante ts zum Flussnetzwerk hinzu und wenden die Balanced Flow Decomposition an. Da $y_{ts} > x_{ts}$ existiert ein gültiger Kreis auf $N(x)$, der (ts) überquert. Dieser entspricht ohne die (ts)-Kante einem gültigen st-Pfad im Ausgangsnetzwerk. Damit haben wir den Widerspruch. □

Natürlich hat auch das Augmenting-Path Theorem im Satz von Berge [7] bzw. von Norman und Rabin [51] bzw. von Petersen [54] sein Analogon in der Matchingtheorie.[1] Wir wollen diesen hier ohne Beweis angeben (man findet diesen z.B. in [43, S. 207]).

Satz 4.10 (Satz von Berge)**.** *Sei G ein Graph mit einem Matching M, dann ist M maximal genau dann, wenn es keinen M-erweiternden Pfad mehr gibt.*

Wir haben damit die wichtigsten Sätze für balancierte Flüsse zusammengestellt. Das Problem bei der Bestimmung eines Max Bal Flow lässt sich nach obiger Argumentation auf das Auffinden eines gültigen st-Pfades bzw. auf den Beweis von dessen Nichtexistenz

[1] Die Aufzählung der verschiedenen Urheber bedauert der Autor. Vermutlich hat Schrijver Recht und der Satz muss Petersen zuerkannt werden [63, S. 433]. Gleichzeitig hat sich aber der Name des „Satzes von Berge“ stark eingebürgert.

zurückführen. Wir bezeichnen dieses Problem auch als das *Balanced-Network-Search Problem* (BNS).
Wir wollen im nächsten Abschnitt kurz auf den Algorithmus von Kocay und Stone [42] zur Lösung des Problems eingehen. Dieser Algorithmus hat mit n der Anzahl der Knoten eine Laufzeit $O(n^2)$. Jungnickel und Fremuth-Paeger haben einen verbesserten Algorithmus vorgeschlagen mit einer Laufzeit (mit m der Anzahl der Kanten) von etwa $O(m)$ [22, S. 105].
Haben wir einen solchen BNS-Algorithmus zur Verfügung ist der Algorithmus zur Lösung des Max Bal Flow Problems einfach: Finden wir keinen gültigen erweiternden Pfad, so ist der Fluss maximal, ansonsten finden wir einen gültigen erweiternden Pfad p und berechnen einen neuen Fluss x_{i+1} aus dem Ausgangsfluss x_i mit:

$$x_{i+1} := x_i + balcap(p)(x_p + x_{p'}) \tag{4.24}$$

Hierbei haben wir die *„balanced capacity“* (*balcap*) eingeführt, die für einen Pfad p definiert ist als:

$$balcap(p) := \min_{(ij')\in p} \left\{ \begin{array}{c} \left\lfloor \frac{rescap(ij')}{2} \right\rfloor \text{ , wenn } (ji') \in p \\ rescap(ij) \text{ sonst} \end{array} \right\} \tag{4.25}$$

Diese Definition macht Sinn, wenn man sich noch einmal in Erinnerung ruft, dass für (ij'), $(ji') \in p$ auch (ij'), $(ji') \in p'$ gilt. Erhöhen wir den Fluss längs p und p' auf einer Kante $(ij') \in A(N)$ simultan und ganzzahlig, so dürfen wir in diesem Fall nicht um mehr als $\left\lfloor \frac{rescap(ij')}{2} \right\rfloor$ erhöhen. Sei z.B. $rescap(ij') = 3$, so gilt $\left\lfloor \frac{3}{2} \right\rfloor = 1$. Würden wir längs p und p' um 2 erhöhen, so würde p den Fluss auf (ij') um 2 und p' den Fluss auf (ij') noch einmal um 2 erhöhen. Damit würde $x_{ij'} > cap(ij')$ gelten.
Jeder Übergang zu einem neuen Fluss nach der obigen Formel wird dabei analog zum Max Flow ein *Erhöhungsschritt* genannt.
Den so beschriebenen Algorithmus bezeichnet man als den „Balanced Augmentation Algorithmus“. Wir halten in dem folgenden Korollar noch kurz dessen Eigenschaft fest.

Korollar 4.11 (Balanced Augmentation Algorithmus)**.** *Der Balanced Augmentation Algorithmus liefert für jedes schiefsymmetrische Netzwerk einen Max Bal Flow.*

Auch an dieser Stelle wollen wir die eingeführten Begriffe kurz anhand eines Beispiels rekapitulieren. Wir betrachten dazu das schiefsymmetrische Netzwerk in (Abb. 4.5). An den Kanten sind jeweils die Restkapazitäten notiert. Ein möglicher *st*-Pfad ist:

$$p_1 = \{(s1), (12'), \overline{(32')}, (33'), \overline{(3'2)}, (21'), (1't)\} \tag{4.26}$$

Dieser Pfad ist aber nicht gültig, da $(12')$ und $(21')$ ein Paar komplementärer Kanten mit Restkapazität 1 darstellt. Würden wir den Fluss längs p erhöhen, so könnten wir den Fluss nicht mehr längs

$$p_1' = \{(1't)', (21')', (3'2)', (33')', (2'3)', (12')', (12')', (s1)'\} \tag{4.27}$$

erhöhen, da die Kanten $(21') = (12')$ und $(12')' = (21')$ keine Restkapazität mehr zur Verfügung hätten.
Betrachten wir nun den Pfad:

$$p_2 = \{(s3), (33'), (3't)\} \tag{4.28}$$

Der Pfad ist gültig, da die Kanten alle eine Restkapazität von 2 haben. Wir können aber den Fluss längs p nicht einfach um 2 erhöhen, da wir den Fluss längs p' dann nicht noch einmal um 2 erhöhen könnten. Vielmehr gibt die balancierte Kapazität $balcap(p_2) = 1$ die maximal mögliche Flusserhöhung längs p_2 an.
Die Flusserhöhung längs p_2 und p_2' wäre dann ein Erhöungsschritt. Danach könnten wir keinen gültigen Pfad mehr finden und $x + 1 \cdot (x_{p_2} + x_{p_2'})$ wäre der Max Bal Flow.

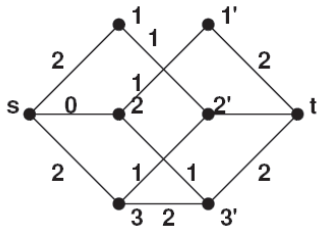

Abbildung 4.5: Schiefsymmetrisches Netzwerk zur Illustration des Balanced Augmentation Algorithmus

Mit dem Balanced Augmentation Algorithmus erhalten wir natürlich auch analog wie in der Argumentation zum Max Flow Algorithmus ein Ergebnis für Schnitte. Wir führen dazu zunächst balancierte Schnitte ein.

Definition 4.12 (balancierter Schnitt)**.** *Wir bezeichnen, analog zu balancierten Flüssen, einen Schnitt (S, T) auf einem balancierten Netzwerk N als balanciert, falls für $(ij') \in (S, T)$ auch $(ji') \in (S, T)$ gilt.*

Wir wollen das folgende Resultat nicht beweisen (der Beweis findet sich z.B. in [41, S. 12]).

Satz 4.13 (Max-Balanced-Flow-Min-Balanced-Cut)**.** *Sei N ein schiefsymmetrisches Netzwerk. Der Flusswert eines Max Bal Flow und die Kapazität eines minimalen balancierten Schnittes sind identisch.*

Auch dieses Ergebnis ist aus der Theorie des bipartiten Matchings bereits bekannt. Das Analogon zum *st*-Schnitt ist hierbei ein Vertex-Cover [35, S. 56-57]:

Definition 4.14 (Vertex-Cover)**.** *Sei $G = (V, E)$ ein Graph und $C \subseteq V$. Dann ist C ein Vertex-Cover, wenn jede Kante $(ij) \in E$ mindestens einen Endknoten in C hat.*

Dem Max-Balanced-Flow-Min-Balanced-Cut Satz entspricht dann der Satz von König [40]:

Satz 4.15 (Satz von König)**.** *Sei $G = (U = V_1 \uplus V_2, E)$ ein bipartiter Graph, dann gilt:*

$$\begin{aligned} & \max\{|M| \mid M \text{ ist ein Matching von } G\} \\ = \; & \min\{|C| \mid C \text{ ist ein Vertex Cover von } G\} \end{aligned} \tag{4.29}$$

4.3 Balanced Network Search

„Always give a bronc a fair shake. Don't pull his head up and don't grab leather. Better yet, don't get on."
— Leslie McKee, Cowboy in Utah (1957) [1]

Wie angekündigt wollen wir nun einen BNS-Algorithmus kurz vorstellen. Wir wählen hier den einfachsten BNS-Algorithmus aus der Arbeit von Kocay und Stone [42].
Das Ziel ist, einen gültigen erweiternden Pfad auf einem schiefsymmetrischen Netzwerk N zu bestimmen. Der Algorithmus von Kocay Stone beruht dabei auf der Breitensuche. Hierbei werden 2 Bäume T und T' sukzessive aufgebaut.
Das Netzwerk wird dabei von s aus durchsucht. Wir definieren dazu folgende Menge:

$$\begin{aligned} ScanQ \;:=\; & \{\text{bisher vom Algorithmus gefundene Knoten, die auf gültigen Pfaden} \\ & \text{von } s \text{ aus erreichbar sind}\} \end{aligned} \tag{4.30}$$

Wir wollen nun die Bäume T und T' iterativ definieren. Beginnt der Algorithmus, so gilt $T = T' = \oslash$. Seien T_{i-1} und T'_{i-1} die Bäume nach der $(i-1)$-ten Iteration des Algorithmus, so gilt mit dem im i-ten Schritt neu gefundenen Knoten j:

$$\begin{aligned} T_i \;=\; & T_{i-1} \cup \big(\{(ij) \mid i \in ScanQ,\ (ij) \in A(N)\} - T'_{i-1}\big) \\ & \cup \big(\{(ji) \mid i \in ScanQ,\ (ji) \in A(N)\} - T'_{i-1}\big) \end{aligned} \tag{4.31}$$

$$T'_i \;=\; \{(ij) \mid (ij)' \in T_i\} \tag{4.32}$$

Mit dieser Definition gilt offensichtlich $T_i \cap T'_i = \oslash$ für jede Iteration des Algorithmus.
Wir wollen den Index i im Folgenden fortfallen lassen und verwenden daher T und T' als Kurzschreibweise für T_i und T'_i.
Der Baum T hat seine Wurzel in s (da anfangs $ScanQ = \{s\}$ gilt) und zu jedem Knoten i, der mit einer Kante in T inzidiert existiert ein gültiger si-Pfad. Der Baum T' ist dann die komplementäre Menge von T. Zu $i' \in T'$ existiert folglich ein gültiger $i't$-Pfad.
Die Gültigkeit der Pfade folgt dabei aus der obigen Definition von T und T': Zu jedem Knoten i, der mit einer Kante in T inzidiert existiert ein si-Pfad. Wäre dieser si-Pfad nicht gültig, müsste mindestens ein Paar komplementärer Kanten in T enthalten sein. Für eine Kante in T ist die komplementäre Kante aber immer in T' enthalten. Damit haben wir den Widerspruch.
Wir definieren nun noch das *Spiegelnetzwerk* M. Der Name ist aus der Arbeit von Kocay und Stone entnommen, ist allerdings etwas irreführend, da es sich lediglich um eine Kantenmenge handelt:

$$M := T \cup T' \tag{4.33}$$

Wir wollen diese verschiedenen Definitionen in folgender Grafik (Abb. 4.6) verdeutlichen. Hierbei ist auf der linken Seite das schiefsymmetrische Netzwerk mit den Kapazitäten der si- und $i't$-Kanten dargestellt. Die Kapazitäten der übrigen Kanten seien alle 1. Auf der rechten Seite finden wir die Bäume T und T' und damit das Spiegelnetzwerk, welches der Algorithmus bisher aufgebaut hat. Die Idee ist also sukzessive Nachbarn von Knoten aus

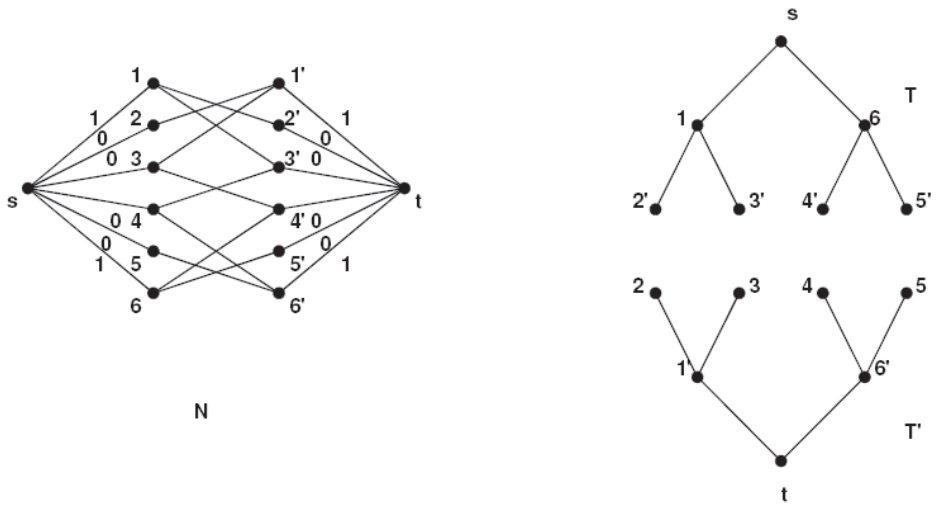

Abbildung 4.6: Auf der linken Seite ist das schiefsymmetrische Netzwerk mit an einigen Kanten notierten Kapazitäten und auf der rechten Seite die Bäume T und T' und damit das Spiegelnetzwerk M bestehend aus T und T'

T aufzunehmen und genauso symmetrisch T' auszubauen. Ist dann t durch einen gültigen st-Pfad p erreichbar, so ist auch p' ein gültiger st-Pfad. Wir wollen p und p' dann, wie wir noch genauer beschreiben werden, aus T und T' extrahieren.
Aus unserer bisherigen Diskussion wissen wir, dass das Problem leider nicht mit einer symmetrischen Erforschung des Netzwerks abgetan ist. Das Problem liegt hierbei in der Verbindung von T und T', die irgendwann natürlich erfolgen muss. Nehmen wir an, T enthalte einen gültigen si-Pfad p_i und einen gültigen sj-Pfad p_j. Damit enthält T' einen gültigen $i't$-Pfad p'_i und einen gültigen $j't$-Pfad p'_j. Reicht es nun aus, die Kanten (ij') und (ji') zu den Pfaden hinzuzufügen um einen gültigen Pfad zu erhalten? Wir haben ja nicht überprüft, ob $p_i \circ (ij') \circ p'_j$ ein gültiger Pfad ist, sondern nur, ob p_i und p'_j einzeln gültig sind.
Bevor wir auf mögliche Lösungen dieses Problems eingehen, wollen wir der Kante (ij') zunächst einen gesonderten Namen geben.

Definition 4.16 (Wechselkante)**.** *Wir nehmen an, unser Algorithmus habe einen gültigen si-Pfad p_i und einen gültigen sj-Pfad p_j gefunden. Im nächsten Schritt werde die Kante (ij') in den Baum T aufgenommen. Wir nennen dann die Kante (ij') für alle Knoten k, die mit einer Kante auf p'_j inzidieren und durch die Konkatenation $p_i \circ (ij') \circ p'_j$ auf einem gültigen Pfad erreichbar werden, die Wechselkante.*

Den Begriff der Wechselkante kann man sich dabei auch über die Richtung der Zeiger auf dem Pfad definieren: Den si-Pfad finden wir, indem wir jeweils den Nachfolger eines Knotens bestimmen. Den $j't$-Pfad finden wir hingegen, indem für den jeweils betrachteten Knoten k', der mit einer Kante auf p'_j inzidiert, den Vorgänger auf dem Pfad p_j bestimmen. Man kann die Wechselkante also auch als die Kante definieren, auf der sich die Richtung der Zeiger zum Nachfolgerknoten ändert. Dies sei auch in der folgenden Abbildung (Abb. 4.7) illustriert Die Pfeile geben dabei die Richtung des *prevpt* in T an. Lediglich aus

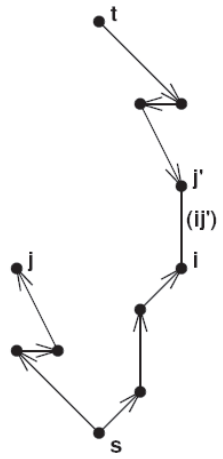

Abbildung 4.7: Wechsel der Richtung der Zeiger auf der Wechselkante

technischen Gründen kann man nun den Begriff der Wechselkante auch auf Knoten in T erweitern. Für Knoten $j \in T$ definiert man dazu die *triviale Wechselkante* als die Kante $(ij) \in A(N)$, die auf dem gültigen Pfad p_j ist. Wir notieren diesen Knoten i auch als $prevpt(j)$ (previous pointer).
Damit haben wir aber die Frage, ob die Konkatenation $p_i \circ (ij') \circ p'_j$ ein gültiger Pfad ist, nicht beantwortet. Eine Möglichkeit, die wir hier verfolgen wollen, ist die Einführung von Kernen. Wir definieren dazu folgende Menge:

$$\begin{aligned} \mathfrak{C} \;:=\; & \{i \mid i \in V(N),\ \text{der Knoten } i \text{ inzidiert mit einer Kante} \\ & \text{auf einem gültigen } si\text{-Pfad in } M\} \end{aligned} \tag{4.34}$$

Definition 4.17 (Kern)**.** *Die selbstkomplementären verbundenen Komponenten von* $\mathfrak{C}$ *heißen Kerne. Sei* $i \in \mathfrak{C}$*, so bezeichnen wir einen Kern, der* i *enthält mit* $\mathfrak{C}(i)$*.*

Unter einer *verbundenen Komponente* verstehen wir hierbei folgendes [35, S. 6]: Zwei Knoten $i, j \in V(N)$ in einem Netzwerk N heißen *verbunden*, wenn zwischen ihnen ein gültiger Pfad existiert. „Verbunden“ ist damit sogar eine Äquivalenzrelation auf der Menge der Knoten $V(N)$ und die Äquivalenzklassen bezüglich dieser Relation sind die verbundenen Komponenten.
Wir wollen uns in (Abb. 4.8) kurz veranschaulichen, was man sich unter einem Kern vorstellen muss. Alle Kanten in dem dargestellten Spiegelnetzwerk haben Restkapazität 1. In einem Schritt des Algorithmus wird nun von s aus der Knoten 3 gefunden. Damit wird $(23')$ in T und $(3'2)$ in T' aufgenommen. Es gibt damit eine selbstkomplementäre verbundene Komponente in $\mathfrak{C}$, einen Kern. Der Kern besteht dabei aus $\mathfrak{C}(1) = \{1, 2', 3', 3, 2, 1'\} = \mathfrak{C}(2')$. Die Eigenschaften selbstkomplementär und verbunden sind dabei offensichtlich.
Kerne sind allerdings nicht nur selbstkomplementär und verbunden, sondern besitzen zusätzlich noch folgende interessante Eigenschaft:

- Jeder Kern $\mathfrak{C}(i)$ enthält einen eindeutigen Knoten $base(i)$, den wir als Basis des Kerns bezeichnen. Jeder gültige sj-Pfad enthält $base(i)$ für alle Knoten $j \in \mathfrak{C}(i)$ und erreicht $base(i)$ über eine Kante mit Restkapazität 1.

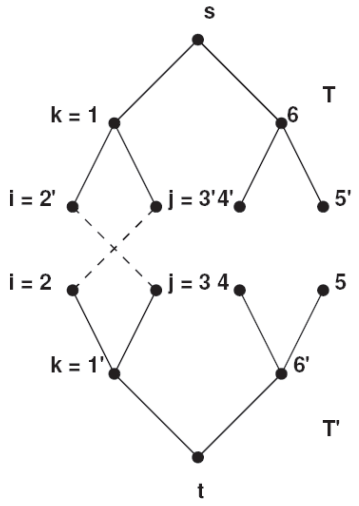

Abbildung 4.8: Der Kern aus 1, 2', 3', 3, 2 und 1' ist durch Hinzunahme der Kanten $(2'3)$ und $(3'2)$ entstanden

Den Beweis dieser Eigenschaft von Kernen findet man in [41, S. 7-12].
Jeder Kern kann also durch seine Basis eindeutig bestimmt werden. Ferner sehen wir, dass wir die Suche nach gültigen Pfaden zu einem Knoten $i \in \mathfrak{C}(i)$ in 2 Schritte aufteilen können: Im ersten Schritt suchen wir die Basis eines Kerns, denn über diese muss jeder gültige Pfad gehen und suchen dann von der Basis weiter. Wir definieren uns dazu einen sogenannten Zeiger, der uns zur Basis des Kerns führt:

$$baseprt(i) := \begin{cases} 0, \ i \ \text{ist nicht in einem Kern} \\ -1, \ i \ \text{ist die Basis des Kerns} \\ j \ , \ j \ \text{ist ein Knoten, der näher an der Basis ist, sonst} \end{cases} \tag{4.35}$$

Um nun 2 Kerne $\mathfrak{C}(i)$ und $\mathfrak{C}(j)$ zu vereinigen, können wir für einen Kern einfach setzen: $baseptr(base(i)) = j$.
Wir sehen zudem, dass ein Kern unsere Suche nach gültigen Pfaden behindert, da die Basis und damit auch der Kern nur von einem der Pfade p oder p' überquert werden darf. Betrachten wir nun unseren schiefsymmetrischen Graphen, so fällt auf, dass, falls dieser vollständig in M aufgenommen wurde, dieser ebenfalls die Anforderungen an einen Kern erfüllt. Irgendwann sind also alle Kanten in einem Kern.
Es ist deshalb sinnvoll und für die Konstruktion der Kerne einfacher, wenn wir anfangs jeden Knoten i in einen sogenannten trivialen Kern aufnehmen.

Definition 4.18 (trivialer Kern)**.** *Ein trivialer Kern ist ein Paar $\{i, i'\}$ für jeden Knoten i, der mit einer Kante in T inzidiert.*

Nach diesen Vorüberlegungen zum Algorithmus, wollen wir zunächst den Pseudocode des Algorithmus angeben und werden dann genauer auf die Funktionsweise des Algorithmus eingehen.

```
BNS-Algorithmus
ScanQ (Menge der von s auf gültigen Pfaden erreichbaren Knoten)
QSize : derzeitige Größe von ScanQ
SwitchEdge[i]: Wechselkante des Knotens i ∈ V(N)
begin
    SwitchEdge[i] := ∅, ∀i ∈ V(N)
    PrevPt[i] := ∅, ∀i ∈ V(N)
    basePtr[i] := 0, ∀i ∈ V(N)
    b_i = 0, i ∈ V(N)
    ScanQ[1] := s
    QSize := 1
    BasePtr[s] := −1
    k := 1
    repeat
        i := ScanQ[k]
        for all j adjazent zu i do
        if (PrevPt[j] ≠ i) and (rescap(ij) > 0) then
        begin
            b_j := FindBase(j)
            if b_j = 0 then
            begin
                PrevPt[j] := i
                QSize := QSize + 1
                ScanQ[QSize] := j
                SwitchEdge[j] := (ij)
                BasePtr[j'] := j
                BasePtr[j] := −1
            end;
            else if (j' von s auf gültigem Pfad erreichbar) and
            (PrevPt[i'] ≠ j') and (b_i ≠ b_j) then
            begin
                w := MakeBlossom(i, j, b_i, b_j)
                b_j := w
                if w' = t then
                begin
                  gebe Pfade p und p' aus
                  STOP
                end;
            end;
        end;
        k:=k+1
    until k > QSize
```

```
end;
```

Wir definieren in diesem Pseudocode zunächst die verschiedenen Variablen. Insbesondere speichern wir die Menge der von s aus auf gültigen Pfaden erreichbaren Knoten in einer sogenannten Queue ab, für die man natürlich eine gute Datenstruktur finden muss. Wir beginnen dann unsere Suche nach einem gültigen Pfad. Der Algorithmus kennt dazu 2 Fälle. Im ersten Fall wurde der im Algorithmus neu gefundene Knoten j adjazent zu i bisher noch nicht untersucht oder bereits in einen (eventuell trivialen) Kern aufgenommen. Bei der Untersuchung wird $prevpt[j] = i$ ausgeschlossen, um nicht einen Vorgängerknoten erneut zu betrachten.
Der erste Fall ist also $b_j = 0$. Wir nehmen dann j mit $prevpt[j] = i$ in den Baum T und mit $ScanQ[QSize] = j$ in $ScanQ$ auf. Da j bisher nicht untersucht wurde, ist j nach unserer vorhergehenden Diskussion zum Aufbau der Bäume von s aus auf einem gültigen Pfad erreichbar. Ferner nehmen wir j in einen trivialen Kern auf, da j jetzt bereits untersucht wurde.
Der 2. Fall tritt ein, wenn j' auf einem gültigen Pfad erreichbar ist. Dann ist ein st-Pfad gegeben durch den si-Pfad, die Kante (ij) und den sj'-Pfad (denn dessen komplementärer Pfad ist ja ein jt-Pfad). Wir vermeiden auch hier $prevpt[i'] = j'$ und fügen die, durch die Kante (ij) verbundenen, Kerne in der Operation $MakeBlossom$ zusammen. Hierbei werden auch die Wechselkanten für Knoten aus T' angepasst. Die Basis des neuen Kerns ist entweder s oder ein beliebiger anderer Knoten. Ist die Basis ein beliebiger Knoten außer s, so wurde durch die $MakeBlossom$ Operation lediglich ein größerer Kern erzeugt und wir machen weiter. Ist die Basis des neuen Kerns s, so reicht der Kern aufgrund der Selbstkomplementarität von s zu t. Damit gibt es nach der Definition von Kernen einen gültigen st-Pfad, den wir nur noch extrahieren müssen.
Hierfür wurden die Wechselkanten eingeführt: Wir bestimmen die Wechselkante von t. Ist diese (ij'), so können wir den Pfad p aufteilen in einen si-Pfad und einen $j't$-Pfad. Wir können nun die Wechselkanten für den si-Pfad und den sj-Pfad bestimmen. Protokollieren wir jeweils die Schritte, so können wir den Pfad über die Wechselkanten extrahieren. Wir kommen damit zur $MakeBlossom$ Operation [42, S. 13-16]. Damit wir diese ausführen, müssen wir zuvor im Algorithmus eine Kante (ij) gefunden haben, die die Bäume T und T' verbindet. In (Abb. 4.8) entspricht diese Kante der Kante $(2'3)$. Wir erhalten damit in unserer Suche einen neuen Kern. Wir wissen damit, dass jeder gültige Pfad über die Basis dieses Kerns laufen muss.
Der erste Schritt von $MakeBlossom$ ist daher die Basis dieses Kerns zu finden. Da die Basis und ihr komplementärer Knoten jeweils auf gültigen Pfaden liegen und Kerne selbstkomplementär sind, gibt es 2 gültige Pfade p und p', die die Basis und ihren komplementären Knoten verbinden. Wir wollen uns dies zunächst in folgender Abbildung (Abb. 4.9) veranschaulichen.
Wir können also von $base(i)$ und $base(j)$ beginnend den gültigen Pfad zu s suchen und jeweils die besuchten Knoten speichern. Diese Operation nennen wir $FindPath$. Vergleichen wir nun die Elemente von p und p', so können wir den letzten gemeinsamen Punkt suchen. Ist dieser Punkt s, so haben wir mit der vorangegangenen Argumentation einen erweiternden Pfad gefunden. Stoppt dagegen die Pfadsuche vor s, so haben wir einen Knoten w gefunden, der auf allen gültigen Pfaden zu Knoten im Kern liegt und zu dem

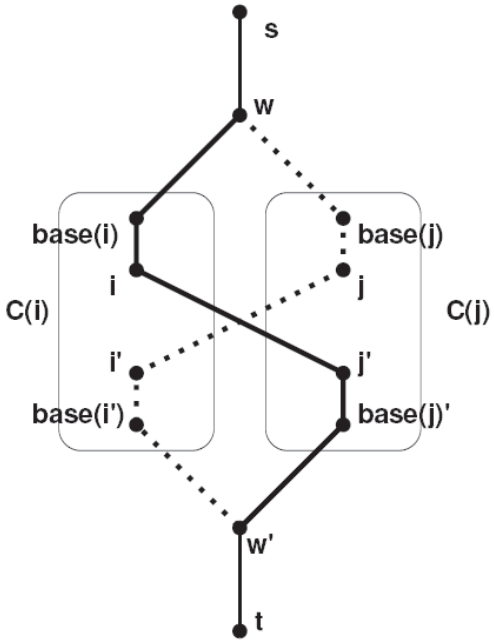

Abbildung 4.9: Veranschaulichung der Begriffe in der *MakeBlossom*-Operation, dick ist zwischen w und w' der Pfad p und gepunktet der Pfad p' eingezeichnet

nur eine Kante mit $rescap(prevpt(w)w) = 1$ führt. Damit ist w die gesuchte Basis des Kerns. Jeder gültige Pfad zu Knoten im Kern muss nun über w laufen und aufgrund der Selbstkomplementarität müssen alle gültigen Pfade den Kern auch wieder über w' verlassen. In unserem Beispiel in (Abb. 4.8) wäre w gerade der Knoten 1.
Damit gibt es 2 Pfade zwischen w und w', wobei der eine über i und der andere über j läuft. Wir müssen daher für die aus T' neu in $ScanQ$ aufgenommenen Knoten die Wechselkante auf (ij') bzw. (ji') setzen (je nachdem, ob wir den Pfad über i oder über j betrachten). In der Abbildung (Abb. 4.9) entspricht dies gerade den Knoten zwischen j' und w' bzw. i' und w'.
Wir könnten nun die Kerne auch aus dem Netzwerk entfernen, da die gültigen Pfade innerhalb des Kerns gefunden sind. Dies ist hier aber, da wir Kerne ohnehin nur durch ihre Basis behandeln, unnötig. Daneben würde das explizite Entfernen der Kerne die Berechnung der Pfade erschweren.
Wir werden später aber im primal-dualen Algorithmus auch ein explizites Schrumpfen von Kernen betrachten.
Wir wollen nach dieser verbalen Beschreibung nun auch den Pseudocode zur Operation *MakeBlossom* angeben:

```
MakeBlossom
begin
```

$p_i := findpath(base(i))$

$p_j := findpath(base(j))$
$length_i := |p_i|$
$length_j := |p_j|$
//finde den letzten gemeinsamen Knoten//
while $(p_i[k] \neq p_j[l])$ and $(i > 0)$ and $(j > 0)$ do

$k = k - 1,\ l = l - 1$

// Suche nach der Basis beginnen vom letzten gemeinsamen Knoten aus//
$w := p_i[k]$
$z := prevpt[w]$
// gehe vom letzten gemeinsamen Knoten weiter, bis w erreicht ist
while $(w \neq s)$ and $(rescap(zw) \geq 2)$ do
begin

$k = k - 1$
$l = l - 1$
$w := p_i[k]$
$z := prevpt[w]$

end;
// setze auf p_i und p'_i die base-pointer und die Wechselkanten//
for $m = length_i - 1$ to k do
begin

$z := p_i[m]$
$baseptr[z] = w$
$baseptr[z'] = w$
// falls die Wechselkanten schon zuvor gesetzt wurden, führe keine neuen ein//
if $z' \notin$ ScanQ then
begin

$SwitchEdge[z'] = (ji')$
$QSize = QSize + 1$
$ScanQ[QSize] = z'$

end;

end;
// nun das gleiche für p_j und p'_j//
for $n = length_j - 1$ to l do
begin

$z := p_j[n]$
$baseptr[z] = w$
$baseptr[z'] = w$
if $z' \notin$ ScanQ then
begin

$SwitchEdge[z'] = (ij')$
$QSize = QSize + 1$
$ScanQ[QSize] = z'$

`end;`

`end;`
`gebe` w `zurück`

`end;`

Wir wollen es bei dieser anschaulichen Beschreibung der Funktionsweise belassen. Da wir uns stark an der Nomenklatur in der Arbeit von Kocay und Stone orientiert haben, verweisen wir hier lediglich auf den Beweis der Korrektheit des Algorithmus in ihrer Arbeit [42, S. 16].
Statt dessen wollen wir das Beispiel in (Abb. 4.6) noch einmal aufgreifen und anhand dieses Beispiels diesen, doch an einigen Stellen trickreichen, Algorithmus erläutern. Außerdem ist in der (Abb. 4.10) das Spiegelnetzwerk nach jedem Schritt des Algorithmus eingezeichnet.
Wir beginnen also mit der Übergabe des schiefsymmetrischen Netzwerks auf der rechten Seite von (Abb. 4.6) an den BNS-Algorithmus von Kocay und Stone. Die Kapazitäten seien dabei wieder entweder in der Grafik an den Kanten notiert oder 1. Anfangs gilt $ScanQ = \{s\}$. Die Bäume T und T' werden im Algorithmus implizit durch die *prevpt*'s dargestellt.
Im ersten Schritt werden die Knoten adjanzent zu s untersucht, also 1 und 6. Keiner der Knoten wurde zuvor untersucht, also gilt $ScanQ = \{s, 1, 6\}$ und $prevpt(j) = s\ \forall j \in \{1, 6\}$.
Im nächsten Schritt wird nun der Knoten 1 untersucht. Wir finden die Knoten $2'$ und $3'$ die ebenfalls bisher nicht untersucht wurden. Wir setzen also $prevpt(j) = 1$ für $j \in \{2', 3'\}$ und $ScanQ := \{1, 6, 2', 3\}$.
Der nächste Knoten in $ScanQ$ ist 6. Wir finden die Knoten $4'$ und $5'$. Diese sind ebenfalls bisher nicht untersucht. Wir setzen also $prevpt(j) = 6$ für $j \in \{4', 5'\}$ und erhalten $ScanQ = \{1, 6, 2', 3', 4', 5'\}$.
Damit kommen wir zum Knoten $2'$. Wir finden den Knoten 3. Dieser ist aber bereits in einem Kern enthalten, da wir $3'$ schon untersucht hatten und damit den trivialen Kern $\mathfrak{C}(3) = \{3, 3'\}$ erstellt haben. Es tritt somit der 2. Fall und damit die $MakeBlossom$-Operation ein. Wir hatten den Kern in diesem Fall bereits in (Abb. 4.8) als $\mathfrak{C}(1) = \{1, 2', 2, 3, 3', 1'\}$ mit Basis 1 identifiziert. $MakeBlossom$ setzt nun für die Knoten $2', 3', 2, 3$ und $1'$ den *baseptr* auf 1. Außerdem wurden für die Knoten $2, 3$ und $1'$ noch keine Wechselkanten definiert. $MakeBlossom$ setzt daher $SwitchEdge[3] = (2'3)$, $SwitchEdge[2] = (3'2)$ und $SwitchEdge[1'] = (2'3)$. Ferner erhalten wir $ScanQ = \{1, 6, 2', 3', 4', 5', 3, 2, 1'\}$.

Für den Knoten $3'$ sind alle Kanten bereits erforscht. Der Fall $b_3 = 1 \neq b_j$ tritt also für einen zu 3 adjazenten Knoten nie ein.
Damit kommen wir zum Knoten $4'$. Wir finden den Knoten 5 und den Knoten $3'$. Beide wurden zuvor bereits untersucht und sind im trivialen Kern $\mathfrak{C}(5) = \{5, 5'\}$ bzw. im Kern $\mathfrak{C}(1)$ enthalten. Wir haben also wieder eine $MakeBlossom$-Operation. Die $MakeBlossom$-Operation findet in diesem Fall einen Kern mit Basis s in dem nun alle Knoten des Netz-

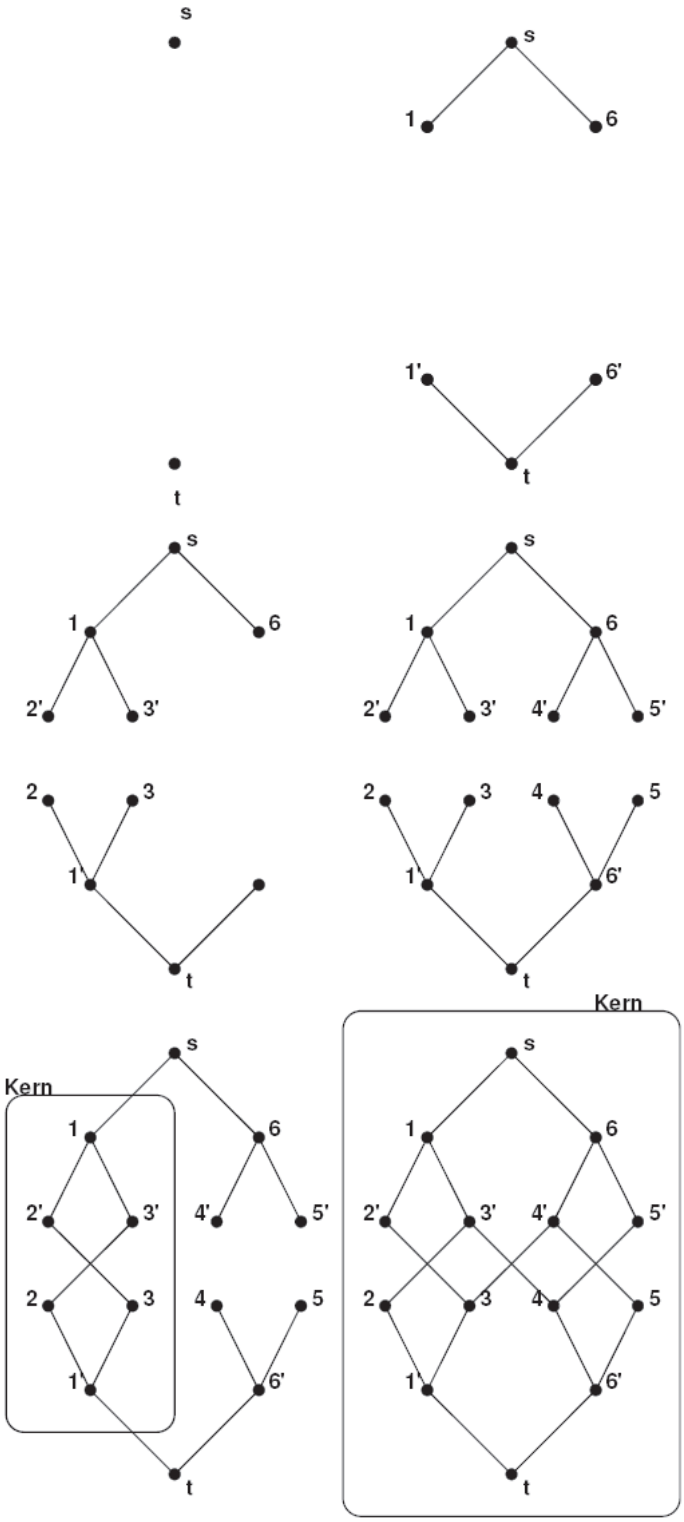

Abbildung 4.10: Das Spiegelnetzwerk nach jedem Schritt des Algorithmus, wie im Text beschrieben

werks enthalten sind. Für alle Knoten wird daher der *baseptr* auf s gesetzt. Außerdem wurden für die Knoten $4, 5, 6$ und t noch keine Wechselkanten definiert. $MakeBlossom$ setzt daher $SwitchEdge[4] = (5'4)$, $SwitchEdge[5] = (4'5)$, $SwitchEdge[6'] = (4'5)$ und $SwitchEdge[t] = (4'3)$. $ScanQ$ enthält nun alle Knoten des Netzwerks.
Da die Basis des entstandenen Kerns s ist, können wir nun über die Wechselkanten einen gültigen Pfad bestimmen. Wir bestimmen dazu die Wechselkante von t und erhalten $(4'3)$. Wir benötigen nun also noch einen $s4'$-Pfad und einen $s3'$-Pfad. Wir bestimmen dazu die Wechselkante von $4'$ und erhalten die triviale Wechselkante $(64')$. Auch 6 besitzt nur eine triviale Wechselkante nämlich $(s6)$. Damit erhalten wir den $s4'$-Pfad $p_{4'} = \{(s6), (64')\}$. Wir führen das gleiche für den Knoten $3'$ durch und erhalten den $s3'$-Pfad $p_{3'} = \{(s1), (13')\}$. Wir erhalten nun den gültigen st-Pfad als Konkatenation von $p_{4'}$, $(4'3)$ und $p'_{3'}$:

$$p = p_{4'} \circ (4'3) \circ p'_{3'} = \{(s6), (64'), (4'3), (31'), (1't)\} \tag{4.36}$$

Dieser Pfad ist in der Tat gültig.
Nach diesem Beispiel zum BNS-Algorithmus kommen wir noch einmal auf das nicht-bipartite Matchingproblem zurück. Wir hatten ja bereits angesprochen, dass die Probleme der Balanced Network Search und des Matching in nicht-bipartiten Graphen ähnlich sind. Die Idee beim nicht-bipartiten Matching ist die Struktur eines bipartiten Graphen auf einen nicht-bipartiten Graphen zu übertragen. Konzeptionell ist der Ansatz also sehr ähnlich zu schiefsymmetrischen Netzwerken. In diesem Fall versuchen wir aber die Knoten des Graphen ohne Hinzunahme weiterer Knoten in innere und äußere Knoten aufzuteilen [35, S. 364-367].
Wir durchsuchen dazu den Graphen von einem nicht gematchten Knoten aus. Wir nehmen dann alle adjazenten Knoten in einen Baum auf. Von diesen Knoten aus, nehmen wir alle adjazenten Knoten auf, die gematcht sind und fahren dann analog fort. Die verschiedenen Lagen des Baumes können wir nun durchnummerieren. Wir erhalten dabei z.B. folgendes Bild (Abb. 4.11), wobei an einigen Knoten die Bezeichnungen eingetragen sind.

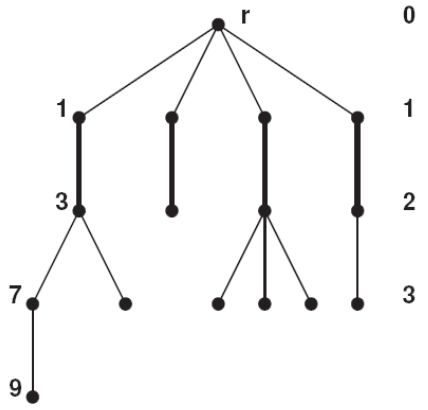

Abbildung 4.11: Beispiel für Lagen im Suchbaum

Es ist sicher nicht verwunderlich, dass wir die Menge der Knoten in ungeradzahligen Lagen

als *innere Knoten* und die Menge der Knoten in geradzahligen Lagen als *äußere Knoten* bezeichnen. Dies ist völlig analog zur Klassifikation der Knoten im schiefsymmetrischen Netzwerk.
Finden wir von einem ungematchten Knoten in einer ungeraden Lage einen ungematchten Knoten, so ist der Pfad von r zu diesem Knoten ein M-erweiternder Pfad. In der (Abb. 4.11) wäre ein solcher M-erweiternder Pfad:

$$p = \{(11), (13), (37), (79)\} \tag{4.37}$$

Die Idee scheint sehr schön einfach, sie ist es aber nur auf den ersten Blick.

Betrachten wir dazu die Suche nach einem erweiternden Pfad in folgendem Graphen (Abb. 4.12).

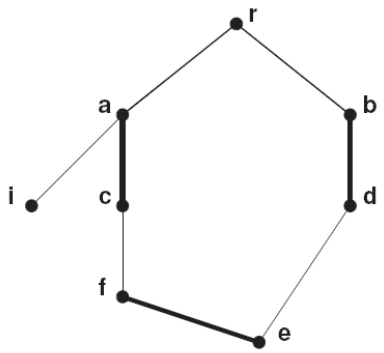

Abbildung 4.12: Matching in einem Graphen, wobei die Matchingkanten wieder dick eingezeichnet sind

Jeder der Knoten $\{a, b, c, d, e, f\}$ könnte, je nach Suchrichtung, entweder ein innerer oder ein äußerer Knoten sein. Für das obige Matching wäre ein erweiternder Pfad offensichtlich $p = \{(rb), (bd), (de), (ef), (fc), (ca), (ai)\}$. Nehmen wir die Kante (fc) nicht auf, da c zuvor bereits als adjazenter Knoten zu a aufgenommen wurde und damit gerade wäre, so finden wir diesen erweiternden Pfad nicht.
Das Problem ist offensichtlich, dass $C = \{(rb), (bd), (de), (ef), (fc), (ca), (ar)\}$ ein Kreis ungerader Länge ist. Diese können im bipartiten Fall nicht auftreten. Dieser ungerade Kreis ist ein typisches Beispiel für eine Blume.
Betrachten wir dies auf schiefsymmetrischen Netzwerken noch etwas formaler: Man nennt eine Kante $(kl') \in A(N)$ in einem schiefsymmetrischen Netzwerk N *erreichbar*, wenn (kl') von einem (kl')-erreichenden Pfad überquert wird. Hierbei ist ein (kl')-*erreichender Pfad* ein Pfad, der bei s beginnt, so dass $p[k']$ kein Knotenpaar (ij'), (ji') mit $rescap(ij) = 1$ enthält. Hierbei bezeichnet $p[k']$ den Teil des Pfades p von s zu k'.
Eine erreichbare Kante $(ij') \in A(N)$ heißt *bi-eulersch*, falls (ji') ebenfalls erreichbar ist und sonst *eulersch*. Die Menge aller bi-eulerschen Kanten bezeichnet man mit $Bic(N)$. Die verbundenen Komponenten in $N[Bic(N)]$ bezeichnet man als *Blüten*.
Es ist nicht verwunderlich, dass die zuvor eingeführten Kerne und die Blüten in Matchingproblemen eng verwandt sind.

Die beiden Begriffe seien anhand der folgenden Abbildung (4.13) noch einmal erläutert, in der, der Einfachheit halber, die symmetrischen Knoten nicht eingezeichnet sind.

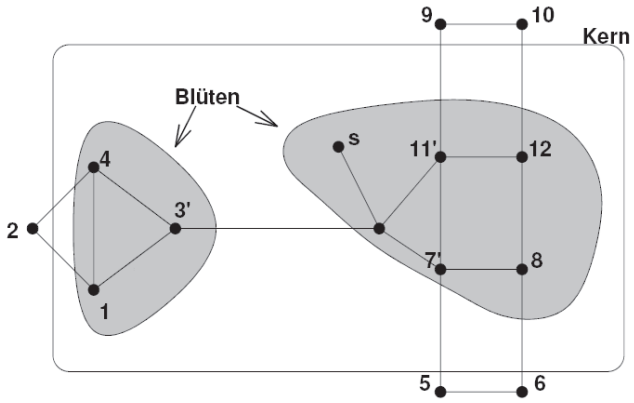

Abbildung 4.13: Blüten und Kerne

Hier existieren 2 Blüten aus $1, 3, 4$ und den komplementären Knoten und $5, 7, 8, 11, 12, 13$ und ihren komplementären Knoten. Die beiden Blüten sind durch die eulersche Kante $(13, 3')$ verbunden und bilden einen Kern.
Dies ist kein Zufall, denn man kann in der Tat zeigen, dass ein Kern immer die disjunkte Vereinigung von Blüten ist.
Mit diesen Analogien können wir den Algorithmus von Kocay und Stone auch als Übertragung des Matching-Algorithmus von Edmonds [20] auf schiefsymmetrische Netzwerke verstehen.

4.4 Kürzeste gültige Pfade

Mit dem obigen Algorithmus haben wir das Problem Max Bal Flow gelöst. Wir haben aber noch keine Aussage über dessen *Komplexität* (also die Worst Case Laufzeit) getroffen. Bei Flussalgorithmen haben wir im Allgemeinen eine Abhängigkeit der Worst-Case Laufzeiten von der Größe des Netzwerks $(n,\ m)$, der maximalen Kapatität einer Kante U und bei Algorithmen, die zusätzlich Kosten auf den Kanten berücksichtigen, auch eine Abhängigkeit von den maximalen Kosten einer Kante C.
Wir unterscheiden nun folgende Fälle [46, S. 5]:

- *pseudopolynomielle Laufzeit*: Die Laufzeit hängt von der Größe (n und m) und den Werten (C und U) der Eingabeinstanzen ab.
- *schwach polynomielle Laufzeit*: Die Laufzeit hängt polynomiell von n und m und nur von den Logarithmen der Eingabeinstanzen ab.
- *stark polynomielle Laufzeit*: Die Laufzeit hängt polynomiell ausschließlich von n und m ab.

Wir wollen hier nicht genauer auf die Laufzeit des Balanced Augmentation Algorithmus eingehen, da er im Allgemeinen nicht stark polynomiell ist [24, S.43].
Dies ist aber auch nicht verwunderlich, da wir unsere erweiternden Pfade bisher nach keinem Schema wählen.
Das Schema, das sich anbietet, ist offensichtlich die Länge der Pfade. Wir führen dazu zunächst Längen, sogenannte *schiefsymmetrische Distanzen*, ein. Diese werden bezüglich des Knotens s definiert:

$$d(i) := \min\{\infty, \{|p| : \text{ p ist ein gültiger } si\text{-Pfad auf } N(x)\}\} \tag{4.38}$$

Hierbei gibt $|p|$ die Anzahl der Kanten in p an.
Man bezeichnet nun einen si-Pfad q mit $|q| = d(i)$ als *minimal gültig* oder $d(i)$-Pfad.
Die Idee ist nun, als erweiternde Pfade nur $d(t)$-Pfade zu verwenden. Hierzu benötigt man BNS-Algorithmen, die lediglich solche bestimmen. Jungnickel und Fremuth-Paeger haben in ihrer Arbeit eine Adaption des Algorithmus von Micali und Vazirani, der sogenannten Double Depth First Search Methode, beschrieben, die dieses leistet [24]. Die Methode ist dabei stark polynomiell mit der Laufzeit $O(m\alpha(m,n))$ [24, S. 50]. Wir wollen hier nicht die Definition von α herausarbeiten. Es sei hier lediglich erwähnt, dass α die Umkehrfunktion der Ackermann-Funktion ist.
Auch diese Idee kommt ursprünglich aus der Matchingtheorie. In ihrer Arbeit [47] beschreiben Micali und Vazirani einen Algorithmus zur Bestimmung zur Bestimmung eines maximalen Matchings auf einem allgemeinen Graphen (Cardinality Matching Problem). Der Algorithmus von Micali und Vazirani ist auch heute noch der schnellste bekannte Algorithmus zur Lösung des Problems [43, S. 219].
Wir werden diesen BNS-Algorithmus hier nicht besprechen und lediglich zeigen, wie sich hiermit ein stark-polynomieller Algorithmus konstruieren lässt.
Hierzu wollen wir zunächst ein Lemma und einen Hilfssatz beweisen.

Lemma 4.19. *Sei p ein $d(t)$-Pfad auf $N(x)$ und (ij') eine Kante auf p, dann überquert p weder $(j'i)$ noch $(i'j)$.*

Beweis. Als Pfad kann p per Definition kein Paar antiparalleler Kanten (ij'), $(j'i)$ enthalten.
Nehmen wir nun an p sei minimal und enthalte ein Paar (ij'), $(i'j)$ von Kanten. Dann können wir den Pfad nach folgender Regel verkürzen:
Regel 1: Gilt $p = p_1 \circ (ij') \circ p_2 \circ (i'j) \circ p_3$ oder $p = p_1 \circ (i'j) \circ p_2 \circ (ij') \circ p_3$ für Spaziergänge p_1, p_2, p_3, so kürzen wir p auf $p := p_1 \circ p_2' \circ p_3$.
Mit dieser Operation verliert p 2 Kanten. p muss kein Pfad mehr sein, aber wir können dann p weiter auf einen Pfad q kürzen. q muss dann sogar gültig sein, da q nur Kanten von p und p' enthält und p und p' jeweils gültig waren. Da q ferner mindestens 2 Kanten weniger enthält als p gilt $|q| < |p|$. Damit haben wir den Widerspruch zur Wahl von p. □

Satz 4.20. *Sei p ein $d(t)$-Pfad auf $N(x)$, $y := x + x_p + x_{p'}$ und q ein $d(t)$-Pfad auf $N(y)$, dann gilt $|p| \leq |q|$.*

Beweis. Ist q gültig auf $N(x)$, dann gilt offensichtlich $|p| \leq |q|$.
Wir müssen nun also den Fall betrachten, dass q nicht gültig auf $N(x)$ ist. Damit existiert

mindestens eine Kante (ij') auf q mit $(j'i) \in p$ oder $(i'j) \in p$.
Die Idee ist nun wieder q zu kürzen und einen auf $N(x)$ gültigen Pfad r mit der Länge $|r| < \max\{|p|,\ |q|\}$ aus p und q zu erhalten.
Wir identifizieren zunächst die auf q zu kürzenden Kanten:

$$A_0 := \{(ij) \in q\ :\ (j'i') \notin q,\ rescap(ij) = 0\} \tag{4.39}$$
$$A_1 := \{(ij) \in q\ :\ (j'i') \in q,\ rescap(ij) = 0\} \tag{4.40}$$
$$A_2 := \{(ij) \in q\ :\ (j'i') \in q,\ rescap(ij) = 1\} \tag{4.41}$$

Da q auf $N(y)$ gültig ist, muss p für jede Kante $(ij) \in A_0 \cup A_2$ die Kante $\overline{(ij)}$ oder ihre komplementäre Kante $\overline{(ij)}'$ überqueren. Für die Kanten (ij) aus A_1 gilt sogar, dass p $\overline{(ij)}$ und $\overline{(ij)}'$ überqueren muss. Wir wollen q nun um die Kanten aus A_0 und A_1 kürzen und für $(ij) \in A_2$ entweder (ij) oder $(ij)'$ aus A_2 entfernen.
Die erste Kürzungsregel hatten wir im vorhergehenden Lemma schon bereitgestellt. Nun nehmen wir noch folgende hinzu:
Regel 2: Gilt $p = p_1 \circ (ij') \circ p_2$ und $q = q_1 \circ (j'i) \circ q_2$ für Spaziergänge $p_1,\ p_2,\ q_1,\ q_2$, dann setze $p := p_1 \circ q_2$ und $q := q_1 \circ p_2$.
Regel 3: Gilt $p = p_1 \circ (ij') \circ p_2$ und $q = q_1 \circ (i'j) \circ q_2$ für Spaziergänge $p_1,\ p_2,\ q_1,\ q_2$, so setze $p := p_1 \circ q_1'$ und $q := p_2' \circ q_2$.
Regel 4: Gilt $r = r_1 \circ (ij') \circ r_2 \circ (j'i) \circ r_2$ oder $r = r_1 \circ (j'i) \circ r_2 \circ (ij') \circ r_3$ für irgendwelche Spaziergänge $r_1,\ r_2,\ r_3$, so setze $r := r_1 \circ r_3$.
Regel 5: Gilt $r = r_1 \circ (ij') \circ r_2$ und $p = p_1 \circ (j'i)$ sei ein geschlossener Spaziergang für Spaziergänge $p_1,\ r_1,\ r_2$, so setze $r := r_1 \circ p_1 \circ r_2$.
Wir wenden nun zunächst die Regeln 1, 2 und 3 an, bis keine dieser Regeln mehr angewendet werden kann. Wir hatten bereits darauf hingewiesen, dass p und q nun nur noch st-Spaziergänge sein müssen. r sei nun der kürzere der beiden Spaziergänge und wir definieren:

$$A_3 := \{(ij) \in r\ :\ rescap(ij) = 0\} \tag{4.42}$$

Nach Konstruktion von r aus p und q gilt notwendigerweise $A_3 \subseteq A_0 \cup A_1 \cup A_2$ und mit $(ij) \in A_3$ gilt auch $\overline{(ij)} \in r$. Wir können nun diese Kanten offensichtlich durch Regel 4 kürzen. Damit spaltet r auf in einen st-Spaziergang und einen 2. geschlossenen Spaziergang r_2. Sind A_3 und r_2 disjunkt, so können wir r_2 fortfallen lassen, ansonsten wird der geschlossene Spaziergang mit Regel 5 erneut verbunden. Wir fahren fort, bis keine Kante (ij) mit $rescap(ij) = 0$ mehr auf r ist.
Hierbei terminiert das Verfahren trotz Regel 5, da bei der kombinierten Anwendung von Regel 4 und Regel 5 insgesamt eine Kante aus dem Pfad gekürzt wird.
Kürzen wir nun r auf einen einfachen st-Pfad, so erhalten wir den gewünschten st-Pfad r mit $|r| < \max\{|p|,\ |q|\}$, der auf $N(x)$ gültig ist.
Wäre q nach dieser Prozedur echt kürzer als p, so würde $|r| < |p|$ gelten im Widerspruch zur Wahl von p. □

Bemerkung: Der Satz ist intuitiv klar, da nach einer Flusserhöhung auf dem kürzesten Pfad in einem Netzwerk dieser kürzeste Pfad ja nicht mehr im Restnetzwerk möglich ist. Bestimmen wir den nächsten kürzesten Pfad, sollte dieser nicht kürzer sein ($|q| \geq |p|$).

Der Beweis dieser Aussage ist aber, wie wir sehen, doch recht aufwändig.
Um ein Schema für die Erhöhung der Flüsse zu finden, ist die Aussage aber sehr interessant: Bei einer Erhöhung des Flusses längs von $d(t)$-Pfaden p_j und p'_j um $balcap(p_j)$ Einheiten kann die Länge der Pfade im Algorithmus nicht abnehmen. Wir können also die gültigen erweiternden Pfade ihrer Länge nach ordnen:

$$phase_i := \{p_j \ : \ 1 \leq j \leq v_i, \ |p_j| = 2i + 1\}, \ i \in \mathbb{N} \tag{4.43}$$

Wir bezeichnen jede Periode eines Algorithmus, in der nur Pfade einer bestimmten Länge verwendet werden als *Phase*. v_i gibt hierbei die Anzahl der Pfade in der $phase_i$ an. Für die Pfade in einer Phase gibt es noch das folgende wichtige Korollar.

