Diplomarbeit

zum Thema

Verfahren zum Kantenzusammenhang in gerichteten Graphen

von
Michael Bussieck

angefertigt am Institut für Mathematische Optimierung
der Technische Universität Carolo Wilhelmina zu Braunschweig
unter Anleitung von
Herrn Professor Dr. Uwe Zimmermann
Braunschweig, im Sommersemester 1992
Inhaltsverzeichnis

1 Einleitung 3

2 Grundlegende Sätze und Definitionen 5

3 Grundlegende Algorithmen 11
   3.1 Ein kürzeste-Wege-Algorithmus .................. 11
   3.2 Der maximale Fluß Algorithmus ................ 13
   3.3 Depth-First-Search ............................ 21
   3.4 Bestimmung starker Zusammenhangskomponenten .. 23

4 Kantenzerlegungszahl $\lambda_G$ 27
   4.1 Einfache Berechnung der Kantenzerlegungszahl .. 27
   4.2 Berechnung der Kantenzerlegungszahl nach Esfahanian und Ha-
        kimi ........................................ 30

5 Einführung Augmentierung 33
   5.1 Problemformulierung ............................ 33
   5.2 NP-vollständige Augmentierungsprobleme ......... 36

6 Augmentierung zu $\lambda_G = 1$ 39

7 Augmentierung gerichteter Graphen 46
   7.1 Der Satz von Frank ............................. 46
   7.2 Der Algorithmus von Frank ..................... 52
   7.3 Greedy-Algorithmus und Kontrapolymatroide .... 55
   7.4 Implementation des Algorithmus von Frank ....... 61
   7.5 Einfache Beispiele ............................. 70

8 Augmentierung von kreisfreien Graphen 75

9 Datenstrukturen und Rechenergebnisse 85
   9.1 Netzwerkgeneratoren ............................ 91
   9.2 Rechenergebnisse ............................. 93
Kapitel 1

Einleitung

Der Zusammenhang gehört zu den grundlegendsten und anschaulichsten Konzepten in der Graphentheorie. Viele Zusammenhangsbegriffe sind eingeführt worden (vgl. [PEROCHE83]), zu nennen sind hier als wichtigste Vertreter der $k$-fache Knotenzusammenhang (kurz $k$-facher Zusammenhang) und der $k$-fache Kantenzusammenhang. Letzterer wird in der vorliegenden Arbeit die zentrale Rolle spielen.


Ab Kapitel 5 beschäftigen wir uns mit Kantenzusammenhangsaugmentierungen. Wir wollen einen gegebenen Graphen durch Hinzunahme von Kanten so vergrößern, daß er gewissen Bedingungen bezüglich des Kantenzusammenhangs genügt.

Betrachten wir folgendes Beispiel:
Die Kapazität eines Telefonnetzes ist nicht mehr ausreichend. Die Telefongesellschaft beschließt, das Telefonnetz zu sanieren, indem sie die bestehenden Leitungen beibehält und noch weitere Verbindungen schafft. Die Bedarfsanalyse liefert die gewünschte (ganzzahlige) Kapazität der Leitungen zwischen je zwei Orten. Beim
Verlegen einer Leitung mit gewisser Kapazität zwischen den Orten A und B entstehen Kosten proportional zur Länge (oder anderen kostenrelevanten Parametern) und Kapazität der Leitung. Aufgabe ist es nun, das Telefonnetz (ganzzahlig) zu vergrößern, daß die geforderte Kapazität erreicht wird und die Gesamtkosten minimal bleiben.

In mathematischer Formulierung heißt das:
Gegeben ist ein (gerichteter) Graph $G = (V, E)$ und eine Bedarfsfunktion $r : V^2 \rightarrow \mathbb{R}_0^+$, weiterhin eine Kostenfunktion $a : V^2 \rightarrow \mathbb{R}_0^+$. Gesucht ist eine Kantenmenge $F \subseteq V^2$ (wir benutzen den Ausdruck Menge sowohl für Mengen als auch für Multi-mengen), so daß die Zahl der kantendisjunkten Wege von $u$ nach $v$ mit $u, v \in V$ in $G^* = (V, E \cup F)$ mindenstens so groß ist wie $r(u, v)$ und $\sum_{e \in F} a(e)$ minimiert wird.


Kapitel 2

Grundlegende Sätze und Definitionen

Ein *gerichteter Graph* ist ein Paar \( G = (V, E) \) einer endlichen Menge \( V \neq \emptyset \) und einer Teilmenge \( E \) von geordneten Paaren \((u, v)\) aus \( V \). Die Elemente aus \( V \) heißen *Knoten*, die Elemente aus \( E \) heißen *Kanten*. Die Mächtigkeit der Mengen \( V \) und \( E \) kürzen wir wie folgt ab: \(|V| = n \) und \(|E| = m \). Statt \((u, v)\) schreiben wir auch \( u \rightarrow v \) oder kurz \( e = uv \). \( v \) ist der Anfangsknoten von \( e \), \( w \) der Endknoten. Beide sind die *Terminalknoten* von \( e \). Die Knoten \( u \) und \( v \) sind mit \( e \) *inzident*. \( u \) und \( v \) heißen *adjazent*.

Ersetzen wir in \( G = (V, E) \) jede Kante \( uv \) durch das ungeordnete Paar \( \{u, v\} \), so erhalten wir den zugrundeliegenden ungerichteten Graphen. Ersetzen wir alle Kanten \( uv \in E \) durch \( vu \), so erhalten wir eine *Umorientierung* von \( G \).

Ein Graph \( \hat{G} = (V, \hat{E}) \) mit \( E \subseteq \hat{E} \) heißt *Obergraph* von \( G \). Ein Graph \( G = (V, E) \) mit \( E \subseteq \hat{E} \) und \( V \subseteq V \) heißt *Teilgraph* von \( G \). Ist \( V = V \), so heißt \( G \) *aufspannender Teilgraph* von \( G \).

Ein *Weg* von \( x_0 \) nach \( x_k \) mit \( x_0, x_k \in V \) ist eine Folge von Kanten aus \( E \) mit \( x_0 \rightarrow x_1, x_1 \rightarrow x_2, \ldots, x_{k-1} \rightarrow x_k \). Ein *Kreis* ist ein geschlossener Weg (d.h. \( x_0 = x_k \)).

Seien \( A, B, X \subseteq V \). \( E(X) := \{uv \in E | u, v \in X \} \) heißt die *von X induzierte Kantenmenge* in \( G \). Sei \( F \subseteq E \).

\[
N^X_F(X) := \{uv \in F | u \in X, \ v \notin X \}
\]

sind die Kanten, die den Anfangsknoten in \( X \) und den Endknoten nicht in \( X \) haben und aus der Menge \( F \) stammen.

\[
N^X_F(X) := \{uv \in F | v \in X, \ u \notin X \}
\]

sind die Kanten, die den Endknoten in \( X \) und den Anfangsknoten nicht in \( X \) haben und aus der Menge \( F \) stammen. Für \( F = E \) schreiben wir kurz \( N^X(X) \) und \( N^X(X) \).
Die Terminalknoten der Kanten aus $N^e(X)$ und $N^e(Y)$, die nicht in $X$ liegen, heißen auch die Nachbarn der Menge $X$.

$$\delta_F(X) := |N^e_F(X)| \quad \varrho_F(X) := |N^e_F(X)|$$

sind die Mächtigkeiten der entsprechenden Mengen.

$$d(A,B) := |\{uv \in E|u \in A - B, v \in B - A \lor u \in B - A, v \in A - B\}|$$

ist die Mächtigkeit der Kantenmenge mit Kanten zwischen Knoten aus $A - B$ und $B - A$.

$$\bar{d}(A,B) := |\{uv \in E|u \in A \cap B, v \in V - (A \cup B) \lor u \in V - (A \cup B), v \in A \cap B\}|$$

ist die Mächtigkeit der Kantenmenge mit Kanten zwischen Knoten aus $A \cap B$ und $V - (A \cup B)$.