Korollar 4.21. *Seien p und q Pfade wie im vorherigen Satz. Wir nehmen an $|q| = |p|$ und (ij') sei eine Kante von q. Dann überquert p weder $(j'i)$ noch $(i'j)$.*

Beweis. Nehmen wir an, p enthalte $(j'i)$ oder $(i'j)$. Dann wären im vorhergehenden Satz die Regeln 2 und 3 anwendbar gewesen und die Anwendung des Satzes hätte die Existenz eines kürzeren Pfades r ergeben. Damit haben wir den Widerspruch. □

Mit diesem Korollar können wir nun die Komplexitätsuntersuchung des Balanced Augmentation Algorithmus mit minimal-gültigen Pfaden angehen.
Das obige Korollar zeigt dabei, dass die einzelnen Erhöhungsschritte einer Phase kommutieren, d.h. ihre Reihenfolge innerhalb einer Phase beliebig geändert werden kann.
Ferner können wir sogar Aussagen über die Anzahl der Erhöhungsschritte in einer Phase machen.
Jeder Pfad $p_k \in phase_l$ mit $k, \ l \in \mathbb{N}$ enthält mindestens eine Kante, die uns davon abhält den Fluss um mehr als $balcap(p_j)$ Einheiten zu erhöhen. Eine solche Kante nennen wir eine blockierende Kante auf p_j.
Ist (ij) eine blockierende Kante und p_k überquert nicht $(ij)'$, so gilt $rescap(ij) = 0$ nach dem k-ten Erhöhungsschritt. Überquert p_k auch $(ij)'$, so gilt natürlich nur $rescap(ij) \leq 1$.
Aufgrund des obigen Korollars können damit (ij) und $(ij)'$ höchstens ein zweites Mal in $phase_l$ auftreten und damit maximal 2 Mal die blockierende Kante sein.
Da mit (ij) auch $(ij)'$ als blockierende Kante (in p') auftreten muss und die Rückwärtskanten mit dem obigen Korollar in $phase_l$ überhaupt nicht auftreten, gilt:

$$|phase_l| \leq m + n \tag{4.44}$$

Hierbei erhalten wir $m + n$, da wir auch die n Kanten zu s berücksichtigen müssen.
Die Länge eines gültigen st-Pfades ist ungerade und offensichtlich durch die Anzahl der Knoten begrenzt, so dass der Algorithmus maximal n Phasen hat. Damit haben wir:

Satz 4.22. *Sind alle erweiternden Pfade minimal gültig, so besteht der Balanced Augmentation Algorithmus aus $O(n)$ Phasen und jede Phase hat $O(m+n)$ Erhöhungsschritte.*

Korollar 4.23 (stark-polynomieller Algorithmus)**.** *Sei die Bestimmung eines minimal-gültigen Pfades in stark-polynomieller Laufzeit $K(m, n)$ möglich, so ist auch der Balanced Augmentation Algorithmus mit minimal-gültigen Pfaden stark-polynomiell mit einer Laufzeit $O((m + n) \cdot n \cdot K(m, n))$.*

4.5 Algorithmus von Anstee

Wir haben mit der obigen Untersuchung gezeigt, dass Max Bal Flow durch Balanced Augmentation mit minimal gültigen Pfaden (und dem BNS-Algorithmus) in stark polynomieller Laufzeit lösbar ist. Der beste Algorithmus zur Lösung von Max Bal Flow greift allerdings nicht auf die Balanced Augmentation zurück, sondern auf die Idee des Cycle Cancelling.
Wir bestimmen hierbei zunächst einen Maximum Flow, symmetrisieren ihn und ziehen dann die nicht-ganzzahligen Kreise ab.
Um diese Idee zu verdeutlichen, wollen wir an dieser Stelle einige Begrifflichkeiten einführen. Wir wollen dies allerdings nur in so weit tun, als die Begriffe für den Primal-Dual Algorithmus, den wir vorstellen wollen, sinnvoll sind und verweisen ansonsten auf die Arbeit von Jungnickel und Fremuth-Paeger [25].
Wir notieren zunächst unsere Definitionsgleichungen für eine balancierte Zirkulation ohne die Ganzzahligkeitsforderung. Wir können diese Gleichungen auch als die Gleichungen des Polytops der rationalen balancierten Zirkulationen verstehen, wobei „rational“ den Verzicht auf die Ganzzahligkeitsforderung ausdrückt. Für dieses Polytop haben wir zusätzlich noch die Kapazitätsbedingungen für Flüsse auf den Kanten $cap(ij)$ und Bedingungen für einen minimalen Flusswert $l(ij)$:

$$\begin{aligned} l(ij) &\leq x_{ij} \; \forall \; (ij) \in A(N) && (4.45)\\ x_{ij} &\leq cap(ij) \; \forall \; (ij) \in A(N) && (4.46)\\ x_{ij'} &= x_{ji'} \; \forall \; (ij) \in A(N) && (4.47)\\ e(i) &= 0 \; \forall i \in V(N) && (4.48) \end{aligned}$$

Das durch diese Ungleichungen bzw. Gleichungen definierte Polytop bezeichnen wir als das *Polytop der rationalen Zirkulationen* $\mathfrak{F}(N)$. Dabei verstehen wir den Fluss als einen Vektor im Vektorraum $\mathbb{R}^{|A(N)|}$.
Die Flüsse, die den Ecken von $\mathfrak{F}(N)$ entsprechen nennt man *Basen*. Wir unterscheiden den *rationalen Teil* des Netzwerks und den *ganzzahligen Teil* des Netzwerks, indem wir die folgenden Kapazitäten definieren:

$$fraccap(ij) := \begin{cases} rescap(ij) \; mod \; 2, & \text{wenn } (ij) \text{ Schleife} \\ rescap(ij) \; mod \; 1, & \text{wenn } (ij) \text{ keine Schleife} \end{cases} \quad (4.49)$$

Damit können wir auch die ganzzahligen Kapazitäten definieren: $intcap(ij) := rescap(ij) - fraccap(ij)$. Die beiden zugehörigen Netzwerke bezeichnen wir mit $N_{frac}(x)$ und $N_{int}(x)$. Man bezeichnet Kanten mit $fraccap(ij) > 0$ als rational. Ferner bezeichnen wir eine Kante als frei, wenn $rescap(ij) > 0$ und $rescap(ji) > 0$.
Einen Kreis nennen wir *ungerade*, wenn er einfach ist und Kanten (ij) und $(ji)'$ immer paarweise enthält. Wir können jeden ungeraden Kreis Q mit einem einfachen ii'-Pfad q schreiben (wobei $i \in Q$ beliebig ist) als: $Q = q \circ \bar{q}'$. q hat dabei aufgrund der bipartiten Struktur des Netzwerks immer eine ungerade Länge.
Die Bedeutung der ungeraden Kreise im Netzwerk illustriert der folgende Satz.

Satz 4.24. *Sei x ein rationaler, balancierter Fluss auf dem schiefsymmetrischen Netzwerk N, dann ist x eine Ecke des Polytops $\mathfrak{F}(N)$, genau dann wenn jeder freie Kreis in $N(x)$ ungerade ist.*

Beweis. in Fremuth Paeger [22, S. 53] □

Zudem haben wir folgende interessante Eigenschaft, die wir hier ebenfalls nicht beweisen wollen [25, S. 203].

Satz 4.25. *Sei x eine Ecke des Polytops $\mathfrak{F}(N)$, dann ist $2 \cdot x$ immer eine ganzzahlige Zirkulation.*

Diese letzte Eigenschaft ist für den Algorithmus von großer Bedeutung. Suchen wir auf den Ecken des Polytops $\mathfrak{F}(N)$ optimale Zirkulationen, so sind diese jedenfalls halbzahlig und die „Reparatur“ des Flusses einfach.
Dies motiviert die folgende Bezeichnung: x ist eine *Pseudobasis*, wenn x halbzahlig ist und die rationalen Kanten paarweise disjunkte Kreise bilden. Diese Kreise $Q_1, \cdots, Q_r$ bezeichnen wir auch als das *System ungerader Kreise* von x.
Damit haben wir die Kreise für das Cycle Cancelling identifiziert.
Zur Illustration der Definition betrachten wir nun das Beispiel in (Abb. 4.14). Die Kapazitäten aller Kanten seien dabei 1 und die unteren Grenzen alle 0.

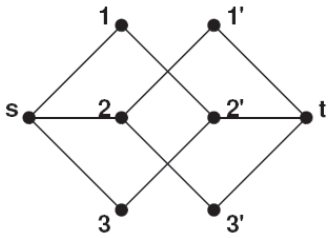

Abbildung 4.14: schiefsymmetrisches Netzwerk zur Illustration des Polytops der rationalen Zirkulationen

Man sieht, dass der maximale Flusswert 2 ist. Wir wollen nun die möglichen maximalen rationalen balancierten Flüsse in der folgenden Grafik (Abb. 4.15) explizit angeben. Die Flusswerte sind dabei jeweils an den Kanten notiert. Wir geben hierbei nur die halbzahligen Lösungen an.
Das Polytop der rationalen balancierten Zirkulationen ist dann ein $|A(N)| = 10$-dimensionalens Polytop mit den 3 Ecken, die den Flüssen in den Grafiken (a), (b) und (c) entsprechen. Wir können dieses Polytop mit einer (hier jetzt nicht näher behandelten) Projektion abstandstreu in den zweidimensionalen Raum projezieren und erhalten dann das Dreieck in (Abb. 4.16), wobei die Ecken andeuten sollen, welchen Flüssen sie entsprechen. Wir sehen, dass wir ein degeneriertes Dreieck erhalten.
Der Algorithmus von Anstee hat nun 2 Phasen (siehe auch [22, S. 63-64]). In der ersten Phase bestimmen wir einen maximalen ganzzahligen Fluss mit einem „normalen“ Max Flow Algorithmus. Wir nennen diesen Fluss x^* und symmetrisieren ihn:

$$x_0(ij') := \frac{1}{2}\left(x^*(ij') + x^*((ij')')\right) \ \forall (ij') \in A(N) \tag{4.50}$$

Dieser Fluss ist damit sicherlich balanciert, aber nur halbzahlig und nicht ganzzahlig.
In der 2. Phase des Algorithmus von Anstee bestimmen wir zunächst das System ungerader Kreise $Q_1, \cdots, Q_r$. Lassen sich 2 dieser Kreise durch einen gültigen Pfad verbinden,

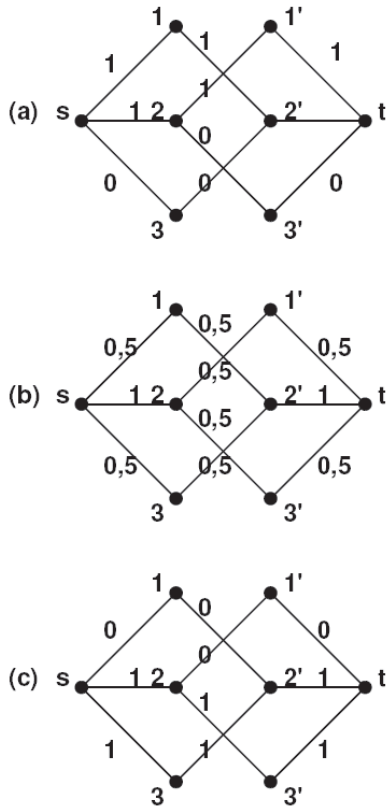

Abbildung 4.15: mögliche maximale rationale balancierte Flüsse

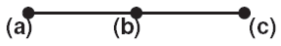

Abbildung 4.16: Polytop der rationalen Zirkulationen

so erhöhen wir den Fluss auf den ungeraden Kreisen und dem verbindenden Pfad.
Ist keine solche Operation mehr möglich, so erhalten wir einen Fluss x, der nur noch unverbundene ungerade Kreise $1, \cdots, t$ hat. Wir reduzieren diesen Fluss nun auf einen ganzzahligen Fluss $int(x)$ (int für integer $\equiv$ engl. ganzzahlig):

$$int(x) :\equiv x - \frac{1}{2}\sum_{i=1}^{t}(x_{q_i} + x_{q_i'}) \tag{4.51}$$

Anstee konnte nun beweisen, dass diese Projektion des Flusses x in der Tat ein maximaler balancierter Fluss ist [22, S. 64].
Falls den Leser die ungeraden Kreise hier an die ungeraden Komponenten in Tuttes 1-Faktor Satz oder seinen allgemeinen f-Faktor Satz erinnern, ist dies vollkommen richtig. In der Tat beschäftigt sich Anstees Arbeit [4] mit einem algorithmischen Beweis des f-Faktor Satzes von Tutte.
Nach Jungnickel und Fremuth-Paeger können wir die Bestimmung des Maximum Flow in Anstees Algorithmus auch durch eine Balanced Augmentation ersetzen und damit eine

Komplexität von $O(mn)$ für das Max Bal Flow Problem erhalten. Dies ist sogar schneller als die heutigen Algorithmen für Max Flow [22, S. 65].
Abschließend wollen wir noch kurz ein Lemma beweisen, dass wir noch benötigen.

Lemma 4.26. *Sei N ein schiefsymmetrisches Netzwerk und x eine Pseudobasis mit dem System ungerader Kreise $Q_1, \cdots, Q_r$. Sei $U \subseteq V(N)$, $U = U'$, so dass kein ungerader Kreis den Schnitt $[U, \bar{U}]$ mit $\bar{U} = V(N) \backslash U$ trifft und $odd(U) := |\{i \ : \ Q_i \subseteq N[U]\}|$. Dann gilt:*

$$odd(U) \equiv x(U, \bar{U}) \ mod \ 2 \tag{4.52}$$

Beweis. Sei $U = W \biguplus W'$ (da U selbstkomplementär ist, lässt sich eine solche Darstellung immer finden), dann gilt:

$$0 = \sum_{i \in W} e(i) = x(\bar{U}, W) - x(W, \bar{U}) + x(W', W) - x(W, W') \tag{4.53}$$

Jeder der Schnitte $[W, W']$, $[W', W]$ spaltet auf in Paare komplementärer Kanten $((ij') \in [W, W'] \ \Rightarrow \ (ji') \in [W, W'])$.
Sei nun $(ij') \in [W, W'] \cup [W', W]$, dann trägt $x_{ij'} + x_{ji'}$ zu $x(W', W) - x(W, W')$ eine ganze Zahl bei, wenn $x_{ij'}$ ganzzahlig ist und eine ungerade Zahl, falls (ij) auf einem ungeraden Kreis ist. Damit haben wir:

$$odd(U) \equiv x(W', W) - x(W, W') \ mod \ 2 \tag{4.54}$$

Und damit auch:

$$\begin{aligned} odd(U) &\equiv x(\bar{U}, W) - x(W, \bar{U}) \equiv x(\bar{U}, W) + x(W, \bar{U}) && (4.55)\\ &\equiv x(\bar{U}, W) + x(\bar{U}, W') \equiv x(\bar{U}, U) && (4.56)\\ &\equiv x(U, \bar{U}) \ mod \ 2 && (4.57) \end{aligned}$$

□

Kapitel 5

Balanced Min Cost Flow

Nach den Vorarbeiten zum Max Bal Flow kommen wir nun zum eigentlichen Problem der minimalen Kostenflüsse auf schiefsymmetrischen Netzwerken mit konvexer Kostenfunktion.

5.1 Formulierung des Problems

Wir gehen zunächst auf das Problem mit linearen Kosten ein und werden später auf das Problem mit konvexen Kosten eingehen.
Bei *linearen Kosten* definieren wir die Kosten eines Flusses x als $\sum_{(ij)\in A(N)} c_{ij}x_{ij}$ mit Kosten $c_{ij} \in \mathbb{R}$ für jede Kante $(ij) \in A(N)$. Im Allgemeinen können Kosten also auch 0 oder negativ sein. Wir werden uns hier allerdings lediglich mit dem Fall positiver Kosten beschäftigen. Das Problem ist nun einen maximalen Fluss bei minimalen Kosten zu erhalten.
Im Allgemeinen müssen die Kosten aber nicht linear sein, sondern können beliebige Funktionen des Flusses x sein.
In praktischen Problemen ist vor allem das Problem konvexer Kosten interessant. Wir wollen zunächst definieren, was wir unter einer konvexen Funktion verstehen [8, S. 6].

Definition 5.1 (konvexe Funktion)**.** *Eine Funktion $k : D \to \mathbb{R}$ von einer Menge nichtnegativer reeller Zahlen $D \subseteq \mathbb{R}^+$ in die Menge der reellen Zahlen nennen wir eine konvexe Funktion, falls für alle $x, y \in D$ und $\alpha,\ \beta \in \mathbb{R}$ mit $\alpha + \beta = 1$ gilt:*

$$k(\alpha x + \beta y) \leq \alpha k(x) + \beta k(y) \tag{5.1}$$

Setzen wir $\alpha,\ \beta = \frac{1}{2}$ und $y = x + 2\Delta$ mit $\Delta \in \mathbb{R}$, so erhalten wir die wichtige Relation, die wir ebenfalls als Definition verwenden könnten:

$$2k(x + \Delta) \leq k(x) + k(x + 2\Delta) \tag{5.2}$$

Wir betrachten allerdings auch nicht beliebige konvexe Kosten, sondern sogenannte *separable konvexe Kosten*. Bei separablen konvexen Kosten haben wir nicht mehr Kosten $c_{ij} \in \mathbb{R}$, sondern konvexe Kostenfunktionen $C_{ij}(x_{ij})$. Wir bezeichnen die Kosten hierbei als separabel, da sie sich als Summe von Kosten auf den einzelnen Kanten schreiben lassen: $\sum_{(ij)\in A(N)} C_{ij}(x_{ij})$.

Die Kosten können wir uns (z.B. bei einem Routenproblem für LKW) als Abstände zwischen den Knoten vorstellen. Die folgende Definition ist deshalb sinnvoll:

Definition 5.2 (schiefsymmetrische Abstände bei gegebenen Kosten)**.** *Sei $P_{i,j}$ die Menge aller gültigen Pfade p im Restnetzwerk $N(x)$ mit Startknoten $i \in V(N(x))$ und Endknoten $j \in V(N(x))$, dann definieren wir den Abstand von i zu j mit den Kosten einer Kante c_{ij} als:*

$$d(i,j) := \min \left\{ \infty, \min_{p \in P_{i,j}} \left\{ \sum_{(ij) \in p} c_{ij} \right\} \right\} \tag{5.3}$$

Mit $d(i)$ für $i \in V(N(x))$ bezeichnen wir den Abstand zu s.

Wir haben damit den Begriff der Kosten eingeführt. Wir können die Kosten allerdings nicht irgendwie minimieren, sondern müssen die Kapazitätsbeschränkungen einhalten und balancierte Flüsse erhalten.
In unserer Diskussion zu Anstee's Algoriothmus hatten wir bereits das Polytop aller rationalen Zirkulationen eingeführt:

$$l(ij) \leq x_{ij} \; \forall \; (ij) \in A(N) \tag{5.4}$$
$$x_{ij} \leq cap(ij) \; \forall \; (ij) \in A(N) \tag{5.5}$$
$$x_{ij} = x_{j'i'} \; \forall \; (ij) \in A(N) \tag{5.6}$$
$$e(i) = 0 \; \forall i \in V(N) \tag{5.7}$$

Um die Darstellung der Algorithmen in dieser Arbeit etwas zu vereinfachen, wollen wir stets $l(ij) = 0 \; \forall (ij) \in A(N)$ betrachten.
Grundsätzlich fehlt in obiger Darstellung aber die Forderung der Ganzzahligkeit. Die Ganzzahligkeit bekommen wir auch nicht „geschenkt“. Wir betrachten dazu folgendes Beispiel mit den Kosten $C_{ij}(x_{ij}) = x_{ij}^2$ für alle Kanten im Ausgangsgraphen (dem Graphen aus dem wir das schiefsymmetrische Netzwerk konstruiert haben) und $c_{ij} = 0$ für die si und $j't$-Kanten mit $i, \; j' \in V(N)$ (Abb. 5.1).

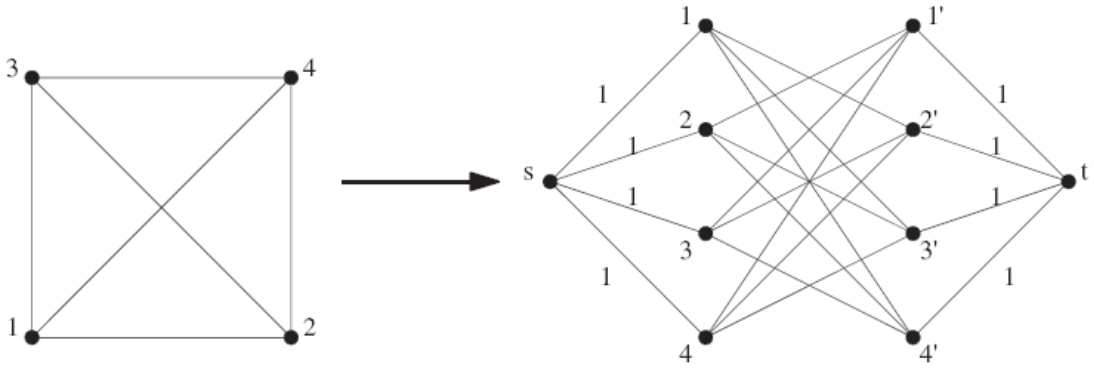

Abbildung 5.1: Konstruktion des schiefsymmetrischen Graphen aus dem Ausgangsgraph

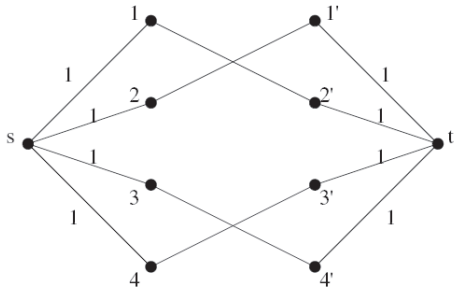

Abbildung 5.2: Ganzzahlige Lösung des Min Cost Flow Problems auf dem Graphen; es sind nur die Kanten eingezeichnet, die einen Fluss von 1 tragen

Die Kapazität sei auf allen Kanten 1. Der maximale Fluss ist dann offensichtlich 4 und eine ganzzahlige Lösung minimaler Kosten hat folgende Form (Abb. 5.2).
Der Fluss ist balanciert und hat die Kosten $1^2 + 1^2 + 1^2 + 1^2 = 4$. In dem Bild (Abb. 5.2) sehen wir zudem, dass die ganzzahlige Lösung immer auch ein Matching liefert.
Lassen wir nun die Ganzzahligkeitsbedingung fallen, haben wir aber folgende Lösung des Min Cost Flow Problems (Abb. 5.3).
Der Fluss ist balanciert, aber nicht ganzzahlig und hat Kosten $12 \cdot \left(\frac{1}{3}\right)^2 = \frac{12}{9} = \frac{4}{3}$.

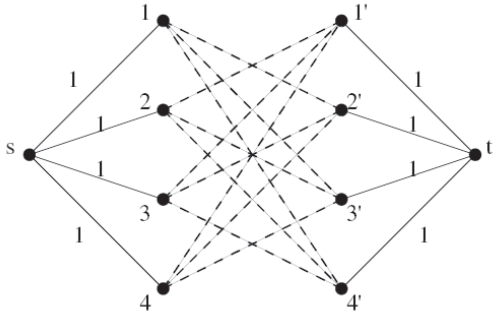

Abbildung 5.3: Lösung des Min Cost Flow-Problems ohne Ganzzahligkeitsbedingung; hierbei stehen die gestrichelten Kanten für Kanten mit $\frac{1}{3}$ Fluss, durchgezogene Kanten stehen wieder für Kanten mit Fluss 1

Dieses Beispiel zeigt auch, dass wir noch nicht einmal den zunächst bewiesenen Satz zu Anstee's Algorithmus über die Ecken des Polytops verwenden können. Maximale Flüsse können nicht nur auf den Ecken des Polytops auftreten, also müssen auch die Lösungen mit minimalen Kosten nicht notwendigerweise halbzahlig sein.
Damit ergeben sich für die Konstruktion von Algorithmen 2 Möglichkeiten. Wir können entweder die Flüsse nur ganzzahlig erhöhen und arbeiten mit obigen Restriktionen weiter oder wir müssen die Ganzzahligkeitsforderung in ein neues Polytop „einbauen".
Der erste Algorithmus, den wir präsentieren, wählt den 2. Weg. Wir werden die andere Möglichkeit aber später auch wählen.
Das Ziel ist dabei Schnitte zu finden, die die ungeraden Kreise einer Pseudobasis aufspüren

und damit die geradzahligen Lösungen von den Pseudobasen unterscheiden [26, S. 211]. Aus dem am Ende der Beschreibung von Anstee's Algorithmus bewiesenen Lemma ergibt sich direkt das folgende Korollar:

Korollar 5.3. *Sei N ein schiefsymmetrisches Netzwerk, x eine balancierte Zirkulation auf N und $V(N) = U \biguplus \bar{U}$ mit einer selbstkomplementären Menge U, dann ist $x(U, \bar{U})$ gerade.*

Bevor wir nun die Ungleichungen vom schiefen Schnitt herleiten können, brauchen wir noch folgende Definition:

Definition 5.4 (nichtrivial)**.** *Eine Knotenmenge $U \subseteq V(N)$ heißt nichttrivial, wenn $N[U]$ (die Einschränkung des Netzwerks auf die Knoten U) und $N[\bar{U}]$ mindestens einen ungeraden Kreis enthält und $N[U]$ verbunden ist.*

Korollar 5.5 (Ungleichungen vom schiefen Schnitt)**.** *Für $V(N) = U \biguplus \bar{U}$ mit einer selbstkomplementären Menge U und $(U, \bar{U}) = A_1 \biguplus A_2$ sei*

$$scap(A_1, A_2) := l(A_2) - cap(A_1) \tag{5.8}$$

Definiere $O_{skew}(N) := \{(A_1, A_2) : A_1 \biguplus A_2$ nichttrivial, $scap(A_1, A_2)$ ungerade$\}$, dann erfüllt jede Zirkulation die Ungleichungen

$$x(A_2) - x(A_1) \geq scap(A_1, A_2) + 1 \quad \text{für} \quad (A_1, A_2) \in O_{skew}(N) \tag{5.9}$$

Wir nennen den Schnitt (A_1, A_2) streng, wenn er die Ungleichung mit Gleichheit erfüllt.

Beweis. Es gilt $x(A_2) - x(A_1) = x(U, \bar{U}) - 2x(A_1)$ mit $x(U, \bar{U}) = x(A_2) + x(A_1)$. $x(U, \bar{U}) - 2x(A_1)$ ist aber aufgrund des vorherigen Korollars gerade. Der Rest ist trivial. □

Wir nennen das Polytop, das zusätzlich zu den Ungleichungen für $\mathfrak{F}(N)$ auch die Ungleichungen vom schiefen Schnitt enthält das *Polytop der balancierten Zirkulationen* $\mathfrak{P}(N)$. In der Arbeit von Jungnickel und Fremuth-Paeger [26] wird nun gezeigt, dass dieses tatsächlich die konvexe Hülle aller balancierten Zirkulationen ist. Damit enthält dieses Polytop die nötigen Bedingungen für die Ganzzahligkeit.
Wir wollen an dieser Stelle noch einmal auf unser Beispiel zum Polytop der rationalen Zirkulationen in (Abb. 4.15) zurückkommen. Die Ecke, die dem Fluss in (b) entspricht, ist offensichtlich nicht ganzzahlig. Alle anderen Ecken erfüllen die Ganzzahligkeitsbedingung. Das Polytop der rationalen balancierten Zirkulationen wäre also in unserem Beispiel wieder 10-dimensional, hätte aber nur noch 2 Ecken. Wir hätten also eine Gerade im 10-dimensionalen Raum. Projezieren wir diese wie zuvor wieder in den 2-dimensionalen Raum, so erhalten wir die (Abb. 5.4).
Es sei hierbei angemerkt, dass die Notation der Ungleichungen nicht völlig eindeutig ist, da die „Reihenfolge" der Mengen A_1 und A_2 nicht festgelegt ist. Es kann somit auch andere Notationen dieser Ungleichungen geben [26, S. 211].
Ferner ist die Einführung von *scap* und O_{skew} nicht notwendig, sondern erleichtert nur die Definition des strengen Schnitts. Wir könnten auch für alle Schnitte $O(N)$ fordern:

$$x(A_2) - x(A_1) \geq 2 \left\lceil \frac{l(A_2) - cap(A_1)}{2} \right\rceil \tag{5.10}$$

Die zusätzlichen Ungleichungen für Schnitte mit geradem $scap(A_1, A_2)$ sind dann natürlich redundant.

Abbildung 5.4: Polytop der balancierten Zirkulationen für das schiefsymmetrische Beispielnetzwerk

5.2 Grundlegende Sätze zur Optimalität

Nachdem wir nun genauer definiert haben, auf welchen Polytopen wir Flüsse optimieren wollen, kommen wir nun zu Kriterien für Optimalität. Aus der Einführung von Kosten auf Graphen ergibt sich nicht nur das Problem der Kostenminimierung eines Flusses, sondern auch das Problem kürzeste gültige Pfade in einem Netzwerk zu bestimmen (Shortest Valid Path).
Allerdings sind sowohl das Shortest Valid Path Problem (SVP), als auch das zuvor studierte Max Bal Flow Problem einfacher als Bal Min Cost Flow (BMCF), da beim Max Bal Flow nur die Kapazitäten und nicht die Kanten und bei SVP nur die Kosten und nicht die Kapazitäten betrachtet werden [36, Kapitel 13, S. 2-3].
Ferner hatten wir bereits beim Balanced Decomposition Satz darauf hingewiesen, dass Zirkulationen und Flüsse eng verwandt sind. Wir wollen die Verwandtschaft zwischen Zirkulationen minimaler Kosten und Flüssen minimaler Kosten durch das folgende Lemma noch etwas formalisieren [36, Kapitel 13, S. 1].

Lemma 5.6. *Balanced Min Cost Circulation (BMCC) und Balanced Min Cost Flow (BMCF) sind äquivalente Probleme.*

Beweis. Wir reduzieren die Probleme jeweils aufeinander, wobei die Reduktionen keine Veränderung der Laufzeit des jeweiligen Algorithmus bedeuten.
Die Reduktion von BMCC auf BMCF ist einfach die Addition von 2 unverbundenen Knoten zum Problem: Der Quelle s und der Senke t (Abb. 5.5). Der Fluss ist damit 0, da von der Quelle zur Senke kein Fluss geschickt werden kann. Der kostenminimale Fluss auf dem Netzwerk ist damit eine kostenminimale Zirkulation auf dem Netzwerk ohne die Knoten t und s. Der BMCF Algorithmus liefert also eine Lösung des BMCC Problems.

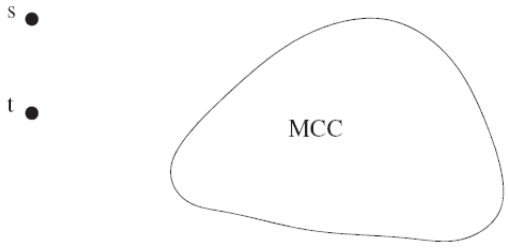

Abbildung 5.5: Reduktion des BMCC Problems auf BMCF

Nun reduzieren wir BMCF auf BMCC. Dazu fügen wir eine Kante (ts) mit unendlicher Kapazität und unendlich negativen Kosten hinzu. Die Lösung des Min Cost Circulation

Problems wird dann einen kostenminimalen maximalen Fluss von s nach t berechnen, was auch in folgender Abbildung illustriert ist (Abb. 5.6).

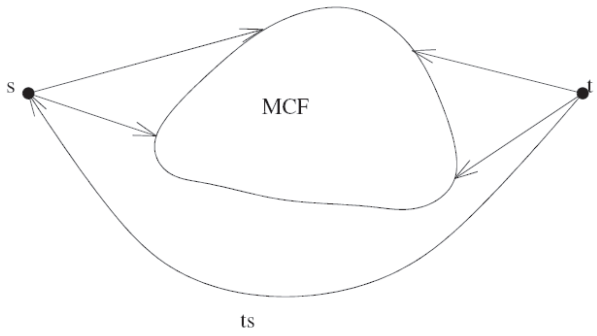

Abbildung 5.6: Reduktion des BMCF Problems auf BMCC

Unendlich ist aber keine Zahl und auch keine, die wir einem Computer geben könnten. In einer wirklichen Implementierung dieses Konzepts müssen wir daher auf die bereits eingeführte maximale Kapazität einer Kante in einem Netzwerk U und die maximalen Kosten einer Kante in einem Netzwerk C zurückgreifen. Wir setzen dann die Kosten auf die minimal möglichen Kosten eines Weges im Netzwerk $-nC$ und die Kapazität auf die maximale Kapazität des Netzwerks mU. □

Wir wollen nun zunächst bei linearen Kosten Optimalitätskriterien herleiten [70, pp. 6-8]. Hierbei betrachten wir zunächst das Problem auf $\mathfrak{F}(x)$:

$$\min C(x) = \sum_{(ij)\in A(N)} c_{ij}x_{ij} \tag{5.11}$$

$$l(ij) \leq x_{ij} \leq cap(ij) \tag{5.12}$$

$$x_{ij'} = x_{ji'} \tag{5.13}$$

$$e(i) = 0 \; \forall i \in V(N) \tag{5.14}$$

Satz 5.7 (Negative Cycle Optimality)**.** *Ein zulässiger Fluss x ist genau dann eine optimale Lösung des BMCF Problems, wenn das Restnetzwerk $N(x)$ keinen gültigen Kreis mit negativen Kosten enthält.*

Beweis. „⇒“ Angenommen x sei ein zulässiger Fluss und $N(x)$ enthalte einen negativen Kreis, dann können die Kosten durch Erhöhen des Flusses längs dieses Kreises verringert werden.
„⇐“ Nehmen wir an, y sei ein optimaler Fluss und x sei ein zulässiger Fluss. Da beide Flüsse gültig sind, muss $y - x$ eine Flusszirkulation sein, die wir nach dem Balanced Decomposition Satz in gültige Kreise zerlegen können. Diese Kreise liegen folglich in $N(x)$ und die Kosten sind $C(y) - C(x)$. Da alle Kreise in $N(x)$ nichtnegativ sind, gilt $C(y)-C(x) \geq 0$. Wegen der Optimalität von y gilt aber $C(y) \leq C(x) \Rightarrow C(y) = C(x)$. □

Ein weiteres Optimalitätskriterium ergibt sich aus der Betrachtung reduzierter Kosten. Diese wollen wir zunächst definieren:

Definition 5.8 (Potential, reduzierte Kosten)**.** *Die Potentialfunktion $\pi : V \to \mathbb{R}$ assoziiert mit jedem Knoten $i \in V$ eine Zahl $\pi(i)$, das Potential von i.*
Die reduzierten Kosten einer Kante sind dann definiert durch: $c_{ij}^{\pi} = c_{ij} - \pi(j) + \pi(i)$.

Wir notieren die Potentialfunktion manchmal auch als Vektor der Potentiale $\pi \in \mathbb{R}^n$. Wir bezeichnen den Vektor π dabei auch als die *duale Lösung* und den Vektor x als die *primale* Lösung. Im Anhang dieser Arbeit werden wir genauer den Zusammenhang zwischen primaler und dualer Lösung erläutern. Die Anleihen hierzu stammen aus der linearen Programmierung und dem Lagrange-Formalismus. An dieser Stelle benötigen wir diese weitergehenden Betrachtungen allerdings nicht.
Wir wollen statt dessen im nächsten Abschnitt einige anschauliche Interpretationen der Potentiale angeben.
Wir notieren nun noch einige Eigenschaften der reduzierten Kosten, deren Beweis wir uns hier sparen:

Korollar 5.9. *Es gilt für die reduzierten Kosten auf einem Netzwerk:*

- *für einen Pfad p von k nach l:*

$$\sum_{(ij)\in p} c_{ij}^{\pi} = \sum_{(ij)\in p} c_{ij} - \pi(l) + \pi(k) \tag{5.15}$$

- *für jeden Kreis K:*

$$\sum_{(ij)\in K} c_{ij}^{\pi} = \sum_{(ij)\in K} c_{ij} \tag{5.16}$$

Satz 5.10 (Reduced Cost Optimality)**.** *Sei x ein gültiger Fluss, dann ist x genau dann optimal, wenn es ein Potential π gibt, so dass für alle Kanten $(ij) \in N(x)$ gilt:*

$$c_{ij}^{\pi} \geq 0 \tag{5.17}$$

Beweis. „⇐“ Wir nehmen an, wir hätten ein Potential π mit $c_{ij}^{\pi} \geq 0 \ \forall (i,j) \in N(x)$. Daraus folgt:

- alle gültigen Kreise in $N(x)$ haben bzgl. der reduzierten Kosten keine negativen Kosten
- mit der vorherigen Eigenschaft der reduzierten Kosten hat $N(x)$ damit auch bzgl. der Originalkosten keine negativen Kreise.

Damit folgt aus dem Satz über die Negative Cycle Optimality auch die Optimalität von x.
„⇒“ Sei x eine Lösung des BMCF-Problems. Wir müssen nun also zeigen, dass ein Potential für diesen Fluss existiert mit $c_{ij}^{\pi} \geq 0 \ \forall (i,j) \in N(x)$. Wir hatten zuvor bereits die schiefsymmetrischen Distanzen $d(i)$ für Knoten $i \in V(N)$ eingeführt. Nach der Definition der schiefsymmetrischen Abstände gilt $d(j) \leq d(i) + c_{ij} \ \forall (i,j) \in N(x)$, da $|p|$ gerade als

Summe der Kosten definiert ist und zwischen i und j eine Kante mit c_{ij} besteht. Damit haben wir aber auch:

$$c_{ij} + (d(i)) - (d(j)) \geq 0 \tag{5.18}$$
$$\Leftrightarrow c_{ij}^{\pi} \geq 0 \text{ für } \pi = d \tag{5.19}$$

□

Abschließend wollen wir noch die Complementary Slackness Optimality oder zu deutsch die Bedingung vom komplementären Schlupf betrachten. Im Gegensatz zu den vorangegangenen Optimalitätsbedingungen bezieht sich diese auf das Original- und nicht auf das Restnetzwerk.

Satz 5.11 (Complementary Slackness Optimality)**.** *Sei x ein zulässiger Fluss, dann ist x optimal genau dann, wenn es ein Knotenpotential π gibt, so dass für alle $(ij) \in A(N)$ gilt:*

- $c_{ij}^{\pi} > 0 \Rightarrow x_{ij} = l(ij)$
- $l(ij) < x_{ij} < cap(ij) \Rightarrow c_{ij}^{\pi} = 0$
- $c_{ij}^{\pi} < 0 \Rightarrow x_{ij} = cap(ij)$

Beweis. Zum Beweis der Complementary Slackness Optimality verwenden wir die Reduced Cost Optimality.
„⇒“ Nehmen wir zunächst an x sei ein optimaler Fluss, dann folgt aus der Reduced Cost Optimality, dass ein Potential $\pi : V(N) \to \mathbb{R}$ mit $c_{ij}^{\pi} \geq 0$ für alle Kanten $(ij) \in A(N(x))$ existiert.
Damit gilt für die 3 betrachteten Fälle:

- $c_{ij}^{\pi} > 0 \Rightarrow (ji) \notin A(N(x))$, da $c_{ji}^{\pi} = -c_{ij}^{\pi} < 0$. Da somit die Gegenkante nicht eingeführt wurde, kann man den Fluss auf dieser Kante nicht mehr verringern und es muss gelten $x_{ij} = 0$.
- Falls $l(ij) < x_{ij} < cap(ij)$, dann sind (ij) und (ji) in $A(N(x))$. Mit der Reduced Cost Optimality folgt $c_{ij}^{\pi} \geq 0$ und $c_{ji}^{\pi} \geq 0$. Zusammen mit $c_{ij}^{\pi} = -c_{ji}^{\pi}$ folgt $c_{ij}^{\pi} = c_{ji}^{\pi} = 0$.
- Falls $c_{ij}^{\pi} < 0$, so folgt aus der Reduced Cost Optimality, dass $(ij) \notin A(N(x)) \Rightarrow x_{ij} = cap(ij)$.

„⇐“ Sei nun π so gegeben, dass die obigen 3 Bedingungen der Complementary Slackness Optimality erfüllt sind, dann wollen wir $c_{ij}^{\pi} \geq 0 \; \forall (ij) \in A(N(x))$ zeigen. Wir unterscheiden dazu wieder die 3 Fälle:

- Sei $c_{ij}^{\pi} > 0$, so erfüllt (ij) die Reduced Cost Optimality. (ji) wird in diesem Fall nach der Complementary Slackness Optimality gar nicht eingeführt.
- Sei $l(ij) < x_{ij} < cap(ij)$ und $c_{ij}^{\pi} = c_{ji}^{\pi} = 0$, dann ist offensichtlich auch in diesem Fall die Reduced Cost Optimality erfüllt.

- Gilt $c_{ij}^{\pi} < 0$, so folgt $x_{ij} = cap(ij)$. Damit ist die Kante aber nicht mehr im Restnetzwerk. Für (ji) gilt dann $c_{ji}^{\pi} = -c_{ij}^{\pi} > 0$ und (ji) erfüllt die Reduced Cost Optimality.

Damit haben wir auch die Rückrichtung gezeigt. □

5.3 Interpretation der dualen Lösung

Wie angekündigt wollen wir uns zum Abschluss dieses Kapitels möglichen Interpretationen der Potentiale und der reduzierten Kosten zuwenden. Es sollen hier 3 Möglichkeiten zur Sprache kommen: Der Stromkreis, der Wanderer im Gebirge und die Lagerkostenminimierung. Dabei soll illustriert werden, dass die duale Lösung nicht nur ein mathematisches Konstrukt ist, sondern oft reale Bedeutung hat. Außerdem soll eine anschaulichere Vorstellung von der dualen Lösung vermittelt werden.
Die direkteste Interpretation ist vermutlich die Interpretation des Netzwerks im Stromkreis wie bei Gramacy [33, S. 9]. Das Netzwerk stellen wir uns in diesem Fall als Stromkreis vor mit der Quelle s als $-$-Pol und der Senke t als $+$-Pol. Die Kanten repräsentieren wir durch Widerstände, wobei die Kosten der Stärke R_{ij} des Widerstands entsprechen. Die Knoten sind jeweils Punkte, an denen Spannungsmessgeräte angeschlossen sind, die die Spannung zwischen den Knoten i und s messen. Den Fluss auf den Kanten interpretieren wir als den elektrischen Fluss I_{ij} durch den Widerstand R_{ij}. Ein Beispiel für diese Interpretation ist die Abbildung (Abb. 5.7).

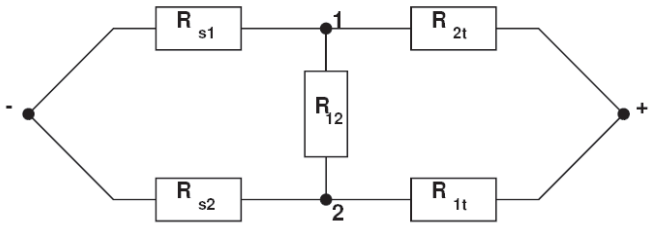

Abbildung 5.7: Interpretation des Netzwerks als Stromkreis

Die Kostenfunktion hat damit die Form $\sum_{(ij)\in A(N)} I_{ij}R_{ij}$. Aus der Schule kennen wir noch das Ohmsche Gesetz $U = R \cdot I$ für die Spannung. Berechnen wir also einen minimalen Kostenfluss, so minimieren wir die Gesamtspannung. Die Natur macht (natürlich) dasselbe um die elektrische Energie $E_{elektr.} = U \cdot I$ im Stromkreis mit festgelegtem Strom (durch Quelle und Senke) zu minimieren. Folglich entsprechen die Potentiale $\pi(i)$ der optimalen Lösung gerade den am Knoten i abgenommenen Spannungen.
Auch die Bedingung $e(i) = 0$ bekommt nun sofort eine Bedeutung, wenn wir sie mit Strömen notieren:

$$\sum_{(ji)\in A(N)} I_{ji} - \sum_{(ij)\in A(N)} I_{ij} = 0 \tag{5.20}$$

Dies ist nicht anderes als Kirchhoffs Gesetz (oder für Quantenfeldtheoretiker: Die Impulserhaltung am Vertex im Feynmangraphen).

Eine andere Interpretation der Potentiale als Höhenmeter stammt von Deppner [18]. Wir betrachten dazu einen Wanderer in einem Gebirge, wobei die Knoten im Netzwerk jeweils bestimmten Stationen entsprechen sollen. Die Potentiale seien jeweils die potentielle Energien des Wanderers der Masse m auf der Station i der Höhe h_i $m \cdot g \cdot h_i$ mit der der Gravitationskonstante g. Die Potentialdifferenz entspricht dann also gerade der Veränderung der potentiellen Energie des Wanderers. Die Kanten entsprechen dann möglichen Wanderpfaden zwischen den Stationen. Die Kosten auf diesen interpretiert man als die Energie, die der Wanderer auf dem Weg von i nach j verbraucht. Diese ist der Potentialdifferenz nicht notwendigerweise identisch, wie jeder wissen wird, der schon einmal durch ein Tal gewandert ist oder einen Berg mit einem Skilift hinaufgefahren ist.
Die reduzierten Kosten c_{ij}^{π} entsprechen mit dieser Interpretation gerade der Energiebilanz auf dem Wanderweg von i nach j. Folglich können wir uns den kürzesten Pfad dann als den Pfad vorstellen, bei dem der Wanderer am wenigsten Energie benötigt.
Eine dritte Interpretation stammt von Bertsekas [10, S. 162-163]. Wir nehmen an, ein Speditionsunternehmen habe verschiedene Läger, die den Knoten des Netzwerks entsprechen sollen. Die Transportwege zwischen diesen Lägern entsprechen folglich den Kanten und die Kosten c_{ij} einer Kante (ij) den Transportkosten einer Stückeinheit zwischen den Lägern i und j. Das Potential $\pi(i)$ interpretieren wir als die Lagerkosten einer Stückeinheit im Lager i.
Der Fluss x_{ij} steht dann für die zwischen i und j ausgetauschten Stückeinheiten. Schickt man nun von i zu j x_{ij} Stückeinheiten, so spart der Spediteur die Lagerkosten $\pi(i)x_{ij}$, muss aber die Lagerkosten $\pi(j)x_{ij}$ und die Transportkosten $c_{ij}x_{ij}$ bezahlen. Folglich ergibt sich für ihn die Kostenbilanz:

$$c_{ij}x_{ij} + \pi(j)x_{ij} - \pi(i)x_{ij} = c_{ij}^{\pi} \tag{5.21}$$

Nehmen wir an, der Spediteur handle gewinnmaximierend, so ergeben sich 3 Fälle, wenn der Standort der Waren keine Rolle spielt:

(i) Gilt $c_{ij}^{\pi} > 0$, so wird er keinen Transport vornehmen.

(ii) Gilt $c_{ij}^{\pi} = 0$, so ist es ihm egal.

(iii) Gilt $c_{ij}^{\pi} < 0$, so wird er möglichst viele Stückeinheiten ($cap(ij)$) austauschen.

Dies ist aber gerade die Complementary Slackness Optimality.

Kapitel 6

Primal-Dualer Algorithmus

Wir wollen nun den ersten Ansatz zur Lösung des gestellten Problems vorstellen. Der primal-duale Algorithmus für lineare Kosten wurde von Jungnickel und Fremuth Paeger vorgeschlagen [27]. Wir wollen hier zeigen, dass man den Algorithmus auch zur Lösung des Problems mit konvexen Kosten verwenden kann.

6.1 Formulierung des Problems

Wir optimieren im Falle des primal-dualen Algorithmus die Kosten auf dem zuvor definierten Polytop $\mathfrak{P}(x)$. Wir betrachten zunächst den Fall positiver linearer Kosten ($c_{ij} > 0 \; \forall \; (ij) \in A(N)$) und haben damit folgendes Problem:

$$\begin{aligned}
\min C(x) &= \sum_{(ij)\in A(N)} c_{ij}x_{ij} && (6.1)\\
l(ij) &\leq x_{ij} \leq cap(ij) && (6.2)\\
x_{ij'} &= x_{ji'} && (6.3)\\
e(i) &= 0 \; \forall i \in V(N) && (6.4)\\
x(A_2) - x(A_1) &\geq scap(A_1, A_2) + 1 \; \forall (A_1, A_2) \in O_{skew}(N) && (6.5)
\end{aligned}$$

Da wir hier zusätzlich die Ungleichungen vom schiefen Schnitt haben, müssen wir leider die zuvor hergeleiteten Optimalitätsbedingungen etwas modifizieren, da wir auch für die möglichen Schnitte „Potentiale“ $\theta \in \mathbb{R}^{|O_{skew}(N)|}$ definieren müssen. Wir führen dazu zunächst den *Vektor χ^{A_1,A_2} eines schiefen Schnitts* (A_1, A_2) ein:

$$\chi^{A_1,A_2} := \begin{cases} +1 & \text{falls } (ij) \in A_1 \\ -1 & \text{falls } (ij) \in A_2 \\ 0 & \text{sonst} \end{cases} \qquad (6.6)$$

Damit gilt für den Vektor eines schiefen Schnitts: $\chi^{A_1,A_2}(\overline{ij}) = -\chi^{A_1,A_2}(ij)$. Damit können wir nun die sogenannte *modifizierte Länge* definieren, die den reduzierten Kosten entspricht:

$$c^{\theta}_{\pi}(ij) := c_{ij} + \pi(i) - \pi(j) + \sum_{(A_1,A_2)\in O_{skew}(N)} \chi^{A_1,A_2}(ij)\theta(A_1, A_2) \qquad (6.7)$$

Mit dieser Definition erhalten wir nun ganz analog zur vorherigen Diskussion den Satz:

Satz 6.1 (modifizierte Complementary Slackness Optimality)**.** *Sei x eine balancierte Zirkulation auf einem schiefsymmetrischen Netzwerk N. Dann ist x optimal genau dann, wenn es ein Paar (π, θ) gibt, so dass:*

$$c_\pi^\theta(ij) \geq 0 \text{ für } rescap(ij) > 0 \tag{6.8}$$

$$\theta(A_1, A_2) = 0 \text{ falls } (A_1, A_2) \in O_{skew}(N) \text{ nicht streng} \tag{6.9}$$

Hierbei können wir das Paar (π, θ), wie die dualen Variablen sehen.
Wir brauchen hier allerdings kein explizites duales Problem wie in der Linearen Optimierung. Statt dessen fordern wir auch für (π, θ) symmetrisch zu sein mit $\theta \geq 0$, $\pi(i) = -\pi(i')$ und $\theta(A_1, A_2) = \theta(A_1', A_2')$ für jeden schiefen Schnitt $(A_1, A_2) \in O_{skew}(N)$. Wir haben also quasi eine symmetrische duale Lösung.
Wir nennen nun x und ein Paar (π, θ), die die Complementary Slackness Bedingungen erfüllen, ein *komplementäres Paar*.
Wir definieren ferner:

$$ucap_0(ij) := \begin{cases} cap(ij) & \text{falls } c_\pi^\theta(ij) \leq 0 \\ l(ij) & \text{falls } c_\pi^\theta(ij) > 0 \end{cases} \tag{6.10}$$

$$lcap_0(ij) := \begin{cases} l(ij) & \text{falls } c_\pi^\theta(ij) \geq 0 \\ cap(ij) & \text{falls } c_\pi^\theta(ij) < 0 \end{cases} \tag{6.11}$$

Wir bezeichnen das Netzwerk aus $V(N)$, $A(N)$ und den Kapazitätsbeschränkungen $ucap_0$ und $lcap_0$ mit N_π^θ oder einfach N_0. $N_0(x)$ enthält nach obiger Wahl der Kapazitäten mit einem komplementären Paar $(x, (\pi, \theta))$ offenbar nur Kanten mit modifizierter Länge 0.
Aus der Complementary Slackness Optimality folgt damit auf dem Netzwerk N_0:

Korollar 6.2. *Seien x und (π, θ) optimal, dann ist x gültig für N_0.*

Beweis. Man betrachte die 3 Fälle der Complementary Slackness Optimality. N_0 besteht gerade aus den Kanten, für die der zweite Fall zutrifft. Damit ist der Beweis offensichtlich. □

6.2 Idee des primal-dualen Algorithmus

Die Idee des primal-dualen Algorithmus ist nun ähnlich der der Balanced Augmentation. Diese wird allerdings ergänzt um Veränderungen der dualen Lösung.
Aufgrund der Complementary Slackness Optimality können wir, ohne die Optimaliät zu verlieren, die Flüsse nur auf den Kanten verändern, die in N_0 liegen. Es liegt also die Idee nahe auf N_0 gültige Pfade zu suchen und dann den Fluss so zu erhöhen, dass auch der neue Fluss x und das alte Paar (π, θ) ein komplementäres Paar bilden.
Hierzu führen wir den Begriff des 0-Pfads ein. Ein *0-Pfad* ist ein gültiger Pfad auf $N_0(x)$, der $(x_p + x_{p'})\chi^{A_1, A_2} = 0$ für jeden strengen Schnitt $(A_1, A_2) \in O_{skew}(N)$ erfüllt.
Das folgende Lemma ist dabei wichtig:

Lemma 6.3. *Seien x und (π, θ) ein komplementäres Paar und p ein 0-Pfad. Dann bildet $y := x + x_p + x_{p'}$ mit (π, θ) ebenfalls ein komplementäres Paar.*

Beweis. Offensichtlich ist y auf N_0 zulässig. Damit erfüllt y auch $c_\pi^\theta(ij) \geq 0$ für Kanten $(ij) \in A(N)$ mit $rescap_x(ij) > 0$. Wir müssen nur noch nachweisen, dass auch die zweite Bedingung aus der Complementary Slackness Optimality erfüllt ist. Dazu sei $(A_1, A_2) \in O_{skew}(N)$. Dann haben wir:

$$\begin{aligned} y(A_2) - y(A_1) &= x(A_2) + (x_p + x_{p'})(A_2) - x(A_1) - (x_p + x_{p'})(A_1) && (6.12) \\ &= x(A_2) - x(A_1) - (x_p + x_{p'})\chi^{A_1,A_2} && (6.13) \end{aligned}$$

Ist (A_1, A_2) streng, so gilt nach der Definition eines 0-Pfads $(x_p + x'_p)\chi^{A_1,A_2} = 0$. Damit folgt die Gültigkeit der Ungleichung aus dem Polytop aus der Ungleichung vom schiefen Schnitt für x. □

Wir bezeichnen x auch als *extremal*, wenn x die optimale Lösung bezüglich aller anderen balancierten Flüsse mit gleicher Flussrate ist.
Die Bestimmung eines 0-Pfads ist nun sehr eng verwandt mit der Bestimmung kürzester Pfade.

Satz 6.4. *Sei x ein balancierter st-Fluss und (π, θ) das symmetrische duale Paar, so dass x und (π, θ) ein komplementäres Paar bilden. Dann ist x extremal.*
Ist p ein 0-Pfad von s nach t, so ist p auch ein kürzester gültiger st-Pfad (SVP) und $c(p) = -2\pi(s)$.

Beweis. Wir wollen auch hier zunächst den Fluss in eine Zirkulation umwandeln, indem wir die Kante (ts) hinzufügen und setzen $l(ts) = x_{ts} = cap(ts) := val(x)$. Dann sind x und (π, θ) immer noch komplementär. Nach der Complementary Slackness Optimality muss x damit extremal sein.
Wenden wir nun das obige Lemma an, dann ist $y := x + (x_p + x_{p'})$ ebenfalls extremal. Dann muss aber, damit die Kosten von y minimal sind, p ein gültiger Pfad gewesen sein. Wir kommen damit zu den Kosten. Für jeden Pfad p in $N_0(x)$ haben wir $c_\pi^\theta(p) = 0$. Alle Potentiale bis auf $\pi(s)$ und $\pi(t)$ heben sich bei der Summation der reduzierten Kosten längs des Pfades auf, also gilt:

$$\begin{aligned} c_\pi^\theta(p) &= c(p) + \pi(s) - \pi(t) = \sum_{(ij)\in p} c_{ij} + 2\pi(s) = 0 && (6.14) \\ \Rightarrow c(p) &= -2\pi(s) && (6.15) \end{aligned}$$

Dies vervollständigt den Beweis. □

Der Satz zeigt damit, dass das SVP-Problem mit dem Problem einen 0-Pfad zu finden eng verwandt ist. Wir stützen uns deshalb in der folgenden Diskussion auch auf die Arbeit von Goldberg und Karzanov zum SVP-Problem [30].
Die Idee des dort beschriebenen Algorithmus ist nun entweder einen SVP zu finden, wobei eventuell die Potentiale auf den Knoten verändert werden müssen oder zu zeigen, dass ein SVP nicht existiert.
Wir nennen dabei die Veränderung der Potentiale auch das *duale Update*.
Für den Algorithmus von Goldberg und Karzanov ist nun der folgende Begriff grundlegend.

Definition 6.5 (Barriere)**.** *Wir nennen ein $k+1$-Tupel $\mathfrak{B} = (A;\ X_1,\ \cdots,\ X_k)$ eine Barriere, falls die folgenden Bedingungen erfüllt sind:*

- *$Y,\ X_1,\ \cdots,\ X_k$ sind paarweise disjunkte Teilmengen von $V(N)$ und $s \in Y$.*
- *$Y \cap Y' = \oslash$*
- *für $l = 1,\ \cdots,\ k$ sind die X_l selbstkomplementär*
- *für $l = 1,\ \cdots,\ k$ gibt es eine eindeutige Kante (il) mit $rescap(il) = 1$ von Y zu X_l.*
- *für $M = V - (Y \cup X_1 \cup \cdots \cup X_k)$ und $l = 1, \cdots, k$ verbindet keine Kante X_l und M.*
- *keine Kante verbindet Y und $Y' \cup M$*

Diese Definition sei in der folgenden Abbildung illustriert (Abb. 6.1).

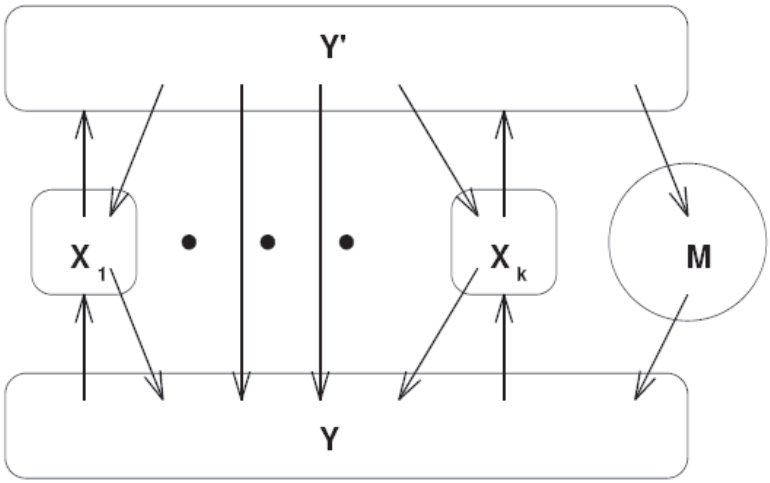

Abbildung 6.1: Illustration einer Barriere

Die Wichtigkeit des Begriffs erläutert der folgende Satz.

Satz 6.6. *Sei N ein schiefsymmetrisches Netzwerk, dann existiert genau dann ein gültiger st-Pfad, falls das Netzwerk keine Barriere hat.*

Beweis. Es ist einfach zu zeigen, dass ein Netzwerk mit einer Barriere keinen gültigen Pfad besitzt, denn hätte es einen solchen, so müsste dieser Y und Y' verbinden. Damit müsste der gültige Pfad über mindestens ein $X_l,\ l \in \{1, \cdots, k\}$ verlaufen. Die X_l sind aber selbstkomplementär und nur über eine Kante (il) mit $rescap(il) = 1$ zu erreichen. Folglich müssen sie auch über eine Kante $(l'i')$ mit $rescap(l'i') = rescap(il) = 1$ verlassen werden. Damit würde der gültige Pfad aber mindestens ein Paar komplementärer Kanten mit Restkapazität 1 enthalten.
Die Rückrichtung können wir nun mit einem Konstruktionsbeweis zeigen. Nehmen wir dazu an, es existiere kein gültiger st-Pfad. Wir wollen nun eine Barriere konstruieren. Sei

dazu S die Menge aller von s auf gültigen Pfaden erreichenbaren Knoten. Wir definieren dann:

$$Y := S - S' \tag{6.16}$$
$$Y' := S' - S \tag{6.17}$$
$$X := S \cap S' \tag{6.18}$$
$$M := V(N) - (S \cup S') \tag{6.19}$$

Wir bezeichnen nun die schwach verbundenen Komponenten des von X induzierten Untergraphen des schiefsymmetrischen Digraphen mit $K_1, \cdots, K_h$. Die $X_l,\ l \in \{1, \cdots, h\}$ sind dann gerade die Knotenmengen der $K_i,\ i \in \{1, \cdots, h\}$. Damit können wir die *kanonische Barriere* $\mathfrak{B} = (Y; X_1, \cdots, X_h)$ definieren.
Unter einer *schwach verbundenen Komponente* verstehen wir hierbei einen maximalen Untergraphen K eines schiefsymmetrischen Digraphen D_G, so dass für jedes Paar von Knoten $i, j \in V(K)$ ein gerichteter Pfad von i zu j und ein ungerichteter Pfad von j zu i existiert oder andersherum [53, S. 172].
Wir müssen nun natürlich noch beweisen, dass diese kanonische Barriere auch tatsächlich eine Barriere im Sinne unserer Definition ist. Für diesen Teil des Beweises verweisen wir auf die Arbeit von Goldberg und Karzanov [30, S. 356-357]. □

Es sei hierbei darauf hingewiesen, dass diese Definition einer Barriere nicht als Analogon zur Definition einer Barriere im Rahmen der Bedingung von Tutte [68] für ein perfektes Matching gesehen werden darf. Die Anzahl der $X_i,\ i \in \{1, \cdots, h\}$ hat hier keine Bedeutung.
Neben dem Begriff der Barriere benötigen wir für den Algorithmus noch den Begriff des Fragments.
Aus dem vorgestellten BNS-Algorithmus kennen wir die Begriffe der Blüte und des Kerns. Eine weitere Verallgemeinerung ist nun der Begriff des Fragments. Ein *Fragment* ist ein Paar $(U, (ij'))$ mit einer selbstkomplementären Menge U, dem sogenannten *Inneren* und einer Kante $(ij') \in N[\bar{U}, U]$, der *Stütze* (prop). Blüten und Kerne sind dabei typische Beispiele für Fragmente. Für diese hatten wir aber den Begriff der Stütze noch nicht eingeführt.
Statt dessen hatten wir uns bisher auf den Begriff der Basis für Kerne beschränkt. Für die Stütze (ij') eines Kerns ist j' gerade die Basis des Kerns. Man kann nun für Blüten und Kerne auch zeigen, dass es tatsächlich nur eine Kante $(ij') \in A(N)$ mit der Eigenschaft $rescap(ij') = 1$ gibt, die auf jedem gültigen Pfad zu Knoten im Kern bzw. in der Blüte liegt. Diese Kante ist dann gerade die erwähnte Stütze $prop(U) = (ij')$.
Den Begriff der Blüte und des Kerns haben wir eingeführt, da sie die BNS Algorithmen „stören“. Wir schrumpfen sie daher (bzw. behandeln sie nur durch ihre Basis) und arbeiten auf einem verkleinerten Netzwerk weiter. Ganz analog behandeln wir hier nun die Fragmente, wobei wir noch erläutern werden, was wir hier mit „stören“ meinen. Zunächst besprechen wir aber das Schrumpfen eines Fragments.
Eine Schrumpfoperation überführt das Originalnetzwerk N in ein neues Netzwerk $\bar{N}$, indem die Kanten aus dem Innern des Fragments gelöscht werden und die Knoten des Fragments durch w und w' ersetzt werden. Hierbei gelten für das Schrumpfen folgende Regeln:

- Die Stützte (kl') wird durch eine Kante (kw') ersetzt, die Co-Stütze $(kl')'$ wird ersetzt durch (wk').
- Jede Kante $(ij') \neq (kl')'$, die U verlässt wird durch eine Kante (wj') ersetzt.
- Jede Kante $(ij') \neq (kl')$, die in U hineinführt, wird durch eine Kante (iw') ersetzt.
- Jede Kante in U wird durch eine Kante $(w'w)$ ersetzt.

Die Operation wird in der folgenden Abbildung (Abb. 6.2) veranschaulicht.

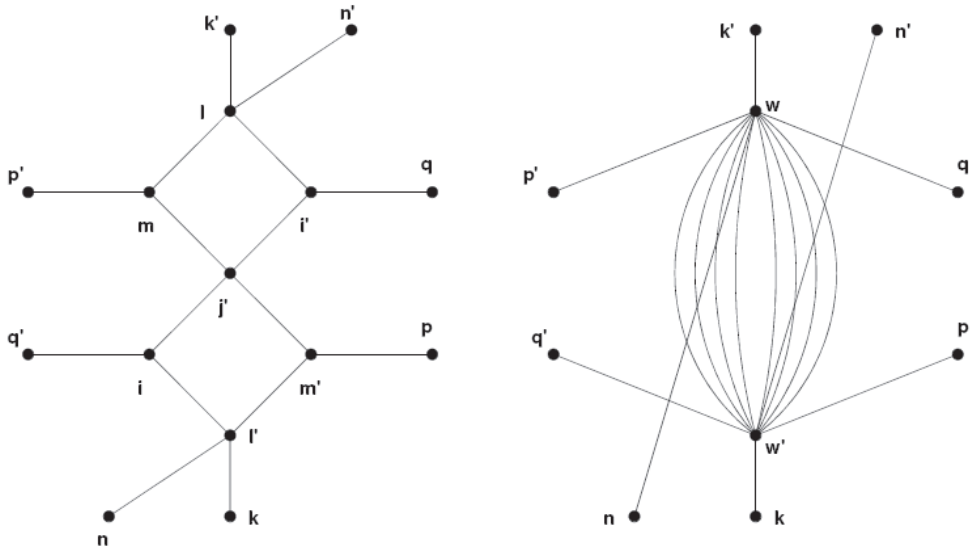

Abbildung 6.2: Schrumpfen von Fragmenten

Der Graph bleibt trotz der Operation offenbar schiefsymmetrisch. Er bleibt sogar schiefsymmetrisch, wenn wir die $(w'w)$-Kanten nicht einführen, aber das sei hier lediglich als technisches Detail erwähnt.
Wir notieren ferner, dass die Menge aller Fragmente $\mathfrak{S}$ eine verschachtelte Familie bildet, also für 2 Fragmente $(U,(kl)),\ (W,(mn)) \in \mathfrak{S}$ gilt $U \subset W$ oder $W \subset U$ oder $U \cap W = \oslash$.
Ein Netzwerk, in dem alle maximalen Fragmente aus einer Familie $\mathfrak{S}$ geschrumpft wurden, bezeichnen wir als *Oberflächengraph*. Hierbei sei angemerkt, dass die Schrumpfoperationen kommutieren und folglich ein Netzwerk $\bar{N}$ existiert, in dem alle Fragmente geschrumpft sind.
Wir müssen nun, bevor wir zum Algorithmus kommen, noch die Verbindung zwischen schiefen Schnitten und Fragmenten aufzeigen. Der erste Schritt ist das folgende Lemma.

Lemma 6.7. *Sei U eine Blüte oder ein Kern im Netzwerk $N_0(x)$, dann gibt es einen strengen schiefen Schnitt mit dem Innern U.*

Beweis. Für $(kl) = prop(U)$ gilt $rescap((kl)) = rescap(\overline{(kl)'}) = 1$. Sei nun $(ij) \in N[U,\bar{U}]\backslash\{(kl)\}$, so muss gelten $rescap(ij) = 0$ oder $rescap(\overline{(ij)}) = 0$, denn sonst gilt

$c_\pi^\theta(ij) = 0$ aufgrund der Komplementarität von x und (π, θ). Dann wäre (ij) aber bieulersch in $N_0(x)$. Wir können also $N[U, \bar{U}]$ schreiben als $N[U, \bar{U}] = A_1 \biguplus A_2 \biguplus \{(kl)\}$ mit:

$$A_1 := \{(ij) \in N[U, \bar{U}] : rescap(ij) = 0,\ (ij) \neq (kl)\} \tag{6.20}$$
$$A_2 := \{(ij) \in N[U, \bar{U}] : rescap(ij) > 0,\ (ij) \neq (kl)\} \tag{6.21}$$

Wir definieren die Flüsse $x(A_1) = cap(A_1)$ und $x(A_2) = l(A_2)$. Ist (kl) eine Kante in Vorwärtsrichtung, so nehmen wir (kl) in A_1 auf und ansonsten in A_2.
Wir brauchen nur noch zu zeigen, dass $l(A_2) - cap(A_1)$ ungerade ist, was aber sofort aus dem Korollar folgt, wonach $x(A_2) - x(A_1)$ gerade ist. Damit haben wir, da (kl) keine komplementäre Kante in $N[U, \bar{U}]$ hat, einen strengen schiefen Schnitt. □

Wir können dieses Konstruktionsprinzip nun auf beliebige Fragmente $(U, (ij))$ erweitern und nennen den resultierenden schiefen Schnitt *assoziiert* mit dem Fragment $(U, (ij))$. Wir drücken die Assoziation eines Fragments mit einem schiefen Schnitt durch einen Doppelpfeil $(U, (ij)) \leftrightarrow (A_1, A_2)$ aus.

6.3 Bestimmung von 0-Pfaden

Nach den Vorarbeiten kommen wir nun zum Herzstück des primal-dualen Algorithmus: Der Berechnung kürzester Pfade unter der Annahme positiver Kosten auf den Kanten und dem dualen Update. Hierbei folgen wir wieder der Darstellung in [30]. Wir stellen zunächst die Bestimmung eines 0-Pfads dar. Wir nehmen also an, die Fragmente $(U,\ (kl'))$ aus $\mathfrak{S}$ würden eine verschachtelte Familie bilden und jedes maximale Fragment erfülle die Vorraussetzungen:

- s inzidiert mit keiner Kante aus U
- $c_\pi^\theta(kl') = 0$
- k ist erreichbar in $N_0(x)$ auf einem Pfad disjunkt von allen Fragmenten

Wir definieren nun $\bar{\mathfrak{S}}$ als die Menge aller Fragmente mit maximalem U.
Der Algorithmus sucht nun in N_0 einen Pfad zu t analog zum Algorithmus von Kocay und Stone, indem er einen Baum $T \subset A(N_0)$ konstruiert. Die Menge der Knoten Y des Baumes erfüllt dabei $Y \cap Y' = \oslash$ und

- für jedes Fragment $(U, (kl')) \in \mathfrak{S}$ ist w' enthalten in Y

Wir beginnen nun mit $\pi = 0$, $\theta = 0$, $T = 0$ und $Y = \{s\}$.
In jeder Iteration sucht der Algorithmus eine Kante $(ij') \in A(N_0)$ mit $i \in Y, j' \notin Y$. Bei der Untersuchung der Kanten müssen wir folgende Fälle unterscheiden:

(i) Ist $j' \notin Y'$, so füge (ij') zu T hinzu. Dies vergrößert Y um ein Element und erhält offensichtlich die Bedingung $A \cap A' = \oslash$.
Nehmen wir nun an, es gelte $j' \in Y'$. In diesem Fall erhalten wir, analog zum BNS-Algorithmus ein Fragment und müssen wieder die Basis bzw. die Stütze des

Fragments aufsuchen. Wir gehen dazu in diesem Algorithmus synchron von i zurück und von j' nach vorn, wobei wir die entstehenden Pfade wieder als p_i und p'_j speichern. Wir stoppen, sobald wir eine Kante (kl') auf p_i oder eine Kante (lk') auf p'_j mit Restkapazität 1 gefunden haben, deren komplementärer Partner bereits auf dem jeweils anderen Pfad gefunden wurde.

(ii) Finden wir kein solches Kantenpaar, so ist s die Basis des neu gefundenen Fragments und die Konkatenation $\tilde{p} = p_i \circ (ij') \circ p'_j$ ein gültiger 0-Pfad in N_0.

(iii) Finden wir hingegen ein solches Kantenpaar (kl'), (lk'), so haben wir den Fall der *MakeBlossom*-Operation aus dem Algorithmus von Kocay und Stone. Über das Innere des Fragments kann nur einer der gültigen Pfade p oder p' laufen. Wir hatten bereits in den Ausführungen zur Schrumpfoperation angesprochen, dass wir hier nicht mit einem base-pointer arbeiten wollen, sondern die Fragmente explizit löschen wollen. Wir betrachten also analog zum BNS-Algorithmus die Teile der Pfade p_i und p'_j, die im Innern des Fragments liegen und definieren die Menge von Kanten dieser beiden Teilpfade als das Innere U des Fragments (die übrigen Kanten müssen wir hier analog zum BNS-Algorithmus nicht betrachten, da wir ja bereits einen gültigen 0-Pfad in U bestimmt haben). Mit der Stütze (kl') führen wir nun die Schrumpfoperation auf dem Fragment $(U, (kl'))$ aus. Das neu entstehende Netzwerk ist, da U selbstkomplementär ist, wieder ein schiefsymmetrisches Netzwerk. Ferner gilt auch mit der neuen Familie der Fragmente $\tilde{\mathfrak{S}}$ für alle Fragmente $(U, (ij')) \in \bar{\tilde{\mathfrak{S}}}$:

(a) s inzidiert mit keiner Kante aus U (denn sonst hätten wir einen st-Pfad gefunden)

(b) $c^{\theta}_{\pi}(ij') = 0$ (denn die Stütze ist in $N_0(x)$ und Fragmente, die in dem neuen Fragment aufgegangen sind, sind nicht in $\bar{\tilde{\mathfrak{S}}}$ enthalten)

(c) i ist erreichbar in $N_0(x)$ auf einem Pfad disjunkt von allen Fragmenten in $\tilde{\mathfrak{S}}$

Daneben bilden die Fragmente in $\tilde{\mathfrak{S}}$ weiterhin eine verschachtelte Familie, analog zu unserer Argumentation bei Kernen. Daneben sind alle Knoten im zuvor definierten Inneren eines Fragments auf einem gültigen 0-Pfad erreichbar.

(iv) Bisher kennen wir alle Operationen bereits aus dem BNS-Problem. Der letzte Fall ist nun, dass es mit der Notation aus der Definition der Barriere, keine Kante von Y nach $V - Y$ gibt. Dann gilt notwendigerweise $c^{\theta}_{\pi} > 0$ für alle Kanten (ij') von Y nach $V - Y$ in N. Sei $M = \bar{V} - (A \cup A')$. Damit sind keine Knoten in M bisher im Algorithmus berücksichtigt worden und damit noch Knoten des Originalnetzwerks. Wir definieren:

$$\epsilon_1 = \min\{c^{\theta}_{\pi}(ij) \ : \ (ij) \in (Y, M)\} \tag{6.22}$$

$$\epsilon_2 = \min\{c^{\theta}_{\pi}(ij) \ : \ (ij) \in (Y, Y')\} \tag{6.23}$$

$$\epsilon = \min\left\{\epsilon_1, \frac{\epsilon_2}{2}\right\} \tag{6.24}$$

Seien $\tilde{Y}$ die Knoten im Ausgangsnetzwerk von Y. Wir führen nun ein Update von π und θ durch:

$$\begin{aligned} \pi(i) &:= \pi(i) - \epsilon\ ,\ i \in \tilde{Y} && (6.25)\\ \pi(i) &:= \pi(i) + \epsilon\ ,\ i \in \tilde{Y}' && (6.26)\\ \theta(A_1, A_2) &:= \theta(A_1, A_2) + \epsilon \text{ für } (A_1, A_2) \leftrightarrow (U, (ij)) \in \bar{\mathfrak{S}} && (6.27) \end{aligned}$$

Man kann nun zeigen, dass das Update c_π^θ folgendermaßen verändert:

$$\begin{aligned} c_\pi^\theta(ij) &:= c_\pi^\theta(ij) - \epsilon\ ,\ \text{ falls } (ij) \in (Y, M) \cup (M, Y') && (6.28)\\ &:= c_\pi^\theta(ij) - 2\epsilon\ ,\ \text{ falls } (ij) \in (Y; Y') && (6.29)\\ &:= c_\pi^\theta(ij) + \epsilon\ ,\ \text{ falls } (ij) \in (M, Y) \cup (Y', M) && (6.30)\\ &:= c_\pi^\theta(ij) + 2\epsilon\ ,\ \text{ falls } (ij) \in (Y', Y) && (6.31) \end{aligned}$$

Damit verändern wir die Kosten keiner Kante in T. Alle Knoten, die bisher auf einem 0-Pfad erreichbar waren, bleiben auf einem solchen erreichbar. Insbesondere gilt dies auch für die Stützen von Fragmenten und das Innere bisher geschrumpfter Fragmente.
c_π^θ bleibt auch für alle Kanten im Netzwerk nichtnegativ. Allerdings gibt es jetzt eine Kante (ij) mit $c_\pi^\theta(ij) = 0$ von Y nach $V - Y$. Damit wird in der nächsten Iteration einer der Fälle (i) - (iii) eintreten. Damit terminiert der Algorithmus in endlicher Zeit und bestimmt den gesuchten 0-Pfad.

Mit einer Implementierung die spezielle Datenstrukturen nutzt, erhalten Goldberg und Karzanov eine Komplexität des Algorithmus von $O(m \min\{\log n,\ \sqrt{\log C}\})$.
Hierbei sei erwähnt, dass wir hier nicht exakt den Algorithmus von Jungnickel und Fremuth-Paeger besprochen haben, sondern wie obig erwähnt, lediglich den Fall nichtnegativer Kosten. Jungnickel und Fremuth-Paeger behandeln den allgemeineren Fall möglicherweise auch negativer Kosten. Hierbei stellen sie die Vermutung auf, dass die obig abgeleitete Komplexität für das duale Update auch im Fall allgemeiner Kosten gelte [27, S. 40]. Sie weisen allerdings auf Probleme bei der dann nötigen Expansion von Fragmenten hin. Wir wollen auf diese Diskussion hier nicht näher eingehen. Es sei hier lediglich angemerkt, dass in unserem Fall die Komplexität unstrittig ist.
Die Implementierung des Primal-Dual Algorithmus ist nun sehr einfach:

```
PD-Algorithmus

    x := 0, π :≡ 0, θ :≡ 0
    until (x := Max Bal Flow)

        bestimme 0-Pfad p in N
        erhöhe den Fluss x := x + (x_p + x_p')
```

Da in jedem Schritt der Fluss so um 2 Einheiten erhöht wird, hängt die Komplexität direkt mit dem maximalen Fluss MF zusammen und ist $O(m \cdot MF \cdot \min\{\log n,\ \sqrt{\log C}\})$.
Auf einen wirklichen Beweis der Korrektheit des Algorithmus wollen wir an dieser Stelle verzichten und verweisen statt dessen noch einmal auf die Arbeit von Jungnickel und Fremuth-Paeger [27, S. 40].

6.4 Enhanced Primal Dual Algorithmus

Der obige einfache Algorithmus hat nur pseudopolynomielle Laufzeit. Jungnickel und Fremuth-Paeger haben nun einen Algorithmus vorgeschlagen, der den primal-dualen mit einem „normalen“ Min Cost Flow Algorithmus kombiniert und so stark polynomielle Laufzeit erhält.
Ein solcher Min Cost Flow Algorithmus ist z.B. das Capacity Scaling von Shamir und Pinto mit einer Komplexität $\mu(n,m) = m \log m(m + n \log m)$ [56]. Es gibt aber noch weitere Möglichkeiten stark polynomieller Algorithmen [46, S. 6].
Die Idee ist hierbei in stark-polynomieller Zeit mit dem „normalen“ MCF-Algorithmus eine Lösung x_0 mit komplementärer Lösung π^0 zu bestimmen. Wir symmetrisieren diese Lösungen mit:

$$x_{ij'} := \frac{1}{2}\left(x_{0_{ij'}} + x_{0_{ji'}}\right) \ \forall (ij') \in A(N) \tag{6.32}$$

$$\pi(i) := \frac{1}{2}\left(\pi_0(i) - \pi_0(i')\right) \ \forall i \in V(N) \tag{6.33}$$

Damit sind π und x auf jeden Fall halbzahlig; x repräsentiert also eine Ecke von $\mathfrak{F}(N)$. Wir erhöhen nun den Fluss auf den ungeraden Kreisen um eine halbe Einheit, um sie ganzzahlig zu machen. Damit haben wir aber notwendigerweise Überschüsse auf Knoten des Kreises. Man verbindet nun die ungeraden Kreise mit künstlichen Kanten zu 2 neu eingeführten Kanten (ml) und $(l'm')$. Den Rückfluss lassen wir über Kanten (mm') laufen. Um die künstlichen Kanten nun wieder entfernen zu können, bestimmen wir nun einen maximalen und kostenminimalen ll'-Fluss mit dem primal-dualen Algorithmus.
Wir geben nun den Enhanced Primal-Dual Algorithmus an und gehen dann kurz auf dessen Korrektheit ein.

Enhanced PD Algorithmus

(i) bestimme MCF x_0 auf N mit dualer Lösung π_0

(ii) setze $\theta :\equiv 0$ und definiere symmetrische Lösung mit:
$x_{ij'} := \frac{1}{2}(x_{0_{ij'}} + x_{0_{ji'}})$
$\pi(i) := \frac{1}{2}(\pi_0(i) - \pi_0(i'))$

(iii) transformiere x in eine Pseudobasis ohne die ganzzahligen Flusswerte auf den Kanten zu verändern.
ermittle die ungeraden Kreise $Q_1, \cdots, Q_{2k}$ von x

(iv) auf jedem ungeraden Kreis Q bestimme ein $i \in V(N)$ mit $\pi(i) \geq 0$
schreibe $Q_i = q_i \circ \overline{q_i}$ mit einem ii'-Pfad q
erhöhe den Fluss auf q_i und q_i' um eine halbe Einheit

(v) füge 2 komplementäre Paare von Knoten m, m' und l, l' hinzu
füge Kanten lm und $m'l'$ mit $x_{lm} = c_\pi^\theta(lm) = 0$ und $cap(lm) = 2 \cdot k$ hinzu
füge 2 parallele Kanten mm' mit $c_\pi^\theta(mm') = 0$ und $x_{mm'} = cap(mm') = k$ hinzu

füge Kanten $m'i$ und $i'm$ mit $c_\pi^\theta(m'i) = 0$ und $x_{m'i} = cap(m'i) = 1$ hinzu allen neuen Kanten wird ein Potential 0 zugewiesen und die modifizierten Längen auf 0 gesetzt

(vi) benutze den normalen Primal-Dual Algorithmus, um einen extremalen Max Bal Flow von l nach l' zu finden

(vii) Ist der Flusswert $2k$, so kann man die künstlichen Kanten fortfallen lassen, ansonsten existiert keine Balanced Circulation.

Man braucht hierbei den 5. Schritt, um den Überschuss bzw. den Defizit auf den Anfangs- und Endknoten der ungeraden Kreise zu entfernen. Wir wollen die Konstruktion hier noch in einer Grafik veranschaulichen (Abb. 6.3).
Wir kommen damit noch zum Beweis der Korrektheit und der Komplexität.

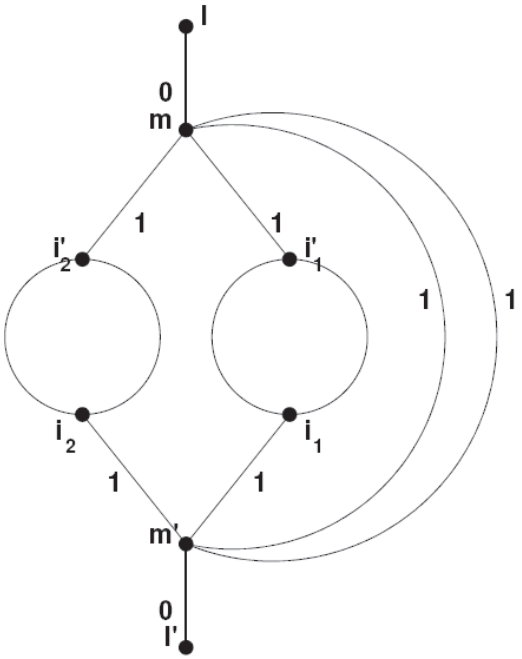

Abbildung 6.3: Transformation im Enhanced Primal-Dual Algorithmus am Beispiel $k = 2$, eingezeichnet sind die anfangs festgelegten Flüsse auf den eingeführten Kanten

Satz 6.8 (Korrektheit und Komplexität des Enhanced Primal-Dual Algorithmus)**.** *Sei $\mu(n, m)$ die Komplexität des verwandten MCF Algorithmus, dann findet der Enhanced Primal-Dual Algorithmus eine balancierte Zirkulation mit minimalen Kosten in einer Zeit $O(\mu(n, m) + nm \min\{\log n, \sqrt{\log C}\})$ oder zeigt, dass keine balancierte Zirkulation existiert.*

Beweis. Wir beweisen zunächst die Korrektheit des Algorithmus. Dazu zeigen wir zunächst, dass der 6. Schritt mit einem komplementären Paar $(x, (\pi, \theta))$ startet. Wir machen dabei

zunächst folgende Beobachtung:

$$\begin{aligned} c_{ij'}x_{ij'} + c_{ji'}x_{ji'} &= \frac{1}{2}c_{ij'}\left(x_{0_{ij'}} + x_{0_{ji'}}\right) + \frac{1}{2}c_{ji'}\left(x_{0_{ji'}} + x_{0_{ij'}}\right) && (6.34)\\ &= \frac{1}{2}c_{ij'}\left(2x_{0_{ij'}} + 2x_{0_{ji'}}\right) && (6.35)\\ &= c_{ij'}x_{0_{ij'}} + c_{ji'}x_{0_{ji'}} && (6.36) \end{aligned}$$

Damit sind die Kosten für die Flüsse x und x_0 gleich. Ferner gilt:

$$\begin{aligned} c^{\pi}_{ij'} &= c_{ij'} + \frac{1}{2}(\pi_0(i) - \pi_0(i')) - \frac{1}{2}(\pi_0(i) - \pi_0(i')) && (6.37)\\ &= \frac{1}{2}(c_{ij'} + \pi_0(i) - \pi_0(j')) + \frac{1}{2}(c_{ji'} + \pi_0(j) - \pi_0(i')) && (6.38)\\ &= \frac{1}{2}c^{\pi_0}_{ij'} + \frac{1}{2}c^{\pi_0}_{ji'} && (6.39)\\ &= c^{\pi}_{ji'} && (6.40) \end{aligned}$$

Es gilt nun $\pi_0 = d$. Damit ist π_0 notwendigerweise symmetrisch und bildet mit der symmetrischen Lösung x ein komplementäres Paar. Da x balanciert aber nicht ganzzahlig ist, haben damit $c^{\pi_0}_{ij'}$, $c^{\pi_0}_{ji'}$ und $c^{\pi}_{ij'}$ alle ein identisches Vorzeichen falls nicht $l(ij') = cap(ij')$ gilt. Damit sind x und π am Ende von Schritt (ii) ein komplementäres Paar.
Die Schritte (iii) und (iv) verändern nur die Flusswerte auf Kanten, die nicht ganzzahlig sind. Da die Kanten damit keine Kapazitätsbeschränkungen übertreten können, bleiben x und π komplementär. Nach diesem Schritt ist x aber ganzzahlig.
Mit Schritt (v) entfernen wir die Überschüsse bzw. Defizite auf den Knoten. Das Netzwerk bleibt, da auch die künstlichen Kanten in komplementären Paaren eingeführt wurden, schiefsymmetrisch und der Fluss balanciert.
Damit endet dieser Schritt mit einer optimalen, balancierten Lösung; allerdings auf einem veränderten Netzwerk. Finden wir nun einen extremalen Fluss von l nach l', so wird entweder der Fluss auf den Kanten von m nach m' gerade ausgeglichen oder nicht. Wird er ausgeglichen, so erhalten wir folglich eine optimale Lösung des BMCF Problems. Wird er nicht ausgeglichen, gibt es keine balancierte Lösung.
Wir kommen damit zur Komplexität. Die einzelnen Phasen (ii), (iv) und (v) haben offensichtlich eine Laufzeit von $O(m)$. Mit einer genaueren Analyse des Algorithmus von Anstee, die wir hier nicht anstreben wollen, findet man auch für die 3. Phase eine Laufzeit von $O(m)$ [27, S. 41]. Maximal gibt es n ungerade Kreise, also auch n Phasen des primal-dualen Algorithmus. Damit folgt die Komplexität. □

6.5 Das Shortest Valid Path Problem

Wir hatten bereits gesagt, dass das SVP-Problem mit nichtnegativen Kosten eng verwandt ist mit dem dualen Update. In der Tat bestimmen wir ja im dualen Update einen gültigen 0-Pfad. Mit dem zuvor bewiesenen Lemma entspricht diesem gerade ein SVP (bei nichtnegativen Kosten) im Originalnetzwerk.
Um diesen zu erhalten müssen wir aber die Fragmente im Netzwerk offensichtlich wieder

expandieren. Goldberg und Karzanov haben gezeigt, dass durch Protokollieren der Fragmentstruktur in einem Wald der Pfad aus dem 0-Pfad in $O(m)$ Zeit rekonstruiert werden kann [30, S. 368-369]. Damit hat das SVP-Problem bei nichtnegativen Kosten ebenfalls die Komplexität $O(m \min\{\log n, \sqrt{\log C}\})$.
Wir werden diesen Algorithmus noch bei der Implementierung des Successive Shortest Path Algorithmus verwenden.
In der Arbeit von Jungnickel und Fremuth-Paeger wird der Fall beliebiger Kosten behandelt. Für das allgemeine SVP Problem geben sie eine Komplexität von $O(mn)$ an [27, S. 40].

6.6 Konvexe Kosten

Bisher haben wir lediglich lineare Kostenfunktionen behandelt. Wir wollen nun eine Methode vorstellen, um mit BMCF-Methoden auch konvexe Kosten zu behandeln. Wir folgen hierbei zunächst der Darstellung in [3, S. 551-554].
Wir nehmen zunächst an, dass wir unsere konvexe Kostenfunktion durch p_{ij} Segmente unterteilt und auf diesen durch lineare Kosten c_{ij}^k approximiert haben. Dies sei in der folgenden Grafik (Abb. 6.4) illustriert.

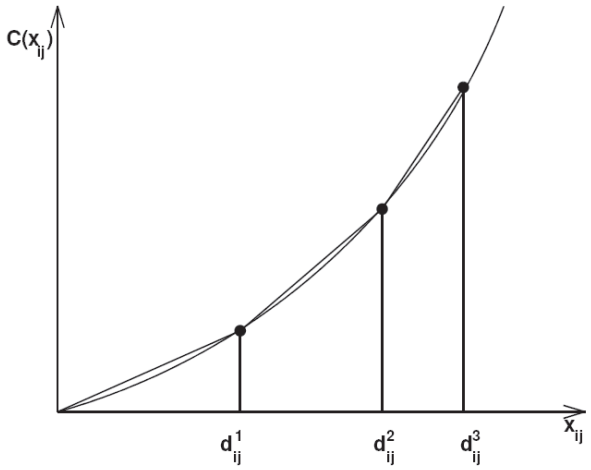

Abbildung 6.4: Lineare Approximation der Kosten am Beispiel von $p_{ij} = 3$ Segmenten

Die Grenzen der einzelnen Segmente seien dabei mit $d_{ij}^k,\ 1 \leq k \leq p_{ij}$ bezeichnet. Hierbei ist die Obergrenze jeweils durch $d_{ij}^k = cap(ij)$ und die Untergrenze durch $l(ij)$ gegeben (in dem Bild willkürlich als 0 gewählt).
Die linearen Kosten erhalten wir dabei als Steigung des entsprechenden Geradenstücks:

$$c_{ij}^k = \frac{C_{ij}(d_{ij}^k) - C_{ij}(d_{ij}^{k-1})}{d_{ij}^k - d_{ij}^{k-1}} \tag{6.41}$$

Wir können nun jeden Fluss x_{ij}, den wir über die Kante schicken wollen, auf die Segmente aufteilen. Nennen wir die Flüsse auf den Segmenten y_{ij}^k, so gilt (mit $d_{ij}^0 = l(ij)$):

$$y_{ij}^k = \begin{cases} d_{ij}^k - d_{ij}^{k-1} & \text{für } x_{ij} \geq d_{ij}^k \\ x_{ij} - d_{ij}^{k-1} & \text{für } d_{ij}^{k-1} \leq x_{ij} \leq d_{ij}^k \\ 0 & \text{sonst} \end{cases} \tag{6.42}$$

Damit gilt natürlich für einen balancierten Fluss im schiefsymmetrischen Netzwerk auch $y_{ij'}^k = y_{ji'}^k$.
Per Definition gilt $x_{ij} = \sum_{k=1}^{p_{ij}} y_{ij}^k$ und $C_{ij}(x_{ij}) = \sum_{k=1}^{p_{ij}} c_{ij}^k y_{ij}^k$. Setzen wir dies ein, erhalten wir das Problem:

$$\begin{aligned}
\min C(x) &= \sum_{(ij)\in A(N)} \sum_{k=1}^{p_{ij}} c_{ij}^k y_{ij}^k && (6.43)\\
y_{ij'}^k &= y_{ji'}^k && (6.44)\\
e(i) &= \sum_{(ij)\in A(N)} \sum_{k=1}^{p_{ij}} y_{ij}^k - \sum_{(ji)\in A(N)} \sum_{k=1}^{p_{ij}} y_{ji}^k = 0 && (6.45)\\
l(ij) &\leq y_{ij}^1 \leq d_{ij}^1 - l(ij) && (6.46)\\
0 &\leq y_{ij}^k \leq d_{ij}^k - d_{ij}^{k-1}, \quad k = 2, \cdots, p_{ij} - 1 && (6.47)\\
0 &\leq y_{ij}^{p_{ij}} \leq cap(ij) && (6.48)
\end{aligned}$$

Dies ist offenbar ein normales BMCF-Problem, allerdings auf einem vergrößerten Netzwerk N'. Wir haben nun für jede Kante zwischen Knoten $i, j \in V(N)$ mit $i \neq s \neq j$, $i \neq t \neq j$ jeweils p_{ij} Kanten $(ij)^1, \cdots, (ij)^{p_{ij}}$ mit den obig angegebenen Kapazitätsgrenzen.
Die Transformation der Kanten wollen wir uns in der Abbildung (6.5) veranschaulichen.

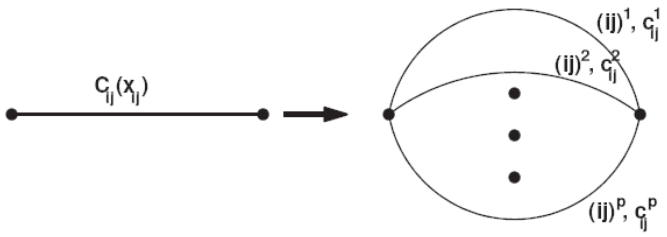

Abbildung 6.5: Illustration der Approximation konvexer Funktionen durch Kantenmultiplizierung

Wir wollen nun noch zeigen, dass der Fluss sukzssive auf den Kanten $(ij)^k, 1 \leq k \leq p_{ij}$ erhöht wird, wobei wir mit sukzessive eine Erhöhung in Reihenfolge der k meinen. Wir bezeichnen dabei die Lösung des modifizierten BMCF Problems als *kontingentierte Lösung*. Eine solche Lösung erfüllt die Bedingung, dass für einen positiven Fluss auf $(ij)^k$ der Fluss auf den Kanten $(ij)^1, \cdots, (ij)^{k-1}$ der Kapazität dieser Kanten entspricht und falls $(ij)^k$ noch freie Kapazität hat, der Fluss auf $(ij)^{k+1}, \cdots, (ij)^{p_{ij}}$ immer 0 ist.

Betrachten wir nun das Problem mit der linearisierten Kostenfunktion, die wir hier mit C_{lin} bezeichnen wollen:

$$\begin{aligned} \min \quad & C_{lin}(x) && (6.49)\\ l(ij) \quad \leq \quad & x_{ij} \leq cap(ij) && (6.50)\\ e(i) \quad = \quad & 0 && (6.51)\\ x_{ij'} \quad = \quad & x_{ji'} && (6.52) \end{aligned}$$

Nehmen wir an, dieses Problem habe eine nicht-kontingentierte Lösung. Dann gibt es mindestens 2 y_{ij}^k, die wir hier mit $k = l, m \;:\; 1 \leq l < m < p_{ij}$ bezeichnen wollen, mit $y_{ij}^l < d_{ij}^l$, $y_{ij}^m < d_{ij}^m$. In diesem Fall können wir aber die Kosten senken, indem wir $\epsilon \in \mathbb{R}$, $\epsilon > 0$ Flusseinheiten über $(ij)^l$ statt über $(ij)^m$ schicken.
Damit haben wir Lösungen zu Problemen auf dem schiefsymmetrischen Netzwerk, die eine linearisierte Kostenfunktion besitzen, aber noch keine Lösung für das eigentliche BMCF-Problem. Hierbei hilft uns die Eigenschaft balancierter Flüsse ganzzahlig zu sein. Sind nur ganzzahlige Flussveränderungen erlaubt, so müssen wir auch die Kostenfunktion nur bei ganzzahligen Flusswerten kennen und können sie auf den Zwischenwerten o.B.d.A. als linear annehmen.
Wir fügen also für jede Kante $cap(ij) - l(ij)$ Kanten ein, die jeweils Kosten $c_{ij}^k = \frac{C_{ij}(d_{ij}^k) - C_{ij}(d_{ij}^k - 1)}{1}$ besitzen. Jede dieser Kanten hat Kapazität 1.
Das so aus dem schiefsymmetrischen Netzwerk entstehende vergrößerte Netzwerk hat allerdings keinen einfachen Graphen mehr, da nun statt einer Kante zwischen den Knoten i und j $cap(ij) - l(ij)$ Kanten existieren. Um das Problem mit einem „normalen“ BMCF-Algorithmus zu lösen, müssen wir also das Problem noch etwas modifizieren.
Bei einer Flusserhöhung müssen wir den Fluss auf den Kanten $(ij) \in A(N)$ minimal um 1 erhöhen, da wir sonst die Ganzzahligkeit verletzen. Da wir aber den Fluss auf p und p' simultan erhöhen wollen, sind mögliche Flusserhöhungen δ auf den Kanten des Netzwerks bei einer Erhöhung des Flusses auf p und p' um 1 $\delta \in \{-2, -1, 0, 1, 2\}$. Erhöhen wir also den Fluss nur minimal, müssen wir nur diese Flusserhöhungen betrachten.
Gleichzeitig muss das Netzwerk einen einfachen Graphen besitzen und schiefsymmetrisch sein. Betrachten wir zunächst für eine beliebige Kante $(ij') \in A(N)$, die keine Schleife ist, die Flusserhöhung bzw. -verringerung um 1. Erhöhen wir den Fluss um 1, so erhöhen wir die Kosten auf der Kante um:

$$C_{ij'}(x_{ij'} + 1) - C_{ij'}(x_{ij'}) =: cc_{ij'} \tag{6.53}$$

Verringern wir hingegen den Fluss um 1, so verändern wir die Kosten auf der Kante um:

$$C_{ij'}(x_{ij'} - 1) - C_{ij'}(x_{ij'}) =: cc_{j'i} \tag{6.54}$$

Wir können die Flusserhöhung um 1 auf der Kante (ij') durch eine Vorwärtskante (ij') mit Kosten $cc_{ij'}$ und Kapazität 1 modellieren. Die Flussverringerung um 1 können wir dann analog durch eine Rückwärtskante $(j'i)$ mit Kosten $cc_{j'i}$ beschreiben.
Für die Flusserhöhungen bzw. -verringerungen um 1 müssen wir also lediglich die Kante (ij') mit ihrer Rückwärtskante (ji') durch eine Vorwärts- und eine Rückwärtskante mit nun unterschiedlichen Kosten ersetzen.

Betrachten wir nun für eine beliebige Kante $(ij') \in A(N)$, die wieder keine Schleife sei, die Flusserhöhung um 2. Eine solche Flusserhöhung tritt ein, wenn (ij') sowohl auf p als auch auf p' liegt. In einem solchen Fall werden 2 Kanten des vergrößerten Netzwerks N' aufgefüllt. Um die Kosten korrekt zu modellieren müssen wir folglich die Kosten auf der letzten Kante betrachten. Diese sind:

$$\frac{C_{ij'}(x_{ij'}+2) - C_{ij'}(x_{ij'}+1)}{1} \tag{6.55}$$

Analog müssen wir bei der Flussverringerung um 2 die Kostenverringerung auf der letzten geleerten Kante betrachten. Diese Kostenverringerung beträgt

$$\frac{C_{ij'}(x_{ij'}-2) - C_{ij'}(x_{ij'}-1)}{1} \tag{6.56}$$

Für die Flusserhöhung bzw. -verringerung um 2 können wir nun aber nicht einfach ein weiteres Kantenpaar (ij') und $(j'i)$ einführen, da der Graph dann nicht mehr einfach wäre. Wir müssen also eine Hilfskonstruktion verwenden, die wir im Folgenden beschreiben wollen.
Wir hatten zuvor bereits auf die Wichtigkeit des Komplementaritätsbegriffs hingewiesen. Offensichtlich ist Komplementarität für Kanten eines schiefsymmetrischen Netzwerks eine Äquivalenzrelation. Wir definieren K als die Menge der Äquivalenzklassen bezüglich Komplementarität für alle Kanten $(ij') \in A(N)$, die keine Schleifen sind.
K enthält also für jedes komplementäre Kantenpaar (ij'), $(ji') \in A(N)$ einen Vertreter.
Für jede Kante $(ij') \in K$ fügen wir nun folgende Knoten zum Netzwerk hinzu:

$$\widetilde{i_{ij'}},\ \widetilde{i'_{ij'}},\ \widetilde{j_{ij'}},\ \widetilde{j'_{ij'}} \tag{6.57}$$

Ferner fügen wir folgende Kanten für jede Kante $(ij') \in K$ hinzu:

$$(i\widetilde{i'_{ij'}}),\ (\widetilde{i'_{ij'}}\widetilde{j_{ij'}}),\ (\widetilde{j_{ij'}}j'),\ (j\widetilde{j'_{ij'}}),\ (\widetilde{j'_{ij'}}\widetilde{i_{ij'}}),\ (\widetilde{i_{ij'}}i') \tag{6.58}$$

Wir wollen diese Konstruktion zunächst anhand einer Abbildung betrachten. Wir betrachten hierzu als Beispiel das Paar komplementärer Kanten $(12')$, $(21')$. In der (Abb. 6.6) sind die, für dieses Kantenpaar hinzugefügten, Kanten in blau eingezeichnet.
Die neu eingeführten Knoten sind also jeweils nur mit einem anderen Knoten verbunden. Wir werden auf die Bedeutung von gestrichelten und ungestrichelten Linien noch eingehen.
In unserem Bild ist nun der Knoten 1 nicht nur über die Kante $(12')$ mit 2' verbunden, sondern auch über den Pfad $p_{12'} = \{(1\widetilde{1'_{12'}})(\widetilde{1'_{12'}}\widetilde{2_{12'}})(\widetilde{2_{12'}}2')\}$. D.h. wir können die Kosten einer Flusserhöhung um 2 auf einer der Kanten auf $p_{12'}$ unterbringen, wenn die Kanten auf $p_{12'}$ eine Kapazität von 2 haben (vorausgesetzt, die Kante im Originalnetzwerk hat genügend freie Kapazität). Wir setzen dazu:

$$cc_{i\widetilde{i'_{ij'}}} = \frac{C_{ij'}(x_{ij'}+2) - C_{ij'}(x_{ij'}+1)}{1} = cc_{\widetilde{i_{ij'}}j'} \tag{6.59}$$

$$cc_{\widetilde{i_{ij'}}i} = \frac{C_{ij'}(x_{ij'}-1) - C_{ij'}(x_{ij'}-2)}{1} = cc_{j'\widetilde{i_{ij'}}} \tag{6.60}$$

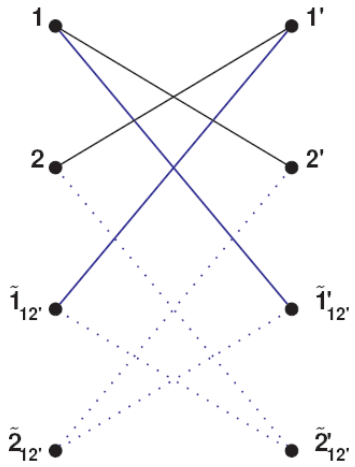

Abbildung 6.6: Illustration der neu eingeführten Kanten und Knoten

Für die übrigen neu eingeführten Kanten setzen wir die Kosten auf 0. Damit auch die reduzierten Kosten 0 sind, setzen wir ferner:

$$\pi(\widetilde{i'_{ij'}}) = \pi(\widetilde{j_{ij'}}) = \pi(i) \tag{6.61}$$

$$\pi(\widetilde{j'_{ij'}}) = \pi(\widetilde{i_{ij'}}) = \pi(j) \tag{6.62}$$

Wir nehmen zudem $\widetilde{i'_{ij'}}$ und $\widetilde{j_{ij'}}$ in das Fragment von i sowie $\widetilde{j'_{ij'}}$ und $\widetilde{i_{ij'}}$ in das Fragment von j auf.
Zum Abschluss betrachten wir nun noch Schleifen. Auf Schleifen kann der Fluss minimal nur um 2 erhöht werden. Wir können es also bei der Einführung einer Kante (ii') mit den Kosten

$$cc_{ii'} = \frac{C_{ii'}(x_{ii'} + 2) - C_{ii'}(x_{ii'} + 1)}{1} \tag{6.63}$$

und der zugehörigen Rückwärtskante $(i'i)$ mit Kosten

$$cc_{i'i} = \frac{C_{ii'}(x_{ii'} - 1) - C_{ii'}(x_{ii'} - 2)}{1} \tag{6.64}$$

belassen. Diese Kapazität der Kanten beträgt 2, soweit die Kante im Originalnetzwerk ausreichend freie Kapazität besitzt.
Das so konstruierte Netzwerk bezeichnen wir als BMCF-Netzwerk. Wir wollen dies nun auch formal definieren. Die Definition ist allerdings wenig erhellend.

Definition 6.9 (BMCF-Netzwerk)**.** *Sei N mit einem Fluss x ein schiefsymmetrisches Flussnetzwerk und K wie zuvor definiert.*
Wir definieren folgende Knotenmenge:

$$\tilde{V}(N) := \{\widetilde{i_{ij'}} | (ij') \in K\} \cup \{\widetilde{i'_{ij'}} | (ij') \in K\} \cup \{\widetilde{j_{ij'}} | (ij') \in K\} \cup \{\widetilde{j'_{ij'}} | (ij') \in K\} \tag{6.65}$$

Ferner definieren wir folgende Kantenmengen:

$$\tilde{A}_2(N) := \{(i\widetilde{i'_{ij'}})|(ij') \in K\} \cup \{(\widetilde{i_{ij'}}i')|(ij') \in K\} \quad (6.66)$$

$$\tilde{A}_0(N) := \{(\widetilde{i'_{ij'}}\widetilde{j_{ij'}})|(ij') \in K\} \cup \{(\widetilde{j_{ij'}}j')|(ij') \in K\}$$
$$\cup\{(j\widetilde{j'_{ij'}})|(ij') \in K\} \cup \{(\widetilde{j'_{ij'}}\widetilde{i_{ij'}})|(ij') \in K\} \quad (6.67)$$

Vorwärts- und Rückwärtskanten weisen eventuell verschiedene Kosten auf. Wir definieren sie folgendermaßen:

$$cc_{ij'} = \frac{C_{ij'}(x_{ij'}+1) - C_{ij'}(x_{ij'})}{1}, \ \forall(ij') \in A(N), \ i \neq j \quad (6.68)$$

$$cc_{ji'} = \frac{C_{ij'}(x_{ij'}) - C_{ij'}(x_{ij'}-1)}{1}, \ \forall(ij') \in A(N), \ i \neq j \quad (6.69)$$

$$cc_{ii'} = \frac{C_{ii'}(x_{ii'}+2) - C_{ii'}(x_{ii'}+1)}{1}, \ \forall(ii') \in A(N) \quad (6.70)$$

$$cc_{i'i} = \frac{C_{ii'}(x_{ii'}-2) - C_{ii'}(x_{ii'}-1)}{1}, \ \forall(ii') \in A(N) \quad (6.71)$$

$$cc_{i\widetilde{i'_{ij'}}} = \frac{C_{ij'}(x_{ij'}+2) - C_{ij'}(x_{ij'}+1)}{1} = cc_{\widetilde{i_{ij'}}i'}, \ \forall(ij') \in K \quad (6.72)$$

$$cc_{\widetilde{i'_{ij'}}i} = \frac{C_{ij'}(x_{ij'}-2) - C_{ij'}(x_{ij'}-1)}{1} = cc_{i'\widetilde{i_{ij'}}}, \forall(ij') \in K \quad (6.73)$$

$$cc_{s\widetilde{i_{ij'}}} = cc_{s\widetilde{j_{ij'}}} = cc_{\widetilde{i_{ij'}}s} = cc_{\widetilde{j_{ij'}}s} = 0 \ \forall(ij') \in K, \ i \neq s, \ j' \neq t \quad (6.74)$$

$$cc_{\widetilde{i'_{ij'}}t} = cc_{\widetilde{j'_{ij'}}t} = cc_{t\widetilde{i'_{ij'}}} = cc_{t\widetilde{j'_{ij'}}} = 0 \ \forall(ij') \in K, \ i \neq s, \ j' \neq t \quad (6.75)$$

Abchließend definieren wir noch die Kapazitätsbeschränkung (wobei wir vereinfachend wieder $l(ij) = 0 \ \forall(ij') \in A(N)$ annehmen wollen):

$$\widetilde{cap} = \begin{cases} 1, \ \textit{falls } cap(ij') - x_{ij'} \geq 1 \\ 0, \ \textit{sonst} \end{cases} \quad \forall(ij') \in A(N), \ i \neq j \quad (6.76)$$

$$\widetilde{cap} = \begin{cases} 1, \ \textit{falls } x_{ij'} \geq 1 \\ 0, \ \textit{sonst} \end{cases} \quad \forall(ij') \in A(N), \ i \neq j \quad (6.77)$$

$$\widetilde{cap}(ii') = \begin{cases} 2, \ \textit{falls } cap(ii') - x_{ii'} \geq 2 \\ 0, \ \textit{sonst} \end{cases} \quad \forall(ii') \in A(N) \quad (6.78)$$

$$\widetilde{cap}(i'i) = \begin{cases} 2, \ \textit{falls } x_{ii'} \geq 2 \\ 0, \ \textit{sonst} \end{cases} \quad \forall(ii') \in A(N) \quad (6.79)$$

$$\widetilde{cap}(i\widetilde{i'_{ij'}}) = \left\{ \begin{array}{l} 2, \ \textit{falls } cap(ij') - x_{ij'} \geq 2 \\ 0, \ \textit{sonst} \end{array} \right\} = \widetilde{cap}(\widetilde{i_{ij'}}i'), \ \forall(ij') \in K \quad (6.80)$$

$$\widetilde{cap}(\widetilde{i'_{ij'}}i) = \left\{ \begin{array}{l} 2, \ \textit{falls } x_{ij'} \geq 2 \\ 0, \ \textit{sonst} \end{array} \right\} = \widetilde{cap}(i'\widetilde{i_{ij'}}), \ \forall(ij') \in K \quad (6.81)$$

$$\widetilde{cap}(s\widetilde{i_{ij'}}) = \widetilde{cap}(s\widetilde{j_{ij'}}) = \widetilde{cap}(\widetilde{i_{ij'}}s) = \widetilde{cap}(\widetilde{j_{ij'}}s) = 0 \ \forall(ij') \in K, \ i \neq s, \ j \neq t \quad (6.82)$$

$$\widetilde{cap}(\widetilde{i'_{ij'}}t) = \widetilde{cap}(\widetilde{j'_{ij'}}t) = \widetilde{cap}(t\widetilde{i'_{ij'}}) = \widetilde{cap}(t\widetilde{j'_{ij'}}) = 0, \ \forall(ij') \in K \ i \neq s, \ j \neq t \quad (6.83)$$

Wir erhalten damit das BMCF-Netzwerk:

$$BN(x) = (V(N) \cup \tilde{V}(N), A(N) \cup \tilde{A}_2(N) \cup \tilde{A}_0(N)) \quad (6.84)$$

Die Kosten eines Flusses $\tilde{x}$ auf $BN(x)$ sind:

$$\sum_{(ij)\in A(BN(x))} cc_{ij}\tilde{x}_{ij} \tag{6.85}$$

$BN(x)$ hat einen einfachen Graphen und ist schiefsymmetrisch.

Wir können also in jedem Schritt das zum schiefsymmetrischen Flussnetzwerk N korrespondierende BMCF-Netzwerk berechnen. Dieses Netzwerk enthält nur die Kanten, die für ein minimales Update des Flusses auf den Kanten um $\delta \in \{-2,-1,0,1,2\}$ notwendig sind. Die mit einer Tilde versehenen Knoten sind dabei lediglich Hilfskonstruktionen, um ein einfaches schiefsymmetrisches Netzwerk zu erhalten. Kosten tragen dabei nur die Kanten, die in unserem Beispiel in (Abb. 6.6) als durchgehende Linien gezeichnet sind.
Um nun unsere langwierige Definition etwas zu erläutern, betrachten wir das Beispiel in (Abb. 6.7). Gezeigt ist dabei ein schiefsymmetrisches Netzwerk, wobei die Tripel $(x_{ij'}, cap(ij'), C_{ij'}(x_{ij'}))$ an den Kanten jeweils den Fluss, die Kapazität und die Kostenfunktion zusammenfassen sollen.

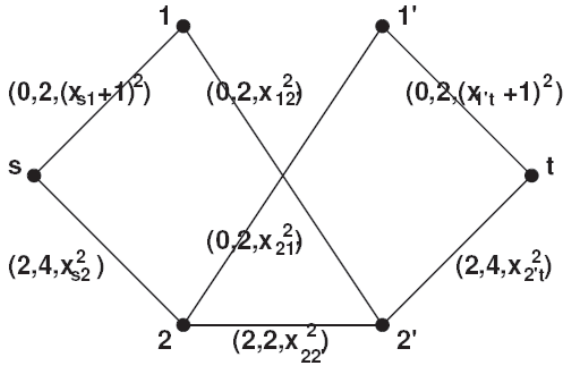

Abbildung 6.7: Beispiel für ein schiefsymmetrisches Netzwerk mit konvexen Kosten

Das BMCF-Netzwerk $BN(x)$ ist in Abbildung (Abb. 6.8) gezeigt, wobei das Dublett $(\widetilde{cap}(ij'), cc_{ij'})$ jeweils die Kosten und die Kapazität zusammenfassen soll. Außerdem sind der Übersichtlichkeit halber nur Kanten eingezeichnet, die auch Kosten tragen können.
Berechnet man ausgehend von $\tilde{x} = 0$ einen Erhöhungsschritt in $BN(x)$, so können wir aus $\tilde{x}$ das Update für den Fluss berechnen. Hierbei entspricht einer Flusserhöhung auf einer Kante $(\widetilde{i_{ij'}i'})$ mit $(ij') \in K$ einer Flusserhöhung um 2 auf der Kante (ji') im Originalnetzwerk. Analog entspricht einer Flusserhöhung um 2 auf der Kante $(\widetilde{ii_{ij'}})$ eine Flusserhöhung um 2 auf der Kante (ij') im Originalnetzwerk. Als Teil des vergrößerten Netzwerks entspricht eine Flusserhöhung um 1 auf (ij') in $BN(x)$ auch eine Flusserhöhung um 1 auf (ij') in $N(x)$. Wir erhalten also x nach der Iteration auf $BN(x_{alt})$ mit $\tilde{x}$ für $(ij') \in K$ folgendermaßen:

$$x_{ij'} = \begin{cases} x^{alt}_{ij'} + \tilde{x}_{ij'}, \text{ falls } \tilde{x}_{ij'} \neq 0 \\ x^{alt}_{ij'} + x_{\widetilde{ii'_{ij'}}}, \text{ falls } \tilde{x}_{\tilde{ij'}} \neq 0,\ (ij') \in K \\ x^{alt}_{ij'} + x_{\widetilde{i_{ij'}i'}}, \text{ falls } \tilde{x}_{\tilde{ij'}} \neq 0,\ (ji') \in K \\ x^{alt}_{ij'}, \text{ sonst} \end{cases} \tag{6.86}$$

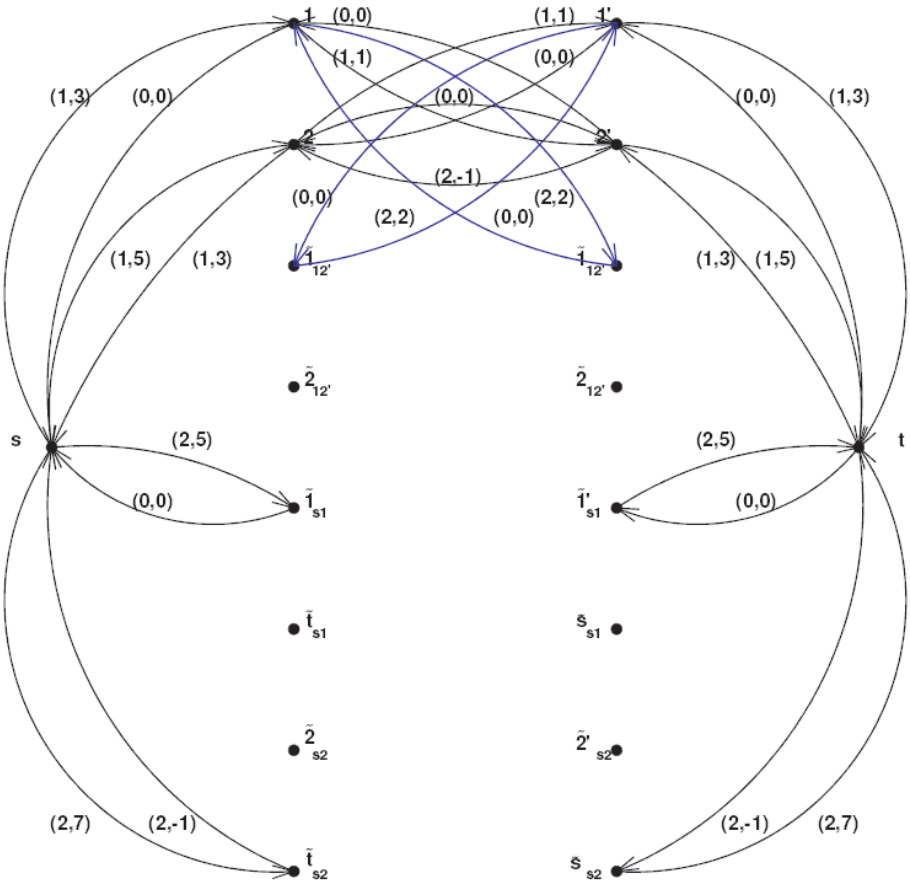

Abbildung 6.8: BMCF-Netzwerk zum vorherigen schiefsymmetrischen Beispielnetzwerk

Und für Schleifen:

$$x_{ii'} = \begin{cases} x_{ii'}^{alt} + \tilde{x}_{ii'}, \text{ falls } \tilde{x}_{ii'} \neq 0 \\ x_{ii'}^{alt}, \text{ sonst} \end{cases} \tag{6.87}$$

Der Fall $\tilde{x}_{ij'} \neq 0$ und $\tilde{x}_{i\widetilde{i'_{ij'}}} \neq 0$ kann nicht auftreten, da $\widetilde{i'_{ij'}}$ nur über i erreichbar ist und folglich i 2 mal auf dem Pfad hätte auftreten müssen. Analog gilt dies auch für $\tilde{x}_{ij'} \neq 0$ und $\tilde{x}_{\widetilde{i_{ij'}}i'} \neq 0$.
Damit können wir das Convex BMCF-Problem mit einem „normalen“ BMCF-Algorithmus lösen, denn in jedem Schritt können wir das Update in $BN(x_{alt})$ bestimmen und den neuen Flus nach obiger Formel aus x_{alt} und dem Fluss $\tilde{x}$ auf $BN(x)$ bestimmen. $BN(x)$ lässt sich dabei aus N in $O(m)$ Zeit bestimmen.
Wir haben allerdings viele neue Kanten eingeführt, sodass eventuell die Komplexität der Algorithmen zur Berechnung kürzester Pfade schlechter werden könnte. Betrachten wir dazu einen beliebigen Pfad p in $BN(x)$. Jeder künstlich eingeführte Knoten, der mit einer Kante auf p inzidiert, kann nur über einen Knoten aus $V(N)$ erreicht werden. Sei also $(ij) \in p$ eine Kante, die mit einem Knoten $i \in V(N)$ beginnt, so wird die übernächste Kante (lk) auf p aufgrund der Struktur von $BN(x)$ mit einem Knoten $k \in V(N)$ inzidieren. Da $|V(N)| = n$ gilt, kann also p maximal $3n$ Kanten enthalten. Da die Anzahl

der Kanten auf einem Pfad in $BN(x)$ also trotz der neu eingeführten Knoten in $O(n)$ ist, verschlechtert sich die Komplexität der Algorithmen zur Suche kürzester Pfade nicht.
Als BMCF-Algorithmus haben wir bereits den primal-dualen Algorithmus kennengelernt. Dieser arbeitet in unserer Version nur mit positiven Kosten. Damit die Kosten aller Vorwärtskanten in $BN(x)$ positiv sind, nehmen wir daher an, die Steigung aller $C_{ij},\ (ij) \in A(N)$ sei stets positiv. Wir wollen uns auch im Folgenden stets auf solche konvexen Kostenfunktionen beschränken.
Wir halten daher noch folgendes Ergebnis fest:

Korollar 6.10 (Komplexität des primal-dualen Algorithmus für Convex BMCF)**.** *Benutzen wir den primal-dualen Algorithmus für Updates im BMCF-Netzwerk $BN(x)$, so können wir eine Lösung des Convex-BMCF Problems in $O(MF \cdot m \cdot \log(n))$ mit dem Max Flow MF erreichen.*

Beweis. In jedem Schritt konstruieren wir $BN(x)$ und führen ein Update mit dem primal-dualen Algorithmus durch. Folglich wird ein Erhöhungsschritt durch die Komplexität des primal-dualen Algorithmus von $O(m \log(n))$ dominiert. Wir erhöhen in jedem Schritt den Fluss im Netzwerk um 1. Es kann also nicht mehr als MF Updates geben, woraus die Komplexität folgt. □

Kapitel 7

Balanced SSP Algorithmus

In diesem Kapitel wollen wir die bei Jungnickel und Fremuth-Paeger [23, S. 12] dargestellte Idee auf kürzesten Pfaden den Fluss zu erhöhen (Successive Shortest Path - SSP) zu einem echten Algorithmus ausformulieren. Bei diesem Algorithmus werden wir auf dem Polytop $\mathfrak{F}(N)$ arbeiten. Wir müssen folglich gesondert sicherstellen, dass wir stets nur ganzzahlig den Fluss erhöhen.
Der Algorithmus ist sehr ähnlich zum primal-dualen Algorithmus. Wir wollen hier allerdings zeigen, dass man tatsächlich lediglich kürzeste Pfade und keine 0-Pfade bestimmen muss, um einen minimalen Kostenfluss zu erhalten. In diesem Sinne ist der Balanced SSP Algorithmus also eine Verallgemeinerung des primal-dualen Algorithmus.