Mit diesen Größen können wir folgendes Lemma formulieren:

**Lemma 2.1** Sei $G = (V,E)$ ein gerichteter Graph und seien $X,Y \subseteq V$. Dann gilt:

$$\delta(X) + \delta(Y) = \delta(X \cap Y) + \delta(X \cup Y) + d(X,Y)$$

und

$$\varrho(X) + \varrho(Y) = \varrho(X \cap Y) + \varrho(X \cup Y) + \bar{d}(X,Y)$$

Gilt zusätzlich $\delta(X \cap Y) = \varrho(X \cap Y)$, so folgt:

$$\delta(X) + \delta(Y) = \delta(X - Y) + \delta(Y - X) + \bar{d}(X,Y)$$

und

$$\varrho(X) + \varrho(Y) = \varrho(X - Y) + \varrho(Y - X) + \bar{d}(X,Y)$$

Beweis: Durch einfaches Abzählen der Kanten bekommen wir das Ergebnis. Wir beweisen die Gleichungen nur für $\varrho$. Die Gleichungen für $\delta$ folgen analog. Betrachten wir folgendes Bild:

![Diagram](image)

Abb. 2.1
Die Buchstaben bezeichnen die Mächtigkeit der entsprechenden Kantenmenge. (z.B. \( A = |\{uv \in E | u \in V - (X \cup Y), v \in X - Y\}|\)). Es gilt:

\[
\varrho(X) = A + C + F + G \quad \text{und} \quad \varrho(Y) = B + D + E + G
\]

\[
\varrho(X \cup Y) = A + B + G, \quad \varrho(X \cap Y) = E + F + G \quad \text{und} \quad \delta(X, Y) = C + D
\]

Addieren wir jeweils auf, so erhalten wir das gewünschte Ergebnis.

Zum Beweis der 2. Ungleichung betrachten wir das nächste Bild:

![Diagramm]

Abb. 2.2

Unter der obigen Voraussetzung gilt: \( E + F + G = H + I + J \). Weiterhin gilt:

\[
\varrho(X - Y) = A + C + H, \quad \varrho(Y - X) = B + D + I \quad \text{und} \quad \delta(X, Y) = G + J
\]

Summieren wir auf so erhalten wir:

\[
\varrho(X) + \varrho(Y) = A + B + C + D + E + F + G + G
\]

\[
= \varrho(X - Y) + \varrho(Y - X) + \delta(X, Y) + E + F + G - (H + I + J)
\]

\[
= \varrho(X - Y) + \varrho(Y - X) + \delta(X, Y)
\]

Eine Funktion \( b : 2^V \rightarrow R \) (\( 2^V \) bezeichnet die Potenzmenge von \( V \)) mit \( b(A) + b(B) \geq b(A \cap B) + b(A \cup B) \) heißt submodular. Eine Funktion \( p : 2^V \rightarrow R \) heißt supermodular, wenn \((-p)\) submodular ist. Mit dem obigen Lemma sind \( \varrho \) und \( \delta \) submodular.

Das 4-Tupel \((G, c, s, t)\) mit \( G = (V, E) \) gerichteter Graph, \( c : E \rightarrow R \) und zwei ausgezeichneten Knoten \( s, t \in V \) heißt Netzwerk. Die Funktion \( c \) heißt Kapazitätsfunktion.

Ein Knoten \( v \) in einem gerichteten Graph heißt
Für die Kanten von $T \subseteq V$ zu den Nachbarn schreiben wir statt $N^>(T)$ auch $(T, \bar{T})$ und nennen diese Mengen einen Schnitt, $\bar{T}$ steht für $V - T$. $(T, \bar{T})$ heißt \{X\}, \{Y\}–Schnitt, wenn $X \subseteq T$ und $Y \subseteq \bar{T}$. Einen Schnitt können wir sowohl in gerichteten Graphen als auch in Netzwerken betrachten. Unter dem Wert eines Schnitts $(T, \bar{T})$ verstehen wir im gerichteten Graphen die Mächtigkeit der Menge $(T, \bar{T})$ also $|(T, \bar{T})|$. In Netzwerken definieren wir den Wert $|(T, \bar{T})|_c := \sum_{uv \in (T, \bar{T})} c(uv)$. Es tauchen häufig Summationen über die Elemente einer Menge auf. Wir vereinbaren für eine Funktion $y : A \to B$ den Ausdruck $y(X)$ mit $X \subseteq A$ als $\sum_{a \in X} y(a)$. Wir können also für $|(T, \bar{T})|_c$ auch $c((T, \bar{T}))$ schreiben. Wir definieren:
\[ M(u, v) := \min (|(T, \bar{T})|, (T, \bar{T}) - u, v–Schnitt). \]

Mit $\lambda(u, v)$ bezeichnen wir die maximale Anzahl der kantendisjunkten Wege von $u$ nach $v$.

Mit diesen Voraussetzungen können wir den $k$–fachen Kantenzusammenhang definieren.

**Definition 2.1** Ein gerichteter Graph $G = (V, E)$ heißt genau dann $k$–fach kantenzusammenhängend, wenn
\[ M(u, v) \geq k \quad \forall u, v \in V \quad \text{mit} \quad u \neq v. \]
Für $k = 1$ heißt $G$ stark zusammenhängend.

Zwischen $\lambda(u, v)$ und $M(u, v)$ besteht ein enger Zusammenhang:

**Satz 2.2** Sei $G = (V, E)$ ein gerichteter Graph und seien $u, v \in V$. Dann gilt:
\[ \lambda(u, v) = M(u, v) \]

Dies ist der ganzzahlige Spezialfall des max-flow–min-cut Satzes von Ford-Fulkerson [FOR62]. Für diesen Satz brauchen wir eine Verallgemeinerung von $\lambda(u, v)$. Sei $(G, c, s, t)$ ein Netzwerk. Eine Funktion $f : E \to R^+_0$ mit
\begin{align}
0 \leq f(uv) \leq c(uv) & \quad \forall \quad uv \in E \\
\sum_{uv \in N^>(u)} f(uv) = \sum_{vu \in N^<(u)} f(vu) & \quad \forall \quad u \in V - \{s, t\}
\end{align}
heißt Fluß. Ein maximaler Fluß ist eine Funktion $f$ die den obigen Bedingungen genügt und $\sum_{sv \in N^>(s)} f(sv)$ maximiert. $\sum_{sv \in N^<(s)} f(sv)$ ist der Wert des Flusses.
Satz 2.3 Sei \((G, c, s, t)\) ein Netzwerk. Dann ist der Wert eines minimalen \(s,t\)-Schnitts gleich dem Wert eines maximalen Flusses.

Beweis: siehe z.B. [JUNGNICHEL87]

Der maximale Fluß in einem Netzwerk \((G, c, s, t)\) entspricht \(\lambda(s, t)\) in einem gerichteten Graphen.

Der \(k\)-fache Kantenzusammenhang kann auf verschiedene Weise definiert werden. Hier sind einige äquivalente Beschreibungen:

Lemma 2.4 Folgende Aussagen sind äquivalent:

1. \(G(V, E)\) ist \(k\)-fach kantenzusammenhängend.
2. \(\forall X \text{ mit } X \neq \emptyset, V \text{ gilt: } \delta(X) \geq k\).
3. \(\forall X \text{ mit } X \neq \emptyset, V \text{ gilt: } \varrho(X) \geq k\).
4. Für jedes Paar \((u, v)\) gilt: \(\lambda(u, v) \geq k\).