7.1 Successive Shortest Path mit linearen Kosten

Wir bedienen uns einer Idee, die man direkt aus dem Beweis der Reduced Cost Optimality ablesen kann. Dort wird das Potential $\pi = d$, für welches man die optimale Lösung erhält direkt angegeben. Die Konstruktion des Algorithmus können wir damit schon erahnen. Man startet mit einem sogenannten *Pseudofluss*, der zwar optimal ist und die Kapazitätsbeschränkungen erfüllt aber die Massenerhaltungen an den Vertizes verletzt. Der Pseudofluss ist also nicht gültig. Diesen Pseudofluss versuchen wir dann durch Flussverschiebung von Überschuss- zu Defizit-Knoten zulässig zu machen.
Wir erinnern uns in diesem Zusammenhang an die Forderung

$$\sum_{\{j:(ij)\in A(N)\}} x_{ij} + \sum_{\{k:(ki)\in A(N)\}} x_{ki} = 0 \tag{7.1}$$

und an die Forderung für den Überschuss auf s und t: $val(x) = e(s) = -e(t)$. Damit ist unser Netzwerk besonders einfach, da wir mit s und t die einzigen Überschuss- bzw. Defizit-Knoten haben.
Wir wollen nun das für den Algorithmus grundlegende Lemma beweisen. Hierbei inkorporieren wir sogleich, dass für einen zulässigen Pfad p auch p' ein zulässiger st-Pfad ist. Analog zum Algorithmus für einen Max Bal Flow können wir also auch hier auf beiden erweiternden Pfaden parallel den Fluss erhöhen.

Lemma 7.1. *Sei x ein balancierter Pseudofluss, der die Reduced-Cost-Optimality bezüglich eines Potentials π erfüllt. Sei ferner d der Vektor der Distanzen $d(i)$ $\forall i \in V(N)$ von s aus mit reduzierten Kosten c_{ij}^{π} als Länge einer Kante (i,j). Dann gilt:*

(i) Der Pseudofluss x erfüllt die Reduced-Cost-Optimality auch bezüglich des Potentials $\pi' = \pi + d$.

(ii) Die reduzierten Kosten $c_{ij}^{\pi'}$ sind 0 längs der kürzesten Pfade p und p'.

Beweis. (i) Da x die Reduced-Cost-Optimality für π erfüllt, gilt $c_{ij}^{\pi} \geq 0$ $\forall (ij) \in N(x)$. Ferner gilt, da d die Shortest-Path Distanzen mit c_{ij}^{π} als Kantenlängen darstellt, folgende Shortest-Path Optimalitätsbedingung:

$$d(j) \leq d(i) + c_{ij}^{\pi} \; \forall (i,j) \in N(x) \tag{7.2}$$

Setzen wir nun die reduzierten Kosten ein:

$$d(j) \leq d(i) + c_{ij} + \pi(i) - \pi(j) \; \forall (i,j) \in N_M(x) \tag{7.3}$$

$$0 \leq c_{ij} + \underbrace{(\pi(i) + d(i))}_{\pi'(i)} - \underbrace{(\pi(j) + d(j))}_{\pi'(j)} \tag{7.4}$$

$$\Leftrightarrow c_{ij}^{\pi'} \geq 0 \tag{7.5}$$

(ii) p ist ein st-Pfad und p' ist ein $t's'$-Pfad, also nach Definition ebenfalls ein st-Pfad. Es gilt für jede Kante $(ij) \in p$: $d(j) = d(i) + c_{ij}^{\pi}$. Ferner gilt auf p' für (j', i'): $d(i') = d(j') + c_{j'i'}^{\pi} = d(j') + c_{ij}^{\pi}$.
Ersetzen wir nun also $c_{ij}^{\pi} = c_{ij} + \pi(i) - \pi(j)$, so ergibt sich:

$$c_{ij}^{\pi} = d(j) - d(i) \tag{7.6}$$

$$c_{j'i'}^{\pi} = d(i') - d(j') \tag{7.7}$$

$$c_{ij}^{\pi'} = c_{ij}^{\pi} - d(j) + d(i) = 0 \tag{7.8}$$

$$c_{j'i'}^{\pi'} = c_{j'i'}^{\pi} - d(i') + d(j') = 0 \tag{7.9}$$

□

Fassen wir somit noch einmal die Optimalitätsbetrachtungen in folgendem Lemma zusammen:

Lemma 7.2. *Angenommen ein Pseudofluss x erfüllt die Reduced-Cost-Optimality bezüglich eines Potentials π. Wenn wir nun x durch balanciertes Abändern des Flusses längs p und p' erhalten, erhalten wir den neuen Fluss x'.*
Dieser Fluss erfüllt nun ebenfalls die Reduced-Cost-Optimality bezüglich des Potentials $\pi' = \pi + d$.

Beweis. Aus dem vorherigen Lemma wissen wir, dass $c_{ij}^{\pi'} = 0$ für jede Kante (ij) auf dem SVP p und seinem komplementären Partner p' gilt.
Flusserhöhungen auf einer dieser Kanten können unter Umständen bewirken, dass die Gegenkante (ji) eingefügt wird. Da aber $c_{ji}^{\pi'} = -c_{ij}^{\pi'} = 0$ für jede Kante auf p und p' gilt, bleibt die Reduced-Cost-Optimality bezüglich π' erhalten. □

Wir müssen nun noch sicherstellen, dass das Netzwerk nach dem Update weiterhin schiefsymmetrisch bleibt. Nur die Potentiale sind eventuell nach einem Update $\pi' = \pi + d$ nicht mehr antisymmetrisch.

Korollar 7.3 (Symmetrisierung der Potentiale)**.** *Sei $N(x)$ ein schiefsymmetrisches Restnetzwerk und p ein kürzester gültiger Pfad. Es gelte $x' = x + balcap(p)(x_p + x_{p'})$ und $\pi' = \pi + d$. Dann ist $N(x')$ wieder ein schiefsymmetrisches Restnetzwerk, wenn wir die Potentiale π' ersetzen durch:*

$$\pi''(i) := \frac{1}{2}(\pi'(i) - \pi'(i')) \; \forall i \in V(N) \tag{7.10}$$

x' erfüllt die Reduced Cost Optimality auch bezüglich π''.

Beweis. x' ist ein balancierter Fluss, also ist $N(x')$ schiefsymmetrisch, falls $c^{\pi}_{ij'} = c^{\pi}_{ji'}$ gilt. Wir haben dies für das wie obig symmetrisierte Potential bereits im Beweis zur Korrektheit des Enhanced Primal-Dual Algorithmus gezeigt. Wir haben dort ebenfalls gezeigt, dass x' mit π'' die Reduced Cost Optimality erfüllt. □

Wir müssen nun allerdings nicht in jedem Schritt zunächst π' berechnen und dann symmetrisieren. Es reicht aus die Abstände zu symmetrisieren, denn:

$$\pi''(i) = \frac{1}{2}((\pi(i) + d(i)) - (\pi(i') + d(i'))) \tag{7.11}$$

$$= \frac{1}{2}(\pi(i) - \pi(i')) + \frac{1}{2}(d(i) - d(i')) \tag{7.12}$$

Wir definieren daher:

Definition 7.4 (symmetrische Abstände)**.** *Sei $d \in \mathbb{R}^{|V(N)|}$ ein Vektor von schiefsymmetrischen Distanzen, dann nennen wir den Vektor $sd \in \mathbb{R}^{|V(N)|}$ mit*

$$sd(i) = \frac{1}{2}(d(i) - d(i')) \tag{7.13}$$

die symmetrischen Abstände.

Wir können nun, da das Netzwerk nach jedem Update schiefsymmetrisch bleibt, für die Berechnung des SVP die zuvor im primal-dualen Algorithmus betrachtete Routine einsetzen. Ferner wollen wir hier noch kurz an den Balanced Augmentation Algorithmus erinnern: Ist ein gültiger st-Pfad gefunden, so können wir vom Fluss x_i zum Fluss $x_{i+1} = x_i + balcap(p)(x_p + x_{p'})$ übergehen mit der balanced capacity:

$$balcap(p) := \min_{(ij)\in p} \left\{ \begin{array}{c} \lfloor \frac{rescap(ij)}{2} \rfloor \text{ für } (ij)' \in p \\ rescap(ij) \text{ sonst} \end{array} \right\} \tag{7.14}$$

Wir können somit den SSP-Algorithmus folgendermaßen implementieren:

```
Balanced SSP-Algorithm
begin
x = 0, π = 0
while(shortest valid st-path exists)
begin:
```

```
    berechne den shortest-valid-path p von s nach t
    berechne die shortest-path-Distanzen d
    π = π + sd
    aktualisiere die reduzierten Kosten c^π_ij
    δ = balcap(p)
    erhöhe den Fluss längs p und p' um δ Einheiten
    aktualisiere N(x)
    end;
end;
```

Korollar 7.5. *Der mit Hilfe des Balanced SSP-Algorithmus berechnete Fluss ist maximal bei minimalen Kosten.*

Beweis. Der berechnete Fluss ist optimal wegen dem Balanced-Augmentation-Theorem. Am Ende existieren keine erweiternden Wege mehr.
Der berechnete Fluss hat auch minimale Kosten, da die Reduced-Cost-Optimality in jedem Schritt erhalten bleibt. □

Man könnte auf die Idee kommen, dieser Algorithmus sei stark-polynomiell, wie wir es für den Balanced Augmentation Algorithmus mit minimal-gültigen Pfaden beweisen konnten. Wir haben uns aber in der dortigen Argumentation darauf abgestützt, dass alle Kanten gleiche Länge haben, was hier leider nicht der Fall ist.
Bevor wir nun konvexe Kosten betrachten, wollen wir an dieser Stelle den Algorithmus noch einmal anhand eines Beispiel betrachten. Wir betrachten hierzu das schiefsymmetrische Netzwerk in (Abb. 7.1). An jeder Kante ist dabei ein Vektor $(cap(ij), c_{ij})$ notiert, der die Kosten und die Kapazitätsbeschränkung zusammenfasst.

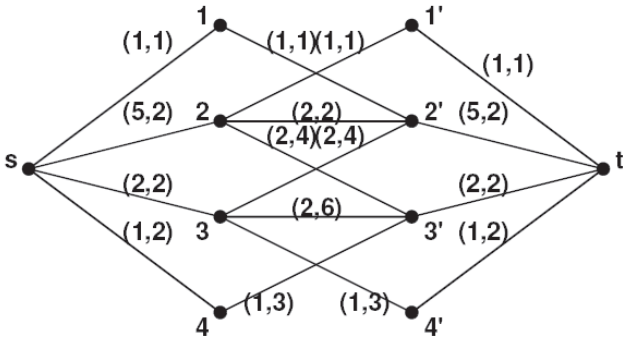

Abbildung 7.1: Beispiel für ein schiefsymmetrisches Netzwerk mit Kosten auf den Kanten

In diesem einfachen Beispiel kann man die kürzesten Pfade direkt „sehen“, so dass wir auf deren Bestimmung hier nicht eingehen wollen. Wir besprechen nun die einzelnen Schritte des Balanced SSP Algorithmus und geben nach jedem Schritt jeweils ein Bild des Restnetzwerks mit an den Kanten eingetragenen reduzierten Kosten und Restkapazitäten als Vektor $(c^{\pi}_{ij}, rescap(ij))$ an.
Im ersten Schritt finden wir den kürzesten Pfad $p_1 = \{(s1), (12'), (2't)\}$. Wir erhöhen den

Fluss längs p_1 und p_1' um $balcap(p) = 1$ Flusseinheit. Die schiefsymmetrischen Distanzen sind $d = (d(s), d(1), \cdots, d(4'), d(s)) = (0, 1, 2, 2, 2, 3, 2, 5, 5, 4)$. Damit ergeben sich die symmetrischen Abstände:

$$sd = (sd(s), \cdots, sd(4'), sd(t)) = \left(-2, -1, 0, -\frac{3}{2}, -\frac{3}{2}, 1, 0, \frac{3}{2}, \frac{3}{2}, 2\right) \tag{7.15}$$

Damit erhalten wir die Potentiale $\pi = sd$.
Wir erhalten also das Restnetzwerk in (Abb. 7.2).

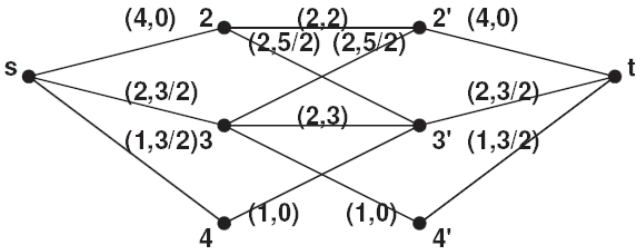

Abbildung 7.2: Restnetzwerk nach dem ersten Schritt

Im nächsten Schritt finden wir den kürzesten Pfad $p_2 = \{(s2), (22'), (2't)\}$. Wir erhöhen den Fluss längs p_2 und p_2' um $balcap(p_2) = 1$ Flusseinheit. Die schiefsymmetrischen Distanzen sind:

$$d = \left(0, \infty, 0, \frac{3}{2}, \frac{3}{2}, \infty, 2, \frac{3}{2}, \frac{3}{2}, 2\right) \tag{7.16}$$

Damit erhalten wir die symmetrischen Distanzen:

$$sd = (-1, 0, -1, 0, 0, 1, 0, 0, 0, 1) \tag{7.17}$$

$$\pi = \pi + sd = \left(-3, -1, -1, -\frac{3}{2}, -\frac{3}{2}, 1, 1, \frac{3}{2}, 3\right) \tag{7.18}$$

Wir erhalten damit das Restnetzwerk in (Abb. 7.3).

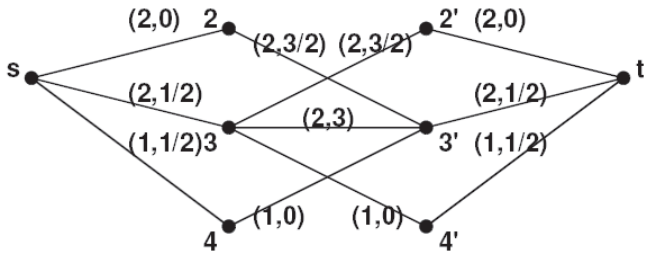

Abbildung 7.3: Restnetzwerk nach dem 2. Schritt

Im dritten Schritt erhalten wir den kürzesten gültigen st-Pfad $p_3 = \{(s3), (34'), (4't)\}$.

Wir erhöhen also wieder den Fluss längs p_3 und p_3' um $balcap(p_3) = 1$ Flusseinheit. Die schiefsymmetrischen Distanzen sind:

$$d = \left(0, \infty, 0, \frac{1}{2}, \frac{1}{2}, \infty, 2, \frac{1}{2}, \frac{1}{2}, 1\right) \tag{7.19}$$

Wir symmetrisieren d wieder:

$$sd = \left(-\frac{1}{2}, 0, -1, 0, 0, 0, 1, 0, 0, \frac{1}{2}\right) \tag{7.20}$$

$$\pi = \pi + sd = \left(-\frac{7}{2}, -1, -2, -\frac{3}{2}, -\frac{3}{2}, 1, 2, \frac{3}{2}, \frac{3}{2}, \frac{7}{2}\right) \tag{7.21}$$

Wir erhalten nun das Restnetzwerk in Abbildung (Abb. 7.4)

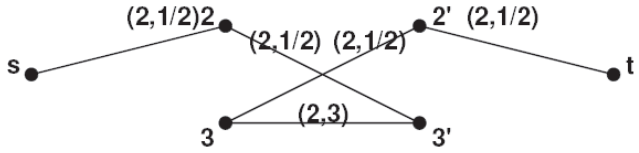

Abbildung 7.4: Restnetzwerk nach dem 3. Schritt

Es sieht so aus, als hätten wir noch den gültigen Pfad $p = \{(s2), (23'), (3'3), (32'), (2't)\}$. Es gilt aber $rescap(3'3) = x_{33'} = 0$ und damit haben wir die Lösung minimaler Kosten erreicht. Wir wollen diese auch einmal in der folgenden Abbildung (Abb. 7.5) explizit angeben, wobei die Flusswerte jeweils an den Kanten notiert sind.

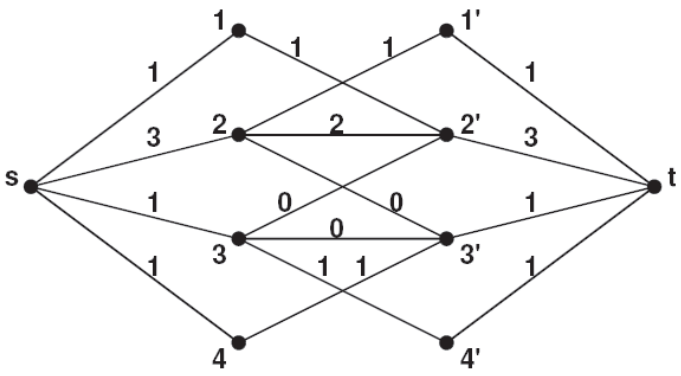

Abbildung 7.5: Netzwerk mit Flusswerten der Lösung minimaler Kosten

7.2 Successive-Shortest-Path Algorithmus mit konvexen Kosten

Wir haben somit den Balanced SSP-Algorithmus für lineare Kosten formuliert. Wir hatten zu Beginn aber einen Algorithmus für konvexe Kostenfunktionen angekündigt.

Dieser ist nun allerdings trivial, denn wir haben in unserer Beschreibung des primal-dualen Algorithmus bereits dargestellt, wie eine „normale“ BMCF-Methode zur Lösung des Convex BMCF-Problems verwendet werden kann. Potentiale und Kosten wollen wir dabei genauso wie im primal-dualen Algorithmus für konvexe Kosten aktualisieren.
Wir geben nun die konkrete Implementierung an:

```
Balanced SSP Algorithm for Convex Costs
begin
x = 0, π = 0
while(shortest valid st-path exists)

    begin
    berechne den shortest-valid-path p in BN(x) von s nach t
    erhöhe den Fluss längs p und p' um 1 Einheit
    aktualisiere den Fluss x
    aktualisiere die reduzierten Kosten cc_ij^π
    aktualisiere BN(x)
    end;

end;
```

Die Komplexität des Balanced SSP Algorithmus für konvexe Kosten ist nun sehr einfach zu ermitteln.

Korollar 7.6 (Komplexität des Balanced SSP Algorithmus für konvexe Kosten)**.** *Der Balanced SSP-Algorithmus für konvexe Kosten berechnet einen ganzzahligen Balanced Min Cost Flow mit maximalem Fluss in* $O(MF \cdot m \log(n))$.

Beweis. Die Korrektheit des Balanced SSP Algorithmus für konvexe Kosten folgt direkt aus der Korrektheit des Balanced SSP Algorithmus für lineare Kosten und unserer Argumentation zu konvexen Kosten beim primal-dualen Algorithmus.
Die Komplexität jedes einzelnen Schritts des Balanced SSP Algorithmus für konvexe Kosten ist dabei offenbar durch die Berechnung des SVP dominiert. Die Distanzen $d(i)$ erhalten wir dabei wie in Satz 6.4 als $\pi(s) - \pi(i)$ ebenfalls aus dem SVP Algorithmus von Goldberg und Karzanov.
Der Fluss wird maximal MF mal um eine Flusseinheit erhöht. Damit haben wir die behauptete Komplexität. □

Zum Abschluss wollen wir auch diesen Algorithmus anhand eines Beispiels betrachten. Wir wählen dazu das Beispiel, welches wir bereits bei der Konstruktion des BMCF-Netzwerks verwandt hatten. Wir starten hierbei natürlich mit dem 0-Fluss. In (Abb. 7.6) ist das schiefsymmetrische Netzwerk noch einmal angegeben, wobei das Dublett $(cap(ij'),\ C_{ij'}(x_{ij'}))$ an den Kanten jeweils die Kapazität und die Kostenfunktion zusammenfassen soll.
Wir konstruieren im Algorithmus zunächst das BMCF-Netzwerk wie in (Abb. 7.7) angegeben.

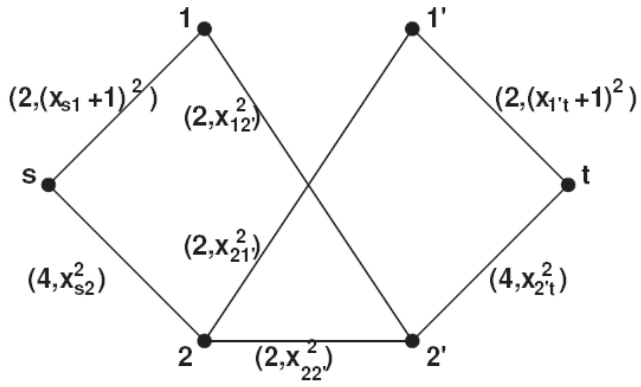

Abbildung 7.6: schiefsymmetrisches Netzwerk mit Kapazitäten und Kostenfunktionen

Auch hier werden wir die kürzesten Pfade nicht explizit berechnen. Aus der Abbildung sehen wir aber den kürzesten Pfad:

$$p_1 = \{(s1), (12'), (2't)\} \tag{7.22}$$

Wir erhöhen also den Fluss längs p_1 und p_1' um 1 Flusseinheit. Die schiefsymmetrischen Distanzen zu Knoten im Originalnetzwerk sind:

$$d = (0, 3, 1, 2, 4, 5) \tag{7.23}$$

Damit erhalten wir die symmetrischen Abstände:

$$sd = \left(-\frac{5}{2}, \frac{1}{2}, -\frac{3}{2}, -\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\right) \tag{7.24}$$

$$\pi = sd \tag{7.25}$$

Wir konstruieren nun wieder das BMCF-Netzwerk, wobei wir wieder an den Kanten jeweils die Kapazität und die reduzierten Kosten angeben (Abb. 7.8).
Der kürzeste Pfad ist wieder:

$$p_2 = \{(s1), (12'), (2't)\} \tag{7.26}$$

Wir erhöhen also wieder den Fluss längs p_2 und p_2' um 1 Flusseinheit. Anschließend berechnen wir wieder die schiefsymmetrischen Distanzen zu den Knoten im Originalnetzwerk.

$$d = (0, 3, 2, 3, 4, 6) \tag{7.27}$$

Ebenso berechnen wir wieder die symmetrischen Abstände und damit das Update der Potentiale:

$$sd = (-3, 0, -1, 0, 1, 3) \tag{7.28}$$

$$\pi = \pi + sd = \left(-\frac{11}{2}, \frac{1}{2}, -\frac{5}{2}, -\frac{1}{2}, \frac{5}{2}, \frac{11}{2}\right) \tag{7.29}$$

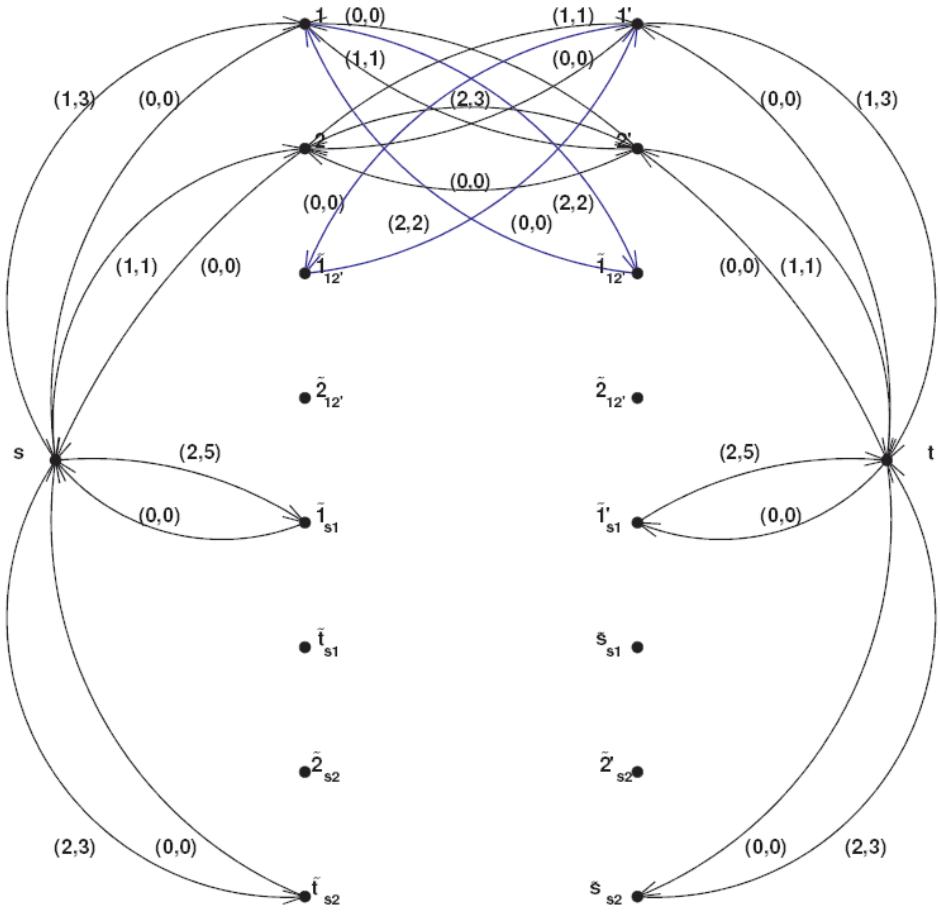

Abbildung 7.7: BMCF-Netzwerk beim Start des Balanced SSP Algorithmus für konvexe Kosten

Damit konstruieren wir nun nochmals das BMCF-Netzwerk (Abb. 7.9).
Der kürzeste Pfad ist nun:

$$p_3 = \{(\widetilde{st}_{s2}), (\widetilde{t_{s2}2'_{s2}}), (\widetilde{2'_{s2}2}), (22'), (2'\widetilde{2_{s2}}), (\widetilde{2_{s2}s_{s2}}), (\widetilde{s_{s2}s})\} \tag{7.30}$$

Wir erhöhen also den Fluss um 1 Einheit längs p_3 und p'_3 und berechnen nun wieder die schiefsymmetrischen Distanzen im Netzwerk:

$$d = (0, \infty, 2, \infty, 5, 7) \tag{7.31}$$

Damit erhalten wir die symmetrischen Abstände:

$$sd = \left(-\frac{7}{2}, 0, -\frac{3}{2}, 0, \frac{3}{2}, \frac{7}{2}\right) \tag{7.32}$$

$$\pi = \left(-9, \frac{1}{2}, -4, \frac{1}{2}, 4, 9\right) \tag{7.33}$$

Wir finden nun keinen gültigen Pfad mehr und haben damit die Lösung des Convex BMCF-Problems bestimmt.

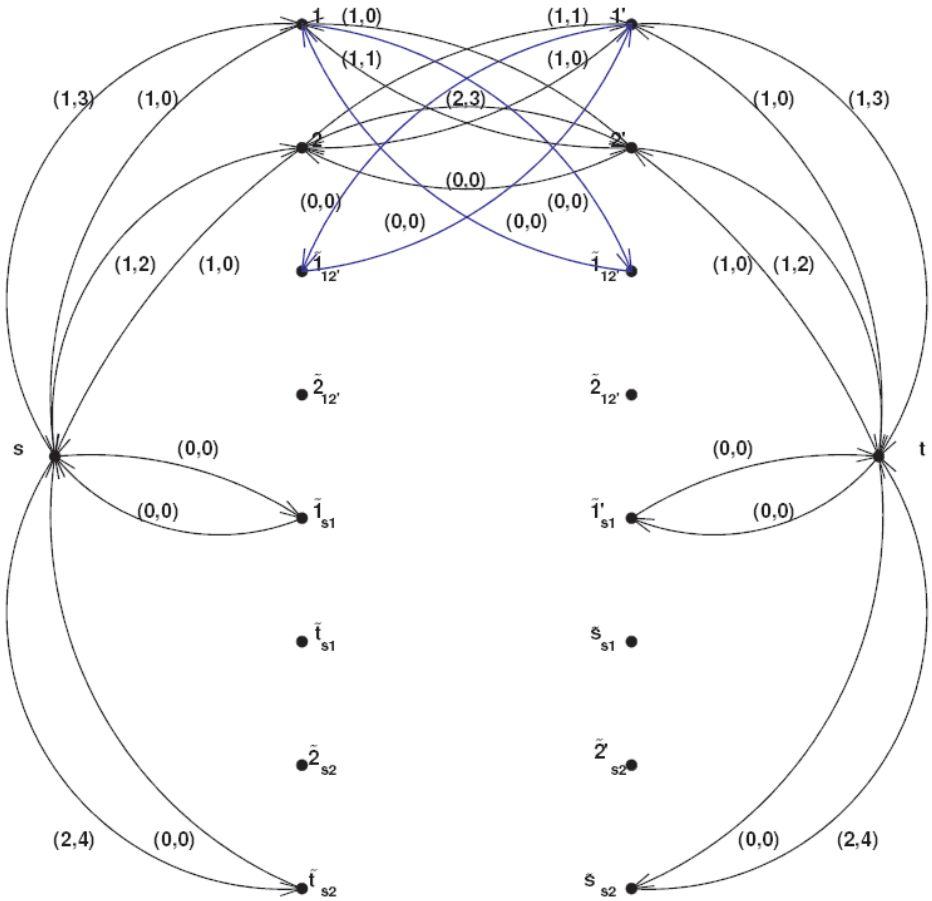

Abbildung 7.8: BMCF-Netzwerk nach dem 1. Schritt des Balanced SSP Algorithmus für konvexe Kosten

Die Lösung sei dabei in der Abbildung (Abb. 7.10) noch einmal angegeben, wobei die Flusswerte an den Kanten notiert sind.

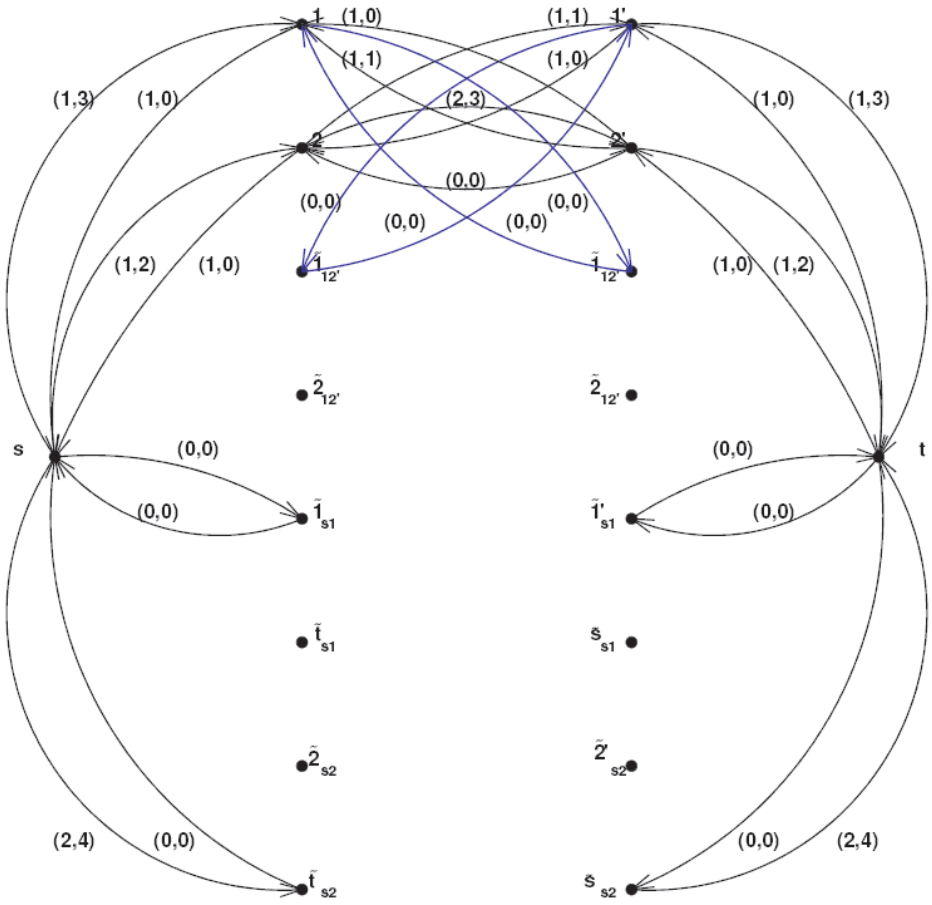

Abbildung 7.9: BMCF-Netzwerk nach dem 2. Schritt des Balanced SSP Algorithmus für konvexe Kosten

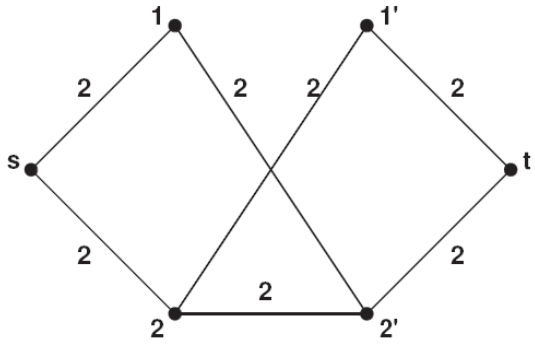

Abbildung 7.10: Lösung auf dem Netzwerk zum Beispielproblem

Kapitel 8

Balanced Capacity Scaling

8.1 Einleitung

Wir haben mit dem Balanced SSP Algorithmus bereits den ersten Successive-Shortest-Path-Algorithmus kennen gelernt. Wir wollen nun auch den Capacity Scaling Algorithmus betrachten. Auf „normalen“ Netzwerken gilt dieser dem SSP Algorithmus als überlegen [46, p. 6].
Die Idee des Balanced Capacity Scaling bleibt als weiterem Successive Shortest Path Algorithmus dieselbe wie beim Balanced SSP Algorithmus, da weiterhin kürzeste Pfade gesucht werden und beide auf der Reduced Cost Optimality beruhen. Beide optimieren auch den Fluss wieder auf $\mathfrak{F}(N)$. Wir müssen also wieder die Erreichung ganzzahliger Flüsse gesondert sicherstellen.
Der Vorteil des Balanced Capacity Scaling mit konvexen Kosten ist die Möglichkeit größere Flusspakete als 1 durch das Netzwerk zu schicken und damit im Fall linearer Kosten sogar einen Algorithmus zu erhalten, der stark polynomiell ist [52].

8.2 Balanced Capacity Scaling mit linearen Kosten

Wir wollen nun den Balanced Capacity Scaling Algorithmus formulieren. Wir beschreiben wieder zunächst eine Version für lineare, nichtnegative Kosten und beschreiben im Anschluss eine Version für konvexe Kosten.
Die Idee des Capacity Scaling besteht darin zunächst einen Untergraphen des Restnetzwerks mit einer Mindestgröße für die Restkapazität auf den Kanten auszuwählen und dann den Fluss auf diesem Untergraphen zu optimieren. Dies geschieht, indem jeweils Flüsse der Mindestgröße auf kürzesten Pfaden zwischen Überschuss- und Defizit-Knoten ausgetauscht werden.
Die Mindestgröße bezeichnen wir gemeinhin mit Δ. Eine Phase des Algorithmus, während derer sich der Wert von Δ im Algorithmus nicht ändert bezeichnen wir als Δ-*scaling phase*. Wir definieren hierzu 2 Mengen:

$$S(\Delta) = \{i \in N_M | e(i) \geq \Delta\} \tag{8.1}$$

$$T(\Delta) = \{i \in N_M | e(i) \leq -\Delta\} \tag{8.2}$$

Hierbei bezeichnet e wieder den Überschuss bzw. das Defizit. Wir gehen ferner, analog zu den Annahmen über das Balanced Min Cost Flow Problem im Balanced SSP-Algorithmus, davon aus, dass $c_{ij} \geq 0 \; \forall (ij) \in A(N)$. Damit erfüllt der Fluss anfangs die Reduced Cost Optimality. Ziel ist es nun den Fluss auch gültig, d.h. primal zulässig zu machen.
Mit der Definition der obigen Mengen sehen wir eine weitere Eigenschaft des Capacity Scaling, die es etwas weniger direkt macht, als den Balanced SSP-Algorithmus. Die Anfangsdefizite auf den Knoten müssen vorgegeben werden. Wir benutzen dazu den im Vorfeld beschriebenen Maximum Balanced Flow Algorithmus und erhalten mit diesem $e(s) = -e(t) =: b$. Für alle anderen Knoten gilt natürlich anfangs: $e(i) = 0 \; \forall i \in V(N)$. Für ein klein genug gewähltes Δ haben wir damit am Anfang $S(\Delta) = \{s\}$ und $T(\Delta) = \{t\}$. Wir definieren nun $U := 1 + |e(s)|$ und beginnen mit $\Delta = 2^{\lfloor \log_2 U \rfloor}$. Dies ist das größte U, das wir definieren können, ohne dass $S = T = \oslash$ anfangs gilt. Zusätzlich definieren wir noch $c_{max} = \sum_{(ij) \in A} c_{ij} + 1$. Den Algorithmus beginnen wir nun mit einem Fluss $x = 0$ auf den Knoten und einem Potential $\pi = 0$.
In einer Δ-scaling phase müssen jeweils kürzeste Wege mit einer Restkapazität mit mindestens Δ gefunden werden. Wir führen deshalb den Begriff des Δ-Restnetzwerks ein:

Definition 8.1 (Δ-Restnetzwerk)**.** *Das Δ-Restnetzwerk $N(\Delta)$ ist definiert als der Untergraph von N, der nur aus den Kanten (ij) mit $rescap(ij) \geq \Delta$ besteht.*
Wir definieren ferner $N(1) := BN(x)$ mit dem bisher erreichten Fluss x.
Offensichtlich ist das Δ-Restnetzwerk im Falle eines schiefsymmetrischen Ausgangsnetzwerks auch schiefsymmetrisch.

Für $\Delta = 1$ müssen wir wieder beachten, dass mögliche Flusserhöhungen auf den Kanten 1 und 2 sein können. Wir müssen also wieder das BMCF-Netzwerk betrachten.
Wir notieren nun den Balanced Capacity Scaling Algorithmus und beweisen wieder im Nachgang dessen Korrektheit. Unsere Version beruht dabei auf der Version in [3, p. 361].

```
Balanced Capacity Scaling
begin
    x := 0, π := 0, e(s) = −e(t) = b, e(i) = 0 ∀i ∈ V(N)
    Δ := 2^⌊log2(1+b)⌋
    while Δ ≥ 1 do
    begin {Δ-Scaling-Phase}
        for jede Kante (ij) im Δ-Restnetzwerk do
            if rescap(ij) ≥ Δ and c^π_ij < 0 then
                erhöhe den Fluss auf (ij) um rescap(ij)
                erhöhe den Fluss auf (j'i') um rescap(ij)
                berechne die e(i)'s neu
        S(Δ) := {i ∈ N|e(i) ≥ Δ}
        T(Δ) := {i ∈ N|e(i) ≤ Δ}
        while (S(Δ) ≠ ⊘) and (T(Δ) ≠ ⊘) do
        begin
```

```
            wähle einen Knoten k ∈ S(Δ) und einen Knoten t ∈
            T(Δ)
            bestimme den kürzesten Pfad p von l zu k in N(Δ) oder,
            falls nicht möglich, verbinde mit einer Kante (lk) =
            p mit Kosten c_max + π(i) − π(j)
            π = π + sd
            erhöhe den Fluss um min{balcap(p), Δ} Einheiten längs
            p und p'
            Update von x, S(Δ), T(Δ), N(Δ)
        end;
        Δ = Δ/2
    end;
end;
```

Satz 8.2. *Der Balanced Capacity Scaling Algorithmus berechnet einen maximalen Fluss x auf dem schiefsymmetrischen Netzwerk N mit minimalen Kosten.*

Beweis. Wir beweisen diesen Satz per Induktion nach der Anzahl der Δ-scaling Phasen. Wir zeigen hierbei zunächst, dass am Ende jeder Δ-scaling phase der berechnete Fluss optimal auf dem Δ-Restnetzwerk im Sinne der Reduced Cost Optimality ist. Ferner zeigen wir, dass das Netzwerk nach jeder Δ-scaling phase auch schiefsymmetrisch ist.
Beginnen wir also mit der Induktionsverankerung. Vor Beginn der ersten Δ-scaling phase gilt $c_{ij} = c_{ij}^{\pi} \geq 0$ und damit die Reduced Cost Optimality. Ferner ist das Ausgangsnetzwerk schiefsymmetrisch.
Nehmen wir nun an, die Lösung sei in der 2Δ-scaling phase optimal und betrachten nun das Problem in der Δ-scaling phase. Zunächst kommen neue Knoten und Kanten hinzu. Da das Problem in der 2Δ-scaling phase optimal bezüglich des 2Δ-Restnetzwerks war, kann es im neuen Δ-Restnetzwerk Kanten (ij) mit negativen reduzierten Kosten geben. Für diese muss gelten $\Delta \leq rescap(ij) < 2\Delta$. Erhöhen wir also den Fluss auf diesen Kanten um $rescap(ij)$, so fallen diese aus dem Δ-Restnetzwerk. Aufgrund der negativen reduzierten Kosten können wir dies tun. Da das Netzwerk zudem schiefsymmetrisch ist, muss auch gelten $\Delta \leq rescap(j'i') = rescap(ij) < 2\Delta$ und aufgrund der simultanen Erhöhung des Flusses auf beiden Kanten bleibt das Netzwerk schiefsymmetrisch. Damit gilt nach der Erhöhung des Flusses im ersten Schritt für alle Kanten $c_{ij}^{\pi} \geq 0$.
Bei der Erhöhung des Flusses wird Fluss von einem Knoten $l \in S(\Delta)$ zu einem Knoten $k \in T(\Delta)$ geschickt. Wir können nun unser Lemma zum Balanced SSP-Algorithmus benutzen: Da die Erhöhung des Flusses längs des kürzesten Weges vorgenommen wird, bleibt die Reduced Cost Optimality erhalten.
Es kann allerdings sein, dass ein solcher Pfad im Restnetzwerk nicht existiert. In diesem Fall erzeugt der Algorithmus eine Kante (lk) mit Kosten c_{max}, die nach Wahl von c_{max} so hoch liegen, dass diese Kante in der Lösung minimaler Kosten nicht auftreten kann. In dem Ausgangsalgorithmus von Ahuja und Orlin werden diese Kanten vor dem Start des eigentlichen Algorithmus erzeugt [3, p. 297]. Wir erzeugen sie hier quasi 'on-the-fly'.
Da wir gültige Pfade im schiefsymmetrischen Restnetzwerk bestimmen, können wir auch

hier den Fluss längs p und p' erhöhen. Das Restnetzwerk bleibt (wieder nach dem Lemma zum Balanced SSP-Algorithmus) schiefsymmetrisch und die Reduced Cost Optimality erfüllt. Damit haben wir den ersten Teil des Beweises.
Wir müssen nun noch zeigen, dass der Algorithmus bei Erreichen eines $\Delta < 1$ auch tatsächlich einen zulässigen Fluss bestimmt hat. Gilt $\Delta < 1$, so kann es nur noch Knoten $i \in V(N)$ mit einem Defizit oder Überschuss $e(i) < 1$ geben. Da das Problem mit ganzzahligen Daten angenommen wurde und im Algorithmus auch nur ganzzahlige Flussveränderungen durchgeführt wurden, kann dies für keine Kante mehr der Fall sein.
Da der Fluss somit zulässig ist und im BMCF-Netzwerk kein Erhöhungsschritt mehr möglich ist, ist dies sogar die optimale Lösung. Die Maximalität des Flusses ergibt sich aus der Tatsache, dass $e(s)$ als Ergebnis der Berechnung des Maximum-Balanced Flow benutzt wurde. □

8.3 Balanced Capacity Scaling mit konvexen Kosten

Wir beschreiben nun einen Balanced Capacity Scaling Algorithmus für konvexe Kosten. Die Idee geht hierbei auf Ahuja und Orlin zurück [3, pp. 556-560]. Wir behandeln hier wieder die Übersetzung der Idee auf die Situation schiefsymmetrischer Netzwerke.
Das offensichtliche Problem des Balanced SSP-Algorithmus für konvexe Kosten ist die Tatsache, dass lediglich Flussveränderungen von 1 vorgenommen werden. Wir wollen nun größere Flussmengen durch das Netzwerk schicken und so den Algorithmus beschleunigen. Die Idee des Algorithmus ist nun in jeder Δ-scaling phase nicht nur ein neues Restnetzwerk zu berechnen, sondern auch die Kosten auf den Kanten sukzessive besser zu approximieren. Diese Idee, die zuerst von Minoux für den Out-of-Kilter Algorithmus formuliert wurde [49], sei zunächst in der folgenden Grafik illustriert (8.1).
Hierbei sei wieder angemerkt, dass wir mit ganzzahligen Flüssen arbeiten. Eine Kostenfunktion muss nur auf den $x_{ij} \in \mathbb{N}$ exakt sein. Ferner haben wir nur einen Algorithmus zur Bestimmung allgemeiner kürzester gültiger Pfade für positive Kosten beschrieben, sodass wir annehmen wollen, die Kostenfunktion habe eine positive Steigung. Damit sind die linearen Kosten jedes Teilstücks positiv.
Auf dem BMCF-Netzwerk haben wir bereits angegeben, wie die Kosten und Kapazitäten der Kanten zu wählen sind.
Für die Δ-Restnetzwerke mit $\Delta > 1$ haben wir es deutlich einfacher, da Flussveränderungen von 2 auf jeder Kante möglich sind. Wir müssen damit nur Flussveränderungen auf den Kanten von $\{-\Delta, 0, \Delta\}$ betrachten. Wir wählen daher die Kosten und Kapazitäten (die wir als cap_Δ kennzeichnen) für die Kanten $(ij') \in A(N(\Delta))$ und deren Rückwärtskanten als:

$$cap_\Delta(ij') := \begin{cases} \Delta, \text{ falls } cap(ij') - x_{ij'} \geq \Delta \\ 0, \text{ sonst} \end{cases} \tag{8.3}$$

$$cap_\Delta(j'i) := \begin{cases} \Delta, \text{ falls } x_{ij'} \geq \Delta \\ 0, \text{ sonst} \end{cases} \tag{8.4}$$

$$cc_{ij'} = \frac{C_{ij'}(x_{ij'} + \Delta) - C_{ij'}(x_{ij'})}{\Delta} \tag{8.5}$$

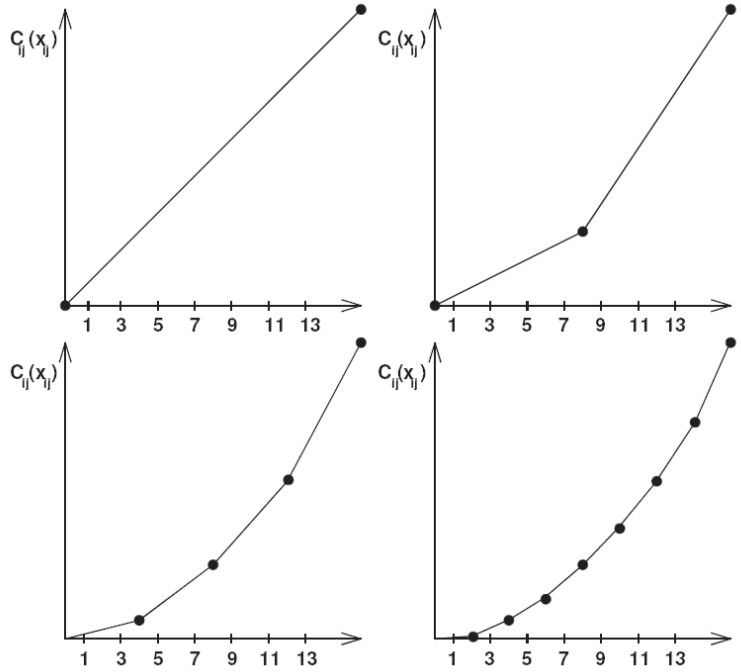

Abbildung 8.1: sukzessive Approximation einer Funktion durch Geradenstücke

$$cc_{j'i} = \frac{C_{ij'}(x_{ij'} - \Delta) - C_{ij'}(x_{ij'})}{\Delta} \tag{8.6}$$

Der Unterschied zum Balanced SSP-Algorithmus besteht nun darin, dass Flussveränderungen $\{-\Delta, 0, \Delta\}$ erlaubt sind und sich deshalb die Definition der cc_{ij}'s verändert.
Wir wollen hier zudem noch auf ein technisches Detail hinweisen. Im obigen Algorithmus hatten wir uns ein c_{max} definiert, als Kosten für künstlich eingeführte Kanten. In diesem Fall müssen wir dieses c_{max} noch etwas verändern. Mit $val(x)$ ist durch den Maximum Balanced Flow der maximale Fluss gegeben. Keine Kante im Ausgangsnetzwerk kann dann die Kosten $cc_{max} = \sum_{(ij)\in A(N)} C_{ij}(val(x)) + 1$ haben.
Wir wollen nun den Balanced Capacity Scaling Algorithmus für konvexe Kosten zunächst notieren und beweisen im Nachgang wieder dessen Korrektheit.

`Balanced Capacity Scaling for Convex Costs`

$x := 0,\ \pi := 0,\ e(s) = -e(t) = b,\ e(i) = 0\ \forall i \in V(N)$
$\Delta := 2^{\lfloor \log_2(1+b) \rfloor}$
`berechne` $N(\Delta)$ `mit entsprechenden Kapazitäten und Kosten while` $\Delta \geq 1$ `do`
`begin{`Δ`-scaling phase}`

`for jede Kante im` Δ`-Restnetzwerk do`

`if` $(c^{\pi}_{ij} \geq 0)$ `and` $(c^{\pi}_{ji} \geq 0)$ `tue nichts`
`if` $(c^{\pi}_{ij} < 0)$ `and` $(c^{\pi}_{ji} \geq 0)$ `erhöhe` x_{ij} `und` $x_{j'i'}$ `um` Δ

```
            if (c^π_ij ≤ 0) and (c^π_ji > 0) verringere x_ij und x_j'i' um
            Δ
        S(Δ) := {i ∈ N|e(i) ≥ Δ}
        T(Δ) := {i ∈ N|e(i) ≤ −Δ}
        while (S(Δ) ≠ ∅) and (T(Δ) ≠ ∅) do
        begin
            wähle einen Knoten l ∈ S(Δ) und einen Knoten k ∈
            T(Δ)
            bestimme den kürzesten, gültigen Pfad p von l zu k
            in N(Δ) oder falls dies nicht möglich ist, verbinde
            l und k mit der Kante (l, k) = p mit Kosten cc_max +
            π(i) − π(j)
            π := π + sd
            erhöhe den Fluss um min{balcap(p), Δ} Einheiten längs
            p und p'
            Update von x, S(Δ), T(Δ), N(Δ) aktualisiere die
            reduzierten Kosten
        end;
        Δ := Δ/2

    end;

end;
```

Wir müssen nun natürlich noch zeigen, dass dieser Algorithmus auch korrekt arbeitet.

Satz 8.3 (Korrektheit des Balanced Capacity Scaling für konvexe Kosten)**.** *Der Balanced Capacity Scaling Algorithmus für konvexe Kosten berechnet einen maximalen Fluss x auf dem schiefsymmetrischen Netzwerk N mit minimalen Kosten.*

Beweis. Wir zeigen zunächst, dass unsere Übersetzung des Problems korrekt ist. In der ersten scaling phase gilt $S = \{s\}$ und $T = \{t\}$. Ferner gilt nach der Wahl von Δ am Anfang $\Delta \leq |e(s)| = |e(t)| < 2\Delta$. Es kann also in der ersten scaling phase nicht mehr als Δ Fluss auf den Kanten verschoben werden, ohne in die nächste scaling phase zu kommen. Nach Wahl von S und T muss dann in jeder folgenden scaling phase gelten $\Delta \leq |e(i)| < 2\Delta \ \forall i \in N(\Delta)$. Somit werden über keine Kante mehr als Δ Einheiten Fluss in einer Iteration der Δ-scaling phase verschoben. Damit reicht es aus, lediglich Verschiebungen um $-\Delta$ oder $+\Delta$ zu betrachten, die in den Kosten der Kanten cc_{ij} und cc_{ji} kodiert sind.
Hierbei müssen wir allerdings beachten, dass $balcap(p) < \Delta$ auftreten kann. Wir müssen hierbei also zeigen, dass auch bei Flusserhöhungen, die kleiner sind als Δ die Reduced Cost Optimality erhalten bleibt. Wir gehen auf diesen Fall später im Beweis ein.
Wir wollen nun zeigen, dass die Reduced Cost Optimality jeweils vor der Definition von S und T in der Δ-scaling phase erfüllt ist. Wir zeigen dies per Induktion nach der Anzahl der Δ-scaling Phasen.

Anfangs gilt $\pi = 0$ und $x = 0$. Für die Kosten gilt:

$$cc_{ij} = c_{ij}^{\pi} = \frac{C(\Delta) - C(0)}{\Delta} \geq 0 \tag{8.7}$$

$$cc_{ji} = c_{ji}^{\pi} = \frac{C(-\Delta) - C(0)}{\Delta} := 0 \tag{8.8}$$

Damit sind die Reduced Cost Optimality Conditions anfangs erfüllt (Induktionsanfang). Es sei hierbei darauf hingewiesen, dass die Definition der Kosten bei negativem Fluss (die ja eigentlich wenig Sinn machen) hier eventuell entsprechend angepasst werden muss.
Nehmen wir nun an, nach der 2Δ-scaling phase seien die Reduced Cost Optimality Bedingungen erfüllt. Nun ändern sich die Kosten auf den Kanten und reduzierte Kosten können eventuell sogar negativ werden.
Sei nun x der Fluss aus der 2Δ-scaling phase. Nun gibt es folgende Fälle für die reduzierten Kosten nach der Neuberechnung in der Δ-scaling phase.

(i) $c_{ij}^{\pi} \geq 0,\ c_{ji}^{\pi} \geq 0$

(ii) $c_{ij}^{\pi} < 0,\ c_{ji}^{\pi} \geq 0$

(iii) $c_{ij}^{\pi} \geq 0,\ c_{ji}^{\pi} < 0$

(iv) $c_{ij}^{\pi} < 0,\ c_{ji}^{\pi} < 0$

Wir zeigen nun zunächst, dass bei konvexen Kosten der Fall (iv) nicht auftreten kann. Damit der Fall (iv) auftritt muss gelten:

$$\frac{C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij})}{\Delta} + \pi(i) - \pi(j) < 0 \tag{8.9}$$

$$\frac{C_{ij}(x_{ij} - \Delta) - C_{ij}(x_{ij})}{\Delta} + \pi(j) - \pi(i) < 0 \tag{8.10}$$

Wir können dies noch etwas anders notieren:

$$+\pi(i) - \pi(j) - \frac{C_{ij}(x_{ij} - \Delta) - C_{ij}(x_{ij})}{\Delta} > 0 \tag{8.11}$$

$$+\pi(i) - \pi(j) + \frac{C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij})}{\Delta} < 0 \tag{8.12}$$

$$\Rightarrow C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij}) < C_{ij}(x_{ij}) - C_{ij}(x_{ij} - \Delta) \tag{8.13}$$

Wir benutzen nun die Definition einer konvexen Kostenfunktion. Danach gilt:

$$2C_{ij}(x_{ij}) \leq C_{ij}(x_{ij} - \Delta) + C_{ij}(x_{ij} + \Delta) \tag{8.14}$$

$$\Rightarrow C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij}) \geq C_{ij}(x_{ij}) - C_{ij}(x_{ij} - \Delta) \tag{8.15}$$

Und damit haben wir den Widerspruch.
Behandeln wir nun die restlichen Fälle. Im Fall (i) sind die Reduced Cost Optimality Bedingungen offensichtlich erfüllt.
Es bleiben also Fall (ii) und (iii). Im Algorithmus behandeln wir Fall (ii), indem wir

x_{ij} um Δ Flusseinheiten erhöhen. Wir zeigen nun, dass dies zum Erfolg führt. Nach der 2Δ-scaling phase gilt:

$$\frac{C(x_{ij}+2\Delta)-C_{ij}(x_{ij})}{2\Delta}+\pi(i)-\pi(j)\geq 0 \tag{8.16}$$

$$C_{ij}(x_{ij}+2\Delta)-C_{ij}(x_{ij})+2\Delta\pi(i)-2\Delta\pi(j)\geq 0 \tag{8.17}$$

Die Kante (ij) erfüllt aber in $N(\Delta)$ nicht mehr die Reduced Cost Optimality Conditions, also:

$$C_{ij}(x_{ij}+\Delta)-C_{ij}(x_{ij})+\Delta\pi(i)-\Delta\pi(j)<0 \tag{8.18}$$

Wir wollen nun zeigen, dass nach erhöhen des Flusses um Δ gilt:

$$C_{ij}(x_{ij}+2\Delta)-C_{ij}(x_{ij}+\Delta)+\Delta\pi(i)-\Delta\pi(j)\geq 0 \tag{8.19}$$

Wir schreiben dazu die Gleichung aus der 2Δ-scaling phase um:

$$\begin{aligned}&[C_{ij}(x_{ij}+2\Delta)-C_{ij}(x_{ij}+\Delta)+\Delta\pi(i)-\Delta\pi(j)]+\\ &\underbrace{[C_{ij}(x_{ij}+\Delta)-C_{ij}(x_{ij})+\Delta\pi(i)-\Delta\pi(j)]}_{<0}\geq 0\end{aligned} \tag{8.20}$$

Und damit folgt die gewünschte Ungleichung.
Die reduzierten Kosten auf $(j'i')$ müssen wegen der Schiefsymmetrie des Netzwerkes identisch sein. Damit lässt sich das Resultat auf $(j'i')$ sofort übertragen. Ferner wird der Fluss im Algorithmus auf (ij) und $(j'i')$ parallel erhöht, so dass die Schiefsymmetrie auch erhalten bleibt.
Völlig analog können wir die Erreichung der Reduced Cost Optimality auch für Fall (iii) zeigen.
Für die while-Schleife haben wir bereits im Fall des linearen Balanced Capacity Scaling Algorithmus gezeigt, dass die Schiefsymmetrie erhalten bleibt und die Reduced Cost Optimality erfüllt ist. Wir müssen lediglich zeigen, dass auch bei Flusserhöhungen $balcap(p)<\Delta$ die Reduced Cost Optimality erhalten bleibt.
Wir nehmen hierzu an, dass $c^{\pi}_{ij}=0$ gilt und wir längs der Kante (ij) den Fluss um weniger als Δ erhöht haben. Dann gilt für die reduzierten Kosten:

$$cc^{neu}_{ij}= \frac{C_{ij}(x_{ij}+balcap(p)+\Delta)-C_{ij}(x_{ij}+balcap(p))}{\Delta}+\pi(i)-\pi(j) \tag{8.21}$$

$$\geq \frac{C_{ij}(x_{ij}+\Delta)-C_{ij}(x_{ij})}{\Delta}+\pi(i)-\pi(j)=0 \tag{8.22}$$

Damit bleibt auf der Vorwärtskante die Reduced Cost Optimality erfüllt. Betrachten wir nun die Rückwärtskante. Wir verwenden hier nochmals:

$$\frac{C_{ij}(x_{ij}+\Delta)-C_{ij}(x_{ij})}{\Delta}+\pi(i)-\pi(j) = 0 \tag{8.23}$$

$$\Leftrightarrow \frac{C_{ij}(x_{ij})-C_{ij}(x_{ij}+\Delta)}{\Delta}-\pi(i)+\pi(j) = 0 \tag{8.24}$$

Es gilt ferner:

$$cc_{ji}^{neu} = \frac{C_{ij}(x_{ij} + balcap(p) - \Delta) - C_{ij}(x_{ij} + balcap(p))}{\Delta} - \pi(i) + \pi(j) \quad (8.25)$$

$$\geq \frac{C_{ij}(x_{ij}) - C_{ij}(x_{ij} + \Delta)}{\Delta} - \pi(i) + \pi(j) = 0 \quad (8.26)$$

Damit bleibt die Reduced Cost Optimality aufgrund der Konvexität der Kostenfunktion auch erhalten, wenn wir um weniger als Δ den Fluss erhöhen.
Am Ende wird ein zulässiger Fluss erreicht, was ebenfalls analog zu dem Argument im Satz zur Korrektheit des Balanced Capacity Scaling gezeigt werden kann. Folglich liefert der Algorithmus also eine Lösung mit maximalem Fluss bei minimalen Kosten. □

Wir wollen nun noch zur Komplexität des Algorithmus kommen:

Satz 8.4 (Komplexität des Balanced Capacity Scaling)**.** *Der Balanced Capacity Scaling Algorithmus hat eine Laufzeit von* $O(\log_2(U) \cdot m \cdot S(n,m))$ *mit der Laufzeit* $S(n,m)$ *zur Berechnung eines kürzesten Pfades.*

Beweis. Am Ende der 2Δ-scaling phase gilt $T(2\Delta) = \oslash$ oder $S(2\Delta) = \oslash$. In der Δ-scaling phase gibt es also maximal $2n\Delta$ „alten“ Überschuss auf den Knoten. Bei Beginn der Δ-scaling phase wird der Fluss auf den Kanten um maximal Δ pro Kante verändert. Damit kann sich maximal ein neuer Überschuss auf den Kanten von $2m\Delta$ ergeben. Wir können damit den Gesamtüberschuss bei Beginn der Δ-scaling-phase mit $2(n+m)\Delta$ abschätzen. In jeder Iteration zum Austausch zwischen Überschuss- und Defizitknoten werden mindestens $\frac{\Delta}{2}$ Flusseinheiten ausgetauscht. Wir haben also in jeder Phase $O(m)$ Iterationen des Algorithmus. Die Komplexität jeder Iteration wird offensichtlich durch die Berechnung des kürzesten Pfades dominiert, sodass die Komplexität jeder Phase $O(mS(n,m))$ beträgt.
Die Anzahl der Phasen ergibt sich aus dem anfangs gewählten $\Delta = 2^{\lfloor \log_2 U \rfloor} \leq U$. In jeder Iteration wird Δ halbiert, sodass wir maximal $\log_2(U)$ Phasen haben. Damit haben wir eine Gesamtkomplexität $O(\log_2 U \cdot m \cdot S(n,m))$. □

Kapitel 9

Balanced Out-of-Kilter

Mit den bisherigen Algorithmen haben wir Algorithmen betrachtet, die versuchten eine dual zulässige Lösung auch primal zulässig zu machen. Der andere Weg ist auch gangbar. Wir werden ihn später noch kurz besprechen. Mit dem Out-of-Kilter Algorithmus wollen wir nun einen Algorithmus vorstellen, der eine unzulässige Lösung primal und dual zulässig macht. Wir wollen ihn hier auch besprechen, da die erste Idee von Minoux für einen schwach polynomiellen Algorithmus zur Lösung des ganzzahligen Min Cost Flow Problems mit konvexen Kosten auf dem Out-of-Kilter Algorithmus basierte [49].
Wir betrachten wieder zunächst den Fall linearer Kosten und gehen später auf den Fall konvexer Kosten ein.

9.1 Färben von Kanten

Die Idee des Out-of-Kilter Algorithmus geht auf Fulkerson [28] und Minty zurück. Wir wollen hier aber unsere Darstellung an der Version in [45] orientieren. Unser Ansatz erlaubt hierbei wie das Capacity Scaling schwach-polynomielle Laufzeit. Wir nutzen hierbei allerdings eine schöne Eigenschaft des von Minoux vorgeschlagenen Algorithmus aus, so dass der Balanced Out-of-Kilter Algorithmus schneller als das vorgeschlagene Balanced Capacity Scaling wird.
Wir betrachten dabei zunächst wieder das BMCF Problem mit linearen Kosten auf dem Polytop $\mathfrak{F}(N)$. Wir werden hierbei sehen, dass wir beim Balanced Out-of-Kilter ebenfalls Barrieren haben, diese hier aber etwas anders auflösen als im primal-dualen Algorithmus, bei dem wir für die Fragmente eigene Potentiale eingeführt hatten.
Wir benutzen bei diesem Algorithmus wieder die bereits erwähnte Reduced Cost Optimality:

$$c_{ij'} + \pi(i) - \pi(j') > 0 \quad \Rightarrow \quad x_{ij'} = l(ij'),\ x_{ji'} = l(ji') \tag{9.1}$$
$$c_{ij'} + \pi(i) - \pi(j') < 0 \quad \Rightarrow \quad x_{ij'} = cap(ij'),\ x_{ji'} = cap(ji') \tag{9.2}$$

Die Idee des Out-of-Kilter Algorithmus ist nun, die Kanten, die die Optimalitätbedingung erfüllen (*in-kilter*), als solche zu behalten und Kanten, die die Kapazitätsbeschränkungen nicht erfüllen (*out-of-kilter*), sukzessive zu in-kilter Kanten zu machen. Dabei müssen wir dann auch die duale Lösung sukzessive verbessern.

Wir benutzen nun die reduzierten Kosten zur Klassifikation der Kanten. Wir geben dabei zunächst nach Fulkerson [28, S. 18] die Möglichkeiten für die reduzierten Kosten einer Kante $(ij) \in A(N)$ bei einer unzulässigen Lösung an und gehen dann auf die gewählte Klassifikation ein:

$$
\begin{aligned}
(\alpha) \quad & c_{ij} + \pi(i) - \pi(j) > 0,\ x_{ij} = l(ij) && (9.3)\\
(\beta) \quad & c_{ij} + \pi(i) - \pi(j) = 0,\ l(ij) \le x_{ij} \le cap(ij) && (9.4)\\
(\gamma) \quad & c_{ij} + \pi(i) - \pi(j) < 0,\ x_{ij} = cap(ij) && (9.5)\\
(\alpha 1) \quad & c_{ij} + \pi(i) - \pi(j) > 0,\ x_{ij} < l(ij) && (9.6)\\
(\beta 1) \quad & c_{ij} + \pi(i) - \pi(j) = 0,\ x_{ij} < l(ij) && (9.7)\\
(\gamma 1) \quad & c_{ij} + \pi(i) - \pi(j) < 0,\ x_{ij} < cap(ij) && (9.8)\\
(\alpha 2) \quad & c_{ij} + \pi(i) - \pi(j) > 0,\ x_{ij} > l(ij) && (9.9)\\
(\beta 2) \quad & c_{ij} + \pi(i) - \pi(j) = 0,\ x_{ij} > cap(ij) && (9.10)\\
(\gamma 2) \quad & c_{ij} + \pi(i) - \pi(j) < 0,\ x_{ij} > cap(ij) && (9.11)
\end{aligned}
$$

Die ersten 3 Fälle (α, β, γ) beschreiben in-kilter Kanten, während die letzten 6 Fälle out-of-kilter Kanten beschreiben.
Wir messen nun die Verletzung der Reduced Cost Optimality in Form der *Kilter-Zahl*, die angibt wie viel (Absolutwert) Fluss zum Erreichen der in-kilter Bedingung notwendig ist [45, S. 144]:

$$
K(x_{ij}) = \begin{cases} |x_{ij} - l(ij)|, \text{ falls } \pi(j) - \pi(i) < c_{ij} \\ l(ij) - x_{ij}, \text{ falls } x_{ij} < l(ij),\ \pi(j) - \pi(i) = c_{ij} \\ x_{ij} - cap(ij), \text{ falls } x_{ij} > c_{ij},\ \pi(j) - \pi(i) = c_{ij} \\ |x_{ij} - cap(ij)|, \text{ falls } \pi(j) - \pi(i) > c_{ij} \end{cases} \tag{9.12}
$$

Insbesondere gilt für schiefsymmetrische Netzwerke offensichtlich $K(x_{ij}) = K(x_{j'i'})$.
Neben der Kilter-Zahl führen wir nun eine *Färbung* des Graphen zur Klassifikation der Kanten ein. Die Methode geht dabei auf Minty zurück.
Wir färben eine Kante grün, falls es möglich ist, den Fluss zu erhöhen oder zu verringern, ohne dass die Kante out-of-kilter geht. Es gilt also:

$$
l(ij) < x_{ij} < cap(ij) \ \text{ und } \ \pi(j) - \pi(i) = c_{ij} \tag{9.13}
$$

Wir färben eine Kante gelb, wenn es möglich ist, den Fluss zu erhöhen, aber nicht ihn zu verringern, ohne die Kilter-Zahl zu erhöhen. Es gibt dafür folgende Fälle:

$$
\begin{aligned}
x_{ij} < cap(ij) \quad & \text{und} \quad \pi(j) - \pi(i) > c_{ij} && (9.14)\\
x_{ij} \le l(ij) \quad & \text{und} \quad \pi(j) - \pi(i) = c_{ij} && (9.15)\\
x_{ij} < l(ij) \quad & \text{und} \quad \pi(j) - \pi(i) < c_{ij} && (9.16)
\end{aligned}
$$

Wir färben eine Kante gelb und drehen ihre Richtung um, falls es möglich ist, den Fluss auf der Kante zu verringern, aber nicht ihn zu erhöhen, ohne die Kilter Zahl zu erhöhen.

$$
\begin{aligned}
x_{ij} > cap(ij) \quad & \text{und} \quad \pi(j) - \pi(i) > c_{ij} && (9.17)\\
x_{ij} \ge cap(ij) \quad & \text{und} \quad \pi(j) - \pi(i) = c_{ij} && (9.18)\\
x_{ij} > l(ij) \quad & \text{und} \quad \pi(j) - \pi(i) < c_{ij} && (9.19)
\end{aligned}
$$

Wir färben eine Kante rot, wenn man den Fluss weder erhöhen noch verringern kann, ohne die Kilter-Zahl zu erhöhen.

$$x_{ij} = cap(ij) \quad \text{und} \quad \pi(j) - \pi(i) > c_{ij} \tag{9.20}$$
$$x_{ij} = l(ij) \quad \text{und} \quad \pi(j) - \pi(i) < c_{ij} \tag{9.21}$$

Es sei hier angemerkt, dass auch die Färbung von Graphen natürlich balanciert ist. Ist also (ij') von einer bestimmten Farbe, so ist es (ji') auch, wenn der Fluss auf dem Netzwerk balanciert ist.
Um das Konzept der Färbung eines Graphen noch etwas zu verdeutlichen, betrachten wir die Bedingungen und Farben in folgender Grafik (Abb. 9.1). Hierbei deuten die senkrechten Linien die Bedingungen für rote und die waagerechte Linie die Bedingungen für grüne Kanten an. Die übrigen Kombinationen von $\pi(i)$, $\pi(j)$ und x_{ij} entsprechen gelben Kanten. Man kann so z.B. aus der Grafik ablesen, dass alle Kanten mit $l(ij) < x_{ij} < cap(ij)$ mit $c_{ij} = \pi(j) - \pi(i)$ grün sind.

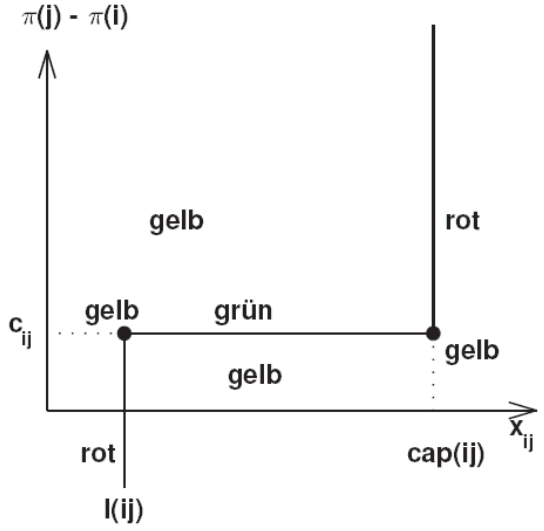

Abbildung 9.1: Färbung der Kanten in einem Graphen

9.2 Grundlagen des Algorithmus

Nachdem wir nun die Klassifikation der Kanten eingeführt haben, wollen wir nun zum Sinn dieser Klassifikation kommen. Bevor wir für die gefärbten Graphen nun den Satz von Minty beweisen, wollen wir noch eine kurze Begriffsbildung nach [35, S. 107] einführen.

Definition 9.1 (Co-Kreis)**.** *Sei G ein Netzwerk mit einer Knotenmenge N . Ein Schnitt ist eine Partition $N = S \biguplus T$ von N in 2 nichtleere Teilmengen. Wir notieren mit (S, T) die Menge aller Kanten inzident mit einem Knoten in T und einem Knoten in S. Jede solche Menge von Kanten nennt man einen Co-Kreis.*

Satz 9.2 (Satz von Minty)**.** *Sei G ein gerichteter Graph mit einer ausgezeichneten Kante (ts). Dann gilt für jede Färbung der Kanten in grün, gelb und rot mit (ts) in gelb genau eine der beiden Alternativen:*

(i) *(ts) ist enthalten in einem Kreis aus nur gelben und grünen Kanten, in dem alle gelben Kanten die gleiche Richtung haben.*

(ii) *(ts) ist enthalten in einem Co-Kreis aus nur gelben und roten Kanten, in dem alle gelben Kanten die gleiche Richtung haben.*

Beweis. Der Beweis ist sehr intuitiv, wenn wir uns den Graphen als eine Straßenkarte vorstellen, auf der es Straßen gibt, die in beiden Richtungen befahrbar sind (grüne Kanten), in nur einer Richtung befahrbar sind (gelbe Kanten) oder für den Verkehr gesperrt sind (rote Kanten).
Entweder gibt es nun einen Weg von s nach t oder es gibt keinen. Nehmen wir an, es gäbe einen solchen Weg, so bestünde er nur aus grünen und gelben Kanten und die gelben Kanten müssten alle von s nach t gerichtet sein (Alternative (i)). Mit der Kante (ts) haben wir damit den gewünschten Kreis.
Nehmen wir nun an, es gäbe keinen solchen Weg von s nach t. Dann könnten wir den beschriebenen Co-Kreis folgendermaßen konstruieren: Sei S die Menge aller Knoten, die für den Verkehr von s erreichbar sind und T die Komplementärmenge. Zwischen S und T könnten nun nach der Definition von T keine grünen Kanten existieren und alle gelben Kanten müssten von T nach S zeigen. Alle anderen Kanten zwischen S und T müssten folglich rot sein. Damit wäre (S, T) der gesuchte Co-Kreis (Alternative (ii)). □

Die Bedeutung des Satzes von Minty wird mit folgender Überlegung deutlich: Betrachten wir eine gelbe Kante (ts) und wenden den Satz von Minty an. Nehmen wir ferner an, es gäbe einen gelb-grünen Kreis C, in dem alle gelben Kanten in die gleiche Richtung zeigen. Würden wir nun alle Kanten umdrehen, deren Richtung umgekehrt wurde, so wäre es möglich den Fluss um $\delta > 0$ zu erhöhen und dabei die Kilter-Zahl zu verringern (war (ts) selbst eine der Kanten, die umgedreht wurde, müssen wir den Fluss natürlich verringern). Damit ist dann also $C - \{(ts)\}$ ein erweiternder Weg von s nach t.
Dieser erweiternde Weg $C - \{(ts)\}$ muss nun natürlich nicht gültig sein. Wir klammern dieses Problem zunächst aus, behalten es aber im Hinterkopf. Es wird sich herausstellen, dass dies im Gegensatz zur naiven Annahme nicht so schwerwiegend ist, wie man zunächst annehmen könnte.
Wie man vermutlich schon sieht, sind die Alternativen (i) und (ii) die Möglichkeiten gegen die Lösung zu iterieren. Im Fall (i) haben wir erweiternde Pfade und im Fall (ii) werden wir sehen, dass wir die Potentiale anpassen können.
Betrachten wir zunächst den Fall (i). Bei diesem wollen wir auf einem erweiternden Pfad den Fluss verändern. Hierbei gibt es verschiedene Fälle für die Flussrichtung, um die Kilter-Zahl zu verringern. Diese seien in der folgenden Grafik zusammengefasst (Abb. 9.2). Man sieht dabei, dass die möglichen Flussveränderungen darauf abzielen, eine der senkrechten oder waagerechten Linien zu erreichen, also in-kilter zu gehen.
Für einen gelb-grünen Kreis seien Y die gelben und G die grünen Kanten. Ferner seien $+$ und $-$ die Indizes für die Kanten aus Y und G, für die der Fluss entweder um δ erhöht

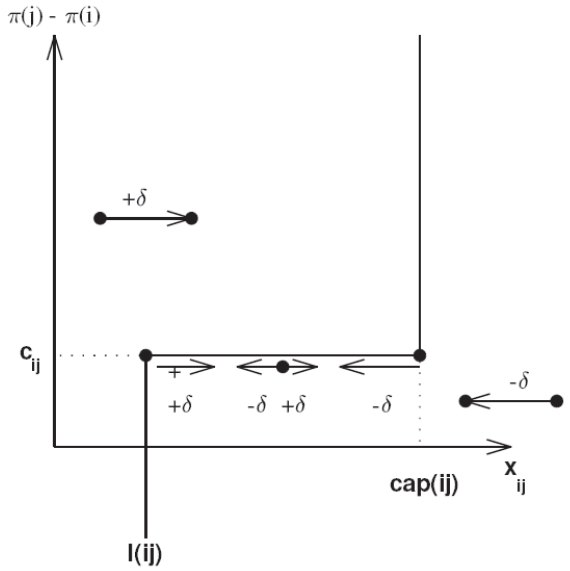

Abbildung 9.2: mögliche Flussveränderungen

oder verringert wird. Keine Kante in-kilter geht out-of-kilter, wenn δ nicht größer als δ_1, δ_2 gewählt wird, wobei:

$$\begin{aligned}
\delta_1 &= \min\{\min\{cap(ij') - x_{ij'} | (ij') \in Y^+ \cup G^+,\ \pi(j') - \pi(i) = c_{ij'},\ (ji') \notin Y^+ \cup G^+\}, \\
&\quad \min\left\{\frac{cap(ij') - x_{ij'}}{2} | (ij') \in Y^+ \cup G^+,\ \pi(j') - \pi(i) = c_{ij'},\ (ji') \in Y^+ \cup G^+\right\}\} \\
\delta_2 &= \min\{\min\{x_{ij'} - l(ij') | (ij') \in Y^- \cup G^-,\ \pi(j) - \pi(i) = c_{ij},\ (ji') \notin Y^- \cup G^-\}, \\
&\quad \min\left\{\frac{x_{ij'} - l(ij')}{2} | (ij') \in Y^- \cup G^-,\ \pi(j) - \pi(i) = c_{ij},\ (ji') \in Y^- \cup G^-\right\}\}
\end{aligned}$$

Wir haben hierbei bereits den Fall komplementärer Kanten auf einem Pfad analog der balanced capacity in unserer Berechnung berücksichtigt.
Ferner wird δ nicht größer als notwendig gewählt, um eine out-of-kilter Kante in-kilter zu bringen. Hierzu definieren wir weitere obere Grenzen für δ:

$$\begin{aligned}
\delta_3 &= \min\{\min\{|cap(ij') - x_{ij'}|\ | (ij') \in Y^+ \cup Y^-,\ \pi(j') - \pi(i) > c_{ij'},\ (ji') \notin Y^+ \cup Y^-\}, \\
&\quad \min\left\{\frac{|cap(ij') - x_{ij'}|}{2}\ | (ij') \in Y^+ \cup Y^-,\ \pi(j') - \pi(i) > c_{ij'}, (ji') \in Y^+ \cup Y^-\right\}\} \\
\delta_4 &= \min\{|x_{ij} - cap(ij)|\ | (ij) \in Y^+ \cup Y^-,\ \pi(j) - \pi(i) < c_{ij},\ (ji') \notin Y^- \cup Y^-\}, \\
&\quad \min\left\{\frac{|cap(ij') - x_{ij'}|}{2}\ | (ij') \in Y^+ \cup Y^-,\ \pi(j') - \pi(i) < c_{ij'}, (ji') \in Y^- \cup Y^-\right\}\}
\end{aligned}$$

Folglich wählen wir:

$$\delta = \min\{\delta_1, \delta_2, \delta_3, \delta_4\} \tag{9.22}$$

Es sei hierbei darauf hingewiesen, dass δ für ein Problem mit endlicher Lösung natürlich endlich sein muss. Erhalten wir also ein unendliches δ hat das BMCF zumindest keine

endliche Lösung.
Wir beschäftigen uns nun mit Alternative (ii). Nehmen wir also an, wir hätten einen gelb-roten Co-Kreis (S, T) mit $s \in S$ und $t \in T$ gefunden, in dem nach dem Satz von Minty alle gelben Kanten in die gleiche Richtung zeigen. Wir drehen nun alle Kanten, die während der Färbung gedreht wurden wieder um.
Erhöhen wir die Potentiale aller Kanten in T um $\epsilon > 0$, $\epsilon \in \mathbb{R}$, so verändert dies offensichtlich nur die Potentialdifferenz auf den Kanten im Co-Kreis. Ferner wird hiermit die Kilter-Zahl keiner Kante erhöht, sondern höchstens verringert. Wir wollen uns diesen Sachverhalt zunächst anhand eines Bildes veranschaulichen (Abb. 9.3). Hierbei deutet $+\epsilon$ eine Veränderung des Potentials an.

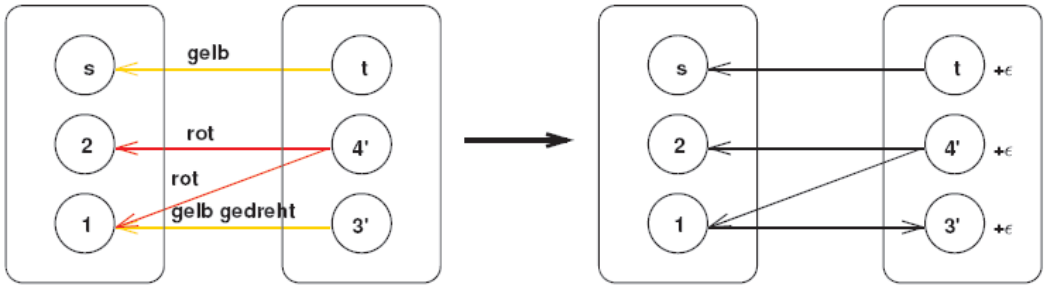

Abbildung 9.3: Veränderung der Kanten und Potentiale in gelb-blau-rotem Schnitt

Wird ϵ nun „klein genug“ gewählt, so wird mit einer solchen Operation keine Kilter-Zahl erhöht. Wir gehen sofort darauf ein, was „klein genug“ in diesem Zusammenhang bedeutet. Zuvor möchten wir allerdings die verschiedenen Fälle für die Erhöhung des Potentials in einer Grafik zusammenfassen (Abb. 9.4).

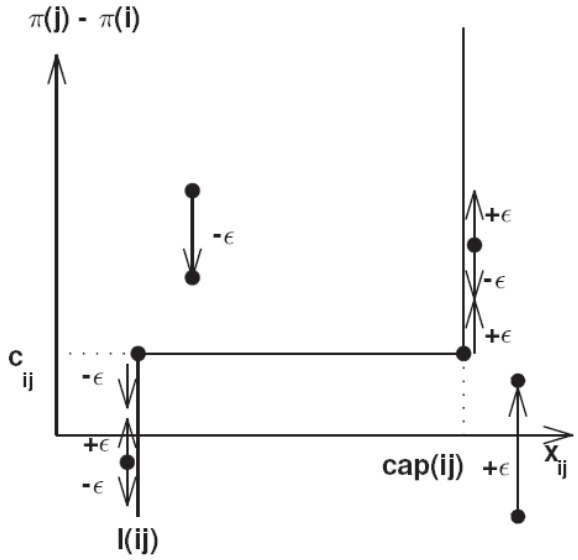

Abbildung 9.4: mögliche Potentialveränderungen

Gehen wir nun auf unsere saloppe Formulierung „klein genug“ ein. Für einen gelb-roten Co-Kreis bezeichnen wir mit Y und R die Teilmengen gelber und roter Kreise von C. Mit Indizies $+$ und $-$ bezeichnen wir hier die Teilmengen der Kanten, für die die Potentialdifferenz $\pi(j) - \pi(i)$ für $(ij) \in A(N)$ erhöht bzw. verringert wird. Keine Kante geht out-of-kilter, falls ϵ nicht größer gewählt wird als ϵ_1, ϵ_2 mit:

$$\epsilon_1 = \min\{\pi(j) - \pi(i) - c_{ij} | (ij) \in R^-,\ x_{ij} = cap(ij)\} \tag{9.23}$$
$$\epsilon_2 = \min\{c_{ij} - \pi(j) - \pi(i) | (ij) \in R^+,\ x_{ij} = l(ij)\} \tag{9.24}$$

Ganz analog zu der Wahl von δ bei den Flussveränderungen, wollen wir auch hier ϵ nicht größer wählen als nötig, um eine Kante in-kilter zu bringen:

$$\epsilon_3 = \min\{\pi(j) - \pi(i) - c_{ij} | (ij) \in Y^-,\ l(ij) \leq x_{ij} < cap(ij)\} \tag{9.25}$$
$$\epsilon_4 = \min\{c_{ij} - \pi(j) + \pi(i) | (ij) \in Y^+,\ l(ij) < x_{ij} \leq cap(ij)\} \tag{9.26}$$

Wir wählen also:

$$\epsilon = \min\{\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4\} \tag{9.27}$$

Für ϵ gibt es nun 3 mögliche Fälle:

(i) ϵ ist unbeschränkt. Dies kann nur auftreten, falls $x_{ij} \geq cap(ij)$ für alle Kanten von S nach T und $x_{ij} \leq l(ij)$ für alle Kanten von T nach S gilt und $x_{ts} < l(ts)$. Der Nettofluss von S nach T ist 0 also:

$$\sum_{i \in S,\, j \in T} l(ij) > \sum_{i \in T,\, j \in S} cap(ij) \tag{9.28}$$

Mit dem Balanced Max Flow Min Cut Satz folgt damit, dass eine gültige Zirkulation nicht existiert.

(ii) ϵ ist endlich und es gilt $\epsilon = \epsilon_3$ oder $\epsilon = \epsilon_4$. Mindestens eine Kante geht nach Definition von ϵ_3 und ϵ_4 in-kilter. Es wird keine Kilter-Zahl erhöht und einige werden verringert.

(iii) ϵ ist endlich und kleiner als ϵ_3 und ϵ_4. Dann geht keine Kante in-kilter. Einige Kilter-Zahlen werden erhöht und einige verringert. Mindestens eine rote Kante wird gelb, wenn sie das nächste Mal gefärbt wird (wieder nach der Definition von ϵ_1 und ϵ_2). Für eine solche Kante $(ij) \in A(N)$ gilt, falls $i \in S, j \in T$, für den Fluss $l(ij) = x_{ij} < cap(ij)$ und falls $i \in T, j \in S$ gilt $l(ij) < x_{ij} = cap(ij)$, da jeweils das Potential des Knotens aus T erhöht wird. Ferner können einige gelbe Kanten rot werden. Für jede dieser Kanten gilt entweder $i \in S, j \in T \Rightarrow l(ij) < x_{ij} = cap(ij)$ oder $i \in T, j \in S \Rightarrow l(ij) = x_{ij} < cap(ij)$. Natürlich werden, da im betrachteten Co-Kreis ja gar keine grünen Kanten enthalten sind, auch keine grünen Kanten verändert. Es sei an dieser Stelle noch angemerkt, dass hier nur Kanten betrachtet werden, mit denen man keine Knoten erreichen kann. Folglich verändert das Rot-Färben gelber Kanten nicht die Menge der Knoten, die von s aus erreichbar sind.

Wir müssen an dieser Stelle noch genauer auf die Symmetrie des Netzwerks, die wir im Algorithmus nutzen wollen, eingehen.
Die symmetrische Behandlung erweiternder Pfade haben wir bereits thematisiert. Beim Balanced Out-of-Kilter Algorithmus müssen aber auch die Potentiale symmetrisch behandelt werden. Betrachten wir dazu noch einmal die 2 Alternativen des Satzes von Minty.
Bei Alternative (i) des Satzes von Minty haben wir bereits festgestellt, dass wir einen erweiternden Pfad $p = C - \{(ts)\}$ bestimmen (der eventuell aber nicht gültig ist). Dann ist in einem schiefsymmetrischen Netzwerk $p' = C' - \{(ts)\}$ ebenfalls ein erweiternder Pfad, da C' aufgrund der Symmetrie der Farbgebung ebenfalls ein gelb-grüner Kreis ist. Wir können also auch hier die Idee der Balanced Augmentation verwenden und den Fluss längs p und p' erhöhen.
Wir haben bisher Alternative (ii) nicht näher untersucht. Die Frage, die wir uns zuerst stellen wollen, ist, ob es zu dem beschriebenen Co-Kreis (S, T) immer auch einen symmetrischen Co-Kreis (T', S') gibt. Wir geben die Antwort in folgendem Lemma:

Lemma 9.3 (Symmetrischer Co-Kreis)**.** *Sei (S, T) ein symmetrischer Co-Kreis im schiefsymmetrischen Netzwerk N. Dann ist*

$$(T', S') := \{(ij')|(j'i) \in (S, T)\} \tag{9.29}$$

ebenfalls ein gelb-roter Co-Kreis, in dem alle gelben Kanten die gleiche Richtung haben.

Beweis. Sei $(ij') \in (S, T)$. Wir hatten bereits die Schiefsymmetrie der Färbung bereits angesprochen. Ist (ij') also von einer bestimmten Farbe, so ist es (ji') auch. Damit ist (T', S') bereits ein gelb-roter Co-Kreis.
Nehmen wir nun an $(ij') \in (S, T)$ sei gelb, dann haben (ij') und $(ij')' = (ji')$ die selbe Richtung. Damit haben alle gelben Kanten in (T', S') die selbe Richtung und sie ist sogar die selbe wie im Co-Kreis (S, T). □

Wir kennen nun den symmetrischen Co-Kreis. Wir behandeln den Co-Kreis (S, T), indem wir die Potentiale $\pi(i),\ i \in T$ um ϵ erhöhen. Bereits aus dem Enhanced Primal Dual Algorithmus wissen wir, dass die Potentiale antisymmetrisch sein müssen. Damit können wir die Behandlung des symmetrischen Co-Kreises direkt ableiten:

$$\pi(i) \to \pi(i) + \epsilon \ \forall i \in T \tag{9.30}$$
$$\Rightarrow \quad \pi(i') \to \pi(i') - \epsilon \ \forall i \in T \tag{9.31}$$
$$\Rightarrow \quad \pi(j) \to \pi(j) - \epsilon \ \forall j \in T' \tag{9.32}$$

Als nächstes brauchen wir noch das Äquivalent zu komplementären Kanten auf einem Pfad. Analog zu komplementären Kanten bei Pfaden sind dies die Kanten, die sowohl im Co-Kreis (S, T), als auch im symmetrischen Co-Kreis (T', S') enthalten sind.

Lemma 9.4 (Gemeinsame Kanten)**.** *Es sei (S, T) ein gelb-roter Co-Kreis und (T', S') sei sein komplementärer Co-Kreis. Genau dann sind Kanten (ij') sowohl in (S, T) als auch in (T', S') enthalten, wenn für die Kanten der komplementäre Partner ebenfalls im Co-Kreis ist.*

Beweis. '⇒' Wir nehmen an, $(ij') \in (S,T)$ und $(ij') \in (S,T) \cap (T',S')$. Dann gilt wegen $(ji') \in (T',S')$ auch $(ji') \in (S,T) \cap (T',S')$.
'⇐' Wir nehmen an, es gebe ein Paar komplementärer Kanten $(ij'),(ji') \in (S,T)$. Dann gilt, da beide komplementären Partner auch in (T',S') sein müssen auch $(ij'),(ji') \in (T',S')$ und damit $(ij') \in (S,T) \cap (T',S')$. □

Dieses Lemma sei mit folgender Abbildung noch einmal verdeutlicht (Abb. 9.5). Man sieht dabei einen Schnitt zwischen zwei Mengen, die durch 2 Boxen angedeutet sind. Gemeinsame Kanten des Co-Kreises und des komplementären Co-Kreises sind dann gerade Paare komplementärer Kanten in einem der beiden Co-Kreise.

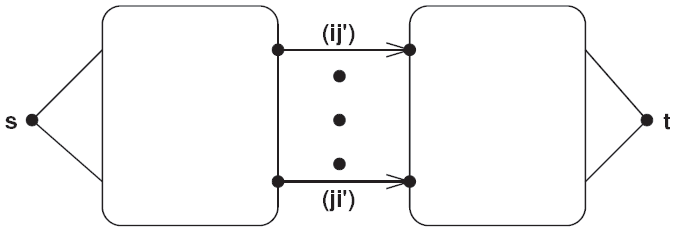

Abbildung 9.5: gemeinsame Kanten von Co-Kreis und komplementärem Co-Kreis

Betrachten wir nun eine Kante $(ij') \in (S,T) \cap (T',S')$, so war unsere antisymmetrische Behandlung der Potentiale offensichtlich noch zu einfach. Im Fall komplementärer Kanten im Schnitt reicht bereits die Hälfte der Potentialänderung aus.
Wir müssen dies also bei der Berechnung des „richtigen" ϵ noch berücksichtigen. Leider führt dies, ähnlich wie bei der Berechnung von δ, zu deutlich längeren Formeln für die $\epsilon_i,\ i \in \{1,2,3,4\}$, die wir bei Berücksichtigung komplementärer Kanten mit $\tilde{\epsilon_i}$ notieren:

$$\begin{aligned}
\tilde{\epsilon_1} &:= \min\{\min\{\pi(j') - \pi(i) - c_{ij'} | (ij') \in R^-,\ x_{ij'} = cap(ij'),\ (ji') \notin R^-\},\\
&\quad \min\left\{\frac{\pi(j') - \pi(i) - c_{ij'}}{2} | (ij') \in R^-,\ x_{ij'} = cap(ij'),\ (ji') \in R^-\right\}\}\\
\tilde{\epsilon_2} &:= \min\{\min\{c_{ij'} - \pi(j') - \pi(i) | (ij') \in R^+,\ x_{ij'} = l_{ij'},\ (ji') \notin R^+\},\\
&\quad \min\left\{\frac{c_{ij'} - \pi(j') - \pi(i)}{2} | (ij') \in R^+,\ x_{ij'} = l(ij'),\ (ji') \in R^+\right\}\}\\
\tilde{\epsilon_3} &:= \min\{\min\{\pi(j') - \pi(i) - c_{ij'} | (ij') \in Y^-,\ l(ij') \le x_{ij'} < cap(ij'),\ (ji') \notin Y^-\},\\
&\quad \min\left\{\frac{\pi(j') - \pi(i) - c_{ij'}}{2} | (ij') \in Y^-,\ l(ij') \le x_{ij'} < cap(ij'),\ (ji') \in Y^-\right\}\}\\
\tilde{\epsilon_4} &:= \min\{c_{ij'} - \pi(j') + \pi(i) | (ij) \in Y^+,\ l(ij') < x_{ij'} \le cap(ij'),\ (ji') \notin Y^+\},\\
&\quad \min\left\{\frac{c_{ij'} - \pi(j') + \pi(i)}{2} | (ij) \in Y^+,\ l(ij') < x_{ij'} \le cap(ij'),\ (ji') \in Y^+\right\}\}
\end{aligned}$$

Um die Notwendigkeit der veränderten Berechnung von ϵ in schiefsymmetrischen Netzwerken deutlich zu machen, führen wir folgende Definition ein:

Definition 9.5 (balancierte Potentialänderung)**.** *Sei (S,T) ein gelb-roter Schnitt im schiefsymmetrischen Netzwerk N und $\tilde{\epsilon_i}$, $i \in \{1,2,3,4\}$ seien definiert wie oben, dann nennen wir*

$$balpot((S,T)) := \min\{\tilde{\epsilon_1}, \tilde{\epsilon_2}, \tilde{\epsilon_3}, \tilde{\epsilon_4}\} \tag{9.33}$$

die balancierte Potentialänderung.

9.3 Algorithmus

Mit diesen Betrachtungen haben wir nun alle Bestandteile zur Formulierung des Algorithmus zusammen. Dieser wird aber nicht notwendigerweise einen Balanced Min Cost Flow auf dem Netzwerk bestimmen, da wir nicht notwendigerweise auf gültigen Pfaden den Fluss erhöhen.
Vielmehr werden wir zeigen, wie wir auf einer bestimmten Klasse von Netzwerken Näherungslösungen bestimmen können. Wir werden später darauf eingehen, wie wir diese Näherungslösungen im Falle des Algorithmus für konvexe Kosten verwenden können.
Die Klasse von Flussnetzwerken, die wir betrachten wollen, habe folgende Eigenschaften:

- alle unteren Grenzen und Kapazitäten seien nichtnegative ganzzahlige Vielfache einer Zahl $cap_{min} \in \mathbb{N}$
- der Anfangsfluss sei auf jeder Kante ein nichtnegatives ganzzahliges Vielfaches von cap_{min}

Wir bezeichnen diese Netzwerke als *cap_{min}-Netzwerke.*
Auf solchen Netzwerken wollen wir Näherungslösungen bestimmen. Hierbei besteht die Näherung darin, dass die Kapazitäten des Netzwerks während des Algorithmus um maximal $\pm\frac{cap_{min}}{2}$ geändert werden. Der Algorithmus bestimmt eine optimale Lösung des BMCF-Problems auf einem solchen veränderten Netzwerk, das wir aber a priori nicht kennen. Wir wollen dies als explizite Definition notieren.

Definition 9.6 (leicht verändertes Netzwerk)**.** *Gegeben sei ein cap_{min}-Netzwerk N und ein weiteres Netzwerk $\tilde{N}$ mit dem gleichen Digraphen. Gilt für alle Kapazitäten $\widetilde{cap}$ von $\tilde{N}$*

$$\widetilde{cap}(ij') \in \left\{cap(ij') + \frac{cap_{min}}{2},\ cap(ij'),\ cap(ij') - \frac{cap_{min}}{2}\right\}\ \forall (ij') \in A(N) \tag{9.34}$$

und für die unteren Grenzen $\tilde{l}$ von $\tilde{N}$

$$\tilde{l}(ij) \in \left\{l(ij') + \frac{cap_{min}}{2},\ l(ij'),\ l(ij') - \frac{cap_{min}}{2}\right\}\ \forall (ij') \in A(N) \tag{9.35}$$

so sagen wir $\tilde{N}$ ist ein leicht verändertes Netzwerk zu N.

Bevor wir nun zum Pseudocode des Algorithmus kommen, wollen wir noch einige Anmerkungen zur Funktionsweise des Algorithmus machen.
Das Färben der Kanten kann analog zur Konstruktion der Kreise im Satz von Minty sukzessive von s und damit auch von t aus vorgenommen werden.
Neben der Färbung müssen auch die Kanten sukzessive untersucht werden. Wir halten hier einige Eigenschaften des Algorithmus fest:

- Der Algorithmus arbeite *symmetrisch*, d.h. untersucht der Algorithmus eine Kante $(ij') \in A(N)$, so untersucht er auch die Kante $(ji') \in A(N)$.
- Knoten $i \in V(N)$ werden *erforscht*: Hierbei werden alle grünen und gelben Kanten, die von dem Knoten ausgehen markiert, als mit dem Knoten verbunden. Rote Kanten werden ebenfalls untersucht, die gefundenen Knoten aber nur vorläufig markiert. Eine Erforschung geht stets nur von endgültig markierten Knoten aus.

Gibt es einen Pfad von s nach t, so erhalten wir einen gelb-grünen Kreis. Gibt es zwischen s und t keinen gültigen Pfad, so enthalte S alle gefärbten Kanten und T sei die Komplementärmenge bezüglich $A(N)$. Es kann nun noch Kanten in T geben, die sich nicht zu einem gültigen Pfad zu t erweitern lassen, so dass (S, T) hier nicht notwendig ein Co-Kreis ist. Das macht hier aber nichts.
Daneben erhalten wir natürlich auch einen symmetrischen gelb-grünen Kreis bzw. einen symmetrischen Co-Kreis.
Wir werden später sehen, dass die vorläufigen und endgültigen Markierungen für die Komplexität des Algorithmus wichtig sind. Eine Markierung setzen wir im Algorithmus durch *label*. Eine Markierung von p' mit *label*$'$. Ein Knoten wird dabei endgültig markiert, wenn er durch einen gelb-grünen Pfad erreichbar ist. Die endgültige Markierung wird durch einen $*$ am *label* angedeutet. Eine vorläufige Markierung zeigt, dass ein Knoten durch einen gelb-grünen Pfad erreichbar sein wird, wenn die Potentiale genug geändert worden sind. Die nötige Änderung ϵ des Potentials wird dabei durch $\eta(j)$ angegeben.

```
Balanced Out-of-Kilter
begin
```

$\pi = 0,\ x = 0$

```
while ((ts) ist out-of-kilter) do
begin (färben)
```

```
färbe eine Kante grün, falls
```
$l(ij) < x_{ij} < cap(ij),\ \pi(j) - \pi(i) = c_{ij}$

```
färbe eine Kante gelb, falls:
```
$x_{ij} < cap(ij),\ \pi(j) - \pi(i) > c_{ij}$
$x_{ij} \leq l(ij),\ \pi(j) - \pi(i) = c_{ij}$
$x_{ij} < l(ij),\ \pi(j) - \pi(i) < c_{ij}$

```
färbe eine Kante gelb und drehe ihre Richtung um, falls:
```
$x_{ij} > cap(ij),\ \pi(j) - \pi(i) > c_{ij}$
$x_{ij} \geq cap(ij),\ \pi(j) - \pi(i) = c_{ij}$
$x_{ij} > l(ij),\ \pi(j) - \pi(i) < c_{ij}$

```
färbe eine Kante rot, falls:
```
$x_{ij} = cap(ij),\ \pi(j) - \pi(i) > c_{ij}$
$x_{ij} = l(ij),\ \pi(j) - \pi(i) < c_{ij}$

```
end;
```
$\eta(i) = \infty\ \forall i \in N$

$label(s) = \oslash$, falls (ts) out-of-kilter
alle anderen Knoten sind nicht markiert
if (alle endgültig markierten Knoten erforscht und t nicht endgültig markiert)
begin (Alternative (ii))

$S := \{$alle endgültig markierten Knoten$\}$
$T := N \backslash S$
for (alle Knoten $i \in T$)

$\pi(i) := \pi(i) + balpot((S,T))$
$\pi(i') := \pi(i') - balpot((S,T))$

if $c_{ij} = \pi(i) - \pi(j)$ then

$label(i) := (label(i))^*$
$label'(i') := (label'(i))^*$

end;
else if (t nicht endgültig markiert)
begin (erforschen)

for (alle Kanten (ij) mit j noch nicht endgültig markiert)

if (ij) gelb oder grün then

$label(j) = i^*$
$label'(j') = i'^*$

for (alle Kanten (ji) mit j noch nicht endgültig markiert)

if (ji) ist grün then

$label(j) = i^*$
$label'(j') = i'^*$

for (alle roten Kanten (ij) mit $x_{ij} = l(ij)$ und $\pi(j) - \pi(i) - c_{ij} < \eta(j)$)

$label(j) = i$
$\eta(j) = \pi(j) - \pi(i) - c_{ij}$
$label'(j') = i'$
$\eta(j') = \pi(i') - \pi(j') - c_{j'i'}$

for (alle roten Kanten (ij) mit $x_{ij} = cap(ij)$ und $\pi(j) - \pi(i) - c_{ij} > \eta(j)$)

$label(j) = i$
$\eta(j) = c_{ji} + \pi(j) - \pi(i)$
$label'(j') = i'$
$\eta(j') = c_{j'i'} + \pi(i') - \pi(j')$

```
end;
else
begin (Balanced Augmentation)
```

finde einen gelb-grünen Kreis, in dem man von t den Weg zu s über die Markierungen sucht
bestimme δ als $\min(\delta_1, \delta_2, \delta_3, \delta_4, cap_{min})$
erhöhe bzw. verringere den Fluss um δ auf diesem erweiternden Pfad
erhöhe bzw. verringere den Fluss um δ auch längs des komplementären Pfades
längs p und p' suche die

(i) grünen Kanten (ij) mit $rescap(ij) = \frac{cap_{min}}{2}$
ist $cap(ij)/cap_{min}$ halbzahlig, so setze $cap(ij) = cap(ij) - \frac{cap_{min}}{2}$
ist $cap(ij)/cap_{min}$ ganzzahlig oder zuvor als gelbe Kante protokolliert, so setze $cap(ij) = cap(ij) + \frac{cap_{min}}{2}$

(ii) grünen Kanten (ij) mit $x_{ij} - l(ij) = \frac{cap_{min}}{2}$
ist $l(ij)/cap_{min}$ halbzahlig, so setze $l(ij) = l(ij) + \frac{cap_{min}}{2}$
ist $l(ij)/cap_{min}$ ganzzahlig oder zuvor als gelbe Kante protokolliert, so setze $l(ij) = l(ij) - \frac{cap_{min}}{2}$

(iii) gelben Kanten (ij) mit $x_{ij} - cap(ij) = \frac{cap_{min}}{2}$
ist $cap(ij)/cap_{min}$ halbzahlig, so setze $cap(ij) = cap(ij) + \frac{cap_{min}}{2}$
ist $cap(ij)/cap_{min}$ ganzzahlig, so setze $cap(ij) = cap(ij) - \frac{cap_{min}}{2}$

(iv) gelben Kanten (ij) mit $l(ij) - x_{ij} = \frac{cap_{min}}{2}$
ist $l(ij)/cap_{min}$ halbzahlig, so setze $l(ij) = l(ij) + \frac{cap_{min}}{2}$
ist $l(ij)/cap_{min}$ ganzzahlig, so setze $l(ij) = l(ij) - \frac{cap_{min}}{2}$

speichere die Änderungen in einer Liste
lösche alle Markierungen

```
end;
end;
```

Wir hatten zuvor bereits angesprochen, dass der Algorithmus nur eine Näherungslösung des BMCF Problems auf dem Netzwerk bestimmt. Wir wollen deshalb die Korrektheit des Algorithmus in diesem Sinne beweisen.

Satz 9.7 (Korrektheit und Komplexität des Balanced Out-of-Kilter)**.** *Sei N ein schiefsymmetrisches cap_{min}-Netzwerk mit einem balancierten Anfangsfluss, dann bestimmt der*

Balanced Out-of-Kilter Algorithmus eine optimale Lösung des BMCF-Problems auf einem leicht veränderten Netzwerk zu N in $O(\frac{K}{cap_{min}} \cdot m)$ mit K der anfänglichen Summe aller Kilter-Zahlen.

Beweis. Wir stellen zunächst fest, dass in einem cap_{min}-Netzwerk die Kilter-Zahlen alle nichtnegative ganzzahlige Vielfache von cap_{min} sind.
Unser Beweis läuft nun in 2 Schritten. Zuerst untersuchen wir die verschiedenen Möglichkeiten aus dem Satz von Minty gegen die Lösung zu iterieren. Hierbei nehmen wir an, die Kilter-Zahlen seien nicht nur anfangs sondern immer ganzzahlige positive Vielfache von cap_{min} oder 0. Im zweiten Schritt zeigen wir, dass diese Annahme korrekt ist. Untersuchen wir also zunächst die Häufigkeit des Auftretens der verschiedenen Fälle aus dem Satz von Minty unter den genannten Annahmen.
Nehmen wir zunächst an, es wird ein gelb-grüner Kreis gefunden. Dann haben wir 2 Pfade gefunden, auf denen wir den Fluss erhöhen. Damit werden mindestens 2 Kilter-Zahlen um $\delta \geq \frac{cap_{min}}{2}$ reduziert. Es gilt hierbei $\delta \geq \frac{cap_{min}}{2}$ nach obiger Annahme für die unteren Grenzen und Kapazitäten und der Wahl von δ bei der Flusserhöhung.
Somit sind nicht mehr als K/cap_{min} Iterationen auf einer Zirkulation möglich, wobei K die Summe aller Kilter-Zahlen für die Anfangszirkulation sei.
Jedes Mal, wenn ein gelb-roter Co-Kreis (und sein symmetrischer Partner) gefunden wird, gehen entweder 2 Kanten in-kilter oder mindestens 2 rote Kanten werden gelb. Der erste Fall reduziert 2 positive Kilter-Zahlen auf 0. Da die Kilter-Zahlen nach Annahme immer positive Vielfache von cap_{min} sind, bringt die Reduktion zweier positiver Kilter-Zahlen auf 0 immer eine Reduktion von K um $2 \cdot cap_{min}$. Damit kann auch dieser Fall nicht häufiger als K/cap_{min} mal auftreten.
Der 2. Fall kann nicht mehr als $n-1$ Mal in Folge auftreten, was wir im Folgenden zeigen werden.
Wir nehmen an, dass für eine Kante (ts) so lange der Satz von Minty angewandt wird, bis ein gelb-grüner Kreis gefunden wird. Jedes Mal wenn ein Co-Kreis (und sein symmetrischer Partner) gefunden wird, wird eine rote Kante gelb und es gibt zu mindestens einem Knoten in T einen zulässigen Pfad. Alle Knoten in S bleiben, wie zuvor bereits erwähnt erreichbar. Analoges gilt für den symmetrischen Co-Kreis.
Somit kann der 2. Fall nur $n-1$ Mal auftreten, bevor entweder ein Kreis gefunden wird, oder eine Kante in-kilter geht.
Man kann den Algorithmus im 2. Fall noch etwas besser machen, wie wir es im vorliegenden Algorithmus mit den vorläufigen Markierungen getan haben. Nach unseren Vorbermerkungen über die Markierungen können wir ausnutzen, dass die Markierungen im 2. Fall erhalten bleiben können, da alle Knoten, die von s aus erreichbar sind, auch erreichbar bleiben. Für Fall 2 reicht also eine Markierung, die wir erst erneuern müssen, wenn entweder ein gelb-grüner Kreis gefunden wurde oder ein gelb-roter Co-Kreis für den der erste Fall zutrifft.
Nun ist K/cap_{min} eine obere Grenze für die Anzahl der Fälle, in denen ein gelb-grüner Kreis gefunden wird oder ein gelb-roter Co-Kreis, für den Fall 1 zutrifft. Somit kann die Markierung nur maximal K/cap_{min} mal durchgeführt werden. Damit ist $O((K/cap_{min}) \cdot m)$ eine obere Schranke für die Laufzeit dieser Version des Balanced Out-of-Kilter Algorithmus.
Wir halten an dieser Stelle zudem fest, dass wir den Fluss immer symmetrisch erhöhen

und die Potentiale immer antisymmetrisch verändern. Folglich bleibt das Netzwerk in jedem Schritt schiefsymmetrisch und der Fluss balanciert.
Wir kommen nun zum 2. Teil unseres Beweises. Wir zeigen per Induktion, dass die Kilter-Zahlen in jedem Schritt positive ganzzahlige Vielfache von cap_{min} oder 0 sind.
Der Induktionsanfang ist mit unserer anfänglichen Bemerkung klar.
Wir unterscheiden für den Induktionsschritt die verschiedenen Fälle für grüne und gelbe Kanten.
Im 1. Fall der Potentialänderung werden Kilter-Zahlen nur auf 0 verändert und die Flüsse bleiben unverändert.
Die Kilter-Zahlen und Flüsse ändern sich sonst nur in einem Erhöhungsschritt. Wir betrachten hier jeweils exemplarisch einen Fall für die grünen und gelben Kanten. Bei einem Erhöhungsschritt gilt in jedem Fall $\delta \in \{\frac{cap_{min}}{2}, cap_{min}\}$ nach Vorraussetzung. Der Fall $\delta = \frac{cap_{min}}{2}$ tritt dabei bei 2 komplementären Kanten auf einem Pfad auf. Betrachten wir nun getrennt die gelben und grünen Kanten.
Es kann für grüne Kanten der Fall $rescap(ij) = \frac{cap_{min}}{2}$ eintreten. Dies würde beim nächsten Erhöhungsschritt $\delta < \frac{cap_{min}}{2}$ ermöglichen. Ist dieser Fall bei der Kante bisher noch nicht aufgetreten, so vergößern wir die Kapazität um $\frac{cap_{min}}{2}$ und protokollieren diesen Schritt. Damit haben wir für die Restkapazität wieder ein Vielfaches von cap_{min}. Die Kapazität ist nach einem solchen Schritt halbzahlig.
Damit können wir das vorherige Auftreten eines solchen Falles sofort registrieren. Gilt wieder $rescap(ij) = \frac{cap_{min}}{2}$, so können wir zur alten Kapazitätsgrenze zurückkehren und erhalten $x_{ij} = l(ij)$ und damit eine gelbe Kante mit Kilter-Zahl 0.
Der andere Fall bei grünen Kanten ist analog.
Betrachten wir nun die gelben Kanten. Hier kann der Fall $x_{ij} - cap(ij) = \frac{cap_{min}}{2}$ eintreten. Dies würde wieder beim nächsten Erhöhungsschritt $\delta < \frac{cap_{min}}{2}$ ermöglichen. Ist dieser Fall bei der Kante bisher noch nicht aufgetreten, so verringern wir die Kapazität um $\frac{cap_{min}}{2}$. Damit erlaubt die Kante im nächsten Schritt wieder eine Erhöhung des Flusses um cap_{min}. Die Kapazität ist nach einem solchen Schritt halbzahlig. D.h. wir können analog zum Fall grüner Kanten diesen Fall registrieren und beim nächsten Auftreten von $x_{ij} - cap(ij) = \frac{cap_{min}}{2}$ zur alten Kapazität zurückkehren. Beim 1. Auftreten des Falles wird die Kilter-Zahl um $\frac{cap_{min}}{2}$ erhöht. Das macht aber nichts, da wir die Kilter-Zahl mit der Erhöhung zuvor um mindestens $2 \cdot \frac{cap_{min}}{2}$ reduziert haben.
Wieder ist der andere Fall für gelbe Kanten analog.
Betrachten wir nun noch den Fall protokollierter gelber Kanten, die grün werden. Wir sehen in diesem Fall, dass die Veränderung der Kapazität, wie im Algorithmus beschrieben, zur alten Kapazität zurückführt und keine Kilter-Zahl erhöht.
Der Fall grüner Kanten, die gelb werden, muss hierbei, da sich die Kilter-Zahl im Algorithmus nicht erhöhen kann, nicht behandelt werden.
Damit haben wir den Induktionsschritt.
Wir halten nun noch fest, dass nach Wahl unserer Kapazitätsveränderung das Netzwerk in jedem Schritt ein leicht verändertes Netzwerk zu N bleibt. Am Ende gilt $K(x_{ij'}) = 0 \ \forall (ij') \in A(N)$. Damit ist die Lösung optimal und zwar auf einem leicht veränderten Netzwerk zu N. □

9.4 Konvexe Kosten

Wir hatten bereits erwähnt, dass die ursprüngliche Idee einer schwach polynomiellen Lösung für das Min Cost Flow Problem mit konvexen Kosten auf Minoux [49] zurückgeht. Wir wollen deshalb nun zeigen, wie der beschriebene Out-of-Kilter Algorithmus zur Lösung von BMCF benutzt werden kann.
Die Idee von Minoux ist ähnlich der, die wir beim Balanced Capacity Scaling angewendet haben. Minoux bezeichnet seine Methode als Approximation p'ter Ordnung. Hierbei ist p wieder die Anzahl der Punkte, in denen die Kostenfunktion linear approximiert wird.
Wir führen zunächst den Begriff des Netzwerks p'ter Ordnung ein:

Definition 9.8 (Netzwerk p'ter Ordnung)**.** *Das Netzwerk p'ter Ordnung $N(p)$ ist das Netzwerk mit den selben Kanten und Knoten wie das Ausgangsnetzwerk N, jedoch mit den Kapazitäten:*

$$cap^p(ij) = 2^p \left\lceil \frac{cap(ij)}{2^p} \right\rceil \tag{9.36}$$

$$l^p(ij) = 2^p \left\lfloor \frac{l(ij)}{2^p} \right\rfloor \tag{9.37}$$

Dies erinnert bereits stark an die Δ-Restnetzwerke aus dem Balanced Capacity Scaling. Natürlich wollen wir auch wieder die Kostenfunktionen sukzessive besser approximieren. Wir bezeichnen dabei unser Ausgangsproblem mit P:

$$\min C(x) = \sum_{(ij)\in A(N)} C_{ij}(x_{ij}) \tag{9.38}$$

$$x \in \mathfrak{P}(x) \tag{9.39}$$

Die Idee ist erneut die Kostenfunktion auf dem Netzwerk $N(p)$ besser zu approximieren. Dazu definieren wir uns Probleme $PA(p)$, bei denen die zugehörige Kostenfunktion in allen Abszissenwerten $k \cdot 2^p$ mit $k \in \mathbb{N}$ und

$$\left\lfloor \frac{l(ij)}{2^p} \right\rfloor \leq k \leq \left\lceil \frac{cap(ij)}{2^p} \right\rceil \tag{9.40}$$

die ursprügliche Kostenfunktion $C_{ij}(x_{ij})$ linear approximiert.
Wir wählen hierbei natürlich ganz analog zum Balanced Capacity Scaling $\bar{p}$ anfangs so, dass mit der maximalen Kapazität U einer Kante im Netzwerk gilt $\bar{p} = \lceil \log_2 U \rceil$. Die Idee ist damit nun vermutlich bereits offensichtlich. Wir bestimmen zunächst eine Lösung des Problems $PA(\bar{p})$, nehmen diese dann als Startlösung für $PA(\bar{p}-1)$ usw..
Diese Idee wollen wir nun mathematisch im Satz von Minoux formulieren. Der Originalbeweis dieses Satzes findet sich in zwei Teilen in [50, S. 366-368] und [49, S. 238]. Der Autor will hier aber eine eigene Version des Beweises anbieten:

Satz 9.9 (Satz von Minoux)**.** *Es gilt für die Probleme $PA(p)$:*

(i) Eine optimale Lösung des Problems $PA(0)$ ist eine optimale, ganzzahlige Lösung des Convex BMCF Problems.