Beweis:

(1) \(\Rightarrow\) (2) Sei \(\emptyset \subset X \subset V\). Dann gibt es ein \(u \in X\) und ein \(v \in \bar{X}\). \((X, \bar{X})\) ist ein \(u, v\)-Schnitt. Nach (1) gilt

\[
\delta(X) = |(X, \bar{X})| \geq \min(||(T, \bar{T})||(T, \bar{T})) \text{ ist ein } u, v\text{-Schnitt} \geq k.
\]

(2) \(\Rightarrow\) (3) Sei \(\emptyset \subset X \subset V\). Dann gilt \(\emptyset \subset X \subset \bar{X} \subset V\). Nach (2) gilt

\[
\varrho(X) = \delta(\bar{X}) \geq k.
\]

(3) \(\Rightarrow\) (1) Seien \(u, v \in V\). Nach (2) gilt:

\[
k \leq \min(\varrho(\bar{X})|u \in X, v \notin X, X \subset V) = \min(|(X, \bar{X})||u \in X, v \notin X) = M(u, v)
\]

(1) \(\Leftrightarrow\) (4) Ist die Aussage von Satz 2.2. \(\square\)

Mit dem Lemma können wir die Kantenzusammenhangszahl \(\lambda_G\) wie folgt definieren:

\[
\lambda_G := \min(M(u, v)|u, v \in V, u \neq v)
\]

\[
= \min(\varrho(X)||\emptyset \subset X \subset V)
\]

\[
= \min(\delta(X)||\emptyset \subset X \subset V)
\]

\[
= \min(\lambda(u, v)|u, v \in V, u \neq v)
\]

Zum Schluß einige Notationen: \(A\) und \(B\) heißen \textit{sich schneidende Mengen}, wenn \(A \cap B \neq \emptyset\), \(A - B \neq \emptyset\) und \(B - A \neq \emptyset\). Sie heißen \textit{sich kreuzend}, wenn sie sich schneiden und \(V - (A \cup B) \neq \emptyset\) gilt.
Eine Familie \( \{X_1, \ldots, X_i\} \) von Teilmengen von \( V \) heißt Teilpartition, wenn die \( X_i \) paarweise disjunkt sind. Sie ist eine echte Teilpartition, wenn \( \{X_1, \ldots, X_i\} \neq \{V\} \) gilt.

Sei \( x \in \mathbb{R}^n \) ein Vektor, dann bezeichnet \( x[i] \in R \) die \( i \)-te Komponente des Vektors. Wir unterscheiden nicht zwischen der einelementigen Menge \( \{x\} \) und dem Element \( x \) selbst.

Im weiteren sprechen wir häufig von Graphen. Ist nicht ausdrücklich von einem ungerichteten Graphen die Rede, so meinen wir immer einen gerichteten Graphen.
Kapitel 3

Grundlegende Algorithmen

3.1 Ein kürzeste-Wege-Algorithmus

Wir wollen in diesem Abschnitt eine sehr einfache Methode zum Auffinden kürzester Wege beschreiben. Gegeben ist ein gerichteter Graph $G = (V, E)$ und zwei Knoten $s$ und $t$. Wir suchen einen Weg $s = v_0 \rightarrow v_1 \rightarrow v_2, \ldots, v_{l-1} \rightarrow v_l = t$ von $s$ nach $t$ mit minimaler Länge. Der Begriff Länge kann unterschiedlich definiert werden und führt zu unterschiedlichen Algorithmen. Sei $a : E \rightarrow R$ eine Längenfunktion. Die Länge eines Weges $v_0 \rightarrow v_1 \rightarrow v_2, \ldots, v_{l-1} \rightarrow v_l$ wird definiert als $\sum_{i=0}^{l-1} a(v_i, v_{i+1})$. Der Algorithmus von Dijkstra löst das Problem für $a \geq 0$ in $O(n^2)$. Für beliebiges $a$ und der zusätzlichen Voraussetzung, dass es keine negativen Kreise (Kreise deren Länge negativ ist) gibt, löst der Algorithmus von Bellman–Ford das Problem in $O(nm)$. Die genannten Algorithmen sind in [EVEN79] ausführlich beschrieben.

Wir werden hier für $a \equiv 1$ eine sehr einfache Methode kennenlernen. Der folgende Algorithmus wurde von Moore [MOORE57] vorgestellt und ist unter dem Namen Breadth First Search (BFS) bekannt. Wir suchen einen kürzesten Weg von $s$ nach $t$.

**BFS-Algorithmus**

1. **Initialisierung**
   
   $d(u) = \infty$ für alle $u \in V - s$, $d(s) = 0$, $i = 0$.

2. **Markiere alle Knoten $v$ mit $d(v) = \infty$, die mit einem Knoten $u$ benachbart sind, für den $d(u) = i$ gilt. Gibt es keinen solchen Knoten, dann STOP.**

3. **Setze $d(v) = i + 1$ für alle in Schritt (2) markierten Knoten $v$.**

4. **Lösch alle Markierungen. Setze $i = i + 1$ und gehe nach (2).**

Führen wir in Schritt (3) folgende Abfrage ein: 'Ist $d(t) < \infty$, dann STOP', so können wir uns die Markierung der restlichen Knoten sparen. Mit einer geeigneten
Datenstruktur (insbesondere Speicherung der Nachbarn eines Knotens) betrachten wir jede Kante genau einmal. Im Schritt 1 setzen wir für alle Knoten das Feld $d$. So erhalten wir die Komplexität $O(n + m)$ für das BFS-Verfahren. Zeigen wir als nächstes die Korrektheit.

**Satz 3.1** Nach Ablauf des BFS-Algorithmus ist $d(v)$ die Länge eines kürzesten Weges von $s$ nach $v$.

Beweis: Ist $d(v) = k$, so gibt es einen Weg von $s$ nach $v$ der Länge $k$. Diesen Weg erhält man wie folgt: Es gibt einen Knoten $v_{k-1}$ der mit $v = v_k$ benachbart ist und für den $d(v_{k-1}) = k-1$ gilt. Genauso gibt es einen Knoten $v_{k-i-1}$ der mit $v_{k-i}$ benachbart ist und für den gilt $d(v_{k-i-1}) = k - i - 1$ für $i = 0, 1, \ldots, k - 1$. Es gilt $v_0 = s$, da $s$ der einzige Knoten mit $d(v) = 0$ ist. $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots, v_{k-1} \rightarrow v_k = v$ ist ein Weg von $s$ nach $v$ der Länge $k$.

Zeigen wir nun noch die Minimalität des Weges. Annahme: Es existiert ein Weg $s = w_0 \rightarrow w_1, w_1 \rightarrow w_2, \ldots, w_{l-1} \rightarrow w_l = v$ mit $l < k$. Es gilt $d(w_i) \leq i$, da der Knoten spätestens entlang des Weges markiert wurde. Damit ist $d(v) \leq l < k$. Widerspruch. □


In einem Algorithmus aus Kapitel 8 wird es erforderlich sein, nicht nur einen Weg von $s$ nach $t$, sondern $k > 1$ kantendisjunkte Wege zu finden. Dazu muß natürlich gewährleistet sein, daß es überhaupt $k$ disjunkte Wege gibt. Durch wiederholte Anwendung des BFS-Verfahren erreichen wir die Lösung in einer Komplexität von $O(km)$.

Hat der Graph keine parallelen Kanten, so gilt $m < n^2$. Existieren parallele Kanten, so können wir eine Komplexität von $O(kn^2 + m)$ erreichen, indem wir in Schritt (2) nur einen Repräsentanten aus der Menge der parallelen Kanten wählen. Diese Erweiterung hat natürlich Auswirkungen auf die Datenstruktur. Wir können z.B. die parallelen Kanten zwischen zwei Knoten $u$ und $v$ in eine Liste eintragen und lassen den Algorithmus nur auf die Kante am Listenanfang zugreifen. Wird die Kante in einem kürzesten Weg benutzt, so entfernen wir sie vom Listenanfang (bzw. ganz aus der Liste). Eine andere parallele Kante nimmt ihren Platz am Listenanfang ein. So betrachtet der Algorithmus pro Aufruf höchstens $n(n - 1)$ Kanten. Zum Aufbau solcher Listen müssen wir jede Kante einmal betrachten und sie in eine der Listen eintragen. Damit erhalten wir für den Aufbau dieser Listen ein Komplexität von $O(m)$. Insgesamt erhalten wir die Komplexität $O(kn^2 + m)$.
3.2 Der maximale Fluss Algorithmus

In diesem Abschnitt wollen wir uns mit dem Problem der Bestimmung eines maximalen Flusses in einem Netzwerk \((G, c, s, t)\) beschäftigen. Die Bestimmung eines maximalen Flusses (genauer eines minimalen Schnitts) wird bei späteren Algorithmen ein notwendiges Hilfsmittel sein. Doch behandeln wir das maximale Flussproblem unabhängig von seinen vielen Anwendungsmöglichkeiten.

Wir suchen einen Fluß \(f : E \longrightarrow R_0^+\), der folgenden Bedingungen genügt:

\[
\begin{align*}
0 \leq f(uv) &\leq c(uv) \quad \forall \ uv \in E \\
\sum_{uv \in N^+(u)} f(uv) &\leq \sum_{vu \in N^+(u)} f(vu) \quad \forall \ u \in V - \{s, t\}
\end{align*}
\]

und der \(\sum_{uv \in N^+(s)} f(uv)\) maximiert.

Wir suchen also einen Fluß, der maximal viele Flußeinheiten von der Quelle \(s\) zur Senke \(t\) verschickt, ohne die Kapazitätsbeschränkungen der Kanten zu verletzen und bei dem aus jedem Knoten (abgesehen von der Quelle und der Senke) genau so viel hinein wie heraus fließt. Wir wollen uns hier nicht weiter mit der Flussetheorie beschäftigen, sondern einen effizienten Algorithmus und seine Implementation vorstellen.

Viele Algorithmen wurden zur Lösung des Problems vorgestellt. (Eine Übersicht findet man in [GOLDBERG90].) Hier betrachten wir den preflow-push-Algorithmus von Goldberg und Tarjan [GOLDBERG90]. Um den Algorithmus zu verstehen, benötigen wir folgende Begriffe:

**Definition 3.1** Eine Funktion \(f : E \longrightarrow R_0^+\) heißt preflow genau dann, wenn (3.1) und folgende Relaxation von (3.2) erfüllt sind:

\[
\sum_{uv \in N^+(u)} f(uv) \leq \sum_{vu \in N^+(u)} f(vu) \quad \forall \ u \in V - \{s, t\}
\]

Mit anderen Worten heißt das, es fließt nicht mehr aus einem Knoten heraus, als hinein fließt. \(G_f = (V, E_f)\) mit \(E_f = E_+ \cup E_-\), \(E_+ = \{uv | uv \in E \text{ und } f(uv) < c(uv)\}\), \(E_- = \{vu | uv \in E \text{ und } f(uv) > 0\}\) heißt Inkrementgraph zu dem Graphen \(G = (V, E)\) und dem preflow \(f\). Die Kanten aus \(E_+\) heißen Vorwärtskanten, die Kanten aus \(E_-\) heißen Rückwärtskanten.

\[
c_f(uv) = \begin{cases} 
  c(uv) - f(uv) & : uv \in E_+ \\
  f(vu) & : uv \in E_-
\end{cases}
\]

heißt Inkrementkapazitätsfunktion. Eine Funktion \(d : V \longrightarrow N_0\) heißt zulässiges labeling bezüglich eines preflows \(f\) genau dann, wenn \(d(t) = 0\) und

\[
\forall uv \in E_f \text{ mit } c_f(uv) > 0 \implies d(u) \leq d(w) + 1
\]
Der Algorithmus erhält als Startdaten ein zulässiges labeling $d$ und einen preflow $f$.

**Bemerkung 3.1** Sei $d$ ein zulässiges labeling, dann ist $d(v)$ eine untere Schranke für die Länge eines kürzesten Weges von $v$ nach $t$ im Inkrementgraphen (die Länge eines Weges sei hier die Anzahl seiner Kanten). Denn sei $v = v_1 \rightarrow v_2, v_2 \rightarrow v_3, \ldots, v_{k-1} \rightarrow v_k = t$ ein kürzester Weg von $v$ nach $t$ in $G_f$, so gilt nach (3.4) $d(v_{k-1}) \leq d(t) + 1$, $d(v_{k-2}) \leq d(v_{k-1}) + 1 = d(t) + 2$, ..., $d(v_1) \leq d(t) + k - 1$. Da $d(t) = 0$, folgt $d(v) \leq k - 1$.

Wir wählen einen zulässigen preflow $f$ wie folgt:

$$f(uv) = \begin{cases} c(uv) : & u = s \\ 0 : & u \neq s \end{cases}$$

Zu diesem preflow wählen wir ein passendes labeling:

$$d(u) = \begin{cases} n : & u = s \\ 0 : & u \neq s \end{cases}$$

**Bemerkung 3.2** Das oben beschriebene labeling wollen wir “einfaches labeling” nennen. Für eine effiziente Implementation lohnt es sich, das labeling möglichst günstig zu wählen. Also $d(v) = \text{Länge eines kürzesten Weges von } v \text{ nach } t$. Dieses können wir durch eine BFS-Suche rückwärts von der Senke $t$ aus realisieren. Dieses labeling heißt “exaktes labeling”. Die Wahl des preflows und damit auch des labeling ist etwas unnatürlich. Wir werden sehen, daß unsere Implementation (bei der highest-first-selection) beginnend mit dem 0-Fluß und $d \equiv 0$ nach der ersten Iteration die obige Situation liefert. Um die Beweise einfach zu halten, wählen wir aber die oben beschriebenen Startdaten.

**Definition 3.2** Die Funktion $ex : V \rightarrow R_0^+ \text{ mit}$

$$ex(u) = \begin{cases} \sum_{vu \in N^0(v)} f(vu) - \sum_{uv \in N^\delta(u)} f(uv) & u = s \\ \infty & u \neq s \end{cases}$$

heißt excess zum preflow $f$. $ex(v)$ steht für die Einheiten, die im Knoten $v$ verschickt werden können. Ein Knoten $u \in V - \{s,t\}$ heißt genau dann aktiv, wenn $ex(v) > 0$.

**preflow-push-Algorithmus**

1. Initialisierung
   - $d(u) = 0$ für alle $u \in V - s$, $d(s) = n$.
   - $f(uv) = 0$ für alle $uv \in E - N^\delta(s)$
   - $f(sv) = c(sv)$ für alle $sv \in N^\delta(s)$.
(2) Existiert kein aktiver Knoten \( v \) mehr, dann STOP, sonst wähle einen aktiven Knoten \( v \).

(3) Existiert eine Kante \( vw \in E_f \), so daß \( d(v) = d(w) + 1 \) und \( c_f(vw) > 0 \) gilt, dann verschick \( \min(c x(v), c_f(vw)) \) Flußeinheiten durch \( vw \). Gehe nach (2).

(4) Wenn für alle \( vw \in E_f \) entweder \( d(v) \leq d(w) \) oder \( c_f(vw) = 0 \) gilt, dann setze \( d(v) = \min(d(w) + 1 \mid vw \in E_f \text{ mit } c_f(vw) > 0) \). Gehe nach (2).

**Bemerkung 3.3** Schritt (3) heißt push. In ihm wird der preflow verändert. Schritt (4) heißt relabel. Dort wird das labeling verändert. Der push (relabel)-Schritt verändert die Zulässigkeit des preflows (labelings) nicht. Im Laufe des Algorithmus wird \( d(v) \) nie kleiner. Existiert ein aktiver Knoten, so ist einer der Schritte (3),(4) immer anwendbar.

Der Algorithmus arbeitet bei einer geeigneten Wahl von aktiven Knoten wie folgt: Er verschickt Flußeinheiten in Richtung Senke, bis dieses nicht mehr möglich ist. Ist dieser Punkt erreicht, so ist ein minimaler Schnitt \( (T \mid \bar{T}) \) gefunden. Es gilt: \( c x(v) = 0 \quad \forall \quad v \in \bar{T} \) (sonst gäbe es einen aktiven Knoten, der seine überflüssigen Einheiten zur Senke schicken könnte). Somit gilt (3.2) für alle \( u \in \bar{T} - t \). D.h. \( f_{\bar{T}} \) ist ein Fluß. Die restlichen Einheiten aus \( V \), d.h. der excess der Knoten aus \( T - s \) wird zur Quelle zurückgeschickt.

Es lassen sich (bei geeigneter Wahl von aktiven Knoten) zwei Phasen erkennen:
1) Suche einen minimalen Schnitt.
2) Finde den zugehörigen maximalen Fluß.