(ii) Für jedes $p < \bar{p}$ (bzw. $p = \bar{p}$) kann von einer Startlösung durch den Balanced Out-of-Kilter von $PA(p+1)$ auf einem leicht veränderten Netzwerk zu $N(p+1)$ (bzw. dem 0-Fluss und dem 0-Potential) eine Lösung von $PA(p)$ auf einem leicht veränderten Netzwerk zu $N(p)$ in einer Zeit $O(m^2)$ mit dem Balanced Out-of-Kilter Algorithmus bestimmt werden.

Beweis. Wir beweisen zunächst Teil (i). Für $p = 0$ wird die Kostenfunktion in allen Punkten $k \in \mathbb{N}$ mit $\left\lfloor \frac{l(ij)}{2^0} \right\rfloor = l(ij) \leq k \leq cap(ij) = \left\lceil \frac{cap(ij)}{2^0} \right\rceil$ approximiert. Die Kostenfunktion wird also in allen ganzzahligen Punkten linear approximiert. Wir hatten bereits im Beweis zur Korrektheit des primal-dualen Algorithmus für konvexe Kosten gezeigt, dass eine optimale Lösung des Problems mit einer in allen ganzzahligen Punkten linear approximierten Kostenfunktion auch die optimale ganzzahlige Lösung des ursprünglichen Problems ist, wenn die Kostenfunktion als konvex angenommen wird. Damit haben wir den ersten Teil. Der Balanced Out-of-Kilter Algorithmus verändert allerdings das Netzwerk, sodass wir für eine optimale ganzzahlige Lösung auf dem Ausgangsnetzwerk diesen nicht verwenden können.
Wir wollen nun auch den 2. Teil des Satzes beweisen. Wir beginnen dabei mit $p = \bar{p}$. Nach Wahl von $\bar{p}$ gilt anfangs für $k \in \mathbb{N}$: $0 \leq k \leq 1$. Damit haben wir gerade den Fall linearer Kosten auf allen Kanten mit:

$$cc_{ij} = \frac{C_{ij}(2^{\bar{p}}) - C_{ij}(0)}{2^{\bar{p}}} \tag{9.41}$$

Auf dem Netzwerk $\bar{p}$'ter Ordnung gilt für die Kapzitäten:

$$cap^{\bar{p}}(ij) = 2^{\bar{p}} \left\lceil \frac{cap(ij)}{2^{\bar{p}}} \right\rceil = 2^{\bar{p}} \geq U \tag{9.42}$$

$$l^{\bar{p}}(ij) = 2^{\bar{p}} \left\lfloor \frac{l(ij)}{2^{\bar{p}}} \right\rfloor = 0 \tag{9.43}$$

Wir verwenden die 0-Lösung und das 0-Potential am Anfang. Damit ist unser Netzwerk anfangs ein cap_{min}-Netzwerk mit $cap_{min} = 2^{\bar{p}}$. Wir können somit zur Lösung des Problems $PA(\bar{p})$ den „normalen“ Balanced Out-of-Kilter Algorithmus verwenden.
Wir kommen damit zur Komplexität. Die Differenz des Flusses auf einer Kante $(ij) \in A(N)$ zur oberen Kapazität und damit die Kilter-Zahl der Kante kann nicht mehr als $2^{\bar{p}}$ betragen. Wir können damit die Summe der Kilter-Zahlen K abschätzen:

$$K \leq m \cdot 2^{\bar{p}} \tag{9.44}$$

Im Beweis zur Komplexität des Out-of-Kilter Algorithmus hatten wir gezeigt, dass wir die Anzahl der möglichen Flusserhöhungen abschätzen können durch $\frac{K}{cap_{min}}$. In diesem Fall gilt:

$$\frac{K}{cap_{min}} \leq m \cdot \frac{2^{\bar{p}}}{2^{\bar{p}}} = m \tag{9.45}$$

Damit ergibt sich mit der Diskussion aus dem Satz über die Komplexität des Balanced Out-of-Kilter Algorithmus eine Komplexität für das Problem $PA(\bar{p})$ von $O\left(\frac{K}{cap_{min}} \cdot m\right) =$

$O(m^2)$.
Nehmen wir nun an, wir hätten eine optimale Lösung von $PA(p+1)$ auf einem leicht veränderten Netzwerk zu $N(p+1)$ bestimmt und würden diese als Startlösung für das Problem $PA(p)$ verwenden. Um die Laufzeit zur Bestimmung einer Lösung von $PA(p)$ auf einem leicht veränderten Netzwerk zu $N(p)$ zu bestimmen, müssen wir zunächst klären, wie wir die stückweise linearen Kosten auf dem Netzwerk implementieren.
Es gilt hierbei aufgrund der Definition von δ im Algorithmus für die Flusserhöhung längs der erweiternden Wege p und p' die untere Grenze $\delta \geq 2^{p-1}$.
Wir stellen ferner fest, dass per Definition im Netzwerk p'ter Ordnung die Kapazitäten der Kanten immer ein ganzzahliges Vielfaches von 2^p sind:

$$cap^p(ij) - l^p(ij) = 2^p \left\lceil \frac{cap(ij)}{2^p} \right\rceil - 2^p \left\lceil \frac{l(ij)}{2^p} \right\rceil \tag{9.46}$$

$$= n2^p,\ n \in \mathbb{N}_0 \tag{9.47}$$

Wir können also analog zum Verfahren im Balanced Capacity Scaling Algorithmus für konvexe Kosten für jede Kante $(ij') \in A(N)$ nun $p_{ij'}$ Kanten mit der Kapazität 2^p einführen [32, S. 206]. Wählen wir allerdings die maximale Flussveränderung auf einer Kante $\delta \leq 2^p$ müssen wir auch wieder nur 2 Kanten mit Kapazität 2^p betrachten, die jeweils die Flussveränderung um $\pm 2^p$ darstellen:

$$cc_{ij} = \frac{C_{ij}(x_{ij} + 2^p) - C_{ij}(x_{ij})}{2^p} \tag{9.48}$$

$$cap_{2^p}(ij) = \begin{cases} 2^p, \text{ falls } cap(ij) - x_{ij} \geq 2^p \\ 0, \text{ sonst} \end{cases} \tag{9.49}$$

$$cc_{ji} = \frac{C_{ij}(x_{ij} - 2^p) - C_{ij}(x_{ij})}{2^p} \tag{9.50}$$

$$cap_{2^p}(ji) = \begin{cases} 2^p, \text{ falls } x_{ij} \geq 2^p \\ 0, \text{ sonst} \end{cases} \tag{9.51}$$

Bei der besprochenen Kapazitätsanpassung grüner und gelber Kanten bedeutet dies natürlich, dass wir die Kapazitäten der einzelnen Kanten $(ij)^l,\ l \in \{1, \cdots, p_{ij}\}$ verändern. $l(ij)/cap_{min}$ und $cap(ij)/cap_{min}$ beziehen sich hierbei natürlich auf die Gesamtkapazität. Dementsprechend muss also nicht die aktuelle Kante $(ij)^l$ protokolliert worden sein, sondern irgendeine der Kanten $(ij)^k,\ k \in \{1, \cdots, p_{ij}\}$ in unserer Bedingung.
Damit haben wir also auch bei dieser Segmentierung der Kanten jeweils Flusserhöhungen, die immer größer oder gleich $\frac{cap_{min}}{2}$ sind.
Wurde die Kapazitätsgrenze einer Kante $(ij)^l$ erreicht, so nehmen wir die nächste Kante $(ij)^{l+1}$ mit neuen reduzierten Kosten cc_{ij} und cc_{ji} in das Netzwerk auf und löschen die alte Kante.
Damit haben wir die Implementierung der stückweise linearen Kosten besprochen. Betrachten wir nun die Startlösung aus dem Problem $PA(p+1)$ noch einmal genauer. Die Flusswerte auf dem leicht veränderten Netzwerk zu $N(p+1)$ sind immer ganzzahlige nichtnegative Vielfache von $\frac{2^{p+1}}{2} = 2^p$. Damit ist unser Netzwerk für das Problem $PA(p)$ ein cap_{min}-Netzwerk mit $cap_{min} = 2^p$.
Wir müssen nun wieder die Kilter-Zahl abschätzen um die Komplexität zu bestimmen.

Die Kilter-Zahl im leicht veränderten Netzwerk zu $N(p+1)$ ist 0. Sie erhöht sich aber beim Übergang zum Netzwerk $N(p)$.
Als erste „Quelle“ für eine Erhöhung der Kilter-Zahl betrachten wir hierbei die Einführung der neuen Kosten auf den Kanten cc_{ij}. Sei dazu die Kante $(ij)^{l'}$ im Netzwerk $(p+1)$'ter Ordnung die letzte voll gefüllte oder die letzte halb gefüllte Kante $l' \in \{1, \cdots p'_{ij}\}$ mit p'_{ij} der Zahl der Punkte, in denen die Kostenfunktion linear approximiert wird. Wir betrachten zunächst den Fall einer voll gefüllten Kante $(ij)^{l'}$. Dann gilt:

$$cc^{p+1}_{(ij)^{l'}} = \frac{C_{ij}(2^{p+1} + (l'-1)\cdot 2^{p+1}) - C_{ij}((l'-1)\cdot 2^{p+1})}{2\cdot 2^p} \leq 0 \tag{9.52}$$

Diese wird im Netzwerk p'ter Ordnung aufgespalten in 2 Kanten, die wir mit $(ij)^l$ und $(ij)^{l+1}$ bezeichnen wollen. Hierbei sei $l \in \{1, \cdots, p_{ij} - 1\}$ wieder die Nummer der Kante und $p_{ij} = 2p'_{ij}$ die Zahl der Punkte, in denen die Kostenfunktion linear approximiert wird. Betrachten wir zunächst die Kante $(ij)^{l+1}$:

$$cc^{p}_{(ij)^{l+1}} = \frac{C_{ij}(2^p + l\cdot 2^p) - C_{ij}(l\cdot 2^p)}{2^p} \geq cc^{p+1}_{(ij)^{l'}} \tag{9.53}$$

Da $(ij)^{l'}$ per Definition in-kilter war, ist die neue Kante eventuell out-of-kilter und die Kilter-Zahl würde sich um 2^p erhöhen.
Betrachten wir nun die zuvor aufgefüllten Kanten. Es gilt hierbei $cc^p_{(ij)^l} \geq cc^p_{(ij)^k}$ mit $k \in \{1, \cdots, l-1\}$ und

$$\begin{aligned} cc^{p}_{(ij)^{l}} &= \frac{C_{(ij)^l}(2^p + (l-1)\cdot 2^p) - C_{(ij)^l}((l-1)\cdot 2^p)}{2^p} && (9.54)\\ &\leq \frac{C_{(ij)^{l'}}(2^{p+1} + (l'-1)\cdot 2^{p+1}) - C_{(ij)^{l'}}((l'-1)\cdot 2^{p+1})}{2^{p+1}} && (9.55)\\ &= cc^{p+1}_{(ij)^{l'}} && (9.56) \end{aligned}$$

Damit bleiben die sonstigen bereits aufgefüllten Kanten also in-kilter.
Betrachten wir nun den Fall $x_{(ij)^{l'}} = 2^p$, also einer halb gefüllten Kante, so kann man analog zeigen, dass auch hier nur die letzte der im Netzwerk p'ter Ordnung eingeführten Kanten out-of-kilter geht.
Durch den Übergang zum Netzwerk p'ter Ordnung erhalten wir damit eine Erhöhung der Kilter-Zahl um maximal $m \cdot 2^p$.
Als nächstes betrachten wir die Veränderung der Kapazitäten und unteren Grenzen. Wir wollen daher eine obere Schranke für die Kapazitätsdifferenz zwischen den Kanten im Netzwerk $(p+1)$'ter Ordnung und den Kanten im Netzwerk p'ter Ordnung ableiten:

$$\begin{aligned} cap^{p+1}(ij) - cap^p(ij) &= 2^{p+1}\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil - 2^p \left\lceil \frac{cap(ij)}{2^p} \right\rceil \geq 0 && (9.57)\\ &= 2^{p+1}\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil - \frac{1}{2}2^{p+1}\left\lceil \frac{cap(ij)}{2^p} \right\rceil && (9.58)\\ &= 2^{p+1}\left(\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil - \frac{1}{2}\left\lceil \frac{cap(ij)}{2^p} \right\rceil\right) && (9.59) \end{aligned}$$

Wir schreiben nun die Kapazität in Binärdarstellung: $cap(ij) = \sum_{i=0}^{k} a_i \cdot 2^i$. Damit gilt:

$$\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil = \left\lceil \frac{\sum_{i=0}^{k} a_i \cdot 2^i}{2^{p+1}} \right\rceil = \left\lceil \sum_{i=0}^{k} a_i \cdot 2^{i-p-1} \right\rceil \tag{9.60}$$

$$\left\lceil \frac{cap(ij)}{2^{p}} \right\rceil = \left\lceil 2 \sum_{i=0}^{k} a_i 2^{i-p-1} \right\rceil \geq 2 \cdot \left\lceil \sum_{i=0}^{k} a_i \cdot 2^{i-p-1} \right\rceil - 1 \tag{9.61}$$

Damit erhalten wir nun:

$$cap^{p+1}(ij) - cap^{p}(ij) = 2^{p+1} \left(\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil - \frac{1}{2} \left\lceil \frac{cap(ij)}{2^{p}} \right\rceil \right) \tag{9.62}$$

$$\leq 2^{p+1} \left(\left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil - 1\frac{1}{2} \cdot 2 \cdot \left\lceil \frac{cap(ij)}{2^{p+1}} \right\rceil + \frac{1}{2} \right) \tag{9.63}$$

$$= 2^{p} \tag{9.64}$$

Analoges gilt für ein Defizit auf einer Kante.
Damit haben wir also aufgrund der Veränderung der unteren Grenzen und der Kapazitäten Überschüsse bzw. Defizite von $\pm 2^p$. Die Kilter-Zahl erhöht sich damit um maximal um $2m \cdot 2^p$.
Als letzten Punkt betrachten wir noch die Kapazitäten im leicht veränderten Netzwerk zu $N(p+1)$. Kehren wir zu den ursprünglichen Kapazitäten zurück, so ergeben sich auf den Kanten maximal Überschüsse bzw. Defizite von $\pm 2^p$. Die Kilter-Zahl erhöht sich damit noch einmal um $2 \cdot m \cdot 2^p$.
Insgesamt hat unsere Startlösung damit eine Kilter-Zahl von $K \leq 5 \cdot m \cdot 2^p$.
Damit haben wir nach unserem Beweis zur Komplexität des Balanced Out-of-Kilter Algorithmus eine Komplexität von $O(\frac{K}{cap_{min}} \cdot m) = O(5 \cdot m^2) = O(m^2)$. □

Den Algorithmus zu notieren ist nun denkbar einfach:

```
Balanced Out-of-Kilter für konvexe Kosten
π := 0, x := 0
p̄ = ⌈log2(U)⌉
löse das Problem PA(p̄) mit dem Out-of-Kilter Algorithmus
for p = p̄ − 1 to 1 do

    berechne die Kapazitäten neu
    verteile die Flüsse auf die neuen Kanten und berechne die Kantenkosten
    im Netzwerk p'ter Ordnung
    löse das Problem PA(p) mit dem Out-of-Kilter Algorithmus, wobei nach
    jeder Flusserhöhung eventuell neue Kanten eingeführt werden

end;
```

Korollar 9.10 (Komplexität des Balanced Out-of-Kilter Algorithmus für konvexe Kosten)**.** *Der Balanced Out-of-Kilter Algorithmus für konvexe Kosten terminiert in einer Laufzeit* $O(\log_2 U \cdot m^2)$.

Beweis. Die Komplexität ergibt sich mit dem dargestellten Algorithmus aus dem 2. Teil des Satzes von Minoux. Die Lösung von $PA(\bar{p})$ erfordert demnach eine Laufzeit von $O(m^2)$. Jeder Schritt der Schleife erfordert nach dem Satz von Minoux ebenfalls eine Komplexität von $O(m^2)$. Das Update der Kosten auf den Kanten von p und p' nach jedem Erhöhungsschritt verändert dabei die Komplexität des Out-of-Kilter Algorithmus offensichtlich nicht.
Damit haben wir eine Gesamtkomplexität von $O(m^2 \cdot \bar{p})$. Es gilt $\bar{p} = \lceil \log_2 U \rceil$, also $\bar{p} = O(\log_2 U)$ und damit folgt die Behauptung. □

Die Frage ist nun, welchen Nutzen diese Näherungslösung hat. Der Algorithmus terminiert mit einer optimalen Lösung auf dem veränderten Netzwerk mit $p = 1$. Die Kilter-Zahlen sind also alle ganzzahlig, wenn wir den Übergang zum Netzwerk 0'ter Ordnung betrachten.
Die Idee ist nun, ähnlich wie im Enhanced Primal-Dual Algorithmus, für den letzten Schritt einen anderen Algorithmus zu verwenden.
Für den letzten Schritt wollen wir dabei den Balanced Capacity Scaling Algorithmus verwenden. Gehen wir zum Netzwerk 0'ter Ordnung über, so haben wir eine Kilter-Zahl $K \leq 5 \cdot m$. Verteilen wir die Überschüsse und Defizite nicht auf die Kanten, sondern auf die Knoten ergibt sich damit eine obere Grenze für die Überschüsse auf den Knoten von $5 \cdot m$. Wir haben gezeigt, dass alle sonstigen Kanten die Complementary Slackness Optimality erfüllen und damit auch die Reduced Cost Optimality.
Wir wollen nun den letzten Schritt mit einer Phase des Balanced Capacity Scaling Algorithmus erledigen. Alle Kanten, die die Reduced Cost Optimality nicht erfüllen, haben, da die Kilter-Zahlen alle ganzzahlig sind, mindestens eine Restkapazität von 1. Wir setzen also $\Delta = 1$ und starten die letzte Phase des Balanced Capacity Scaling.
Für die Laufzeit einer Phase des Balanced Capacity Scaling entnehmen wir aus unseren Überlegungen zum Balanced Capacity Scaling und der Abschätzung der Überschüsse $O(5 \cdot m \cdot S(n, m))$. Wir erhalten damit die balancierte Lösung des Problems in einer Laufzeit $O(m^2 \log_2 U + mS(n, m))$.
Dies mag zunächst verwundern, da wir uns in diesem Algorithmus an keiner Stelle, ausser im letzten Schritt um gültige Pfade gekümmert haben. In der Tat sind aber alle Pfade gültig, wenn die Kapazitäten der Kanten alle größer als 2 sind. Dementsprechend braucht man die Bestimmung gültiger Pfade erst im allerletzten Schritt.
Man kann diese Idee auch auf das Capacity Scaling übertragen und auch dort erst im letzten Schritt gültige Pfade bestimmen. Man erhält dann eine ähnlich gute Laufzeit wie für den hier beschriebenen Balanced Out-of-Kilter Algorithmus.
Wir wollen außerdem darauf hinweisen, dass wir nur für den Balanced Capacity Scaling Algorithmus in der letzten Phase des Balanced Out-of-Kilter die Annahme konvexer Kosten mit positiver Steigung benötigen. Wir können im Balanced Capacity Scaling Algorithmus auch den Algorithmus zur Bestimmung allgemeiner gültiger kürzester Pfade nach Jungnickel und Fremuth-Paeger verwenden. In diesem Fall müssen wir uns nicht auf den Fall konvexer Kosten mit positiver Steigung beschränken. Der SVP-Algorithmus hat nun allerdings lediglich noch eine Komplexität von $O(mn)$. Wir erhalten also mit dem Balanced Out-of-Kilter unter Verwendung dieses abgewandelten Balanced Capacity Scaling in der letzten Phase eine Komplexität für das allgemeine Convex-BMCF Problem

von $O(m^2 \log_2 U + m^2 n)$. Die Komplexität ist also nur wenig schlechter als im Fall des Convex BMCF-Problems mit einer steigenden konvexen Kostenfunktion.

Kapitel 10

Enhanced Balanced Capacity Scaling

Wir wollen die letzte Idee zu einer Weiterentwicklung des Capacity Scaling noch etwas weiter ausformulieren. Sind wir im Balanced Capacity Scaling Algorithmus in einer Δ-scaling-phase mit $\Delta \geq 2$, so sind im Δ-Restnetzwerk in jedem Schritt nur Kanten mit einer Restkapazität von mindestens 2 enthalten. Auf einem Pfad im Δ-Restnetzwerk kann also in diesem Fall keine Kante $(ij') \in A(N(\Delta))$ existieren mit $rescap(ij') = 1$. Damit sind kürzeste Pfade auch gleichzeitig kürzeste gültige Pfade. Kürzeste Pfade bei positiven Kosten können wir mit einer Implementierung des Dijkstra-Algorithmus mit sogenannten Fibonacci-Heaps in $O(n \log(n) + m)$ berechnen (siehe auch [3, S. 116]). Nutzen wir in der letzten Phase $\Delta = 1$ den Algorithmus von Goldberg und Karzanov zur Berechnung des SVP mit $S(n, m) = O(m \log(n))$, so erhalten wir eine Gesamtkomplexität für diese Version des Balanced Capacity Scaling von $O(m^2 \log_2 U + n \log(n) \log_2 U + m^2 \log(n))$.
Nutzen wir für den Balanced Out-of-Kilter Algorithmus in der letzten Phase ebenfalls den Algorithmus von Goldberg und Karzanov zur Berechnung der kürzesten gültigen Pfade, so erhalten wir eine Komplexität von $O(m^2 \log_2 U + m^2 \log(n))$. Die Komplexität der beiden Algorithmen ist also fast identisch.

Kapitel 11

Vergleich und weitere Algorithmen

Wir haben als schwach-polynomielle Algorithmen zur Lösung des Convex BMCF Problems bisher den Enhanced Balanced Capacity Scaling und den Balanced Out-of-Kilter Algorithmus betrachtet.
Beide Algorithmen sind nur für konvexe Kostenfunktionen mit positiver Steigung beschrieben worden und beide beginnen mit dem 0-Fluss und dem 0-Potential. Auch die Idee der Approximation der Kostenfunktion ist in beiden Fällen sehr ähnlich.
Die beiden Algorithmen unterscheidet allerdings die Herangehensweise. Im Balanced Capacity Scaling werden die duale und die primale Lösung immer gemeinsam verbessert, während der Balanced Out-of-Kilter Algorithmus explizit zwischen dem dualen und dem primalen Update unterscheidet. Ferner müssen wir im Balanced Out-of-Kilter Algorithmus keine künstlichen Kanten einführen. Dagegen benötigt das Balanced Capacity Scaling die vom Autor entwickelte Kapazitätsveränderung.
Insgesamt halten wir an dieser Stelle fest, dass die Komplexität der Algorithmen auf „normalen“ Netzwerken praktisch identisch ist zu der Komplexität der hier beschriebenen Algorithmen auf schiefsymmetrischen Netzwerken.
Es muss an dieser Stelle aber erwähnt werden, dass die Komplexität des Balanced Out-of-Kilter im Falle des allgemeinen Convex BMCF-Problems ohne die Annahme einer steigenden Kostenfunktion deutlich besser als die Komplexität des Capacity Scaling ist.
Die Frage ist nun, ob diese Komplexität noch weiter verbessert werden kann. Der bisher noch nicht dargestellte Weg sind duale Ansätze. Wir wollen diese im Anhang der Arbeit genauer betrachten.
Der erste Ansatz ist der Balanced Relaxation Algorithmus. Bisher haben wir immer Algorithmen betrachtet, die ganzzahlige Lösungen bestimmen. In diesem Algorithmus lassen wir diese Bedingung fallen. Damit bestimmen wir dann aber auch kein Matching im eigentlichen Sinne mehr. Interessant sind an diesem Algorithmus aber die Ideen aus der nichtlinearen Optimierung (Lagrange Formalismus) die einfließen. Leider ist die Laufzeit des Algorithmus aber schlecht, sodass der Autor den Balanced Relaxation Algorithmus eher als Denkanstoß sieht, um über bessere Varianten nachzudenken. Zudem sollen die Ansätze aus der nichtlinearen Optimierung, die vor allem von Bertsekas [10] vorangetrieben wurden, hier nicht ganz ausgelassen werden.
Die aussichtsreichste bessere Variante ist die Cancel and Tighten Methode von Karzanov und McCormick [38]. Er basiert im Grunde auf einer Idee von Goldberg und Tarjan

[31]: Aus der Negative Cycle Optimality kommt die Idee negative Kreise zu suchen und sie durch Flusserhöhungen bzw. Potentialänderungen sukzessive zu eliminieren. Goldberg und Tarjan suchen in ihrem Algorithmus immer spezielle Kreise, sogenannte Mean Min Cycles, für das Cycle Cancelling. Eine Weiterentwicklung dieser Idee für konvexe Kosten ist der Cancel and Tighten Algorithmus von Karzanov und McCormick, der im Anhang beschrieben wird. Dieser besitzt eine Laufzeit von $O(m \log(n) \log(nU))$. Der Autor bespricht im Anhang auch einige Ideen zur möglichen Adaption des Algorithmus für das Convex BMCF Problem.

Kapitel 12

Und noch einmal Physik

Sicherlich ist dem Leser bereits aufgefallen, dass in unserem Anfangsbeispiel etwas gemogelt wurde. Wir hatten die Flüsse ja als $x_{ij} = n_i - n_j$ definiert. Dies impliziert aber $x_{ij} = -x_{ji}$! Dies ist nun das genaue Gegenteil eines balancierten Flusses.
Wir haben aber symmetrische Kosten auf den Kanten und damit auch ein schiefsymmetrisches Netzwerk. Die Verbindung dieser beiden Tatbestände wollen wir mit dem Konzept der antibalancierten Flüsse herstellen.

12.1 Anti-balancierte Flüsse

Nach den phänomenologischen Beobachtungen in der Einleitung wollen wir dem „Gegenteil eines balancierten Flusses“ zunächst einen Namen geben:

Definition 12.1 (anti-balancierter Fluss)**.** *Ein anti-balancierter Fluss x ist ein Fluss auf den Kanten eines schiefsymmetrischen Netzwerks N, der folgende Anforderungen erfüllt:*

- $x_{ij'} = -x_{ji'} \; \forall$ *Kanten* $(ij') \in A(N)$
- $x_{ij'}$ *ist ganzzahlig für alle Kanten* $(ij') \in A(N)$
- $x_{ii'} = 0$ *für alle Schleifen* $(ii') \in A(N)$

Der Fluss auf dem schiefsymmetrischen Netzwerk, das wir zuvor für das SOS-Modell beschrieben hatten, ist also anti-balanciert.
Damit ergibt sich auch sofort die Anti-Balanced Flow Decomposition. Hierbei sei zuvor noch darauf hingewiesen, dass wir bei anti-balancierten Flüssen natürlich keine Einschränkung der erweiternden Pfade auf gültige Pfade haben.

Satz 12.2 (Anti-Balanced Flow Decomposition)**.** *Seien x und y zwei unterschiedliche anti-balancierte Zirkulationen auf dem schiefsymmetrischen Netzwerk N, dann gibt es Kreise $p_1, \cdots, p_k$ im Restnetzwerk $N(x)$, so dass:*

$$y - x = \sum_{i=1}^{k} (x_{p_i} - x_{p_i'}) \tag{12.1}$$

Beweis. Wir setzen $\delta := y - x$ und betrachten, wie im Beweis zur Balanced Flow Decomposition $Supp(\delta)$, A_δ^+ und A_δ^-. Setzen wir $\delta(ij) := -\delta(\overline{(ij)}) \; \forall \; (ij) \in A_\delta^-$, so ist $N(x)[Supp(\delta)]$ auch hier ein schiefsymmetrisches Netzwerk, nur mit antibalanciertem Fluss.
Sei nun, ganz analog zum Beweis der Balanced Flow Decomposition, p ein Spaziergang auf $N(x)[Supp(\delta)]$, so dass p jede Kante (ij) aus dem Träger von δ maximal $\delta(ij)$ mal häufiger als p' überquert. Wir nehmen wieder an, p sei ein maximaler Spaziergang mit dieser Eigenschaft und wir zeigen wieder, dass p ein geschlossener Spaziergang ist.
Für jeden Knoten $i \in V(N)$ haben wir wieder die Kontinuitätsgleichung auf jedem Knoten:

$$\sum_{(ij) \in Supp(\delta)} \delta(ij) = \sum_{(ij) \in Supp(\delta)} \delta(ij) \tag{12.2}$$

Sei nun i der Anfangsknoten und j der Endknoten von p. Für $i \neq j$ gibt es mindestens eine Kante $(ij) \in Supp(\delta)$, die noch nicht $\delta(ij)$ mal häufiger von p als von p' überquert wurde und damit auch eine Kante $(j'i')$, die noch nicht $\delta(ij)$ mal häufiger von p' als von p überquert wurde. Wir können also den Pfad p genau um diese Kante erweitern.
p lässt sich als geschlossener Spaziergang in endlich viele Kreise $(p_1, \cdots, p_l)$ zerlegen. Mit analoger Argumentation zur Balanced Flow Decomposition erhalten wir damit den Beweis. □

Natürlich macht mit anti-balancierten Flüssen Max Bal Flow keinen Sinn. Wir können aber sehr wohl einen *betragsmäßig maximalen Fluss* definieren mit der Flussrate $aval(x) := \sum_{(ij) \in A(N)} |x_{ij}|$. Damit erhalten wir dann ganz analog zur Balanced Augmentation:

Korollar 12.3 (Anti-Balanced Augmenting Path Theorem)**.** *Sei x ein anti-balancierter Fluss auf dem schiefsymmetrischen Netzwerk N, dann ist x ein betragsmäßig maximaler antibalancierter Fluss, wenn es keinen erweiternden Pfad in $N(x)$ gibt.*

Wir können damit den Balanced SSP Algorithmus auf antibalancierte Flüsse übertragen. Die Berechnung des kürzesten Pfades kann dabei mit den angesprochenen schnelleren Verfahren für kürzeste Pfade (Dijkstra-Algorithmus mit Fibonacci Heaps) auf „normalen“ Netzwerken geschehen. Wir müssen nun nur den Fluss auf p erhöhen und auf p' verringern. Der Algorithmus hat nun folgende Form:

```
begin
x := 0, π := 0
while(shortest augmenting st-path exists)

    begin
    berechne kürzesten erweiternden st-Pfad p
    erhöhe den Fluss längs p um 1 Einheit und verringere den Fluss längs
    p' um 1 Einheit
    aktualisiere die reduzierten Kosten cc^π_ij längs p und p'
    aktualisiere N(x)
    end;

end;
```

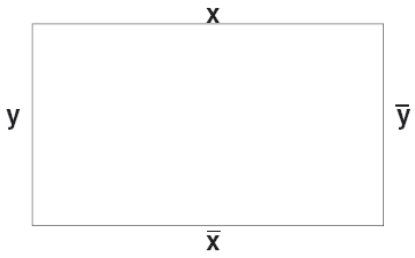

Abbildung 12.1: Illustration der Randbedingungen

Uns bleibt nur noch das Problem der Randbedingungen. Da man in der Regel auf kleinen Gittern arbeitet bieten sich sogenannte periodische Randbedingungen an. Hierbei werden an den „Rändern“ des Gitters jeweils „auf beiden Seiten gleiche“ Randbedingungen angesetzt. Wir lassen also per Zufallszahlengenerator die x_{ij}^r der Randknoten (Flüsse der Randknoten gekennzeichnet mit r) für die in der folgenden Grafik (Abb. 12.1) dargestellten Linien x und y zu nicht mehr im Gitter befindlichen Knoten berechnen.
$\bar{y}$ und $\bar{x}$ erhalten dann die gleichen Randbedingungen wie x und y.
Es gilt $e(i) = 0$ für Knoten im schiefsymmetrischen Netzwerk. Folglich sind die x_{ij}^r Überschüsse bzw. Defizite, die wir auf die Kanten zu s bzw. zu t legen und wir setzen folglich:

$$cap(si) := \begin{cases} x_{ij}^r & \text{für } i \text{ Randknoten} \\ 0 & \text{sonst} \end{cases} \tag{12.3}$$

$$cap(i't) := \begin{cases} x_{ij}^r & \text{für } i \text{ Randknoten} \\ 0 & \text{sonst} \end{cases} \tag{12.4}$$

Damit haben wir die Beschreibung unseres Algorithmus komplettiert.

12.2 Physikalisches Resultat

Sicherlich ist dem Leser bei unserer Beschreibung längst klar geworden, dass wir schiefsymmetrische Netzwerke hier nicht notwendigerweise einführen müssen. Blasum und Rieger haben eine andere Übertragung auf ein Netzwerkproblem beschrieben [15]. Wir wollen hier ihre Resultate [58] kurz vorstellen.
Um die superraue Phase zu identifizieren berechnet man die Korrelationsfunktion K der Höhen h_i. Die zugehörigen Orte auf dem Gitter seien mit $\vec{r_i}$ bezeichnet. In der Arbeit von Toner und diVincenzo [66] wird (und das ist alles andere als trivial) gezeigt, dass

$$K(\vec{r_i}, \vec{r_j}) \propto \log(|\vec{r_i} - \vec{r_j}|) \text{ in der rauen Phase} \tag{12.5}$$

$$\propto \log^2(|\vec{r_i} - \vec{r_j}|) \text{ in der superrauen Phase} \tag{12.6}$$

Da wir hier lediglich auf einem kleinen Gitter arbeiten, müssen wir die logarithmische Abhängigkeit noch durch eine Abhängigkeit von einer komplizierteren Funktion $P_L(|\vec{r_i} - \vec{r_j}|) = P_L(r)$ ersetzen [58, S. 2].

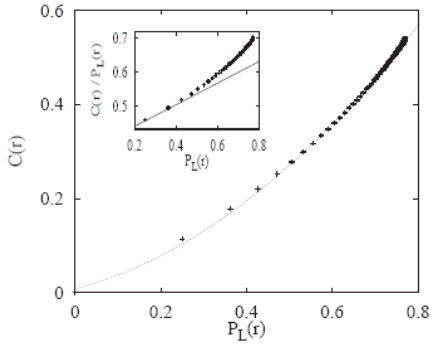

Abbildung 12.2: Korrelationsfunktion der Höhen aufgetragen gegen $P_L(r)$

Bei 2000 Untersuchungen mit verschiedenen Randbedingungen erhielten Rieger und Blasum folgendes Ergebnis für die Korrelationsfunktion (Abb. 12.2).

Man sieht deutlich, dass die Korrelationsfunktion sich wie eine quadratische Funktion verhält. Dies war die erste Untersuchung der superrauen Phase bei $T = 0$. Das Resultat ist dabei durchaus bemerkenswert, da wir den Hamiltonian aus der Renormierungsgruppentheorie direkt für die Simulation verwendet haben. Es zeigt, dass sie superraue Phase bis zu $T = 0K$ Bestand hat und kein weiterer Phasenübergang auftritt.
Der raue Untergrund könnte also tatsächlich der Grund für die Bilder im Sand sein.
Rieger hat mit diesem Modell noch weitere Untersuchungen zu Anregungszuständen der superrauen Phase gemacht [55]. Deren Interpretation ist aber, aufgrund der Tatsache, dass $T = 0K$ in diesem Modell absolut gilt, schwieriger.

12.3 Warum schiefsymmetrische Netzwerke?

Angesichts dieser Resultate stellt sich die Frage nach der Sinnhaftigkeit unseres Ansatzes mit schiefsymmetrischen Netzwerken.
Der Autor glaubt dabei, dass der Ansatz nicht nur Spielerei ist, da die Übersetzung auf ein Netzwerkproblem nach Blasum und Rieger die gleiche Anzahl von Kanten benötigt. Gleichzeitig muss aber bei einem Update der Kanten überprüft werden, ob $x_{ij} > 0$ oder $x_{ij} < 0$ gilt, da bei Blasum und Rieger jeweils nur positive Flüsse betrachtet werden, um sich des anti-balancierten Flusses zu entledigen. In unserer Umsetzung ist dies nicht nötig und man kann die negative Erhöhung des Flusses auf p' völlig implizit machen. Das Update müsste also in unserer Version schneller sein.
Ferner erscheint dem Autor die Umsetzung mit schiefsymmetrischen Netzwerken natürlicher, da die Symmetrie der Wechselwirkung explizit eingeht.
Tests der Geschwindigkeit dieses Ansatzes sind also durchaus interessant. Insbesondere könnte dies relevant werden, wenn man das Modell auf größeren Gittern testen würde, um die angesprochene $\log^2(r)$-Abhängigkeit direkt zu sehen.
Auch könnte man so testen, in wie weit die Terme höherer Ordnung in Simulationen berücksichtigt werden müssen, die im Sinne der Renormierungsgruppentheorie irrelevant sind.

Kapitel 13

Das Minkonvex Problem

Wir hatten am Anfang angesprochen, dass wir unser einleitendes Problem „optimales Rudern“ als sogenanntes Minsquare-Problem notieren können. Wir wollen in diesem Kapitel zunächst genauer definieren, was wir unter einem solchen Problem verstehen und im Anschluss die Lösung mittels BMCF Algorithmen mit konvexen Kosten darstellen.

13.1 Minsquare, Minkonvex und optimales Rudern

Wir hatten bereits bei der Einführung von Matching-Problemen Knotengrade $deg(i)$ für Knoten $i \in V$ in einem Graphen eingeführt.
Beim Minsquare-Problem betrachten wir einen einfachen Graphen $G = (V, E)$ und suchen auf diesem einen Teilgraphen $M := (V, F)$ konstanter Kardinalität $|F| = k,\ k \in \mathbb{N}$ [8, S. 6]. In unserem Ruderproblem entspricht $k = 8$ der Anzahl der möglichen Fahrten. Es soll nun ein Teilgraph F bestimmt werden, bei welchem die Anzahl der Kantenpaare (also die Anzahl von Matchingkanten mit einem gemeinsamen Knoten) minimal ist (möglichst keine Schüler, die mehrmals rudern). Die Anzahl der Kantenpaare ist dabei gegeben durch:

$$\begin{aligned} \text{Kantenpaare}(M) &:= \sum_{i \in V} \binom{deg_M(i)}{2} && (13.1)\\ &= \frac{1}{2}\sum_{i \in V}(deg_M(i))^2 - \frac{1}{2}\sum_{i \in V} deg_M(i) && (13.2)\\ &= \frac{1}{2}\sum_{i \in V}(deg_M(i))^2 - k && (13.3) \end{aligned}$$

Damit ist das Ruderproblem ein sogenanntes *Minsquare-Problem*:

$$Minsquare(G, K) := \min\left\{\sum_{i \in V}(deg_M(i))^2 \mid M = (V, F),\ \sum_{i \in V} deg_M(i) = 2k\right\} \quad (13.4)$$

Den konstanten Faktor k können wir bei der Minimierungsbedingung natürlich fortfallen lassen.

Allgemeiner können wir natürlich die Funktion $\sum_{i \in V}(deg_M(i))^2$ durch eine beliebige separable, konvexe Funktion $C(deg_M) = \sum_{i \in V} C_i(deg_M(i))$ ersetzen [8, S. 8].
Damit erhalten wir das allgemeinere sogenannte *Minkonvex-Problem*:

$$Minsquare(G, K) := \min\{C(deg_M) \mid M = (V, F), \sum_{i \in V} deg_M(i) = 2k\} \tag{13.5}$$

13.2 Lösung des Minkonvex-Problems mit BMCF-Algorithmen

Wir hatten bereits angesprochen, dass sich das Minsquareproblem mit Methoden auf schiefsymmetrischen Netzwerken lösen lässt. Wir wollen hier gleich den allgemeinen Fall des Minkonvex-Problems behandeln.
Die Lösung von Faktor-Problemen hatten wir bereits angesprochen. Die Konstruktion des schiefsymmetrischen Netzwerks zur Lösung des Minkonvex-Problems ist nun ähnlich. Wir konstruieren zu dem Graphen, auf dem wir das Minkonvex-Problem lösen wollen den schiefsymmetrischen Graphen. Die Knotengrade im Graphen entsprechend dann per Konstruktion gerade den Flüssen auf den (si)-Kanten im schiefsymmetrischen Graphen. Wir setzen also die Kosten auf den (si)-Kanten (und damit auch auf den $(i't)$-Kanten) auf:

$$C_{si}(x_{si}) = C_i(deg_M(i)) \; \forall i \in V(N) \tag{13.6}$$
$$C_{i't}(x_{i't}) = C_i(deg_M(i)) \; \forall i \in V(N) \tag{13.7}$$

Wie im Faktorproblem benötigen wir jede Paarung nur einmal und setzen folglich die Kapazität auf den (ij')-Kanten auf 1 für alle $i, j \in V(N)$. Die Kosten auf den (ij')-Kanten können wir auf 0 setzen, da wir ja nur Kosten haben, die vom Knotengrad abhängen. Wir müssen nun noch die Kardinalität des Matchings auf k begrenzen. Da jede gematchte Kante im Graphen durch 2 Kanten repräsentiert wird, muss der Gesamtfluss im Netzwerk auf $2k$ begrenzt werden. Dies können wir einfach sicherstellen, indem wir $e(s) = 2k$ festlegen. Offensichtlich entspricht dann die Lösung des entstehenden BMCF Problems einer Lösung des Minkonvex-Problems. Die Lösung des Minkonvex-Problems erhalten wir dabei über die zuvor definierte Bijektion zwischen Matchings und balancierten Flüssen.
Mit dem Balanced SSP Algorithmus erfordert die Lösung des Minkonvex-Problems so $O(km \log(n))$. Dieses Limit ist sehr gut, wenn man bedenkt, dass der Balanced SSP Algorithmus sehr einfach ist.

13.3 Vergleich mit anderen Algorithmen

Für das Minsquare- und das Minkonvex-Problem sind erst seit kurzem polynomielle Lösungen bekannt. Nicolas Apollonio und András Sebö konnten in ihrer Arbeit [5] zeigen, dass das Minsquare-Problem in $O(n^{5,5})$ lösbar ist. In einer weiteren Arbeit [6] gaben sie auch einen polynomiellen Algorithmus zur Lösung des allgemeinen Minkonvex-Problems an. Sie reduzierten dabei das Problem auf die Bestimmung eines allgemeinen f-Faktors.

Ihre Reduktion ist dabei aber sehr kompliziert [8, S. 4]. Berger gibt in ihrer Arbeit [8] eine einfachere Reduktion auf ein f-Faktor Problem an. Die Laufzeit beträgt in diesem Fall $O(m^2 \log(n))$.
Damit bietet unser Algorithmus eine Verbesserung der Laufzeit um einen Faktor $\frac{m}{k}$ für den Fall, dass die Steigung der Kostenfunktion immer positiv ist. Hierbei gilt es allerdings auch zu bemerken, dass der Balanced SSP Algorithmus, wie bereits gesagt, sehr einfach ist. Dies zeigt wie effizient und einfach Matching Probleme auf schiefsymmetrischen Netzwerken gelöst werden können.
Der Balanced Capacity Scaling Algorithmus und der Balanced Out-of-Kilter Algorithmus geben keine besseren Komplexitätslimits. Sie erlauben aber noch kompliziertere Probleme in ähnlicher Zeit zu lösen. So könnten wir die Knotengrade auf den si-Kanten noch durch Kapazitäten begrenzen und Kosten auf den ij'-Kanten einführen, ohne die Komplexität signifikant zu verschlechtern.

Kapitel 14

Zusammenfassung und Ausblick

Wir haben in dieser Arbeit zunächst ein Problem aus der Physik der kondensierten Materie und ein Minsquare-Problem betrachtet. Wir haben daraufhin schiefsymmetrische Netzwerke eingeführt und gezeigt, wie für diese maximale balancierte Flüsse zu bestimmen sind. Als Vorbereitung zur Bestimmung kostenminimaler Flüsse haben wir dann das Balanced Min Cost Flow Problem zunächst präzisiert und dann Optimalitätskriterien für dieses Problem betrachtet.
Wir haben im Anschluss mehrere Algorithmen zur Lösung des BMCF-Problems mit konvexen Kosten betrachtet. Abschließend haben wir gezeigt, dass sich sowohl das Minsquare-Problem als auch das Problem aus der Physik der kondensierten Materie mit BMCF-Algorithmen für konvexe Kosten effizient lösen lässt.
Wir wollen die verschiedenen Probleme und Komplexitätslimits hier noch einmal in Tabellenform zusammenfassen. Neue Komplexitätslimits bzw. neue Algorithmen sind dabei mit (neu) gekennezeichnet.
BMCF-Problem mit positiven Kosten: Berechnung eines kostenminimalen Flusses auf einem schiefsymmetrischen Netzwerk mit linearen nichtnegativen Kosten (Tab. 14.1).

Algorithmus	Komplexität
Primal-Dual	$O(MF \cdot m \log(n))$ (neu)
Enhanced Primal-Dual	$O(m \log(n) \log(nU) + nm \log(n))$
Balanced SSP (neu)	$O(MF \cdot m \log(n))$
Balanced Capacity Scaling (neu)	$O(m^2 \log(n) \cdot \log_2 U)$

Tabelle 14.1: Komplexitäten der Algorithmen zur Lösung des BMCF-Problems

Convex BMCF-Problem mit steigender Kostenfunktion: Berechne einen kostenminimalen Fluss auf einem schiefsymmetrischen Netzwerk, wobei die Kostenfunktion auf den Kanten konvex sei und nichtnegative Steigung habe (Tab. 14.2).

Convex BMCF-Problem: Berechne einen kostenminimalen Fluss auf einem schiefsymmetrischen Netzwerk, wobei die Kostenfunktion auf den Kanten konvex sei (Tab. 14.3).
Insbesondere die neue Komplexität für das Minkonvex-Problem scheint dabei sehr interessant zu sein. Man kann sich viele praktische Minkonvex-Probleme überlegen [8, S. 49-50].

Algorithmus	Komplexität
Primal-Dual	$O(MF \cdot m \log(n))$ (neu)
Balanced SSP (neu)	$O(MF \cdot m \log(n))$
Balanced Capacity Scaling (neu)	$O(m^2 \log(n) \cdot \log_2 U)$
Balanced Out-of-Kilter (neu)	$O(m^2 \log_2 U)$
Enhanced Balanced Capacity Scaling (neu)	$O(m^2 \log_2 U + m^2 \log(n))$

Tabelle 14.2: Komplexitäten der Algorithmen zur Lösung des Convex BMCF-Problems

Algorithmus	Komplexität
Balanced Out-of-Kilter (neu)	$O(m^2 \log_2 U + m^2 n)$

Tabelle 14.3: Komplexitäten der Algorithmen zur Lösung des Convex BMCF-Problems

Hierbei wäre allerdings interessant, auch einmal zu untersuchen, wo und in welcher Form solche Probleme tatsächlich in praxi auftreten, um auch praktische Anwendungsmöglichkeiten zu testen. Daneben wäre auch zu überprüfen, inwiefern die Beschränkung auf konvexe Kostenfunktionen mit positiver Steigung tatsächlich eine Beschränkung ist. Vermutlich ist die Annahme der positiven Steigung für viele praktische Beispiele erfüllt.
In diesem Zusammenhang wäre auch eine Version der Cancel and Tighten Methode für schiefsymmetrische Netzwerke interessant. Mit dieser könnte man eventuell eine noch bessere Komplexität für das Convex BMCF Problem und das Minkonvex Problem finden.
In dieser Arbeit wurden die meisten Ideen zur Lösung von Min Cost Flow Problem mit konvexen Kosten auf schiefsymmetrische Netzwerke übertragen. Keiner unserer Algorithmen ist allerdings stark polynomiell. Dem Autor ist auch kein MCF-Algorithmus mit stark-polynomieller Laufzeit für konvexe Kosten bekannt, der eine ganzzahlige Lösung berechnen würde. Karzanov und McCormick haben in ihrer Arbeit [38] allerdings für ähnliche Probleme stark polynomielle Laufzeiten erreicht. Die Suche nach stark polynomiellen Algorithmen für das MCF-Problem erscheint also nicht aussichtslos, wohl aber aufgrund der vielen praktischen Anwendungsmöglichkeiten nicht nur im Rahmen dieser Arbeit sehr interessant.
Daneben möchte der Autor das Problem der Komplexität des SVP-Algorithmus von Goldberg und Karzanov ansprechen. Wir haben in dieser Arbeit den Algorithmus nur für positive Kosten formuliert. Jungnickel und Fremuth-Paeger weisen, wie wir erwähnten, im allgemeinen Fall auf Probleme mit der Expansion von Fragmenten hin [27, S. 40]. Goldberg und Karzanov geben allerdings in ihrer Arbeit durchaus an, wie sich die Expansion von Fragmenten effizient durchführen lässt [30, S. 368-369]. Dies tun sie allerdings nur im Rahmen der Bestimmung von Pfaden, wo man die expandierten Fragmente danach nicht wiederverwenden muss. Der Autor gibt unumwunden zu, dass das Problem eher technisch ist. Gleichzeitig erhält man mit einer Lösung des Problems sofort eine neue Komplexitätsgrenze für das f-Faktor Problem [8, S. 20] und man könnte auch den Balanced Capacity Scaling Algorithmus auf den allgemeinen Fall erweitern. In diesem Sinne erscheint die Lösung des Problems durchaus interessant.
In diesem Zusammenhang möchte der Autor zudem auf das Problem der Komplexität des Max Bal Flow Algorithmus von Anstee hinweisen. Fremuth-Paeger gibt zwar eine Kom-

plexität von $O(mn)$ für das Problem an, formuliert den Algorithmus aber nicht aus. Eine genauere Analyse dieses Komplexitätslimits erscheint daher sinnvoll.
Abschließend möchte der Autor noch kurz auf die Fussnote zum „Satz von Berge“ hinweisen. Der Autor hält die Tatsache, dass 3 verschiedene Urheber für den Satz genannt werden, für äußerst unbefriedigend. Es wäre also durchaus sinnvoll und vielleicht sogar interessant die 3 Arbeiten zu vergleichen und den korrekten Urheber dieses, für die Matching-Theorie so wichtigen, Satzes zu ermitteln.

Anhang A

Balanced Relaxation Algorithmus

A.1 Einleitung

Bisher haben wir verschiedene Scaling Algorithmen und den primal-dualen Algorithmus betrachtet. Alle liefern uns, egal ob wir auf dem Polytop $\mathfrak{F}(N)$ oder $\mathfrak{P}(N)$ arbeiten, die ganzzahlige balancierte Lösung. Die Ganzzahligkeit ist dabei von großer Bedeutung, da die Lösung sonst offensichtlich keinem Matching entspricht. Insbesondere haben wir in unserem Beispiel in Kapitel 5 gesehen, dass uns die Ganzzahligkeit nicht „geschenkt“ wird und wir noch nicht einmal erwarten können, dass wir eine halbzahlige optimale Lösung erhalten werden.
Lassen wir also die Ganzzahligkeitsbedingung fallen, so bestimmen wir lediglich ein nicht-ganzzahliges Matching (Kanten werden anteilig zueinander gematcht mit rationalen Koeffizienten), wie wir in unserem Beispiel ebenfalls gesehen haben.
Man könnte vermuten, dass ein fraktionaler BMCF, der zu einem nicht-ganzzahligen Matching korrespondiert, einfacher zu bestimmen ist. Leider ist dies, mit den dem Autor bekannten Algorithmen, nicht der Fall. Wir werden hier keine Komplexitätsanalyse anstreben. Der Relaxation Algorithmus ist allerdings nur pseudopolynomiell und hat eine schlechte Worst-Case Laufzeit [11, S. 108]. Dem Autor sind auch keine Anwendungen bekannt, die die Berechnung eines fraktionalen Matchings erfordern würden. Der Algorithmus soll an dieser Stelle trotzdem dargestellt werden, um eine weitere Klasse von Ansätzen für Min Cost Flow Algorithmen mit konvexen Kostenfunktionen zu präsentieren [29, p. 358]. Der Autor sieht den vorgestellten Algorithmus eher als Denkanstoß für Relaxation Algorithmen, die ganzzahlige Lösungen berechnen.
Der Ansatz ist dabei in so fern interessant, als dass er aus der nichtlinearen Optimierung kommt und somit direkt auf konvexen Kosten arbeitet.

A.2 Formalismus

Wir betrachten hier einen Algorithmus, der sich stark an dem in [13] vorgestellten Algorithmus orientiert und behandeln hier lediglich die Spezialisierung und Symmetrisierung des dortig vorgestellten Algorithmus für Min Cost Flow für den Fall des Balanced Min Cost Flow. Beim Relaxation-Algorithmus betrachten wir in der Regel Zirkulationen. Hier-

zu führen wir im Ausgangsnetzwerk wieder die Kante (ts) ein und erlauben damit eine Zirkulation im gesamten Netzwerk. Ferner muss bei diesem Algorithmus zuvor das Defizit auf den verschiedenen Knoten im Netzwerk vorgegeben werden. Wir müssen also vor dem Start des Algorithmus einen Max Bal Flow Algorithmus verwenden, um den maximalen Fluss im Netzwerk zu berechnen und dann $val(x) = e(s) = -e(t)$ auf den Knoten vorzugeben. Wir betrachten dann das Problem mit Zirkulationen x:

$$\begin{aligned} \min C(x) &= \sum_{j \in A} C_{ij}(x_{ij}) && \text{(A.1)} \\ \text{u.d.N.: } Ex &= 0 && \text{(A.2)} \\ x &\quad \text{balanciert, nicht notwendig ganzzahlig} && \text{(A.3)} \end{aligned}$$

Hierbei bezeichnet E die sogenannte *Netzwerk-Inzidenzmatrix*. Für die Einträge $(E)_{ij}$ gilt:

$$(E)_{ij} = \begin{cases} 1 \text{ falls } (ij) \in A(N) \\ -1 \text{ falls } (ji) \in A(N) \\ 0 \text{ sonst} \end{cases} \qquad \text{(A.4)}$$

Die Struktur dieser Matrix ist offensichtlich sehr einfach und wir werden die Struktur dieser Matrix im übernächsten Kapitel noch genauer untersuchen. Hier ist $Ex = 0$ lediglich eine andere Schreibweise für $e(i) = 0 \; \forall i \in V(N)$. Dies ergibt sich aus der Darstellung $e(i) = \sum_{j \in A} e_{ij} x_{ij} \; \forall i \in V(N) \Rightarrow e = Ex$ mit dem Vektor der Defizite der Knoten e. Wir schreiben auch $x \in \mathfrak{K}$ mit $\mathfrak{K} = \{x | Ex = 0, \text{ x balanciert, aber nicht notwendig ganzzahlig}\}$.
Wir wollen an dieser Stelle noch darauf hinweisen, dass die die Netzwerkinzidenzmatrix natürlich ebenfalls die Schiefsymmetrie des Netzwerks widerspiegelt: $(E)_{ij'} = (E)_{ji'}$
Für den Algorithmus gibt es nun einige Annahmen an das Problem, die wir hier zusammenfassen wollen:

(i) Die Funktionen $C_{ij}(x_{ij}) : X \to \mathbb{R}$ sind konvex und unterhalbstetig auf dem Intervall X und es gibt mindestens eine gültige Lösung.

(ii) Die *konjugierte*, konvexe Funktion für jedes $C_{ij}(x_{ij})$ ist definiert durch:

$$G_{ij}(t_{ij}) = \sup_{x_{ij}} \{t_{ij} x_{ij} - C_{ij}(x_{ij})\} \qquad \text{(A.5)}$$

Ferner definieren wir noch $G(t) = \sum_{(ij) \in A(N)} G_{ij}(t_{ij})$ mit t dem Vektor der t_{ij}. Es sei hierbei auf die Ähnlichkeit zur Legendre-Transformation von Funktionen hingewiesen. Wir nehmen ferner an G_{ij} sei reellwertig: $-\infty < G_{ij}(t_{ij}) < \infty \; \forall t_{ij} \in \mathbb{R}$.

Damit gilt auch sofort $C_{ij}(x_{ij}) > -\infty \; \forall x_{ij} \in X$. Wir betrachten nun das Intervall, das durch die reellen Punkte von C_{ij} aufgespannt wird und bezeichnen die Grenzen mit:

$$\begin{aligned} cap(ij) &= \sup \{x_{ij} | C_{ij}(x_{ij}) < \infty\} && \text{(A.6)} \\ l(ij) &= \inf \{x_{ij} | C_{ij}(x_{ij}) < \infty\} && \text{(A.7)} \end{aligned}$$

Wir bezeichnen $cap(ij)$ und $l(ij)$ natürlich auch hier als die oberen und unteren Kapazitätsgrenzen von C_{ij}.