Wir wollen jetzt kurz die Korrektheit und die Komplexität des Algorithmus betrachten.

**Satz 3.2 ([GOLDBERG90])** Angenommen der Algorithmus terminiert, so ist der preflow \( f \) ein maximaler Fluß.

Beweis: Wenn der Algorithmus terminiert, gilt \( c x(v) = 0 \) für alle \( v \in V - \{s,t\} \), d.h. (3.2) gilt und damit ist \( f \) ein Fluß.

Es bleibt zu zeigen, daß \( f \) maximal ist. Wir zeigen, daß es keinen Weg \( W \) in \( G_f \) (bzw. keinen zunehmenden Weg in \( G \)) von \( s \) nach \( t \) gibt. (\( W \) zunehmender Weg \( \iff \min_{w \in W} c_f(uv) > 0 \)). Damit ist \( f \) maximal (s.[FORD62]). Annahme: Es gibt einen solchen Weg \( W : s = v_0 \rightarrow v_1, v_1 \rightarrow v_2, \ldots, v_{i-1} \rightarrow v_i = t \). Es gilt o.B.d.A. \( l < n \) (sonst abkürzen) und \( v_i v_{i+1} \in E_f \) mit \( c_f(v_i v_{i+1}) > 0 \). Da \( d \) ein zulässiges labeling ist, folgt \( d(v_i) \leq d(v_{i+1}) + 1 \) für \( 0 \leq i < l \). Damit gilt \( d(s) = d(v_0) \leq d(v_l) + l = d(t) + l \).

Da \( d(t) = 0 \) gilt \( d(s) \leq l < n \). Widerspruch. \( \square \)
Zu zeigen bleibt, daß der Algorithmus terminiert. Wir können zeigen, daß die Anzahl der Update-Operationen (push/relabel) beschränkt ist. Somit ist die Zahl der Iterationen endlich und der Algorithmus terminiert.

**Lemma 3.3** Für alle \( v \) gilt während Ablauf des Algorithmus \( d(v) \leq 2n - 1 \).

**Beweis:** Für \( v \in \{s,t\} \) ist die Aussage klar, da \( s \) und \( t \) nie aktiv werden. Sei \( v \in V - \{s,t\} \). Da das labeling nur im Schritt (4) geändert wird, brauchen wir den Beweis nur für aktive Knoten zu führen. Ist ein Knoten aktiv, so gibt es einen Weg zur Senke oder einen Weg zur Quelle. Sei \( W : v = v_0 \rightarrow v_1, v_1 \rightarrow v_2, \ldots, v_{l-1} \rightarrow v_l \) solch ein Weg in \( G_f \). Es gilt \( oBdA l \leq n - 1 \). Da \( d \) zulässiges labeling ist, folgt (wie oben) \( d(v) \leq d(v_l) + l \leq n + l \leq n + n - 1 = 2n - 1 \) mit \( d(v_l) \in \{0, n\} \). \( \square \)

**Korollar 3.4** Die Zahl der relabel-Operationen ist durch \((n - 2)(2n - 1) < 2n^2\) beschränkt.

**Beweis:** Es kann höchstens \( n - 2 \) aktive Knoten geben, für die pro Knoten maximal \( 2n - 1 \) relabel-Operationen ausgeführt werden müssen. \( \square \)

**Definition 3.3** Eine push-Operation auf einer Kante \( uv \) heißt gesättigt (ungesättigt) genau dann, wenn nach der Operation \( c_f(uv) = 0 \) (\( c_f(uv) > 0 \)) gilt.

**Lemma 3.5** Die Zahl der gesättigten push-Operationen ist durch \( nm \) beschränkt.

**Beweis:** Sei \( uv \) die Kante, für die eine gesättigte push-Operation durchgeführt wird. Danach kann auf derselben Kante (bzw. auf der Rückwärtskante) keine push-Operation mehr erfolgen, bis das label von \( v \) um 2 erhöht wurde. Erfolgt der push, so muß das label von \( u \) erst um 2 erhöht werden, bevor \( u \) wieder nach \( v \) zurücksenden kann. Da \( d(u) \leq 2n - 1 \) gilt, kann dieses höchstens \( n \) mal auftreten. Mit \( m \) Kanten kommen wir zur Schranke \( nm \). \( \square \)

**Lemma 3.6** Die Zahl der ungesättigten push-Operationen ist durch \( 2n^2m \) beschränkt.

**Beweis:** (s. [GOLDBERG90]) Dort befinden sich auch die Beweise der oben aufgeführten Sätze und Lemmata. \( \square \)

Wir haben gesehen, daß die Zahl der Update-Operationen beschränkt ist. Da immer eine dieser Operationen möglich ist, terminiert der Algorithmus nach \( O(n^2m) \) Update-Schritten. Die komplexitätsbestimmende Schranke ist die Schranke für die ungesättigten push-Operationen. Wir wollen mit Hilfe einer geeigneten Wahl von aktiven Knoten in Schritt (2) diese Schranke auf \( O(n^2) \) reduzieren.

Auswahlregel (highest-first-selection) : Wähle einen Knoten \( v \) für den \( d(v) = \max(d(u)|u \text{ ist aktiv}) \) gilt.

Für das folgende Lemma benötigen wir einen weiteren Begriff.
**Definition 3.4** Ein Abschnitt des preflow-push-Algorithmus besteht aus einer maximalen Anzahl von Iterationen, in denen sich das labeling des durch die highest-first-selection ausgewählten Knotens nicht ändert.

![Diagram](image)

**Abb. 3.1**

**Lemma 3.7** Die Anzahl der Abschnitte im preflow-push-Algorithmus bei der highest-first-selection ist durch $4n^2$ beschränkt.

Beweis: Sei $\Phi = \max(d(u)|u$ ist aktiv $)$ mit $\Phi = 0$, falls kein aktiver Knoten existiert. (Bem. $\Phi$ hängt von $d$ und damit von $f$ ab.)

1) Nach Korollar 3.4 gibt es höchstens $2n^2$ Abschnitte, in denen eine oder mehr relabel-Operationen ausgeführt werden.


3) Zum Beginn und zum Ende des Algorithmus ist $\Phi = 0$. Also ist die Anzahl der Abschnitte ohne relabel-Operation beschränkt durch die Summe der Zunahmen von $\Phi$ (während des gesamten Algorithmus).

4) Wie unter 2) bereits erwähnt, kann nur die relabel-Operation das label eines Knotens und damit $\Phi$ erhöhen. Vergrößert die relabel-Operation das label
eines Knotens um $k$, so wächst auch $\Phi$ um $k$. Mit Korollar 3.4 ist die Summe der Zunahmen durch $2n^2$ beschränkt und damit die Anzahl der Abschnitte ohne relabel-Operation. Zusammen mit den $2n^2$ Abschnitten mit relabel-Operation ergibt das maximal $4n^2$ Abschnitte. \hfill \Box

**Satz 3.8** Der preflow-push-Algorithmus mit highest-first-selection terminiert nach $O(n^3)$ Update-Schritten.

Beweis: Wir brauchen nur noch zu zeigen, dass die Anzahl der ungesättigten push-Operationen durch $O(n^3)$ beschränkt ist.

Beh.: Es gibt höchstens eine ungesättigte push-Operation pro Knoten in einem Abschnitt. Denn nach einer ungesättigten push-Operation besitzt der Knoten $v$ keine Flussinheiten mehr. Er kann dann wieder etwas verschicken, wenn ein anderer Knoten ihm etwas schickt. Dieses kann aber erst in einem anderen Abschnitt geschehen. Annahme: Innerhalb des Abschnitts bekommt Knoten $v$ erneut Flussinheiten von Knoten $u$. Dann muss $d(u) > d(v)$ gelten. Andererseits war Knoten $v$ aktiver Knoten in dem Abschnitt. Der Abschnitt ist noch nicht beendet, darum gilt $d(u) = d(v)$. Widerspruch. Mit Lemma 3.7 folgt die $O(n^3)$ Schranke für Anzahl der ungesättigten push-Operationen. \hfill \Box

Im folgenden werden wir uns mit einer effizienten Implementation des oben beschriebenen Algorithmus beschäftigen. Es wird im wesentlichen darum gehen, die Update-Operationen in konstanter Zeit durchzuführen. Im Hinblick auf die Anwendungen innerhalb dieser Arbeit werden wir nur die erste Phase des Algorithmus implementieren und dort mit Hilfe einer gap-Suche die Laufzeit des Algorithmus wesentlich verbessern.