Betrachten wir nun ob G_{ij} auch wohldefiniert ist, in dem Sinne, dass ein x_{ij} aus der Menge der gültigen Lösungen $dom(\mathfrak{K}) = \{x|C(x) < \infty\}$ existiert, für welches das Supremum auch tatsächlich angenommen wird.
Gilt $-\infty < \sup_{x_{ij}} \{t_{ij}x_{ij} - C(x_{ij})\} < \infty$ mit $x_{ij} \in [l_{ij}, c_{ij}]$, so muss für $t_{ij} \neq 0$ das Supremum für x_{ij} in einem $|x_{ij}| < \infty$ erreicht werden, da sonst $G_{ij}(t_{ij}) < \infty$ verletzt wäre. Damit gilt aber auch $x_{ij} \in dom(\mathfrak{K})$.
Wir betrachten nun das Verhalten von $C_{ij}(x_{ij})$ für $|x_{ij}| \to \infty$. Dazu müssen wir natürlich annehmen, dass die Kostenfunktion für solche x auch definiert ist. Für das asymptotische Verhalten nehmen wir zunächst an $C_{ij}(x_{ij}) \leq M$ mit einem $M \in \mathbb{R},\ M > 0$. Dann gilt für $G_{ij}(t_{ij})$:

$$\begin{aligned} G_{ij}(t_{ij}) &= \sup_{x_{ij}\ in\mathbb{R}} \{t_{ij}x_{ij} - C_{ij}(x_{ij})\} && \text{(A.8)} \\ &\leq \sup_{x_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - M\} && \text{(A.9)} \end{aligned}$$

Für $t_{ij} \neq 0$ gilt dann $G_{ij}(t_{ij}) \to \pm\infty$ je nachdem, ob t_{ij} positiv oder negativ ist. Dies steht allerdings im Widerspruch zur Annahme der Endlichkeit von G_{ij}. Da $C_{ij}(x_{ij})$ als konvex und unterhalbstetig angenommen wurde, folgt damit $\lim_{|x_{ij}|\to\infty} C_{ij}(x_{ij}) = \infty$.
Folglich muss $C(x)$ im Intervall $dom(C)$ als unterhalbstetige Funktion ihr Minimum annehmen und es gibt somit mindestens einen optimalen Fluss.
Wir hätten dies hier auch einfach vorraussetzen können, wollen hier aber aufzeigen, wie gering die Anforderungen an die Kostenfunktion sind.
Das so formulierte Minimierungsproblem wird im allgemeinen Fall auch als *Optimal Distribution Problem* bezeichnet. In unserem Fall handelt es sich also um das *Balanced Optimal Distribution Problem*.
Wir haben obig bereits die konjugierte, konvexe Funktion $G_{ij}(t_{ij})$ definiert. Um den Begriff der konjugierten Funktion noch etwas weiter zu motivieren und die Eigenschaften von G_{ij} zu betrachten benötigen wir den folgenden Satz aus [10, S. 427-428].

Satz A.1. *Sei $C_{ij} : X \to \mathbb{R}$ eine strikt konvexe Funktion über dem Intervall X und $G_{ij}(t_{ij}) = \sup_{x_{ij}\in X}\{t_{ij}x_{ij} - C_{ij}(x_{ij})\}$.*

(i) Es gilt:

$$\sup_{t_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - G_{ij}(t_{ij})\} = \begin{cases} C_{ij}(x_{ij}) & \text{falls } x_{ij} \in X \\ \infty & \text{sonst} \end{cases} \qquad \text{(A.10)}$$

Die folgenden Aussagen sind dabei für $x_{ij} \in X$ und $t_{ij} \in \mathbb{R}$ äquivalent:

- *(a) $x_{ij}t_{ij} = C_{ij}(x_{ij}) + G_{ij}(t_{ij})$*
- *(b) Für x_{ij} wird das Supremum in der Definition von G_{ij} angenommen.*
- *(c) Für t_{ij} wird das Supremum in der obigen Gleichung angenommen.*

(ii) Es gilt für den Fluss $x_{ij}(t_{ij})$, für den das Supremum in der Definition von G_{ij} angenommen wird und für ein differenzierbares G_{ij}:

$$\nabla G_{ij}(t_{ij}) = x_{ij}(t_{ij}) \qquad \text{(A.11)}$$

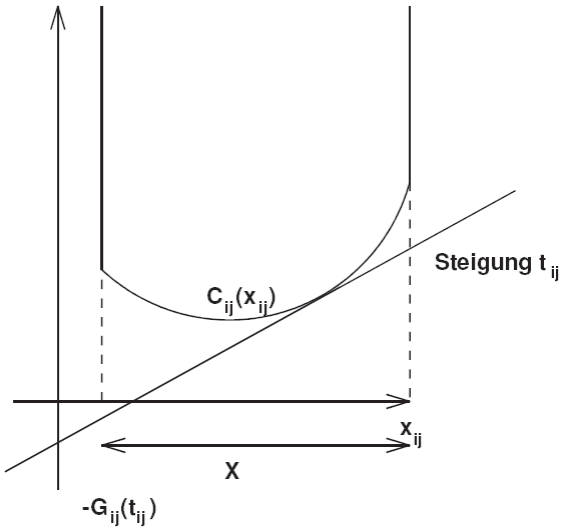

Abbildung A.1: 1. Teil des geometrischen Beweises zu Aussage (i)

Beweis. Wir wollen an dieser Stelle keinen strengen mathematischen Beweis dieser Aussagen anstreben, sondern geben lediglich die geometrischen Argumente wie in dem Buch von Bertsekas [10, S. 428]. Eine exaktere Herleitung müsste auf die konvexe Analysis zurückgreifen. Wir wollen dies an dieser Stelle aber unterlassen und verweisen den Leser auf das Buch von Rockafellar [59].
Wir beweisen nun zunächst die Aussage aus (i). Hierzu betrachten wir folgende Grafik (A.1).
Für jedes t_{ij} erhalten wir $G_{ij}(t_{ij})$ geometrisch, indem man die Unterstützungsgerade mit Steigung t_{ij} zur konvexen Menge (dem *Epigraph* von C_{ij})

$$\{(x_{ij}, \gamma) | x_{ij} \in X;\ C_{ij}(x_{ij}) \leq \gamma\} \tag{A.12}$$

konstruieren. $-G_{ij}(t_{ij})$ ist dann gerade der y-Achsenabschnitt der Unterstützungsgeraden. Für ein differenzierbares C_{ij} ist dies unmittelbar einsichtig. Hierfür schreiben wir $G_{ij}(t_{ij}) = \inf_{x_{ij} \in X}\{C_{ij}(x_{ij}) - t_{ij}x_{ij}\}$. Nun können wir die Bedingung für das Minimum x^*_{ij} einer differenzierbaren Funktion verwenden:

$$\begin{aligned} 0 &= \frac{d}{dx_{ij}}(C_{ij}(x_{ij} - t_{ij}x_{ij}))|_{x_{ij}=x^*_{ij}} && \text{(A.13)} \\ &= C'_{ij}(x^*_{ij}) - t_{ij} \ \Rightarrow C'_{ij}(x^*_{ij}) = t_{ij} && \text{(A.14)} \\ &\Rightarrow G_{ij}(t_{ij}) = C'_{ij}(x^*_{ij})t_{ij} - C_{ij}(x^*_{ij}) && \text{(A.15)} \end{aligned}$$

Wir können nun die Gleichung für die Unterstützungsgerade $U(x_{ij})$ direkt angeben und den Nulldurchgang bestimmen:

$$\begin{aligned} U(x_{ij}) &= C'_{ij}(x^*_{ij})(x_{ij} - x^*_{ij}) + C_{ij}(x^*_{ij}) && \text{(A.16)} \\ \Rightarrow U(0) &= C_{ij}(x^*_{ij}) - C'_{ij}(x^*_{ij})x^*_{ij} = -G_{ij}(t_{ij}) && \text{(A.17)} \end{aligned}$$

Nach diesen Vorüberlegungen gehen wir an den tatsächlichen Beweis von (i). Hierzu betrachten wir folgende Grafik (A.2).

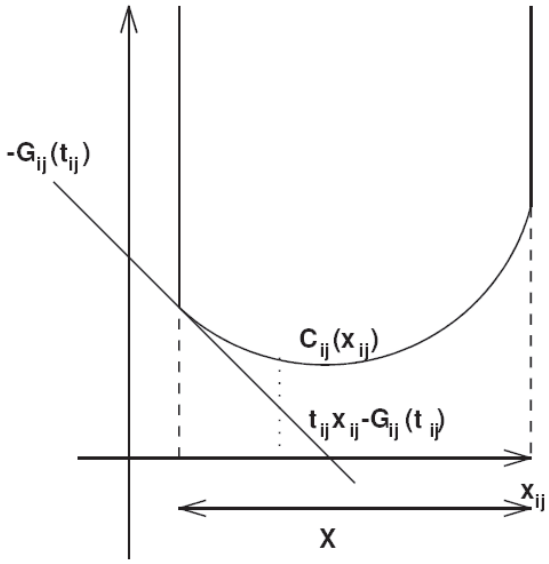

Abbildung A.2: 2. Teil des geometrischen Beweises zu Aussage (i)

Für ein gegebenes $x_{ij} \in X$ erhält man $t_{ij}x_{ij} - G_{ij}(t_{ij})$ als Schnittpunkt einer Geraden mit Steigung t_{ij}, die den Epigraph von C_{ij} stützt mit der Vertikalen durch $(x_{ij},\ C_{ij}(x_{ij}))$. Dies ergibt sich sofort aus der vorherigen Betrachtung, da $-G_{ij}(t_{ij})$ gerade der y-Achsenabstand der Untersützungsgeraden mit Steigung t_{ij} ist.
Betrachten wir nun das Supremum von $t_{ij}x_{ij} - G_{ij}(t_{ij})$ in Abhängigkeit von t_{ij}. Es gilt $t_{ij}x_{ij} - G_{ij}(t_{ij}) \leq C_{ij}(x_{ij})$, da $C_{ij}(x_{ij})$ konvex ist. Wir sehen aber in (A.1), dass die Gleichheit für korrekt gewähltes $t_{ij} \in \mathbb{R}$ erreicht wird:

$$\Rightarrow\ C_{ij}(x_{ij}) = \sup_{t_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - G_{ij}(t_{ij})\} \ \text{ für } \ x_{ij} \in X \tag{A.18}$$

Für $x \notin X$ zeigt die Konstruktion mit entsprechender Wahl für t_{ij}, dass $t_{ij}x_{ij} - G_{ij}(t_{ij})$ beliebig groß werden kann. Also:

$$\sup_{t_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - G_{ij}(t_{ij})\} = \infty \tag{A.19}$$

Mit unserer Konstruktion sehen wir ferner, dass die Äquivalenzaussagen (b) und (c) jeweils äquivalent sind zu (a), was damit auch den zweiten Teil von (i) zeigt.
Wir kommen damit zu Teil (ii). Da das Supremum für $G_{ij}(t_{ij})$ für jedes $t_{ij} \in \mathbb{R}$ für ein $x_{ij} \in X$ auch angenommen wird, ist G_{ij} eine reelle konvexe Funktion. Lassen wir $t_{ij} \in \mathbb{R}$ fix und betrachten die Gleichung aus der Äquivalenzaussage (a):

$$t_{ij}x_{ij} = C_{ij}(x_{ij}) + G_{ij}(t_{ij}) \tag{A.20}$$

Wir hatten G_{ij} als differenzierbar (bezüglich t_{ij}) angenommen. Differenzieren wir die obige Gleichung bezüglich t_{ij}, so ist $C_{ij}(x_{ij})$ nur eine Konstante und es gilt:

$$x_{ij} = \nabla G_{ij}(t_{ij}) \tag{A.21}$$

□

Wir wollen an dieser Stelle noch auf ein wichtiges Faktum für schiefsymmetrische Netzwerke hinweisen.
Wir können die Transformation für die konjugierte Funktion natürlich auch umkehren und C_{ij} als konjugierte Funktion [60, S. 331] von G_{ij} schreiben mit:

$$C_{ij}(x_{ij}) = \sup_{t_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - G_{ij}(t_{ij})\} \tag{A.22}$$

In einem schiefsymmetrischen Netzwerk soll nun gelten:

$$\begin{aligned} C_{ij}(x_{ij}) &= \sup_{t_{ij}\in\mathbb{R}} \{t_{ij}x_{ij} - G_{ij}(t_{ij})\} && \text{(A.23)}\\ &= C_{j'i'}(x_{j'i'}) = \sup_{t_{j'i'}\in\mathbb{R}} \{t_{j'i'}x_{j'i'} - G_{j'i'}(t_{j'i'})\} && \text{(A.24)}\\ &= \sup_{t_{j'i'}\in\mathbb{R}} \{x_{ij}t_{j'i'} - G_{ij}(t_{j'i'})\} && \text{(A.25)} \end{aligned}$$

Dies lässt sich sofort sicherstellen durch $t_{ij} = t_{j'i'}$, sodass wir die Schiefsymmetrie auch auf das duale Problem übertragen können. Dies ist dann das, bereits im primal-dualen Algorithmus angesprochene, symmetrische duale Problem.
Bevor wir auf die bereits angesprochene Dualität genauer eingehen, wollen wir zunächst das symmetrische duale Problem angeben [60, p. 345]:

$$\begin{aligned} \min G(t) &= \sum_{(ij)\in A(N)} G_{ij}(t_{ij}) && \text{(A.26)}\\ \text{u.d.N.:} \quad & t \in \mathfrak{K}^{\perp} && \text{(A.27)} \end{aligned}$$

Hierbei gilt $t \in \mathbb{R}^{|A(N)|}$, da wir in Annahme (ii) die Endlichkeit von den G_{ij} für alle t_{ij} vorrausgesetzt hatten. Man bezeichnet t_{ij} auch als die Spannung der Kante (ij). Der Raum $\mathfrak{K}^{\perp}$ ist dabei der Raum der zulässigen Lösungen. Wir werden ihn jetzt noch etwas genauer fassen.

Lemma A.2 (Spannungsunterraum)**.** *Sei $\mathfrak{K} = \{x|Ex = 0\}$, dann gilt für den Spannungsunterraum $\mathfrak{K}^{\perp} = \left\{t|t = E^T p\right\}$.*

Beweis. Betrachten wir die Gleichung $Ex = \begin{bmatrix} E_1^t x \\ \vdots \\ E_n^t x \end{bmatrix} = 0$. Dies gilt folglich genau dann, wenn x orthogonal zu den Zeilen von E ist. Sei nun x auch orthogonal zu Linearkombinationen aus den Zeilen von E, so gilt $t^T x = p^T Ex = 0$. □

Mit dieser Darstellung von t muss es folglich Zahlen $\pi(i), i \in V$ geben, so dass $t_{ij} = \pi(i) - \pi(j) = t_{j'i'} = \pi(j') - \pi(i')$. Dies sind dann wieder die Potentiale, die wir auch zuvor schon bei den reduzierten Kosten hatten. Wir arbeiten hierbei allerdings nicht mit den Kanten im primalen Problem, sondern direkt auf den Potentialen.
Wir können nun ein sogenanntes Potential untersuchen:

$$\Psi(v) = -\sum_{(ij)\in A} G_{ij}(v(ij)) \tag{A.28}$$

Nach den obigen Annahmen ist das Potential für alle $v(ij) \in \mathbb{R}$ endlich und reell. Nach den gemachten Annahmen über die G_{ij} besteht nun das sogenannte *Balanced Optimal Differential Problem* darin $\Psi(v)$ (mit v dem Vektor der $v(ij)$) zu maximieren. Man bezeichnet dann v auch als die optimale Lösung, wenn sie Ψ maximiert.
Der Begriff des Potentials deutet bereits an, dass es sich hierbei um eine deutliche Analogie zum Lagrange-Formalismus der theoretischen Physik handelt. Man kann diese Analogie noch etwas deutlicher machen, indem man $v(ij)$ als Gradienten einer Funktion $u(ij)$ schreibt. Diesen noch etwas verallgemeinerten Fall wollen wir hier aber nicht behandeln. Es sei ebenfalls darauf hingewiesen, dass der Begriff des Potentials die Analogie zur theoretischen Physik ausmacht und keinesfalls der Lagrange Formalismus selbst der theoretischen Physik zugeordnet werden soll.
Statt dessen wollen wir genauer auf die Dualität eingehen. Da wir C_{ij} als Supremum definiert hatten, gilt (lediglich mit der Ersetzung $v(ij) \leftrightarrow t_{ij}$):

$$C_{ij}(x_{ij}) \geq x_{ij}v(ij) - G_{ij}(v(ij)) \tag{A.29}$$

Ferner gilt, da die t_{ij} im dualen Problem aus $C^{\perp}$ gewählt sind für das Skalarprodukt: $\sum_{(ij)\in A(N)} x_{ij}v(ij) = 0$. Addieren wir nun die obige Ungleichung, so erhalten wir:

$$\sum_{(ij)\in A(N)} C_{ij}(x_{ij}) \geq -\sum_{(ij)\in A(N)} G_{ij}(v(ij)) \tag{A.30}$$

$$\Rightarrow C(x) \geq \Psi(v) \tag{A.31}$$

Dies ist die sogenannte schwache Dualität. Es gilt hierbei sogar nicht nur die schwache, sondern sogar die starke Dualität nach [60, p. 349].

Satz A.3 (Network Duality)**.** *Hat mindestens eines der beiden Probleme (Balanced Optimal Distribution Problem, Balanced Optimal Differential Problem) eine gültige Lösung, so*

$$\begin{bmatrix} \textit{inf im balanced optimal} \\ \textit{distribution problem} \end{bmatrix} = \begin{bmatrix} \textit{sup im balanced optimal} \\ \textit{differential problem} \end{bmatrix} \tag{A.32}$$

Formulieren wir nun das Potential-Problem noch etwas um. Zunächst haben wir

$$\sup\{\Psi(v)|v \in \mathbb{R}^n\} = \inf\{-\Psi(v)|v \in \mathbb{R}^n\} \tag{A.33}$$

Ferner haben wir die Existenz einer optimalen Lösung bereits gezeigt (unter den gemachten Annahmen) und erhalten damit als äquivalente Formulierung des Problems unser symmetrisches duales Problem:

$$\min \sum_{(ij)\in A} G_{ij}(t_{ij}) \tag{A.34}$$

$$t \in \mathfrak{K}^{\perp} \tag{A.35}$$

Mit den Potentialen können wir jetzt das duale Problem mit dem dualen Funktional $G(\pi) = \sum_{(ij)\in A} G(\pi(i) - \pi(j))$ (π ist der Vektor der Potentiale) auch schreiben als:

$$\min G(\pi) \tag{A.36}$$

$$\text{unter keinen Nebenbedingungen} \tag{A.37}$$

Interessant an dieser Formulierung des dualen Problems ist die Abwesenheit von Nebenbedingungen. Diese können wir nun mit einem Satz aus der konvexen Analysis direkt ausnutzen.

Satz A.4 (Differenzierbarkeit der konjugierten Funktion)**.** *Eine geschlossen eigentlich konvexe Funktion ist strikt konvex genau dann, wenn ihre konjugierte Funktion glatt ist.*

Beweis. siehe [59, p. 253] □

Korollar A.5. *Für eine strikt konvexe Funktion ist die konjugierte Funktion auch differenzierbar.*

Mit dem Korollar haben wir damit die Differenzierbarkeit der dualen Kostenfunktion. Damit haben wir auch die zuvor bereits für differenzierbare duale Kostenfunktionen bewiesene Gleichung $\nabla G_{ij}(t_{ij}) = x_{ij}$ zur Verfügung. Wir berechnen nun die partiellen Ableitungen nach den Potentialen:

$$\frac{\partial q(\pi)}{\partial \pi(k)} = \frac{\partial}{\partial \pi(i)} \left(\sum_{(ij)\in A(N)} G_{ij}(\pi(i) - \pi(j)) \right) \tag{A.38}$$

$$= \sum_{(ij)\in A(N)} \nabla G_{ij}(t_{ij}) \frac{\partial t_{ij}}{\partial \pi(k)} \tag{A.39}$$

$$\frac{\partial t_{ij}}{\partial \pi(k)} = \left\{ \begin{array}{c} 1 \text{ falls } i = k \\ -1 \text{ falls } j = k \\ 0 \text{ sonst} \end{array} \right\} = e_{ij} \tag{A.40}$$

$$\Rightarrow \frac{\partial q(\pi)}{\partial \pi(k)} = \sum_{(ij)\in A(N)} e_{ij} \nabla G_{ij}(t_{ij}) \tag{A.41}$$

Da also die dualen Kosten differenzierbar und die primalen Kosten strikt konvex sind, wird hierdurch ein Algorithmus wie beim Gauss-Seidel Verfahren (Einzelschrittverfahren) nahe gelegt.
In unserem schiefsymmetrischen Netzwerk werden wir wieder eine symmetrische Behandlung des Problems versuchen. Dabei greifen wir Paare von Knoten mit positivem und negativem Defizit heraus und erhöhen bzw. verringern das Potential $\pi(i)$ und $\pi(i')$ bis zu dem Punkt, in dem $\frac{\partial G(\pi)}{\partial \pi(i)} = 0$.
Die Herangehensweise ist dabei auch für Parallelrechnung interessant, da auf verschiedenen Rechnern getrennt Knoten-Paare berechnet werden können [12].
Die wichtige Eigenschaft strikt konvexer Kostenfunktionen ist nun, dass für $C(x)$ nur ein minimales x existiert. Ferner sei daran erinnert, dass durch die strikte Konvexität auch die Differenzierbarkeit der dualen Kostenfunktion erreicht wird.
Als letzte Folgerung aus der strikten Konvexität wollen wir an dieser Stelle das Äquivalent zur Complementary Slackness Optimality betrachten.
Seien dazu C_{ij}^- und C_{ij}^+ die rechts- und linksseitigen Ableitungen von C_{ij}, dann erfüllt $t_{ij} \in \mathbb{R}$ die Relation:

$$C_{ij}^-(x_{ij}) \leq t_{ij} \leq C_{ij}^+(x_{ij}) \tag{A.42}$$

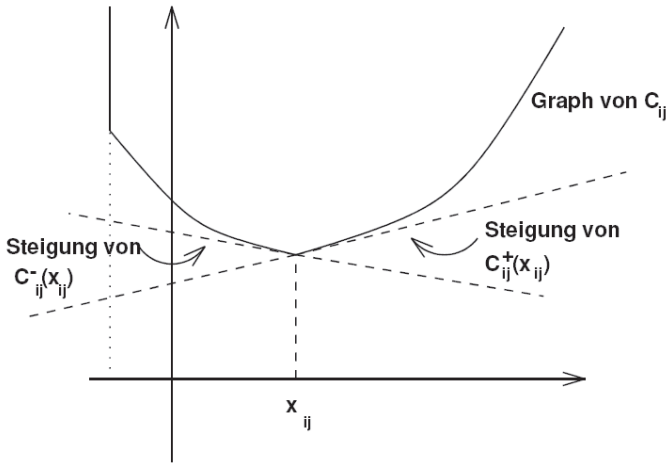

Abbildung A.3: Illustration der Complementary Slackness Bedingung

genau dann, wenn x_{ij} den Ausdruck $C_{ij}(\xi) - \xi t_{ij}$ maximiert. Dies kann nur für das Supremum für $t_{ij} \in \mathbb{R}$ der Fall sein. Dies wird auch in dem folgenden Bild (A.3) noch einmal erläutert. Für eine genauere mathematische Begründung sei hier auf [60, S. 332] verwiesen.
Abschließend definieren wir noch für $l_{ij} = c_{ij}$ die links- und rechtsseitigen Ableitungen mit $C_{ij}^{-}(l_{ij}) = -\infty, C_{ij}^{+}(x_{ij}) = \infty$.

A.3 Algorithmus

Mit den obigen Ergebnissen können wir nun beginnen den Relaxation-Algorithmus zu formulieren. Hierzu nutzen wir unser Resultat wonach $x_{ij} = \nabla G_{ij}(t_{ij})$. Hiermit können wir nun wieder das Defizit der einzelnen Knoten notieren:

$$e_i(\pi) = \sum_{(ij)\in A(N)} e_{ij} \nabla G_{ij}(t_{ij}) \tag{A.43}$$

Wir notieren mit $e(\pi)$ den Vektor der $e_i(\pi)$.

Lemma A.6. *Für das Defizit des Knotens k gilt $e_k(\pi) = \frac{\partial G(\pi)}{\partial \pi(k)}$. Ferner gilt aufgrund der Schiefsymmetrie $e_k(\pi) = -e_{k'}(\pi)$.*

Beweis. Wir leiten direkt ab und erhalten:

$$\frac{\partial G(\pi)}{\partial \pi(k)} = \sum_{(ij)\in A(N)} \nabla G_{ij}(t_{ij}) \frac{\partial t_{ij}}{\partial \pi(k)} \tag{A.44}$$

$$= \sum_{(kj)\in A(N)} x_{kj} e_{kj} = e_k(\pi) \tag{A.45}$$

Wir benutzen nun noch: $x_{kj} = \nabla G_{kj}(t_{kj}) = x_{j'k'} = \nabla G_{j'k'}(t_{j'k'})$. Und wir erhalten die zweite Aussage des Lemmas:

$$e_{k'}(\pi) = \sum_{(kj)\in A(N)} \nabla G_{kj}(t_{kj}) \frac{\partial t_{j'i'}}{\partial \pi(k')} \tag{A.46}$$

$$= \sum_{(kj)\in A(N)} \nabla G_{kj}(t_{kj})(-e_{kj}) = -e_k(\pi) \tag{A.47}$$

□

Mit dem obigen Lemma sehen wir also, dass wir den Algorithmus symmetrisch formulieren können.
Wir wollen nun den Algorithmus zunächst angeben und danach beweisen, dass dieser tatsächlich eine Lösung berechnet. Es sei $0 < \delta < 1$:

```
Symmetric Iteration for strictly convex arc costs
if e_i(π) = 0 ∀i ∈ V then STOP
else
    wähle irgendeinen Knoten i, setze β = e_i(π)

    if β = 0 tue nichts
    if β > 0 then verringere π(i) so, dass 0 ≤ e_i(π) ≤ δ · β
                  erhöhe π(i') so, dass 0 ≥ e_i'(π) ≥ −δ · β
    if β < 0 then erhöhe π(i)so, dass 0 ≥ e_i(π) ≥ δ · β
                  verringere π(i) so, dass 0 ≤ e_i'(π) ≤ −δ · β
```

Für die Wahl der Knoten in der *Relaxation* machen wir hier lediglich die Annahme, dass jeder Vertex in der Relaxation unendlich oft verwendet wird.
Wir wollen zunächst darauf hinweisen, dass mit dem obigen Lemma noch nicht vollständig geklärt wurde, ob eine symmetrische Behandlung im Algorithmus möglich ist, da der Fall auftreten kann, dass die Ableitungen nach den einzelnen Potentialen nicht unabhängig sind. Dies ist genau bei Schleifen der Form (kk') der Fall. Wir müssen hierbei beachten, dass Schleifen jeweils doppelt gezählt werden [23, p. 4] und erhalten damit das duale Funktional in der Form: $G(\pi) = \sum_{(ij)\in A(N),(ij)\neq(kk'),(k'k)} G_{ij}(\pi(i) - \pi(j)) + G_{kk'}(\pi(k) - \pi(k')) + G_{k'k}(\pi(k') - \pi(k))$. Wir können nun die Ableitung z.B. nach $\pi(k)$ direkt berechnen und erhalten:

$$\begin{aligned}
\frac{\partial G(\pi)}{\partial \pi(k)} &= \frac{\partial}{\partial \pi(k)} \sum_{(ij)\in A(N),(ij)\neq(kk'),(k'k)} (G_{ij}(\pi(i) - \pi(j)) + \\
&\quad G_{kk'}(\pi(k) - \pi(k')) + G_{k'k}(\pi(k') - \pi(k))) && \text{(A.48)} \\
&= \frac{\partial}{\partial \pi(k)} G_{kk'}(\pi(k) - \pi(k')) + \frac{\partial}{\partial \pi(k)} G_{k'k}(\pi(k') - \pi(k)) + \\
&\quad \frac{\partial}{\partial \pi(k)} \sum_{(ij)\in A(N),(ij)\neq(k'k),(kk')} G_{ij}(\pi(i) - \pi(j)) && \text{(A.49)} \\
&= \frac{\partial}{\partial \pi(k)} \sum_{(ij)\in A(N),(ij)\neq(k'k),(kk')} G_{ij}(\pi(i) - \pi(j)) && \text{(A.50)}
\end{aligned}$$

Wir können somit das Problem ganz einfach durch Streichen der Schleifen in $G(\pi)$ beheben:

$$G(\pi) = \sum_{(ij)\in A(N),(ij)\neq(k'k),(kk')} G_{ij}(\pi(i) - \pi(j)) \tag{A.51}$$

Um die Notation allerdings etwas zu vereinfachen wollen wir annehmen, der Graph habe keine Schleifen, was wir nach obiger Rechnung o.B.d.A. tun können.
Beginnen wir nun damit zu zeigen, dass die Relaxation im obigen Algorithmus wohldefiniert ist.

Lemma A.7. *In jedem Schritt des Algorithmus lässt sich $\pi(i)$ jeweils so erhöhen bzw. verringern, dass $e_i(\pi)$ in den angegebenen Grenzen ist.*

Beweis. Nehmen wir an $\beta > 0$ und es existiere kein $\Delta < 0$, so dass $e_i(\pi + \Delta \cdot \hat{e_i}) \leq \delta\beta$, wobei $\hat{e_i}$ den i-ten Einheitskoordinatenvektor bezeichne und i wieder aus der Menge der Knoten gewählt sei. Dann gilt:

$$\lim_{\Delta \to -\infty} e_i(\pi(i) + \Delta\hat{e_i}) = \lim_{\Delta \to -\infty} \sum_{(ij) \in A(N)} e_{ij} \nabla G_{ij}(t_{ij} + \Delta\hat{e_i}) \tag{A.52}$$

Wir können dies nun etwas umschreiben und die Potentiale einsetzen:

$$\lim_{\Delta \to -\infty} \sum_{(ij) \in A(N)} e_{ij} \nabla G_{ij}(e_{ij}(\pi(i) - \pi(j) + \Delta\hat{e_i})) \tag{A.53}$$

$$= \lim_{\Delta \to -\infty} \sum_{(ij) \in A(N)} e_{ij} \nabla G_{ij}(e_{ij}\pi(i) + \Delta\hat{e_i} - e_{ij}\pi(j)) \tag{A.54}$$

Damit entspricht dem Grenzübergang für $\Delta \to -\infty$ ein Grenzübergang für das Potential:

$$\text{für } e_{ij} > 0 \quad \Rightarrow \quad \pi(i) \to -\infty \tag{A.55}$$

$$\text{für } e_{ij} < 0 \quad \Rightarrow \quad \pi(i) \to \infty \tag{A.56}$$

Bei unendlich hohem Potential muss der Fluss der Kante (ij) maximal (also $cap(ij)$) sein und bei unendlich niedrigem Potential muss der Fluss der Kante (ij) minimal (also $l(ij)$) sein. Damit erhalten wir:

$$\sum_{e_{ij} > 0,\ (ij) \in A(N)} e_{ij} l(ij) + \sum_{e_{ij} < 0,\ (ij) \in A(N)} e_{ij} cap(ij) \geq \delta\beta > 0 \tag{A.57}$$

Dies führt nun aufgrund der Definitionen von $l(ij)$ und $cap(ij)$ als Flussunter- bzw. -obergrenzen zum Widerspruch. Ganz analog kann man dies für $\beta < 0$ zeigen.
Für die schiefsymmetrischen Kanten folgt die Aussage ebenfalls mit der obigen Argumentation, da für den Knoten i' gilt: $e_{ij} = e_{j'i'} = -e_{i'j'}$ und damit für die obige Summe:

$$\sum_{e_{i'j'} < 0,\ (ij) \in A(N)} e_{i'j'} l(i'j') + \sum_{e_{i'j'} > 0,\ (ij) \in A(N)} e_{i'j'} cap(i'j') \leq -\delta\beta < 0 \tag{A.58}$$

Und erneut bekommen wir den Widerspruch mit den Definitionen von $l(i'j')$ und $cap(i'j')$. □

Wir müssen nun zeigen, dass unser Algorithmus Flussvektoren so erzeugt, dass wir auch primal zulässig werden und den Unterraum $\mathfrak{K}$ erreichen. Hierbei gilt:

$$\mathfrak{K} = \{x | Ex = 0,\ x \text{ balanciert, aber nicht notwendig ganzzahlig}\} \tag{A.59}$$

Dazu werden wir eine untere Grenze für die Verbesserung in jedem Schritt angeben. Wir werden dann zeigen, dass, falls der Unterraum $\mathfrak{K}$ nicht erreicht wird, das duale Funktional selbst durch eine positive Zahl als Untergrenze zu charaktersieren ist, was für die primären Kosten $-\infty$ bedeutet. Damit erhalten wir dann den Widerspruch zur Endlichkeit der primalen Kosten.
Wir führen nun zunächst ein wenig Notation ein. Wir schreiben den Potentialvektor nach der r-ten Iteration $\pi^r, r \in \mathbb{N} \cup \{0\}$ und die Knoten auf denen die Relaxationen ausgeführt werden i^r, i'^r. Ferner schreiben wir $t^r = E^T p^r$ und $x_{ij}^r = \nabla G_{ij}(t_{ij}^r)$ $((ij) \in A(N))$. Ferner sei x^r der Vektor mit den Koordinaten x_{ij}^r.
Für jeden Kreis C des Netzwerkes bezeichnen wir mit C^+ die Vorwärtskanten und mit C^- die Rückwärtskanten in Y.
Bevor wir nun auf den eigentlichen Konvergenzsatz kommen, zeigen wir zunächst 3 Hilfssätze.

Satz A.8. *Es gilt für alle r, so dass $\pi^{r+1} \neq \pi^r$:*

$$G(\pi^r) - G(\pi^{r+1}) \geq \sum_{j \in A(N)} \left[C_{ij}(x_{ij}^{r+1}) - C_{ij}(x_{ij}^r) - (x_{ij}^{r+1} - x_{ij}^r)t_{ij}^r\right] > 0 \tag{A.60}$$

wobei die erste Ungleichung mit Gleichheit erfüllt ist, wenn Line-Minimization benutzt wird ($e_{k^r}(\pi^{r+1}) = 0,\ e_{k'^r}(\pi^{r+1}) = 0$).

Beweis. Sei r ein fester Index $r \geq 0, k = k^r$ und $\Delta = \pi^{r+1}(k) - \pi^r(k),\ \Delta' = \pi(k')^{r+1} - \pi(k')^r = -\Delta$. Wir haben nun:

$$G(\pi) = \sum_{(ij) \in A(N)} G_{ij}(\underbrace{\pi(i) - \pi(j)}_{t_{ij}}) = \sum_{(ij) \in A(N)} \sup_{x_{ij}} \{t_{ij}x_{ij} - C_{ij}(x_{ij})\} \tag{A.61}$$

Nun gilt aber:

$$x_{ij}^r = \arg\max_{x_{ij}} \{t_{ij}x_{ij} - C_{ij}(x_{ij})\} \tag{A.62}$$

$$\Rightarrow G(\pi^r) = \sum_{(ij) \in A(N)} \left[t_{ij}^r x_{ij}^r - C_{ij}(x_{ij}^r)\right]\ \forall r \geq 0 \tag{A.63}$$

$$\begin{aligned} \Rightarrow \quad & G(\pi^r) - G(\pi^{r+1}) \\ = \quad & \sum_{(ij) \in A(N)} \left[t_{ij}^r x_{ij}^r - C_{ij}(x_{ij}^r)\right] - \sum_{(ij) \in A(N)} \left[t_{ij}^{r+1} x_{ij}^{r+1} - C_{ij}(x_{ij}^{r+1})\right] \end{aligned} \tag{A.64}$$

Und damit:

$$= \sum_{(ij) \in A(N)} \left[t_{ij}^r x_{ij}^r - C_{ij}(x_{ij}^r)\right] - \sum_{(ij) \in A(N)} \left[\left(t_{ij}^r + e_{kj}\Delta + e_{k'j'}\Delta'\right) x_{ij}^{r+1} - C_{ij}(x_{ij}^{r+1})\right] \tag{A.65}$$

$$= \sum_{(ij) \in A(N)} \left[C_{ij}(x_{ij}^{r+1}) - C_{ij}(x_{ij}^r) - (x_{ij}^{r+1} - x_{ij}^r)t_{ij}^r - e_{kj}\Delta x_{ij}^{r+1} - e_{k'j'}\Delta' x_{ij}^{r+1}\right] \tag{A.66}$$

$$= \sum_{(ij) \in A(N)} \left[C_{ij}(x_{ij}^{r+1}) - C_{ij}(x_{ij}^r) - (x_{ij}^{r+1} - x_{ij}^r)t_{ij}^r\right] - 2\Delta \sum_{(ij) \in A} e_{kj}x_{ij}^{r+1} \tag{A.67}$$

$$= \sum_{(ij) \in A(N)} \left[C_{ij}(x_{ij}^{r+1}) - C_{ij}(x_{ij}^r) - (x_{ij}^{r+1} - x_{ij}^r)t_{ij}^r\right] - 2\Delta e_k(\pi^{r+1}) \tag{A.68}$$

Da $e_k(\pi^{r+1}) \leq 0$ folgt damit die Behauptung der rechten Seite.
Die linke Seite folgt mit $t_{ij}^r = \nabla C_{ij}(x_{ij}^r)$ (da ja C_{ij} und G_{ij} konjugiert sind) und der strikten Konvexität von C_{ij}. □

Satz A.9. *Die Folge $\{x^r\}$ ist beschränkt.*

Beweis. Wir halten zunächst fest, dass bei keiner Iteration die Summe der Defizite größer wird.

$$\sum_{i \in V} |e_i(\pi^{r+1})| \leq \sum_{i \in V} |e_i(\pi^r)| \tag{A.69}$$

Dies folgt aus der Tatsache, dass Flussveränderungen auf einer Kante zu einer Veränderung des Defizits auf dem Anfangsknoten und zu einer umgekehrten Veränderung des Defizits auf dem Endknoten führen. Die Defizite auf den Relaxationsknoten ändern nicht ihr Vorzeichen und ihr Absolutwert sinkt nur. Somit kann sich das Gesamtdefizit auf den Knoten, die nicht Relaxationsknoten sind, nicht erhöhen. Ferner kann e_k auf den Relaxationsknoten sich ebenfalls nicht erhöhen, wie im Algorithmus festgelegt ist.
Somit ist die Folge $\{e(\pi^r)\}$ beschränkt.
Sei nun $\{x^r\}$ unbeschränkt. Dann muss es mindestens ein Kantenpaar $(ij), (j'i')$ geben und eine Teilfolge $\{x^r\}$, so dass $|x_{ij}^r| \to \infty$, $|x_{j'i'}^r| \to \infty$ für $r \to \infty$.
Nun ist aber $\{e(\pi^r)\}$ beschränkt. Indem wir eventuell zu einer Teilfolge übergehen aber müssen aber wegen

$$\nabla G_{ij}(t_{ij}) = \arg\max_{x_{ij}} \{t_{ij}x_{ij} - C_{ij}(x_{ij})\} = \begin{cases} +\infty \\ -\infty \end{cases} \quad (t_{ij} \neq 0) \tag{A.70}$$

$$e_i(\pi) = \sum_{(ij) \in A(N)} e_{ij} \nabla G_{ij}(t_{ij}) \tag{A.71}$$

zwei Kreise C, C' existieren mit:

$$x_{ij}^r \to \infty \; \forall (ij) \in C^+ \text{ und } x_{ij}^r \to -\infty \; \forall (ij) \in C^- \tag{A.72}$$

$$x_{j'i'}^r \to \infty \; \forall (j'i') \in C'^+ \text{ und } x_{j'i'}^r \to -\infty \; \forall (j'i') \in C'^- \tag{A.73}$$

Hierbei sind jeweils die Grenzübergänge für $r \to \infty$ gemeint.
Nun haben wir noch die Complementary Slackness Bedingungen:

$$C_{ij}^-(x_{ij}^r) \leq t_{ij}^r \leq C_{ij}^+(x_{ij}^r) \tag{A.74}$$

$$C_{ji}^-(x_{j'i'}^r) \leq t_{j'i'}^r \leq C_{j'i'}^+(x_{ij}^r) \tag{A.75}$$

Verbinden wir dies mit den Bedingungen für den Kreis $\sum_{(ij) \in C^+} t_{ij}^r - \sum_{(ij) \in C^-} t_{ij}^r = 0, \sum_{(j'i') \in C'^+} t_{j'i'} - \sum_{(j'i') \in C'^-} t_{j'i'}^r = 0$, so erhalten wir für alle r:

$$\sum_{(ij) \in C^+} C_{ij}^-(x_{ij}^r) - \sum_{(ij) \in C^-} C_{ij}^+(x_{ij}^r) \leq 0 \tag{A.76}$$

$$\sum_{(j'i') \in C'^+} C_{ij}^-(x_{j'i'}^r) - \sum_{(j'i') \in C'^-} C_{ij}^+(x_{j'i'}^r) \leq 0 \tag{A.77}$$

Und damit erhalten wir den Widerspruch, da $x_{ij}^r \to \infty$ bedeutet, dass $C_{ij}^-(x_{ij}^r) \to \infty$, während $x_{ij}^r \to -\infty \Rightarrow C_{ij}^+(x_{ij}^r) \to -\infty$. □

Im nächsten Satz zeigen wir nun, wie die Folge der Preisvektoren sich der optimalen Lösung annähert. Das Ergebnis hängt mit der Monotonie der $\nabla G_{ij}(t_{ij})$ zusammen.

Satz A.10. *Gegeben sei ein $\pi \in \mathbb{R}^{|V|}$. Sei k ein Knoten und $\bar{\pi}$ der duale Preisvektor nach der Relaxation von π unter der Benutzung von k. Ferner sei $\bar{\pi}$ so gewählt, dass wenn $e_k(\pi) > 0$ dann $e_k(\bar{\pi}+\alpha(\pi-\bar{\pi})) > 0\ \forall\alpha > 0$ und wenn $e_k(\pi) < 0$ dann $e_k(\bar{\pi}+\alpha(\pi-\bar{\pi})) < 0\ \forall\alpha > 0$. Daraus folgt natürlich für $e_{k'}(\pi)$ jeweils die umgekehrte Relation. Wir haben dann für alle optimalen, dualen Potentialvektoren π^* und alle $k \in V$:*

$$\min\{\pi(i)-\pi^*(i)|i\in V\} \leq \bar{\pi}(k)-\pi^*(k) \leq \max\{\pi(i)-\pi^*(i)|i\in V\} \quad \text{(A.78)}$$

$$\min\{\pi(i)-\pi^*(i)|i\in V\} \leq \bar{\pi}(k')-\pi^*(k') \leq \max\{\pi(i)-\pi^*(i)|i\in V\} \quad \text{(A.79)}$$

Bemerkung Die obige Annahme für $e_k(\pi) < 0$ bzw. $e_k(\pi) > 0$ ist äquivalent zu der Annahme, dass $\bar{\pi}(k)$ größer (kleiner) oder gleich dem minimierenden Punkt der dualen Kosten entlang der k-ten Komponente bei Start von π aus gewählt wird. Anders formuliert nähern wir uns dem Minimum im ersten Fall von größeren und zweiten Fall von kleineren π-Werten an.
In unserem symmetrischen Fall nähern wir uns also mit k und k' jeweils von verschiedenen Seiten dem Minimum an.
Die Anforderung ist also automatisch erfüllt, falls die dualen Kosten ein eindeutiges Minimum längs der Geraden $\{\pi + \alpha\hat{e_k}|\alpha\in\mathbb{R}\}$ haben.

Beweis. Sei π^* ein optimaler, dualer Preisvektor und $\tilde{\pi}$ ein beliebiger Preisvektor. Sei ferner k so gewählt, dass $\tilde{\pi}(k)-\pi^*(k) = \max\{\tilde{\pi}(i)-\pi^*(i)|i\in V\}$. Damit haben wir also $\tilde{\pi}(k)-\pi^*(k) \geq \tilde{\pi}(i)-\pi^*(i)\ \forall i \neq k$. Und damit auch:

$$\tilde{\pi}(k)-\tilde{\pi}(i) \geq \pi^*(k)-\pi^*(i)\ \forall(k,i)\in A \quad \text{(A.80)}$$

$$\tilde{\pi}(i)-\tilde{\pi}(k) \leq \pi^*(i)-\pi^*(k)\ \forall(i,k)\in A \quad \text{(A.81)}$$

Wir können nun ausnutzen, dass $\nabla G_{ij}(t_{ij})$ eine nichtfallende Funktion ist und erhalten:

$$\nabla G_{ki}(\tilde{\pi}(k)-\tilde{\pi}(i)) \geq \nabla G_{ki}(\pi^*(k)-\pi^*(i)) \quad \text{(A.82)}$$

$$\nabla G_{ik}(\tilde{\pi}(i)-\tilde{\pi}(k)) \geq \nabla G_{ik}(\pi^*(i)-\pi^*(k)) \quad \text{(A.83)}$$

Damit gilt also $e_k(\tilde{\pi}) \geq e_k(\pi^*) = 0$. Damit können wir nun die Behauptung beweisen. Wir unterscheiden 3 Fälle:
Sei $e_k(\pi) = 0$, dann gilt $\pi = \bar{\pi}$ und damit die Behauptung.
Sei nun $e_k(\pi) < 0$. Dann betrachten wir den Vektor $\tilde{\pi}$, der die extremste Änderung in einer Relaxation beschreibt:

$$\tilde{\pi}(i) = \begin{cases} \pi(i), & i \neq k, k' \\ \pi^*(k) + \max\{\pi(j)-\pi(j)^*|j\in V\}, & i = k \\ \pi^*(k') + \min\{\pi(j)-\pi(j)^*|j\in V\}, & i = k' \end{cases} \quad \text{(A.84)}$$

Dann haben wir wir $\tilde{\pi}(k)-\pi(k)^* = \max\{\tilde{\pi}(j)-\pi(j)^*|j\in V\} = \max\{\pi(j)-\pi(j)^*|j\in V\}$. Mit den obigen Ausführungen erhalten wir damit $e_k(\tilde{\pi}) \geq 0$.
Damit gilt natürlich auch $\tilde{\pi}(k')-\pi(k')^* = \min\{\tilde{\pi}(j)-\pi(j)^*|j\in V\} = \min\{\pi(j)-\pi(j)^*|j\in V\}$ und damit auch $e_{k'}(\tilde{\pi}) \leq 0$. Nach Wahl von $\tilde{\pi}$ gilt für den behandelten Fall $\bar{\pi}(k) \leq \tilde{\pi}(k)$. Aufgrund der Relaxation gilt $\pi(k) < \bar{\pi}(k),\ \pi(k') > \bar{\pi}(k')$ und $\pi(i) = \bar{\pi}(i)\ \forall i \in V, i \neq k, k'$. Damit folgt unsere Behauptung. Der Beweis für $e_k(\pi) > 0$ ist nun offensichtlich. □

Insbesondere zeigt der obige Satz, dass die Folge $\{\pi^r\}$ beschränkt ist. Wir müssen jetzt also noch zeigen, dass wir sogar gegen die optimale Lösung iterieren. Damit kommen wir zum wichtigsten und letzten Satz zu dem vorgestellten Algorithmus.

Satz A.11 (Konvergenz des Balanced Relaxation Algorithmus)**.** *Sei $\{\pi^r, x^r\}$ eine Folge, die durch den Balanced Relaxation Algorithmus mit strikt konvexen Kosten erzeugt wurde, dann gilt:*

(i) $\lim_{r\to\infty} e(\pi^r) = 0$

(ii) $\lim_{r\to\infty} x^r = x^*$*, mit dem optimalen Fluss* x^*

(iii) $\lim_{r\to\infty} G(\pi^r) = -C(x^*) = \inf_\pi G(\pi)$

(iv) Wird die Bedingung aus dem letzten Hilfssatz in jeder Iteration erfüllt und das duale Problem ist die optimale Lösung, dann gilt:

$$\lim_{r\to\infty} \pi^r \to \pi^* \tag{A.85}$$

mit einem optimalen Potentialvektor π^**.*

Beweis. Wir beginnen mit (i) und zeigen zunächst für eine beliebigen beliebigen Knoten $l \in V$ in seiner r-ten Relaxation:

$$\lim_{r\to\infty} e_{l^r}(\pi^r) = 0, \ \lim_{r\to\infty} e_{l^r}(\pi^r) = 0 \tag{A.86}$$

Gelte dies nicht, so gäbe es ein $\epsilon > 0$ und eine (Teil-)Folge R, so dass $|e_{l^r}(\pi^r)| \geq \epsilon \ \ \forall r \in R$. Ferner würde aufgrund der Symmetrie für die selbe (Teil-)Folge R gelten, dass $|e_{l^r}(\pi^r)| \geq \epsilon \ \forall r \in R$ gelten. O.B.d.A. können wir also annehmen $|e_{l^r}(\pi^r)| \geq \epsilon, \ |e_{l^r}(\pi^r)| \geq \epsilon \ \forall r \in R$ annehmen.
Aufgrund des Relaxationsalgorithmus wissen wir $\delta|e_{l^r}(\pi^r)| \geq |e_{l^r}(\pi^{r+1})|, \ \delta|d_{l^r}(\pi^r)| \geq |e_{l^r}(\pi^{r+1})|$. Damit können wir nun die Flussänderung in der r-ten Iteration auf den Kanten inzident zu l^r betrachten.
Hierzu betrachten wir $e_{l^{r+1}}(\pi^r) = \sum_{(lj)\in A(N)} e_{lj}\nabla G_{lj}(t^r_{lj}) = \sum_{(lj)\in A(N)} e_{lj}x^r_{lj}$.
Die Flussänderung, die nötig ist, um einen Überschuss $|e_{l^r}(\pi^{r+1})|$ zu erreichen, muss größer sein, als die nötige Flussänderung um das Defizit $\delta|e_{l^r}(\pi^r)|$ zu erreichen. Wir betrachten also zunächst die nötige Flussänderung für $\delta|e_{l^r}(\pi^r)|$:

$$\delta|e_{l^r}(\pi^r)| \ = \ \left|\sum_{(lj)\in A} e_{lj}(x^r_{lj} - \Delta_{lj})\right| \tag{A.87}$$

Die kleinsten Flussänderungen Δ auf den Kanten (die wir hier o.B.d.A. positiv wählen wollen) erhalten wir bei absolutwertmäßig äquivalenter Verteilung auf allen Kanten, also:

$$\delta|e_{l^r}(\pi^r)| \leq \left|\sum_{(lj)\in A(N)} e_{lj}x^r_{lj}\right| - |A|\Delta \tag{A.88}$$

Und mit $\delta|e_{l^r}(\pi^r)| \geq \delta\epsilon$ erhalten wir damit $\Delta \geq \frac{(1-\delta)\epsilon}{|A|}$.
Indem wir nun eventuell wieder zu einer Teilfolge übergehen, können wir annehmen, dass die Flussänderung jeweils immer auf der selben Kante (mn) stattfindet und damit $x^{r+1}_{mn} - x^r_{mn} \geq \Delta \; \forall r \in R$.
Benutzen wir nun die Beschränktheit von $\{x^r\}$, die wir im Hilfssatz (A.9) gezeigt haben, so können wir nach dem Satz von Bolzano-Weierstraß (eventuell wieder unter Auswahl einer Teilfolge) annehmen, dass die Folge $\{x^r_{mn}\}_{r\in R}$ gegen den Grenzwert x_{mn} konvergiert. Analoges gilt natürlich auch wieder für $x_{n'm'}$.
Benutzen wir nun die Konvexität der Kostenfunktion und den ersten Hilfssatz, so haben wir:

$$\begin{aligned} G(\pi^r) - G(\pi^{r+1}) &\geq C(x^{r+1}_{mn}) - C(x^r_{mn}) - \\ &\quad (x^{r+1}_{mn} - x^r_{mn})t^r_{mn} - (x^{r+1}_{n'm} - x^r_{n'm'})t^r_{n'm'} && \text{(A.89)} \\ &\geq C(x^r_{mn} + \Delta) - C(x^r_{mn}) - \Delta t^r_{mn} - t^r_{n'm'} && \text{(A.90)} \\ &= C(x^r_{mn} + \Delta) - C(x^r_{mn}) - 2\Delta t^r_{mn} && \text{(A.91)} \\ &\geq C(x^r_{mn} + \Delta) - C(x^r_{mn}) - 2\Delta C^+_{mn}(x^r_{mn}) && \text{(A.92)} \end{aligned}$$

Wir gehen nun zum Grenzwert für $r \to \infty, r \in R$ über. Hierbei nutzen wir aus, dass $x^r_{mn} \to x_{mn}$ und $\lim_{r\to\infty} C^+_{mn}(x^r_{mn}) = C^+_{mn}(x_{mn})$. Die letzte Aussage ergibt sich dabei aus der Oberhalbstetigkeit von C^+_{mn}. Wir erhalten damit:

$$\begin{aligned} \liminf_{r\to\infty,\, r\in R} \left[G(\pi^r) - G(\pi^{r+1})\right] &\geq C(x_{mn} + \Delta) - C(x_{mn}) - 2\Delta C^+_{mn}(x_{mn}) > 0 && \text{(A.93)} \\ &\Rightarrow \lim_{r\to\infty} G(\pi^r) = -\infty && \text{(A.94)} \end{aligned}$$

Aus der Definition von G erhalten wir aber sofort $G(\pi) \geq -\sum_{(ij)\in A(N)} C_{ij}(x_{ij})$ und damit den Widerspruch.
Nach diesem Vorresultat gehen wir nun an den eigentlichen Beweis von (i). Hierbei müssen wir nun nicht nur die Schritte betrachten, in denen eine Relaxation auf dem Knoten ausgeführt wird. Wählen wir also ein beliebiges $k \in V(N)$. Sei $\epsilon > 0$ und reell und R die Menge von Indizes r, so dass $e_k(\pi(r)) > 2\epsilon$. O.B.d.A. nehmen wir an, dass $e_k(\pi^r) < \epsilon$ für alle $k = l^r$ (also nach der Relaxation). Nach dem ersten Beweisschritt ist dies möglich.
Für jedes $r \in R$ sei nun $\tilde{r}$ der erste Index mit $\tilde{r} > r$ für den $k = l^{\tilde{r}}$. Während der Relaxationen $r, r+1, \ldots, \tilde{r}-1$ wird also k nicht zur Relaxation verwendet, während das Defizit sich von $e_k(\pi^r) > 2\epsilon$ (nach Wahl von r) auf $e_k(\pi^{\tilde{r}}) < \epsilon$ verringert. Wir wollen nun zeigen, dass sich das totale Defizit während dieser Iterationen um mehr als 2ϵ verringert.
Wir verwenden wieder die Aussage des Hilfssatzes darüber, dass das totale Defizit nicht ansteigen kann. Nun betrachten wir die Iterationen $r, \ldots, \tilde{r}-1$. Wir bezeichnen nun die Menge aller Relaxationen, bei denen das Defizit von k verringert wird mit $\bar{R}$ und folglich die Indizes dieser Relaxationen mit $\bar{r}$. Die zugehörigen Knoten müssen (bei $e_k(\pi^{\bar{r}}) > 0$) ein negatives Defizit $e_{l^{\bar{r}}}(\pi^r)$ haben. Die Verringerung des Defizits in der Relaxation sei χ. Da der Anstieg von $e_{l^{\bar{r}}}$ in der Relaxation mit Verringerungen positiver Defizite von benachbarten Kanten von $l^{\bar{r}}$ aufgebracht werden muss und Vorzeichenwechsel der Defizite in der Relaxation nicht vorkommen, wird das absolute Defizit verringert um mindestens $4\min\{\chi, e_k(\pi^r)\}$ während der Iteration (Faktor 2 von 2 betrachteten Knoten und Faktor 2 von der symmetrischen Relaxation). Folglich muss das totale Defizit sogar um mehr als

4ϵ sinken.
Damit erhalten wir insbesondere, dass die zuvor definierte Menge R nicht unendlich sein kann. Ferner gilt damit $\limsup_{r\to\infty} e_k(\pi^r) \leq 0$. Wir erhalten nun völlig analog $\liminf_{r\to\infty} e_k(\pi^r) \geq 0$ und damit $e_k(\pi^r) \to 0$. Ferner gilt natürlich mit obiger Argumentation, dass $\limsup_{r\to\infty} e_{k'}(\pi^r) \leq 0$ und dann wieder ganz analog $\liminf_{r\to\infty} e_{k'}(\pi^r) \leq 0$. Damit folgt dann auch $e_{k'}(\pi^r) \to 0$.
Zeigen wir nun (ii). Für alle r aus der Menge der Relaxationen und alle Kanten $(ij) \in A(N)$ gilt die Complementary Slackness Bedingung $C_{ij}^-(x_{ij}^r) \leq t_{ij}^r \leq c_{ij}^+(x_{ij}^r)$.
Wir hatten bereits im Beweis zum Satz zur Beschränktheit der Folge $\{x^r\}$ mit den dortigen Bezeichnungen gezeigt:

$$\sum_{(ij)\in C^+} C_{ij}^-(x_{ij}^r) - \sum_{(ij)\in C^-} C_{ij}^+(x_{ij}^r) \leq 0 \leq \sum_{(ij)\in C^+} C_{ij}^+(x_{ij}^r) - \sum_{(ij)\in C^-} C^-(x_{ij}^r) \tag{A.95}$$

Sei $\{x^r\}_{r\in R}$ eine Teilfolge, die gegen $\bar{x}$ konvergiere. Dann erhalten wir aus der Unter- (Ober-) halbstetigkeit von C_{ij}^- (C_{ij}^+) und der obigen Ungleichung:

$$\sum_{(ij)\in C^+} C_{ij}^-(\bar{x}_{ij}) - \sum_{(ij)\in C^-} C_{ij}^+(\bar{x}_{ij}) \leq 0 \leq \sum_{(ij)\in C^+} C^+(\bar{x}_{ij}) - \sum_{(ij)\in C^-} C^-(\bar{x}_{ij}) \tag{A.96}$$

Mit der gleichen Argumentation erhalten wir das selbe Resultat natürlich auch für die Teilfolge $\{x'^r\}_{r\in R}$ mit der gleichen ausgewählten Teilfolge. Damit ist ebenfalls nach dem Beweis zum Satz über die Beschränktheit der Folge $\{x^r\}$ die Complementary Slackness Bedingung erfüllt. Die Lösung ist also optimal. Mit Teil (i) haben wir damit nun sogar die Eigenschaft, dass $\bar{x}$ die Gleichung $E\bar{x} = 0$ erfüllt.
Beweisen wir nun (iii). Für jede Kante (ij) mit $l(ij) < cap(ij)$ und damit auch $l(j'i') < cap(j'i')$ haben wir 3 Möglichkeiten:

(i) $\{t_{ij}^r\}$ ist beschränkt und $\{t_{j'i'}^r\}$ ist ebenfalls beschränkt.