Wir benötigen eine effiziente Bearbeitung der inzidenten Kanten eines Knotens. Dazu eignet sich eine Inzidenzliste mit beliebiger aber fester Reihenfolge der Kanten für einen Knoten. Weiterhin sei $x : V \rightarrow E$ eine Funktion, die einen Knoten auf eine Kante seiner Inzidenzliste abbildet. Für die Implementation der ersten Phase wählen wir $f \equiv 0$ und das labeling $d$ wie folgt:

$$d(u) = \begin{cases} 0 & : u = t \\ 1 & : u \neq t \end{cases}$$

**Bemerkung 3.4** Auch hier können wir mit dem BFS-Verfahren ein exaktes labeling berechnen.

Auch den Begriff aktiv werden wir ein wenig abwandeln. So erhalten wir die geeignete Auswahl von aktiven Knoten, die den Algorithmus in die beiden Phasen aufteilt. Ein Knoten $v \in V$ (einschließlich $s, t$) heißt aktiv genau dann, wenn $ex(v) > 0$ und $0 < d(v) < n$. Wir sehen, dass die Quelle der erste aktive Knoten ist. Gilt $d(v) \geq n$ für ein $v \in V$, so ist sichergestellt, dass $v$ zu der Seite des Schnitts gehört,
zu der auch $s$ gehört. Zur Bestimmung des minimalen Schnitts braucht $v$ nicht mehr herangezogen zu werden. $A$ sei die Menge der aktiven Knoten.

**Minimale-Schnitt-Algorithmus**

1. **Initialisierung**
   \[ d(u) = 1 \text{ für alle } u \in V - t, \quad d(t) = 0. \]
   \[ f \equiv 0. \]
   \[ x(v) = \text{"erste Kante in der Inzidenzliste von } v\". \]

2. **Ist** $A = \emptyset$, dann **STOP**

3. Wähle $v \in A$. Sei $x(v) = vw$.

4. **Ist** $d(w) = d(v) + 1$ und $c_f(vw) > 0$, dann verschicke $\min(\varepsilon x(v), c_f(vw))$ Flußeinheiten von $v$ nach $w$. Gehe nach (2).

5. **Ist** $vw$ die letzte Kante in der Inzidenzliste von $v$, dann setze $d(v) = \min(d(w) + 1 | vw \in E_f mit c_f(vw) > 0)$ und setze $x(v) = \text{"erste Kante in der Inzidenzliste von } v\"$. sonst setze $x(v) = \text{"nächste Kante in der Inzidenzliste von } v\"$. Gehe nach (2).

Durch den Zeiger $x(v)$, der auf die zuletzt bearbeitete Kante eines Knotens $v$ zeigt, müssen wir nicht bei erneuter Auswahl des Knotens $v$ die gesamte Inzidenzliste durchlaufen, sondern greifen sofort auf die aktuelle Kante zu. Die Kanten, die vor $x(v)$ in der Inzidenzliste stehen, sind zu diesem Zeitpunkt für einen push ungeeignet.

Wir sehen, daß die Update-Operationen in konstanter Zeit ablaufen. Mit Hilfe einer verketteten Liste und einem Zeiger auf den höchsten Eintrag können wir den Zugriff auf die aktiven Knoten (highest-first-selection) ebenfalls in konstanter Zeit realisieren (s. Kapitel 9) und erreichen damit eine Komplexität von $O(n^2)$.

Cheriyan und Maheswari [CHERIYAN87] konnten sogar eine Komplexität von $O(n^2 \sqrt{m})$ für die highest-first-selection zeigen. Auch eine Verwaltung der aktiven Knoten in einer Schlanke führt zu einer Komplexität von $O(n^2)$ [GOLDBERG90].

Als nächstes wollen wir die gap-Suche erläutern.

**Definition 3.5** Sei $f$ preflow, $d$ zulässiges labeling. Dann heißt $z \in \{1,2,\ldots,n-1\} \in gap$ genau dann, wenn $d(v) \neq z \ \forall \ v \in V$ und es einen Knoten $u \in V$ mit $n > d(u) > z$ gibt.

**Lemma 3.9 ([DERIG89])** Sei $z \in gap$. Dann ist
\[
d'(u) = \begin{cases} 
  n : & d(v) > z \\
  d(v) : & d(v) < z
\end{cases}
\]
$z$ ein zulässiges labeling.
Beweis: Annahme: $d'$ ist kein zulässiges labeling. Es gibt $vw \in E_f$ mit $c_f(vw) > 0$ und $d'(v) > d'(w) + 1$.

1. Fall $d(v) < z$, $d(w) < z \implies d'(v) = d(v) \leq d(w) + 1 = d'(w) + 1$. Widerspruch.

2. Fall $d(v) > z$, $d(w) > z \implies d'(v) = d'(w) = n \implies n > n + 1$. Widerspruch.

3. Fall $d(v) < z$, $d(w) > z \implies d'(v) = d(v) < z < n = d'(w) < d'(w) + 1$. Widerspruch.

4. Fall $d(v) > z$, $d(w) < z \implies d'(w) = d(w) \implies d(w) + 1 \leq z < d(v)$. Widerspruch zu $d$ zulässiges labeling. \qed

Ändern wir also jedesmal, wenn ein gap aufgetreten ist, das labeling wie im Lemma beschrieben, dann wird die Anzahl der aktiven Knoten echt verkleinert. Ein Knoten mit $d(v) \geq n$ wird in Phase 1 nicht mehr aktiv und wir erreichen die Abbruchbedingung $A = \emptyset$ schneller. Wir müssen aber erreichen, daß solche gaps effizient gefunden werden. Dazu führen wir einen Zähler für jedes label-Niveau ein, der die Zahl der Knoten auf dem label-Niveau enthält. Dieser Zähler muß nur beim relabel-Schritt verändert werden. Wird ein Zähler 0 und gibt es einen positiven Zähler auf höherem Niveau, so ist ein gap gefunden. Diese kleine Änderung verringert die Laufzeit erheblich (s. [DERIG89]).
3.3 Depth-First-Search


**DFS-Algorithmus**

1. **Initialisierung**
   Markiere alle Kanten als unbenutzt. $k(v) = 0$ und $f(v)$ sei undefiniert
   $\forall v \in V, i = 0, v = s$.

2. $i = i + 1, k(v) = i$.

3. **Gibt es keine unbenutzte mit $v$ inzidente Kante, dann gehe nach (5).**

4. **Wähle eine unbenutzte Kante $vw$ und markiere sie als benutzt. Ist $k(w) \neq 0$, dann gehe nach (3). Sonst setze $f(w) = v, v = w$ und gehe nach (2).**

5. **Ist $f(v)$ definiert, dann setze $v = f(v)$ und gehe nach (3).**

6. **Gibt es einen Knoten $u$, für den $k(u) = 0$ gilt, dann setze $v = u$ und gehe nach (2).**

7. **STOP.**

Der Algorithmus benutzt die Listen $k$ und $f$. In $k(v)$ steht die Nummer der Iteration, in der der Knoten $v$ zum erstenmal besucht wurde. $f(v)$ enthält den Knoten, von dem aus $v$ besucht wurde. Die Kanten $f(v)v$ heißen *Baumkanten*. Die beim DFS-Verfahren entstehenden Baumkanten bilden einen spannenden, gerichteten Wald. Wir wollen weitere Kantentypen unterscheiden.
(i) **Baumkanten:** $f(v)v$.

(ii) **Vorwärtskanten:** Eine Kante $vw$ heißt Vorwärtskante, wenn sie zum erstenmal benutzt wird und $k(w) > k(v)$ gilt. Da $vw$ noch unbenutzt ist, mußten wir den Rückwärtsschritt (5) für $v$ noch nicht anwenden. Alle Knoten mit größerer Markierung $k$ sind Nachfahren von $v$; d.h. sie sind durch Baumkanten von $v$ aus erreichbar.