(ii) $x_{ij}^* = cap(ij) < \infty,\ x_{ij}^r \leq x_{ij}^*$ und $-\infty < \liminf_{r\to\infty} t_{ij}^r \leq \limsup_{r\to\infty} t_{ij}^r = +\infty$
$x_{j'i'}^* = cap(ij) < \infty, x_{j'i'}^r \leq x_{j'i'}^*$ und $-\infty < \liminf_{r\to\infty} t_{j'i'}^r \leq \limsup_{r\to\infty} t_{j'i'} = +\infty$

(iii) $x_{ij}^* = l(ij) > -\infty,\ x_{ij}^r \geq x_{ij}^*$ und $-\infty = \liminf_{r\to\infty} t_{ij}^r \leq \limsup_{r\to\infty} t_{ij}^r < \infty$
$x_{j'i'}^* = l(ij) > -\infty,\ x_{j'i'}^r \geq x_{j'i'}^*$ und $-\infty = \liminf_{r\to\infty} t_{j'i'}^r \leq \limsup_{r\to\infty} t_{j'i'}^r < \infty$

Dies ist nichts anderes als eine duale Version für etwas, was uns aus den Scaling-Algorithmen längst bekannt ist. Dort haben wir ebenfalls eine Unterscheidung in die 3 Fälle der positiven, negativen und verschwindenden reduzierten Kosten.
Für eine Kante mit $l(ij) = cap(ij)$ (und damit auch $l(j'i') = cap(j'i')$) gilt natürlich $x_{ij}^* = x_{ij}^r\ \forall r$. Wir können nun eine Teilfolge auswählen, so dass $(x_{ij}^r - x_{ij}^*)t_{ij}^r < 0$ für alle Kanten des 2. Falles und $(x_{ij}^r - x_{ij}^*)t_{ij}^r < 0$ für alle Kanten des 3. Falles. Damit gilt nun:

$$\sum_{(ij)\in A(N)} t_{ij}^r(x_{ij}^r - x_{ij}^*) \leq \sum_{(ij)\in B(N)} t_{ij}^r(x_{ij}^r - x_{ij}^*) \tag{A.97}$$

wobei $B(N)$ die Menge der Kanten mit beschränktem t_{ij}^r (1. Fall) ist.
Wir wissen nach den Hilfssätzen ferner $t^r \in C^\perp$, $x^* \in C$ und damit $\sum_{(ij)\in A(N)} t_{ij}^r x_{ij}^* = 0$.

Damit erhalten wir nun:

$$C(x^r) + G(\pi^r) = \sum_{(ij)\in A(N)} t_{ij}^r x_{ij}^r = \sum_{(ij)\in A(N)} t_{ij}^r (x_{ij}^r - x_{ij}^*) \quad \text{(A.98)}$$

$$\leq \sum_{(ij)\in B(N)} t_{ij}^r (x_{ij}^r - x_{ij}^*) \quad \text{(A.99)}$$

Wir haben bereits $x_{ij}^r \to x_{ij}^*$ für $r \to \infty$ gezeigt. Ferner ist $\{t_{ij}^r\}_R$ beschränkt für $(ij) \in B(N)$. Nehmen wir nun den Grenzwert für $r \to \infty$, so gilt: $\lim_{r\to\infty} \sum_{(ij)\in B(N)} t_{ij}^r (x_{ij}^r - x_{ij}^*) = 0$, da x und t konvergieren und damit $C(x^*) + \lim_{r\to\infty} G(\pi^r) \leq 0$. Wir haben allerdings $C(x^*)+G(\pi) = C(x^*)+\sum_{(ij)\in A} \sup_{x_{ij}} \{t_{ij}x_{ij} - C(x_{ij})\} \geq 0$. Und damit erhalten wir (iii):

$$\lim_{r\to\infty} G(\pi^r) = -C(x^*) = \inf_{\pi} G(\pi) \quad \text{(A.100)}$$

Hierbei folgt die letzte Gleichung, da $G(\pi^r)$ für r nicht steigend ist.
Wir beweisen nun noch (iv). Mit dem Hilfssatz (A.9) gilt dass $\{\pi^r\}$ beschränkt ist. Sei nun $\{\pi^r\}_{r\in R}$ eine konvergente Teilfolge mit dem Grenzwert π^*. Sei dann $t^* = E^T p^*$, dann gilt für alle $(ij) \in A$ wieder $C_{ij}^-(x_{ij}^r) \leq t_{ij}^r \leq C_{ij}^+(x_{ij}^r)$. Dann folgt mit (ii) und der Unter- bzw. Oberhalbstetigkeit von C_{ij}^-, C_{ij}^+ auch für alle $(ij) \in A$: $C_{ij}^-(x_{ij}^*) \leq t_{ij}^* \leq C_{ij}^+(x_{ij}^*)$ mit einem optimalen Flussvektor x^*. Damit erfüllt t^* mit x^* die Bedingung vom komplementären Schlupf und muss daher dual optimal sein. Der Hilfssatz (2.8) zeigt noch, dass $\{\pi^r\}$ keine 2 Häufungspunkte haben kann und wir sind fertig. □

A.4 Weitere Implementierungen

In der Arbeit von Bertsekas, Hosein und Tseng wird neben dem hier präsentierten Algorithmus auch eine Version des Algorithmus für gemischt lineare und konvexe Kosten vorgestellt [13, S. 1228-1239]. Der Autor nimmt an, dass sich auch dieser Algorithmus mit den bisher geleisteten Vorarbeiten auf schiefsymmetrische Netzwerke anwenden lässt.
Die Komplexität des hier beschriebenen Algortithmus wurde nicht besprochen. Wir haben lediglich darauf hingewiesen, dass die worst-case Laufzeit schlecht ist [11, S. 108]. Bertsekas, Hosein und Tseng haben für den Algorithmus auf 'normalen' Netzwerken allerdings einige Computersimulationen durchgeführt und konnten zeigen, dass der Algorithmus für verschiedene Probleme effizient arbeitet [13, S. 1240-1242]. Für eine verbesserte Version des Algorithmus auf 'normalen' Netzwerken sei hier auf die Arbeit von Bertsekas und Tseng [14] verwiesen.
Die meisten Ansätze der nichtlinearen Programmierung sind ähnlich zu dem hier vorgestellten. Eine weitere Klasse von Algorithmen bedient sich allerdings der Newton-Methode als Ansatz (vergleiche hierzu die Arbeit von Ahlfeld et. al. [2] oder die Arbeit von Klincewicz [39]). Es sei aber darauf hingewiesen, dass diese eine zwei Mal differenzierbare Kostenfunktion auf den Kanten benötigen.

Anhang B

Cancel-and-Tighten Method

Wir wollen abschließend noch einen besseren dualen Ansatz präsentieren. Dieser basiert in seinen Ideen auf dem Mean-Min-Cycle-Cancelling von Goldberg und Tarjan [31].
Wir beschreiben hier zunächst die in [37] beschriebene Cancel and Tighten Methode von Goldberg und Karzanov. Wir wollen abschließend noch einige Ideen für einen Algorithmus auf schiefsymmetrischen Netzwerken ansprechen.

B.1 Theorie der Cancel and Tighten Methode

Die Cancel and Tighten Methode wurde von McCormick und Karzanov für den allgemeinen Fall der separablen konvexen Optimierung auf bestimmten Räumen beschrieben. Diese Allgemeinheit wollen wir hier nicht ausnutzen. Es sei aber darauf hingewiesen, dass unsere Ergebnisse hier in einem allgemeineren Rahmen betrachtet werden können.
Wir wollen zunächst wieder das Problem betrachten. Die Netzwerkinzidenzmatrix E hatten wir bereits im Balanced Relaxation Algorithmus eingeführt. Wir fügen auch hier wieder die Kante (ts) zum Netzwerk hinzu und betrachten nun den allgemeinen Fall nicht-balancierter Zirkulationen $x_{ij} \in \mathbb{R}$. Wir erhalten so folgendes Optimierungsproblem:

$$\begin{aligned} \min C(x) &= \sum_{(ij)\in A(N)} C_{ij}(x_{ij}) && \text{(B.1)} \\ Ex &= 0 && \text{(B.2)} \\ l(ij) \leq\ & x_{ij} \leq cap(ij) && \text{(B.3)} \end{aligned}$$

Wir wollen das durch die Nebenbedingungen beschriebene Polytop mit L bezeichnen.
Damit haben wir das Problem formuliert. Es wird vermutlich bereits deutlich, dass wir ein wenig auf die Theorie des Balanced Relaxation Algorithmus zurückgreifen können. Ganz analog zum Balanced Relaxation Algorithmus können wir wieder das duale Problem betrachten. Das zugehörige Polytop bezeichnen wir hier mit $L^{\perp}$. Dort hatten wir die besondere Struktur der Netzwerkinzidenzmatrix E bereits angesprochen. Wir wollen dies mit einer Definition noch genauer fassen [45, S. 160].

Definition B.1 (unimodular)**.** *Wir nennen eine Matrix unimodular, falls die Matrix eine Determinante* ± 1 *hat. Wir nennen die Matrix total unimodular, falls jede Untermatrix jeweils unimodular ist.*

Die Netzwerkinzidenzmatrix ist also ein typisches Beispiel für eine total unimodulare Matrix.
Wir hatten im Balanced Relaxation Algorithmus auch die strikte Konvexität der Kostenfunktionen bereits genauer untersucht. Wir bezeichnen auch hier wieder die rechts- und linksseitigen Ableitungen mit C_{ij}^+ und C_{ij}^-. Wir hatten bereits das Analogon zur Complementary Slackness Optimality betrachtet und erhielten: $C_{ij}^-(x_{ij}) \leq C_{ij}^+(x_{ij})$. Mit der strikten Konvexität der C_{ij} erhalten wir dann für $x_{ij} < \tilde{x}_{ij}$ sofort:

$$C_{ij}^-(x_{ij}) \leq C_{ij}^+(x_{ij}) \leq C_{ij}^-(\tilde{x}_{ij}) \leq C_{ij}^+(\tilde{x}_{ij}) \tag{B.4}$$

Damit sind insbesondere C_{ij}^- und C_{ij}^+ monoton und nicht-fallend.
Karzanov und McCormick nehmen nun an, dass die Funktionen C_{ij} lediglich durch ein sogenanntes Orakel gegeben seien. Dieses Orakel O habe folgende Eigenschaften:

(i) für einen Punkt $x_{ij} \in \mathbb{R}$ gibt es $C_{ij}^-(x_{ij})$ und $C_{ij}^+(x_{ij})$ zurück.

(ii) für eine gegebene Steigung $t_{ij} \in \mathbb{R}$ gibt es einen Punkt x_{ij} zurück mit $C_{ij}^- \leq t_{ij} \leq C_{ij}^+$

Es ist meist einfach ein solches Orakel zu konstruieren, indem man spezielle Suchalgorithmen verwendet [38, S. 1254].
Wir haben obig bereits angesprochen, dass wir wieder Zirkulationen betrachten wollen. In der Balanced Decomposition haben wir bereits gezeigt, dass jede balancierte Zirkulation aus Paaren gültiger Kreise aufzubauen ist. Für allgemeine Zirkulationen kann man ganz ähnlich beweisen, dass auch diese sich immer als Linearkombinationen von Zirkulationen auf Kreisen ξ^i mit $i \in \{1, \cdots, k\}$ schreiben lassen:

$$x = \alpha_1 \xi^1 + \cdots + \alpha_k \xi^k \tag{B.5}$$

$$\xi^1, \cdots, \xi^k \in L,\ \alpha_1, \cdots, \alpha_k \in \mathbb{N} \tag{B.6}$$

Wir können nun die Kreise so wählen, dass gilt: $\xi_{ij}^p \xi_{ij}^q \geq 0\ \forall p, q \in \{1, \cdots, k\},\ (ij) \in A(N)$. Man bezeichnet dies auch als *konforme Zerlegung* von x.
Wir wollen nun auf die bereits angesprochenen Minimum Mean Cycles kommen. Hierzu führen wir zunächst eine neue Schreibweise für Kreise ein. Wir können dabei jeden der obig eingeführten Kreise als Vektor im Raum $\mathbb{R}^{|A(N)|}$ beschreiben, wobei jeweils die Flussrate auf der Kante $(ij) \in A(N)$ einem Eintrag entspricht. Sei der Fluss auf dem Kreis x_ξ, so haben wir dann einen $(0, \pm x)$-Vektor $\xi \in \mathbb{R}^{|A(N)|}$. Wir schreiben nun den Kreis ξ als ein Paar $F_\xi = (A_\xi, B_\xi)$ mit $A_\xi := \{(ij) \in A(N) \mid \xi_{ij} > 0\}$ und $B_\xi := \{(ij) \in A(N) \mid \xi_{ij} < 0\}$. Diese Konstruktion lässt sich völlig analog auch auf die bereits im Balanced Out-of-Kilter eingeführten Co-Kreise übertragen. Ist ξ ein Kreis (bzw. Co-Kreis), so bezeichnen wir auch F_ξ als Kreis bzw. Co-Kreis. Die Menge der Kreise F_ξ bezeichnen wir mit $\mathfrak{F}$ und die Menge der Co-Kreise F_ξ bezeichnen wir mit $\mathfrak{F}^\perp$. Die Kardinalität eines Kreises F definieren wir nun natürlich als $|F| := |A| + |B|$.
Für eine Funktion $f : S \to \mathbb{R}$ schreiben wir wieder für eine Teilmenge $S \subseteq S'$: $f(S') := \sum\{f_{ij} : (ij) \in S'\}$. Für festes $x \in L$ und einen Kreis $F = (A, B)$ schreiben wir $\sum_{(ij)\in A} C_{ij}^+(x_{ij}) =: c^+(A)$ und $\sum_{(ij)\in B} C_{ij}^-(x_{ij}) =: c^-(B)$.
Erhöhen wir nun den Fluss längs des Kreises F um $\epsilon > 0$, so müssen wir auf den Kanten

aus B den Fluss um ϵ verringern und auf den Kanten aus A um ϵ erhöhen. Der neue Fluss x' ergibt sich damit auf jeder Kante aus:

$$x'_{ij} = \begin{cases} x_{ij} + \epsilon \\ x_{ij} - \epsilon \\ x_{ij} \text{ sonst} \end{cases} \tag{B.7}$$

Verändern wir die Kosten auf dem Kreis um eine infinitesimale Einheit ϵ, so verändern sich die Kosten um $(c^+(A) - c^-(B)) \cdot \epsilon$. Wir bezeichnen damit $c^+(A) - c^-(B)$ auch als die Kosten von F bei x und schreiben:

$$c(x, F) := c^+(A) - c^-(B) \tag{B.8}$$

Die durschnittlichen Kosten eines Kreises sind dann $\bar{c}(x, F) := \frac{c(x,F)}{|F|}$.

Definition B.2 (Minimum Mean Cycle)**.** *Wir bezeichnen F als einen Minimum Mean Cycle, falls F der Kreis aus $\mathfrak{F}$ ist, für den $\bar{c}(x, F)$ sein Minimum annimmt.*

Damit definieren wir nun:

Definition B.3 (Absolute Mean Value)**.** *Für eine gegebene Menge von Kreisen $\mathfrak{F}$ und einen Fluss x auf einem gegebenen Flussnetzwerk N gilt:*

$$\lambda(x) := \max\{0, \ -\min_{F \in \mathfrak{F}} \bar{c}(x, F)\} \tag{B.9}$$

Gilt $\lambda(x) > 0$, so nennen wir $\lambda(x)$ den absolute mean value von x.

Mit diesen Definitionen erhalten wir die Negative Cycle Optimality in folgender Form:

Lemma B.4 (Negative Cycle Optimality für Minimum Mean Cycles)**.** *$x \in L$ ist eine optimale Lösung des Problems, genau dann wenn $\lambda(x) = 0$ gilt.*

Wir müssten hier eigentlich noch die Übertragung unseres Resultats für lineare Kosten auf konvexe Kosten leisten. Der Beweis ist aber leider eher technisch und verweisen den Leser daher auf die Arbeit von McCormick und Karzanov [38, S. 1254].
Einen Teil des Beweises wollen wir im folgenden Lemma leisten, dass für unseren Algorithmus sehr grundlegend ist. Wir sehen bereits in der obigen Definition, dass $\lambda(x)$ offensichtlich eine wichtige Größe ist. Im folgenden Lemma wollen wir λ genauer charakterisieren.

Lemma B.5 (MMCM Lemma)**.** *Ein reelles λ ist eine obere Grenze für $\lambda(x)$, genau dann wenn ein $h \in L^\perp$ existiert, so dass für die reduzierten Kosten gilt:*

$$C^+_{ij}(x_{ij}) - h_{ij} \geq -\lambda \ \forall (ij) \in A(N) \tag{B.10}$$
$$h_{ij} - C^-_{ij}(x_{ij}) \geq -\lambda \ \forall (ij) \in A(N) \tag{B.11}$$

Ferner gilt für die Kanten des Minimum Mean Cycle $F = (A, B)$ mit $\lambda = \lambda(x) > 0$ und $h \in L^\perp$:

$$C^+_{ij}(x_{ij}) - h_{ij} = -\lambda \ \forall (ij) \in A \tag{B.12}$$
$$h_{ij} - C^-_{ij}(x_{ij}) = -\lambda \ \forall (ij) \in B \tag{B.13}$$

Beweis. Wir bezeichnen einen Vektor mit lauter Einsen mit u und betrachten folgendes Optimierungsproblem:

$$\min\left(\frac{C^+x^+ - C^-x^-}{ux^+ + ux^-}\right) \tag{B.14}$$

$$E(x^+ - x^-) = 0 \tag{B.15}$$

$$x^+, x^- \geq 0 \tag{B.16}$$

Hat dieses Programm eine optimale Lösung, so können wir den Vektor (x^+, x^-) als Vektor für die Mengen A und B auffassen, wobei x^+ der Menge A und x^- der Menge B entspricht. Die Lösung muss dabei eine Zirkulation sein, da $E(x^+ - x^-) = 0$ gefordert wird. Damit ist die Lösung ein Minimum Mean Cycle.
Das Optimierungsproblem ist homogen.Wir können also das Problem normalisieren, indem wir $ux^+ + ux^- = 1$ setzen. Damit erhalten wir nun folgendes Problem:

$$\min C^+x^+ - C^-x^- \tag{B.17}$$

$$E(x^+ - x^-) = 0 \tag{B.18}$$

$$ux^+ + ux^- = 1 \tag{B.19}$$

$$x^+, x^- \geq 0 \tag{B.20}$$

Wir erhalten damit das duale Programm:

$$\max \delta \tag{B.21}$$

$$\pi E - \delta u \leq C^+ \tag{B.22}$$

$$-\pi E + \delta u \leq -C^- \tag{B.23}$$

$$\pi,\ \delta \text{ frei} \tag{B.24}$$

Hierbei sind natürlich die Ungleichungen für alle Komponenten der Vektoren gemeint. Da π frei ist, ist πE ein beliebiger Vektor $h \in L^\perp$ und wir können das duale Programm folgendermaßen notieren:

$$\max \delta \tag{B.25}$$

$$-\delta u \leq C^+ - h \tag{B.26}$$

$$\delta u \leq -C^- + h \tag{B.27}$$

$$\pi,\ \delta \text{ frei} \tag{B.28}$$

Damit bestimmen wir also den größten Wert δ, der die obigen Bedingungen des MMCM Lemmas erfüllt. Damit haben wir den ersten Teil des Beweises aus der Dualität der linearen Programmierung.
Den 2. Teil des Beweises erhalten wir aus der Definition von F als Minimum Mean Cycle. Gelte $C^+_{ij}(x_{ij}) - h_{ij} > -\lambda(x)$ für ein Element aus A, so hätten wir, da λ ja eine obere Grenze ist, auch $\bar{c}(x, F) > -\lambda$. Damit hätten wir dann aber einen Widerspruch zu $\lambda = \lambda(x)$, da wir ja $\lambda = \delta u$ für einen Minimum Mean Cycle berechnen. □

Wir sehen damit, dass uns λ offensichtlich eine obere Grenze (und sogar eine beste obere Grenze) für die reduzierten Kosten liefert. Damit ist $\lambda(x)$ auch ein Maß für die

duale Konvergenz. Im Balanced Relaxation Algorithmus konnten wir nur sehr schlechte Grenzen für die Konvergenz angeben. Dies wollen wir hier natürlich verbessern. Als erstes wollen wir daher ein Maß für die Qualität der Lösung einführen.
Betrachten wir dazu ein $x \in L$ und ein $h \in L^{\perp}$. Für jede Kante $(ij) \in A(N)$ definieren wir eine Art Skalarprodukt:

$$< x_{ij}, h_{ij} > := \begin{cases} h_{ij} - C_{ij}^{+}(x_{ij}) & \text{falls } C_{ij}^{+}(x_{ij}) < h_{ij} \\ C_{ij}^{-}(x_{ij}) - h_{ij} & \text{falls } C_{ij}^{-}(x_{ij}) > h_{ij} \\ 0 & \text{falls } C_{ij}^{-} \leq h_{ij} \leq C_{ij}^{+}(x_{ij}) \end{cases} \tag{B.29}$$

$$< x, h > := \max_{(ij) \in A(N)} < x_{ij}, h_{ij} > \tag{B.30}$$

$< x, h >$ ist damit der Absolutwert der negativsten reduzierten Kosten für x und h. Ferner gilt natürlich $< x, h > \geq 0$. $< x, h >$ können wir uns dabei in etwa vorstellen wie das duale Analogon zur Kilter-Zahl aus dem Balanced Out-of-Kilter Algorithmus. Wir bezeichnen $< x, h >$ hier als die *Kostendistanz* von x zu h.
Gilt $< x, h >= 0$, so ist x optimal.

B.2 Die Idee der Cancel and Tighten Methode

Die, von Goldberg und Tarjan in ihrer Arbeit vorgeschlagene, Methode des Min Mean Cycle Cancelling versucht nun $\lambda(x)$ direkt zu berechnen und dann ähnlich wie in unserer Idee zum Algorithmus von Anstee diese Kreise durch Flusserhöhungen aus der Lösung zu „canceln“. Das Auffinden der Kreise ist allerdings aufwändig [38, S. 1260].
Anstatt $\lambda(x)$ explizit wie im Min Mean Cycle Cancelling zu berechnen, wollen wir uns daher mit einem dualen Vektor $h \in L^{\perp}$ begnügen, der die Optimalitätsbedingungen näherungsweise erfüllt. Wir werden sehen, dass wir in der Tat die Min Mean Cycles dann nie berechnen müssen.
Die Idee der Cancel and Tighten Methode ist dabei folgende: In der ersten Phase des Algorithmus, der sogenannten Cancel-Phase versuchen wir die primale Lösung x zu verbessern, während wir in der zweiten Phase, der sogenannten Tighten-Phase versuchen die duale Lösung h zu verbessern. Die Cancel-Phase beruht dabei auch auf der Idee des Cycle Cancelling. Wir wollen hier aber nur auf den Kreisen den Fluss erhöhen, die „weit genug“ von der Optimalität entfernt sind.
Betrachten wir die Ideen für die Cancel and Tighten Methode nun genauer. Zunächst benötigen wir natürlich auch für ein $h \in L^{\perp}$, das die Optimalitätsbedingungen näherungsweise erfüllt, einen Parameter wie $\lambda(x)$. Wir definieren daher $\bar{\lambda} =< x, h >$. Damit ist $\bar{\lambda}$ eine obere Grenze für $\lambda(x)$ und erfüllt damit nach dem MMCM-Lemma auch die Bedingungen:

$$C_{ij}^{+}(x_{ij}) + h_{ij} \geq -\bar{\lambda} \; \forall (ij) \in A(N) \tag{B.31}$$

$$-h_{ij} - C_{ij}^{-}(x_{ij}) \geq -\bar{\lambda} \; \forall (ij) \in A(N) \tag{B.32}$$

Als nächstes wollen wir darauf eingehen, was „weit genug von der Optimalität entfernt“ bedeuten soll. Wir führen dazu zunächst einige Begriffe ein:

Definition B.6 (positiv und negativ nah)**.** *Wir nennen eine Kante $(ij) \in A(N)$ positiv (bzw. negativ) nah zu einem Vektor $h \in L^{\perp}$, falls $C_{ij}^{+}(x_{ij}) - h_{ij} \geq -\frac{\bar{\lambda}}{2}$ (bzw. $h_{ij} - C_{ij}^{-}(x_{ij}) \geq -\frac{\bar{\lambda}}{2}$). Sonst nennen wir die Kante positiv (bzw. negativ) fern.*

Damit haben wir einen Begriff für die Nähe zur Optimalität. Wir bezeichnen die positiv fernen Kanten mit P^{far} und die negativ fernen Kanten mit N^{far}. Einen Kreis $F = (A, B)$ mit $A \subseteq P^{far}$ und $B \subseteq N^{far}$ nennen wir einen fernen Kreis. Die Elemente nah zu h fassen wir in der Menge $Cl = A(N) - (P^{far} \cup N^{far})$ zusammen.
Der Cancel-Schritt besteht nach unserer Beschreibung aus einer Flusserhöhung auf den fernen Kreisen. Dabei sind die reduzierten Kosten für die Kanten aus A negativ und für die Kanten aus B positiv. Wir müssen also den Fluss auf den Kanten in A erhöhen und auf den Kanten in B verringern, um die Complementary Slackness Optimality zu erreichen. Wir wählen also für jede Kante zunächst ϵ_{ij} zunächst so, dass $C_{ij}^{-}(x_{ij} \pm \epsilon_{ij}) \leq h_{ij} \leq C_{ij}^{+}(x_{ij} \pm \epsilon_{ij})$ (hierbei steht das $+$ für die Kanten aus A und das $-$ für die Kanten aus B). Mit entsprechenden Suchalgorithmen lässt sich die Bestimmung eines solchen ϵ effizient implementieren [38, S. 1254].
Die Erhöhung des Flusses auf einem fernen Kreis wählen wir damit natürlich als $\epsilon = \min\{\epsilon_{ij} \ : \ (ij) \in (A \cup B)\}$. Es gilt dann $\epsilon > 0$.
Als nächstes wollen wir zeigen, dass eine solche Cancel-Phase sinnvoll ist. Als erste Vorbereitung wollen wir folgendes Lemma beweisen:

Lemma B.7 (fallendes $\bar{\lambda}$)**.** *Wurde x' durch eine Iteration des Algorithmus aus x gewonnen, so gilt $\bar{\lambda}' = < x', h > \leq < x, h > = \bar{\lambda}$.*

Beweis. Wir zeigen dieses Lemma mit der Definition von $< x, h >$. Betrachten wir eine Kante $(ij) \in A(N)$. Sei für diese Kante $C_{ij}^{+}(x_{ij}) < h_{ij}$ (der Fall $C_{ij}^{-}(x_{ij}) > h_{ij}$ ist analog), dann gilt $h_{ij} - C_{ij}^{+}(x_{ij}) \geq h_{ij} - C_{ij}^{+}(x'_{ij})$, da $x'_{ij} > x_{ij}$ gilt. Damit gilt aber $< x_{ij}, h_{ij} > \geq < x'_{ij}, h_{ij} >$ für alle Kanten $(ij) \in A(N)$. Damit folgt dann $< x, h' > \leq < x, h >$ und damit die Behauptung. □

Damit kommen wir nun zu dem eigentlichen Lemma zum Cancel-Schritt:

Lemma B.8 (Far Cycle-Cancelling)**.** *Ein Cancel-Schritt erhält die Bedingungen aus dem MMCM Lemma (mit λ) und benötigt maximal m Iterationen.*

Beweis. Im ersten Schritt des Beweises weisen wir darauf hin, dass keine Flusserhöhung auf einem Kreis die Bedingungen aus dem MMCM Lemma mit $\bar{\lambda}^p$ verletzt.
1. Behauptung
Für alle Iterationen $p = 1, \cdots, m$, bei denen der Kreis $F^p = (A^p, B^p)$ gerade so gewählt ist, dass jede Kante $(ij) \in A^p$ positiv fern von h und jede Kante $(ij) \in B^p$ negativ fern von h ist, gilt mit dem Fluss x^p nach der Iteration:

$$C_{ij}^{+}(x_{ij}^{p}) - h_{ij} \geq -\bar{\lambda}^p \tag{B.33}$$

$$h_{ij} - C_{ij}^{-}(x_{ij}^{p}) \geq -\bar{\lambda}^p \tag{B.34}$$

Beweis. Der Beweis ist trivial, denn gerade so haben wir ja $\bar{\lambda}^p$ gewählt. □

Der nächste Schritt ist nicht so einfach, denn wir müssen noch zeigen, dass wir keine Kanten fern von h erzeugen.

Behauptung 2
In jeder Iteration p erzeugen wir keine neue Kante fern von h und mindestens eine neue Kante nah zu h.

Beweis. Wir zeigen zunächst, dass, falls $(ij) \in A(N)$ zu den positiv nahen Kanten P (bzw. den negativ nahen Kanten N) gehört, dies auch nach der Iteration der Fall ist.
Betrachten wir dazu eine positiv nahe Kante $(ij) \in A(N)$ (der Beweis für eine negativ nahe Kanten ist analog). Nun könnte (ij) nur dann positiv nah werden, falls $C^+_{ij}(x_{ij})$ fallen würde und damit $(ij) \in B^{p-1}$ gelten müsste. Dann ist aber (ij) negativ fern von h bei Iteration $(p-1)$ (also $-C^-_{ij}(x^{p-1}_{ij}) < -h_{ij} - \frac{\bar{\lambda}^{p-1}}{2}$), da wir ja nur negativ ferne Elemente in B^{p-1} haben.
Wir wissen, dass $\bar{\lambda}^{p-1} = \max_{(ij)\in A(N)}\{C^-_{ij}(x^{p-1}_{ij}) - h_{ij}\}$ gilt. Die Flusserhöhung wählen wir minimal, sodass auch nach der Flusserhöhung von x^{p-1}_{ij} zu x^p_{ij} gilt: $C^+_{ij}(x^p_{ij}) \geq h_{ij}\ \forall (ij) \in B^{p-1}$. Damit gilt:

$$\begin{aligned} C^+_{ij}(x^p_{ij}) &\geq h_{ij} && \text{(B.35)}\\ &= C^-_{ij}(x^{p-1}_{ij}) - (C^-_{ij}(x^{p-1}_{ij}) - h_{ij}) && \text{(B.36)}\\ &\geq C^-_{ij}(x^{p-1}_{ij} - \bar{\lambda}^{p-1} && \text{(B.37)}\\ &> h_{ij} + \frac{\bar{\lambda}^{p-1}}{2} - \bar{\lambda}^{p-1} && \text{(B.38)}\\ &= h_{ij} - \frac{\bar{\lambda}^{p-1}}{2} && \text{(B.39)} \end{aligned}$$

Damit ist (ij) nicht positiv fern von h nach Iteration p.
Wir wollen nun zeigen, dass in der Iteration p mindestens eine neue Kante nah zu h wird. Hierbei gilt wieder aufgrund der minimalen Erhöhung des Flusses, dass für mindestens eine Kante $(ij) \in A$ gilt $C^+_{ij}(x^p_{ij}) = h_{ij}$ oder für mindestens eine Kante $(ij) \in B$ gilt $C^-_{ij}(x^p_{ij}) = h_{ij}$. Betrachten wir den Fall $(ij) \in A$. Der Fall $(ij) \in B$ ist analog. In diesem Fall gilt $C^+_{ij} - h_{ij} = 0$ und damit natürlich auch $C^+_{ij}(x^p_{ij}) - h_{ij} > -\frac{\bar{\lambda}}{2}$. Damit ist (ij) nah zu h. □

Wir können natürlich nicht mehr ferne Kreise 'canceln' als Kanten verfügbar sind. Es kann damit nur m solche Iterationen geben. □

Wir wollen später darauf zurückkommen, wie wir ferne Kreise auf einem Netzwerk bestimmen. Wir wollen nun zunächst zur Tighten-Phase kommen.
Hier nehmen wir eine Anleihe aus den „inneren Punkt Methoden“ der linearen Programmierung [62, S. 190-194]. Wir suchen eine „Richtung zur Optimalität“. Wir definieren zunächst:

Definition B.9 (ausreichend verbessernde Richtung)**.** *Einen Vektor $d \in \mathbb{R}^{|A(N)|}$ mit einem Flussnetzwerk N nennen wir eine ausreichend verbessernde Richtung, falls folgende Bedingungen erfüllt sind:*

- *d ist ein ganzzahliger Vektor in $L^\perp$*

- *für alle Kanten* $(ij) \in A(N)$ *gilt* $|d_{ij}| \leq m-1$
- *für alle Kanten* $(ij) \in P^{far}$ *gilt* $d_{ij} \geq 1$
- *für alle Kanten* $(ij) \in N^{far}$ *gilt* $d_{ij} \leq -1$

Lemma B.10 (ausreichend verbessernde Richtung)**.** *Ist* d *eine ausreichend verbessernde Richtung und ersetzen wir* h *durch* $h' := h + \left(\frac{\bar{\lambda}}{2m}\right) d$*, dann wird* $\bar{\lambda}$ *mindestens reduziert auf* $\bar{\lambda}' := \left(1 - \frac{1}{2m}\right) \bar{\lambda}$*. Ferner ist* $\bar{\lambda}'$ *wieder eine obere Grenze für* $\lambda(x')$ *mit* h'*.*

Beweis. Wir verwenden im Beweis die Charakteristika (i)-(iv) aus der Definition.
Mit (i) gilt $h' \in L^{\perp}$.
Zeigen wir nun mit (ii) - (iv), dass $\bar{\lambda}'$ wieder eine obere Grenze für $\lambda(x')$ mit h' ist.
Zunächst betrachten wir Kanten (ij), die nicht positiv fern von h sind. Dann gilt:

$$\begin{aligned} C_{ij}^{+}(x_{ij}) + h'_{ij} &= C_{ij}^{+}(x_{ij}) + h_{ij} + d_{ij}\frac{\bar{\lambda}}{2m} && \text{(B.40)} \\ &\geq C_{ij}^{+}(x_{ij}) + h_{ij} - (m-1)\frac{\bar{\lambda}}{2m} && \text{(B.41)} \\ &= C_{ij}^{+} + h_{ij} - \frac{\bar{\lambda}}{2} + \frac{\bar{\lambda}}{2m} && \text{(B.42)} \end{aligned}$$

Da (ij) nicht positiv fern von h ist, gilt $C_{ij}^{+}(x_{ij}) + h_{ij} \geq -\frac{\bar{\lambda}}{2}$. Damit erfüllen die Kanten $(ij) \in A(N)$, die nicht positiv fern von h sind, die Bedingung $C_{ij}^{+}(x_{ij}) + h_{ij} \geq -\bar{\lambda}$ aus dem MMCM-Lemma. Man kann dies analog für die Kanten $(ij) \in A(N)$, die nicht negativ fern von h sind und die Bedingung $-h_{ij} - C_{ij}^{-}(x_{ij}) \geq -\bar{\lambda}$ aus dem MMCM Lemma zeigen.
Betrachten wir nun also noch die übrigen Fälle. Sei $(ij) \in P^{far}$, dann gilt:

$$\begin{aligned} C_{ij}^{+}(x_{ij}) + h'_{ij} &= C_{ij}^{+}(x_{ij}) + h_{ij} + d_{ij}\frac{\bar{\lambda}}{2m} && \text{(B.43)} \\ &\geq C_{ij}^{+}(x_{ij}) + h_{ij} + \frac{\bar{\lambda}}{2m} && \text{(B.44)} \end{aligned}$$

Damit erfüllen auch die Kanten $(ij) \in P^{far}$ die erste Bedingung aus dem MMCM-Lemma. Den Fall $(ij) \in N^{far}$ kann man wieder analog behandeln. □

Wir haben damit gezeigt, dass mit einer „ausreichend verbessernde Richtung“ die duale Lösung verbessert werden kann. Wir wollen nun etwas genauer darauf eingehen, wie wir eine ausreichend verbessernde Richtung bestimmen können.
Wir notieren dazu zunächst, was wir in einer Cancel-Phase tun, etwas anders. Wir bestimmen einen Kreis nur aus fernen Kanten. Dieser muss eine Lösung z minimalen Trägers des folgenden Ungleichungssystems sein:

$$\begin{aligned} Ez &= 0 && \text{(B.45)} \\ z_{ij} &\geq 0 \text{ für } (ij) \in P^{far} && \text{(B.46)} \\ z_{ij} &\leq 0 \text{ für } (ij) \in N^{far} && \text{(B.47)} \\ z_{ij} &= 0 \text{ für } (ij) \in Cl && \text{(B.48)} \end{aligned}$$

Die Anleihen aus der linearen Programmierung können wir nun noch etwas weiter treiben, indem wir Farkas' Lemma [48] benutzen (siehe auch [62, S. 89-90]).

Lemma B.11. *Eine Kante* $\widehat{(ij)} \in P^{far} \cup N^{far}$ *ist in keinem fernen Kreis enthalten, genau dann wenn es einen Co-Kreis* $\tilde{\xi} \in L^{\perp}$ *mit* $\tilde{\xi}_{\widehat{(ij)}} \neq 0$ *gibt, so dass* $\tilde{\xi} \geq 0$ *für alle* $(ij) \in P^{far}$ *und* $\tilde{\xi}_{ij} \leq 0$ *für alle* $(ij) \in N^{far}$.

Beweis. Wir wenden Farkas' Lemma auf unser vorheriges Ungleichungssystem an, wobei wir zusätzlich $z_{\widehat{ij}} \neq 0$ fordern. Dann ist $\widehat{(ij)}$ genau dann nicht in einem fernen Kreis, falls ein Vektor $\xi = y^T E$ existiert, so dass:

$$\xi_{ij} \geq 0 \; \forall (ij) \in P^{far} \tag{B.49}$$

$$\xi_{ij} \leq 0 \; \forall (ij) \in N^{far} \tag{B.50}$$

$$\xi_{\widehat{ij}} \neq 0 \tag{B.51}$$

Hat nun dieses Ungleichungssystem eine Lösung in $L^{\perp}$, so ist die Lösung offensichtlich ein Co-Kreis $\tilde{\xi}$.
Umgekehrt erfüllt jedes $\tilde{\xi}$ auch das Ungleichungssystem und beweist damit wieder mit Farkas' Lemma, dass kein ferner Kreis $\widehat{(ij)}$ enthalten kann. □

Um nun eine ausreichend verbessernde Richtung zu bestimmen, bestimmen wir eine Teilmenge B^{far} von $P^{far} \cup N^{far}$, so dass die Menge

$$\begin{aligned}\{\tilde{\xi}^{lk} \mid \tilde{\xi}^{lk} \;&\text{ ist ein Co-Kreis und enthält eine Kante} \\ &(lk) \in B^{far} \;\text{ , die in keinem fernen Kreis ist}\}\end{aligned} \tag{B.52}$$

eine Basis von

$$\begin{aligned}\{\tilde{\xi}^{lk} \mid \tilde{\xi}^{lk} \;&\text{ ist ein Co-Kreis und enthält eine Kante} \\ &(lk) \in (N^{far} \cup P^{far}) \;\text{ , die in keinem fernen Kreis ist}\}\end{aligned} \tag{B.53}$$

bildet. Wir berechnen dann $d = \sum_{(lk) \in B^{far}} \tilde{\xi}^{lk}$.
Wir müssen nun natürlich noch zeigen, dass d auch wirklich eine ausreichend verbessernde Richtung ist.

Lemma B.12. *Das wie obig berechnete* d *ist eine ausreichend verbessernde Richtung.*

Beweis. Wir beweisen, dass die verschiedenen Annahmen in der Definition einer ausreichend verbessernden Richtung, zutreffen.
d ist die Summe von Inzidenzvektoren von Co-Kreisen und erfüllt damit sicherlich (i).
B^{far} ist die Basis eines Untervektorraums von $L^{\perp}$. Maximal kann B^{far} also $m-1$ Elemente enthalten (die Dimension von $L^{\perp}$). Damit gilt sicherlich auch $|d_{ij}| \leq m-1$, da ein Inzidenzvektor nur Einträge $\{0, \pm 1\}$ besitzt. Damit haben wir (ii).
Betrachten wir nun eine Kante $(ij) \in (P^{far} \cup N^{far})$. Mit dem vorherigen Lemma gilt $\tilde{\xi}^{lk}_{ij} \geq 0$ für $(ij) \in P^{far}$ und $\tilde{\xi}^{lk}_{ij} \leq 0$ für $(ij) \in N^{far}$. Da wir die Menge der $\{\tilde{\xi}^{lk} \; : \; (lk) \in B^{far}\}$ als Basis von $\{\tilde{\xi}^{lk} \; : \; (lk) \in (P^{far} \cup N^{far})\}$ definiert hatten und d als Summe geschrieben hatten, gilt damit auch $d_{ij} \geq 0$ für $(ij) \in P^{far}$ und $d_{ij} \leq 0$ für $(ij) \in N^{far}$ und damit (iii) und (iv). □

B.3 Implementierung der Methode

Bisher haben wir vor allem über den Nutzen von fernen Kreisen gesprochen. Wir wollen nun zumindest andeuten, wie wir ferne Kreise auch finden können.
Wir definieren uns dafür zunächst mit den Kanten von P^{far} und den gedrehten Kanten von N^{far} einen sogenannten *Ferngraphen*. Ferne Kreise entsprechen dann gerade den Kreisen auf diesem Ferngraphen. Wir drehen hierbei die Kanten aus N^{far} um, damit auch alle Kanten im Kreis die gleiche Richtung besitzen, ganz analog zum Balanced Out-of-Kilter Algorithmus.
Die Idee ist nun mit einer Tiefensuche den Ferngraphen zu untersuchen. Hierbei geht die Suche nur entlang von Kanten des Ferngraphen. Wird ein neuer Knoten gefunden wird dieser mit einer fortlaufenden Markierung $l_i,\ i \in \mathbb{N}$ versehen. Wird ein besuchter Knoten erneut gefunden, so haben wir einen fernen Kreis gefunden.
Zur Berechnung der Kreise haben Sleator und Tarjan [64] eine spezielle Datenstruktur für Bäume entwickelt (dynamic trees). Mit dieser Datenstruktur haben Goldberg und Tarjan [31, S. 881] einen Algorithmus entwickelt, der das Auffinden eines Kreises und die Flusserhöhung in $O(\log n)$ erreicht. Mit dem Far-Cycle-Lemma hat die Cancel-Phase damit eine Gesamtkomplexität von $O(m \log n)$.
Die Tighten Phase kann man mit den Markierungen sogar noch schneller implementieren. Wir betrachten die Markierungen im nach der Cancel-Phase verbleibenden Graphen. Wir definieren $\Delta l_{ij} := l_j - l_i$. Dann gilt $-(m-1) \leq \Delta l_{ij} \leq m-1$. Da die Knoten in den fernen Kreisen jeweils in Richtung der positiven Kanten (die negativen hatten wir gedreht) durchsucht werden, gilt sogar $\Delta l_{ij} \geq 0$ für $(ij) \in P^{far}$ und $\Delta_{ij} \leq 0$ für $(ij) \in N^{far}$. Die Δl_{ij}'s sind damit die Einträge einer ausreichend verbessernden Richtung [31, S.881-882]. Damit kann die Tighten-Phase sogar in $O(n)$ Zeit beendet werden.
Wir erhalten damit für Cancel- und Tighten-Phase eine Gesamtlaufzeit von $O(m \log n)$.

B.4 Ganzzahlige Lösungen

Bisher haben wir noch die Ganzzahligkeitbedingung vernachlässigt. Dies wollen wir nun nachholen und damit die Beschreibung des Algorithmus komplettieren.
Betrachten wir dazu unsere linearisierte Kostenfunktion $C_{lin}(x) = \sum_{(ij)\in A(N)} \sum_{k=1}^{p_{ij}} c_{ij}^k y_{ij}^k$ aus unserer Betrachtung konvexer Kosten für den primal-dualen Algorithmus. Diese Kostenfunktion verändert gemäß unserer Annahme im primal-dualen Algorithmus ihre Steigung jeweils bei ganzzahligen Werten. Da wir im Algorithmus den Fluss jeweils so erhöhen, dass wir uns auch dual verbessern, muss folglich jede Flusserhöhung größer als 1 sein, wenn der Fluss ganzzahlig ist, denn ansonsten sind wir optimal. Da wir aber nur minimal erhöhen, erhöhen wir auch den Fluss folglich immer nur ganzzahlig und erhalten damit nach jedem Erhöhungsschritt eine ganzzahlige Lösung [38, S. 1262].
Ferner haben wir gezeigt, dass eine ganzzahlige Lösung des Problems mit der linearisierten Kostenfunktion auch die optimale, ganzzahlige Lösung mit der ursprünglichen Kostenfunktion darstellt. Wir können damit o.B.d.A. die obige linearisierte Kostenfunktion verwenden.
Unser erster Ansatz um nun das Problem mit der linearisierten Kostenfunktion zu lösen

mag dabei zunächst widersinnig anmuten, wir werden aber sehen, dass wir diesen in der Tat sinnvoll verwenden können.
Wir wollen nämlich annehmen, es gelte $c_{ij}^k \in \mathbb{Z}\ \forall (ij) \in A(N)\ \forall k \in \{1, \cdots, p_{ij}\}$. Wir beweisen nun das folgende wichtige Lemma für diesen Fall:

Lemma B.13. *Wir nehmen an, unser Problem habe eine linearisierte Kostenfunktion mit ganzzahligen Steigungen c_{ij}^k wie in obiger Diskussion. Gilt dann $\bar{\lambda} < \frac{1}{n}$, so ist die Lösung die exakte optimale Lösung.*

Beweis. Sei $h \in L^\perp$ ein dualer Vektor und $\bar{\lambda} = < h, x > < \frac{1}{n}$. Dann gilt mit dem MMCM Lemma, dass die reduzierten Kosten für jede Kante in einem fernen Kreis minimal $-\frac{1}{n}$ sein können. Damit gilt aber:

$$\begin{aligned} c(x, F) &= c^+(A) - c^-(B) && \text{(B.54)} \\ &> -\frac{|F|}{n} \geq -1 && \text{(B.55)} \end{aligned}$$

$c(x, F)$ muss aber eine ganze Zahl sein, denn die reduzierten Kosten sind nach Annahme über die Kostenfunktion immer ganzzahlig. Damit gilt also $c(x, F) \geq 0$ und damit ist die Lösung optimal. □

Betrachten wir nun noch kurz die Laufzeit des Algorithmus. Mit dem Lemma über die ausreichend verbessernde Richtung wissen wir, dass wir in jedem Tighten-Schritt eine neue obere Grenze der reduzierten Kosten $\bar{\lambda}' = (1 - \frac{1}{2m})\bar{\lambda}$ erreichen. Betrachten wir die maximalen Kosten C einer Kante, so gilt auch $C_{ij}^+(x_{ij}) \leq C$. Anfangs können wir, da wir positive c_{ij}^k's angenommen hatten $h_{ij} = 0$ wählen. Damit erhalten wir $\bar{\lambda} \leq C$. Wir können also die Anzahl der benötigten Iterationen q bestimmen aus:

$$\begin{aligned} \frac{1}{n} &= \left(1 - \frac{1}{2m}\right)^q C && \text{(B.56)} \\ \frac{1}{nC} &= \left(1 - \frac{1}{2m}\right)^q && \text{(B.57)} \\ -\log(nC) &= q \cdot \log\left(1 - \frac{1}{2m}\right) && \text{(B.58)} \\ q &= -\frac{\log(nC)}{\log\left(1 - \frac{1}{2m}\right)} \leq \log(nC) && \text{(B.59)} \end{aligned}$$

Damit erhalten wir eine Laufzeit des Algorithmus von $O(m \log(n) \log(nC))$.
Nun haben wir aber keine ganzzahligen c_{ij}^k's, sondern wir haben eine linearisierte Kostenfunktion, bei der jeweils die Übergänge zwischen verschiedenen Steigungen bei ganzzahligen Werten liegen.
Die Lösung des Problems liegt in unseren vorherigen Betrachtungen zur Dualität im Balanced Relaxation Algorithmus. Dort hatten wir gezeigt, dass das duale Problem eine ganz ähnliche Struktur hat:

$$\min_{h \in L^\perp} \sum_{(ij) \in A(N)} G_{ij}(h_{ij}) \qquad \text{(B.60)}$$

Interessant ist nun zu betrachten, welche Eigenschaften $G_{ij}(h_{ij})$ besitzt, wenn $C_{ij}(x_{ij})$ stückweise linear ist. Im Balanced Relaxation Algorithmus hatten wir gezeigt, dass G_{ij} die konjugierte Funktion zu C_{ij} ist:

$$C_{ij}(x_{ij}) = \sup_{h_{ij} \in L^{\perp}} \{h_{ij}x_{ij} - G_{ij}(h_{ij})\} \tag{B.61}$$

Ist also $C_{ij}(x_{ij})$ stückweise linear, so muss es $G_{ij}(h_{ij})$ auch sein.
Da $G_{ij}(h_{ij})$ aber nur eine Funktion von h_{ij} ist, müssen die Steigungen der Geradenstücke von $G_{ij}(h_{ij})$ gerade den Abszissenwerten entsprechen, bei denen $C_{ij}(x_{ij})$ seine Steigung ändert [38, S. 1271]. Damit ist das duale Problem gerade ein Problem, dass wir aufgrund des obigen Lemmas mit der Cancel and Tighten Methode behandeln können. Die maximalen Kosten im dualen Problem entsprechen gerade der maximalen Kapazität im Ausgangsproblem und wir erhalten die optimale Lösung in einer Laufzeit $O(m \log(n) \log(nU))$.

B.5 Ideen für einen Algorithmus auf schiefsymmetrischen Netzwerken

Um den Balanced Cancel and Tighten Algorithmus für schiefsymmetrische Netzwerke verwenden zu können, müssen wir offensichtlich den Cancel-Schritt und den Tighten-Schritt symmetrisieren.
Im Cancel-Schritt müssen wir dazu gültige Kreise finden. Hierzu müsste der Algorithmus von Goldberg und Tarjan zum Auffinden ferner Kreise so verändert werden, dass gültige Kreise gefunden werden. Es ist hierbei zu klären, in wie weit die Datenstruktur der dynamic trees dies erlaubt. Die Möglichkeit zur Suche gültiger Kreise auf dem Ferngraphen ergibt sich hierbei wieder aus dem Balanced Decomposition Satz.
Haben wir eine effiziente symmetrische Cancel-Phase implementiert, so ist die Tighten-Phase einfach zu symmetrisieren, da die Markierungen Δl_{ij} dann bereits symmetrisch sein müssen.
Insgesamt glaubt der Autor, dass eine symmetrische Implementierung der Cancel and Tighten Methode auf schiefsymmetrischen Netzwerken möglich ist und eine ähnliche Komplexität wie für den Fall „normaler“ Netzwerke erreicht werden kann.

Literaturverzeichnis

[1] Edward Abbey. *Desert Solitaire*. Ballantine Books, 1985.

[2] David P. Ahlfeld, John M. Mulvey, Ron S. Dembo, and Stavros A. Zenios. Nonlinear programming on generalized networks. *ACM Transactions on Mathematical Software*, 13(4):350–367, December 1987.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[4] Richard P. Anstee. An algorithmic proof of Tutte's f-factor theorem. *J. Algorithms*, 6(1):112–131, 1985.

[5] Nicola Apollonio and András Sebö. Minsquare factors and maxfix covers of graphs. In George L. Nemhauser and Daniel Bienstock, editors, *IPCO*, volume 3064 of *Lecture Notes in Computer Science*, pages 388–400. Springer, 2004.

[6] Nicola Apollonio and András Sebö. Minconvex factors of prescribed size in graphs. *Les Cahiers du Laboratoire Leibniz*, 2006.

[7] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Science*, 43:842–844, September 1957.

[8] Annabell Berger. Minkonvexe Faktoren fixer Kantenzahl. Master's thesis, Fernuniversit

[9] Annabell Berger and Winfried Hochst

[10] D. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[11] Dimitri P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14:105–123, 1988.

[12] Dimitri P. Bertsekas and Didier El Baz. Distributed asynchronous relaxation methods for convex network flow problems. *SIAM Journal on Control and Optimization*, 25(1):74–85, 1987.

[13] Dimitri P. Bertsekas, Partick A. Hossein, and Paul Tseng. Relaxation methods for network flow problems with convex arc costs. *SIAM J. Control Optim.*, 25(5):1219–1243, 1987.

[14] Dimitri P. Bertsekas and Paul Tseng. A epsilon-relaxation method for generalized separable convex cost network flow problems. In *IPCO*, pages 85–93, 1996.

[15] U. Blasum, W. Hochstättler, C. Moll, and H. Rieger. Using network-flow techniques to solve an optimization problem from surface-physics. *J. Phys. A*, 29:L459, 1996.

[16] L.B. Cebik. The balanced-l network, http://www.cebik.com/link/l-bal.html, 2002.

[17] P.M. Chaikin and T.C. Lubensky. *Principles of Condensed Matter Physics*. Cambridge University Press, 1995.

[18] Ingo Deppner. private communication, 2007.

[19] Jack Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[20] Lester R. Ford, Jr. and Delbert R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the Hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218, 1957.

[21] Christian Fremuth-Paeger. *Degree Constrained Subgraph Problems and Network Flow Optimization*. Friedrich Pukelsheim, Augsburg, 1997.

[22] Christian Fremuth-Paeger and Dieter Jungnickel. Balanced network flows. I. A unifying framework for design and analysis of matching algorithms. *Networks*, 33(1):1–28, 1999.

[23] Christian Fremuth-Paeger and Dieter Jungnickel. Balanced network flows. III. Strongly polynomial augmentation algorithms. *Networks*, 33(1):43–56, 1999.

[24] Christian Fremuth-Paeger and Dieter Jungnickel. Balanced network flows. V. Cycle-canceling algorithms. *Networks*, 37(4):202–209, 2001.

[25] Christian Fremuth-Paeger and Dieter Jungnickel. Balanced network flows. VI. Polyhedral descriptions. *Networks*, 37(4):210–218, 2001.

[26] Christian Fremuth-Paeger and Dieter Jungnickel. Balanced network flows. VII. Primal-dual algorithms. *Networks*, 39(1):35–42, 2002.

[27] D. R. Fulkerson. An out-of-kilter method for minimal-cost flow problems. *Journal of the Society for Industrial and Applied Mathematics*, 9(1):18–27, 1961.

[28] M.J. Todd G.L. Nemhauser, A.H.G. Rinooy Kan, editor. *Volume 1 Optimization*. Handbooks in Operations Research and Management Science. Elsevier Science B.V., Amsterdam, third edition, 1998.

[29] Goldberg and Karzanov. Path problems in skew-symmetric graphs. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[30] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.

[31] Michel Gondran and Michel Minoux. *Graphs and Algorithms*. John Wiley and Sons Inc., 1984.

[32] Robert B. Gramacy. Shortest path and network flow algorithms for electrostatic discharge analysis. Master's thesis, University of California Santa Cruz, 2001.

[33] Alexander K. Hartmann and Heiko Rieger. *Optimization Algorithms in Physics*. Wiley VCH, 2001.

[34] Dieter Jungnickel. *Graphs, Networks and Algorithms (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[35] David Karger. Lecture notes on advanced algorithms - minimum cost flow problem, 1999.

[36] Alexander V. Karzanov and S. Thomas McCormick. Polynomial methods for separable convex optimization in unimodular spaces. In *SODA*, pages 78–87, 1995.

[37] Alexander V. Karzanov and S. Thomas McCormick. Polynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM J. Comput.*, 26(4):1245–1275, 1997.

[38] John G. Klincewicz. A newton method for convex separable network flow problems. *Networks*, 13(3):427–442, 1983.

[39] William Kocay and Douglas Stone. Balanced network flows. *Bulletin of the Institute of Combinatorics and its applications*, 7:17–32, June 1992.

[40] William Kocay and Douglas Stone. An algorithm for balanced flows. *Journal of Combinatorical Mathematics and Combinatorical Computing*, 19:3–31, 1995.

[41] Dénes König. Graphs and matrices (auf ungarisch). *Matematikaizs Fizikai Lapok*, 38:116–119, 1931.

[42] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer-Verlag, 2000.

[43] I. A. Kurkova. A load-balanced network with two servers. *Queueing Syst. Theory Appl.*, 37(4):379–389, 2001.

[44] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, Winston, 1976.

[45] Philipp Neuner Martin Gruber and Jacob Puchinger. Klassifikation der Min Cost Flow algorithmen http://www.ads.tuwien.ac.at/teaching/ws04/algograph/solutionu2a5.pdf, 2004.

[46] Silvio Micali and Vijay V. Vazirani. An o(sqrt(abs(v)) abs(e)) algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27. IEEE, 1980.

[47] Hermann Minkowski. *Geometrie der Zahlen*, volume 1. Teil, Erste Lieferung. Teubner Verlag, Leipzig, 1896.

[48] Michel Minoux. Solving integer minimum cost flows with separable convex cost objective polynomially. *Mathematical Programming Study*, 26:237–239, 1986.

[49] Michel Minoux. *Mathematical programming.* John Wiley and Sons, West Sussex, England, 1995.

[50] Robert Z. Norman and Michael O. Rabin. An algorithm for a minimum cover of a graph. *Proceedings of the American Mathematical Society*, 10(2):315–319, April 1959.

[51] James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 377–387, New York, NY, USA, 1988. ACM Press.

[52] Sriram Pemmaraju and Steven Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Cambridge University Press, 2003.

[53] J. Petersen. Die theorie der regulären graphen. *Acta Mathematica*, 15:193–220, 1891.

[54] Frank Pfeiffer and Heiko Rieger. Dislocations in the ground state of the solid-on-solid model on a disordered substrate. *Journal of Physics A*, 2489, 2000.

[55] Yaron Pinto and Ron Shamir. Efficient algorithms for minimum-cost flow problems with piecewise-linear convex costs. *Algorithmica*, 11(3):256–276, 1994.

[56] H. Rieger and U. Blasum. Ground state properties of solid-on-solid models with disordered substrates. *Physical Review B*, 55:7394R, 1997.

[57] R. Tyrell Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

[58] R. Tyrell Rockafellar. *Network Flows and Monotropic Optimization.* Pure and Applied Mathematics. John Wiley and Sons Inc., Princeton, New Jersey, 1984.

[59] Yasser Roudi and Peter E. Latham. A balanced memory network, 2007.

[60] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[61] Alexander Schrijver. *Combinatorical Optimization - Polyhedra and Efficiency.* Springer Verlag, Heidelberg, 2003.

[62] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122, New York, NY, USA, 1981. ACM Press.

[63] John Toner. Dirt roughens real sandpiles. *Phys. Rev. Lett.*, 66(6):679–682, Feb 1991.

[64] John Toner and D. P. DiVincenzo. Super-roughening: A new phase transition on the surfaces of crystals with quenched bulk disorder. *Phys. Rev. B*, 41(1):632–650, Jan 1990.

[65] Yan-Chr Tsai and Yonathan Shapir. Kinetic roughening in surfaces of crystals growing on disordered substrates. *ERRATUM-IBID.*, 69:3420, 1992.

[66] W. T. Tutte. The factorisation of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947.

[67] Philip Christian Weber. Minimum Cost Flow Grundlagen und erste algorithmen http://www.informatik.uni-trier.de/ naeher/professur/courses/ws2005/exercises/seminar/ausarbeitungweber.pdf, 2005.

Abbildungsverzeichnis

Tabellenverzeichnis