(iii) **Rückwärtskanten:** Eine Kante $vw$ heißt Rückwärtskante, wenn sie zum erstenmal benutzt wird, $k(w) < k(v)$ und $w$ Vorfahre von $v$ ist; d.h. $v$ ist durch Baumkanten von $w$ erreichbar.

(iv) **Querkanten:** Eine Kante $vw$ heißt Querkante, wenn sie zum erstenmal benutzt wird, $k(w) < k(v)$ und $w$ kein Vorfahre von $v$ ist. $w$ liegt auf einem anderen Zweig des Baumes.

![Diagram](image)

Abb. 3.2

Wie schon beim BFS-Verfahren sehen wir, daß der DFS-Algorithmus jede Kante genau einmal betrachtet. Im Schritt 1 werden die Listen $k$ und $f$ für alle Knoten gesetzt. Alle anderen Berechnungen geschehen in konstanter Zeit. Daher hat auch das DFS-Verfahren die Komplexität $O(n + m)$. 
3.4 Bestimmung starker Zusammenhangskomponenten

Ein gerichteter Graph heißt stark zusammenhängend, wenn \( \lambda(v, w) \geq 1 \) für alle \( v, w \in V \). Wir führen für jeden gerichteten Graphen folgende Relation \( \sim \) ein. Es gilt \( v \sim w \) genau dann, wenn gilt \( \lambda(v, w) \geq 1 \) und \( \lambda(w, v) \geq 1 \). Es ist leicht einzusehen, daß \( \sim \) eine Äquivalenzrelation ist. Damit kommen wir zu einer Klasseneinteilung von \( V \). Die Äquivalenzklassen nennen wir starke Zusammenhangskomponenten. (Ist \( G = (V, E) \) stark zusammenhängend, so ist \( V \) die einzige Äquivalenzklasse.)

Die Kantenanzahl innerhalb der Komponenten ist mindestens 1, d.h. der Graph, eingeschränkt auf eine dieser Komponenten, ist stark zusammenhängend. Seien \( C_1, C_2, ..., C_k \) die starken Zusammenhangskomponenten von \( G \), dann heißt der Graph \( \hat{G} = (\hat{V}, \hat{E}) \) der Kondensationsgraph von \( G \), mit \( \hat{V} = \{C_1, C_2, ..., C_k\} \) und \( \hat{E} = \{(C_i, C_j) | i \neq j \text{ und es gibt eine Kante } vw \text{ in } G \text{ mit } v \in C_i \text{ und } w \in C_j\} \). (Sollten hier parallele Kanten entstehen, so können wir uns entscheiden, ob wir sie in den Kondensationsgraphen aufnehmen wollen oder nicht. Für die weitere Betrachtung auch in späteren Kapiteln ist es egal welcher Richtung wir folgen.) Der Graph \( \hat{G} = (\hat{V}, \hat{E}) \) enthält keinen gerichteten Kreis (denn sonst würden alle Komponenten auf dem Kreis zu einer Komponente zusammenfallen). Mit dieser Superstruktur läßt sich in Kapitel 6 eine Augmentierung zu einem stark zusammenhängenden Graphen sehr leicht durchführen. Das Problem besteht darin, diese Superstruktur aus einem gegebenen Graphen zu erhalten.

Wie wir eben eingesehen haben, kann es in \( \hat{G} \) keinen Kreis geben, also gibt es mindestens eine Komponente \( C_i \), für die \( \delta_{\hat{G}}(C_i) = 0 \) gilt. Wir durchlaufen den Graphen mit dem DFS-Verfahren. Folgende Änderung ist in den DFS-Algorithmus einzubringen: Ein besuchter Knoten wird auf einem Kellerspeicher (LIFO = last-in-first-out) abgelegt.

Sei \( r \) der erste Knoten von \( C_i \), der vom DFS-Algorithmus besucht wird. Da \( C_i \) starke Zusammenhangskomponente ist, werden alle Knoten von \( C_i \) von \( r \) aus besucht. Das heißt alle Knoten \( v \) aus \( C_i \) erhalten eine Marke \( k(v) \), die größer ist als \( k(r) \). Sind alle Knoten aus \( C_i \) besucht, wird das Zentrum der Aktivität beim DFS-Verfahren wieder in den Knoten \( r \) verlegt. (Beachte: Es werden keine Knoten außerhalb von \( C_i \) besucht, da \( \delta_{\hat{G}}(C_i) = 0 \).) Ist man also wieder bei \( r \) angelangt, so bilden alle Knoten, die hinunter bis einschließlich \( r \) auf dem Keller liegen, die starke Zusammenhangskomponente \( C_i \). Die Schwierigkeit besteht darin, den Knoten \( r \) als den ersten Knoten einer Komponente zu erkennen. Dazu führen wir ein weiteres Feld \( L \) ein. \( L(v) \) zeigt auf die kleinste Marke \( k(u) \) eines Knotens \( u \), der durch Baumkanten und höchstens eine Rückwärts- oder Querkante erreicht werden kann und in derselben Komponente wie \( v \) liegt. Wir werden sehen, daß der folgende Algorithmus von Tarjan [TARJAN72] die starken Zusammenhangskomponenten richtig bestimmt.
Starke Zusammenhangskomponenten-Algorithmus

(1) Iniialisierung
   Markiere alle Kanten als unbenutzt. \( k(v) = 0 \) und \( f(v) \) sei undefined
   \( \forall v \in V \). Der Keller \( S \) sei leer. \( i = 0, \ v = s \).

(2) \( i = i + 1, \ k(v) = i, \ L(v) = i \). Lege \( v \) auf \( S \).

(3) Gibt es keine unbenutzte mit \( v \) inzidente Kante, dann gehe nach (7).

(4) Wählen eine unbenutzte Kante \( vw \) und markiere sie als benutzt. Ist \( k(w) = 0 \), dann setze \( f(w) = v, v = u \) und gehe nach (2).

(5) Ist \( k(w) > k(v) \) (\( vw \) ist Vorwärtskante), dann gehe nach (3). Ist \( w \) nicht auf \( S \) (\( v, w \) gehören nicht zur selben Komponente), dann gehe nach (3).

(6) (Es gilt \( k(w) < k(v) \) und beide Knoten gehören zur selben Komponente.) Setze \( L(v) = \min(L(v), k(u)) \) und gehe nach (3).

(7) Ist \( L(v) = k(v) \), dann löse alle Knoten von \( S \) bis einschließlich \( v \). Diese Knoten bilden eine starke Zusammenhangskomponente.

(8) Ist \( f(v) \) definiert, dann setze \( L(v) = \min(L(f(v)), L(v)), v = f(v) \) und gehe nach (3).

(9) Gibt es einen Knoten \( u \), für den \( k(u) = 0 \) gilt, dann setze \( v = u \) und gehe nach (2).

(10) STOP.

Lemma 3.10 Sei \( r \) der erste Knoten für den in Schritt (7) \( L(r) = k(r) \) gilt. Dann bilden alle Knoten auf \( S \) hinunter bis einschließlich \( r \) eine starke Zusammenhangskomponente von \( G \).

Beweis: Im ersten Schritt zeigen wir, daß zwischen allen aus \( S \) entfernten Knoten \( v, w \quad \lambda(v, w) \geq 1 \) ist. Im zweiten Schritt werden wir sehen, daß diese Menge maximal ist, d.h. es gibt keinen Knoten \( x \), der zur Menge hinzugenommen werden kann, ohne den Zusammenhang zu zerstören.

Alle Knoten, die auf \( S \) oberhalb von \( r \) liegen, sind nach \( r \) entdeckt worden. Der Schritt (8) kann nach Voraussetzung für den Knoten \( r \) noch nicht ausgeführt worden sein. Also sind diese Knoten Nachfahren von \( r \). Zeigen wir nun, daß \( r \) von jedem dieser Knoten \( v \) aus erreichbar ist. Der Schritt (8) ist für \( v \) bereits ausgeführt, es gilt \( L(v) < k(v) \), denn \( r \) ist der erste Knoten für die Gleichheit gilt. Also gibt es einen Knoten \( u \) mit \( k(u) = L(v) \), der von \( v \) aus erreichbar ist. Wegen Schritt (8)
gilt \( L(r) \leq L(v) \) und mit \( k(u) = L(v) \geq L(r) = k(r) \) gilt \( k(u) \geq k(r) \). Also ist \( u \) ein Nachfahre von \( r \). Gilt \( r \neq u \), so können wir die obige Prozedur wiederholen, um einen Vorfahren von \( u \) zu finden, der gleichzeitig Nachfahre von \( r \) ist. Nach endlich vielen Schritten gilt \( r = u \) und somit ist ein Weg von \( v \) nach \( r \) gefunden.

Zu zeigen bleibt, daß kein zusätzlicher Knoten zur Komponente gehört. Beim Auftreten der Gleichheit in Schritt (7) liegen alle Knoten mit Marke größer als \( k(r) \) oberhalb von \( r \) auf \( S \). Soll noch ein weiterer Knoten \( x \) zur Komponente gehören, muß er auf \( S \) liegen (denn von einem Knoten der Komponente muß er mindestens erreicht werden) und zwar unterhalb von \( r \). Es muß also mindestens eine Rückwärts- oder Querkante \( vx \) geben, mit \( k(v) \geq k(r) > k(x) > 0 \) und \( v, r, x \) aus einer Komponenten. Also wurde in Schritt (6) \( L(v) = \min(L(v), k(x)) \) ausgeführt und es gilt \( L(v) \leq k(x) \). Damit folgt \( L(r) \leq L(v) \leq k(x) < k(r) \). Widerspruch.

Wie das Lemma zeigt, finden wir die erste Komponente \( C_1 \) korrekt. Durch Entfernen von \( C_1 \) aus dem Graphen \( G \), d.h. löschen aller Knoten und inzidenten Kanten, erhalten wir einen Graphen \( G' \). Nach dem Löschen von \( C_1 \) können wir mit \( G' \) so fortfahren, als ob wir mit \( G'' \) begonnen hätten. Liegt der Startknoten \( s \) in \( C_1 \), so ist jetzt der Stapel leer. Der Zustand des Algorithmus ist exakt wie zu Beginn. Liegt \( s \) nicht in \( C_1 \), so enthält der Keller \( S \) gerade alle Knoten, die der Algorithmus vor “Betreten” von \( C_1 \) bearbeitet hat. Nach Entfernen arbeitet er so weiter, als ob es \( C_1 \) nie gegeben hat. (Natürlich sind die Marken \( k(v) \) um die Mächtigkeit der Komponente vergrößert.)

So explizit wie eben beschrieben, arbeitet der Algorithmus nicht. Es treten keine Modifizierungen von \( G \) auf. Doch wir werden sehen, daß die Bearbeitung im Algorithmus zum Entfernen der Komponente aus dem Graphen äquivalent ist.

Um die Knoten von \( C_1 \) zu entdecken, mußte der Algorithmus die Kante \( f(r)r \) entlang gehen. Keine andere Kante von einem Knoten außerhalb von \( C_1 \) zu einem Knoten innerhalb \( C_1 \) wurde bis zu diesem Zeitpunkt benutzt. Nach Entfernen von \( C_1 \) kann keine Kante \( xv \) mit \( x \in V - C_1, \ v \in C_1 \), die in \( C_1 \) eintritt, Einfluß auf den Algorithmus nehmen. Wird diese Kante in Schritt (4) gewählt, so tritt in Schritt (5) der zweite Fall ein (da \( v \) nicht im Keller ist). Der Algorithmus “überliest” die Kante. Betrachten wir nun noch die kritische Kante \( f(r)r \). Die einzige Möglichkeit zur Einflußnahme liegt in Schritt (8). Dort wird der Zeiger \( L(r) \) neu gesetzt und kann im weiteren Verlauf auch andere Knoten beeinflußten. Es gilt aber:

\[
L(f(r)) = \min(L(f(r)), L(r)) \\
= \min(L(f(r)), k(r)) \\
\leq \min(k(f(r)), k(r)) = k(f(r))
\]

Also bleibt diese Anweisung ohne Wirkung. Der Algorithmus arbeitet weiter als ob es die Komponente \( C_1 \) nie gegeben hätte. Das beweist die Korrektheit des Verfahrens.

Die Komplexität des Algorithmus läßt sich sehr einfach bestimmen. Im Prinzip
Kapitel 4

Bestimmung der Kantenzusammenhangszahl

4.1 Einfache Berechnung der Kantenzusammenhangszahl

In Kapitel 2 haben wir die Kantenzusammenhangszahl $\lambda_G = \min(M(v, w) \mid v, w \in V)$ für einen gerichteten Graphen $G = (V, E)$ definiert, wobei $M(v, w)$ die Mächtigkeit eines minimalen Schnitts (oder auch trennenden Kantenmenge) war.

Nach dem Satz 2.3 können wir $M(v, w)$ mit einem Aufruf des MFMC–Algorithmus (s. Kapitel 3) in einem Netzwerk berechnen, mit $G$ als zugrunde liegendem Graphen und der Kapazität 1 auf allen Kanten.

Bemerkung 4.1 Wir können auch für ein Netzwerk $(G, c, *, *)$ die Kantenzusammenhangszahl $\lambda_{G,c}$ durch $\lambda_{G,c} = \min(M_c(v, w) \mid v, w \in V)$ definieren. Mit $M_c(v, w) := \min(||(T, T)||, ||(T, T)||)$ ist $v, w$-Schnitt.

Die Bestimmung der Kantenzusammenhangszahl wird also eine geeignete Folge von Aufrufen des MFMC–Algorithmus sein.

Eine naive Möglichkeit wäre die Untersuchung für alle möglichen Paare $(v, w)$. Es sind also $n(n - 1)$ MFMC–Aufrufe zur Bestimmung von $\lambda_G$ nötig. Wir werden im nächsten Abschnitt die Zahl der Aufrufe auf $n/2$ reduzieren.

Seien $v_0, w_0$ zwei Knoten für die $M(v_0, w_0) = \min(M(v, w) \mid v, w \in V)$ gilt. Zu diesen Knoten gehört ein minimaler $v_0, w_0$-Schnitt $(L, R)$. Diesen Schnitt wollen wir minimalen Schnitt (des Graphen) nennen.

Lemma 4.1 Sei $(L, R)$ minimaler Schnitt eines Graphen. Für jeden Knoten $v \in L$ und jeden Knoten $w \in R$ gilt $M(v, w) = ||(L, R)||$. 

27
Das Lemma besagt, daß es egal ist von welchem Knoten aus $L$ und zu welchem Knoten aus $R$ wir den maximalen Fluß berechnen.

Beweis: Annahme: Es existieren Knoten $v \in L, w \in R$ mit $M(v, w) \neq |(L, R)|$. Es muß $M(v, w) > |(L, R)|$ gelten, da $(L, R)$ minimaler Schnitt des Graphen. Sei $(T, \bar{T})$ der minimale $v, w$-Schnitt. Dann gilt $|(T, \bar{T})| > |(L, R)|$. $(L, R)$ ist ebenfalls ein $v, w$-Schnitt, das ist aber ein Widerspruch zur Minimalität von $|(T, \bar{T})|$. \hfill \Box

**Algorithmus zur Kantenzusammenhangszahl I**

1. Wähle einen Knoten $v$.
2. Suche einen Knoten $w$ auf der “anderen Seite des minimalen Schnitts”.
3. Setze $\lambda_G = \min(M(v, w), M(w, v))$. STOP.

Nach dem Lemma 4.1 ist der Algorithmus I korrekt. Das Problem liegt in Schritt (2), in der Wahl des Knotens $w$. Wie sieht man einen Knoten an, ob er auf der richtigen Seite liegt oder nicht. Betrachten wir den folgenden Algorithmus:

**Algorithmus zur Kantenzusammenhangszahl II**

1. Wähle einen Knoten $v$.
2. Setze $\lambda_G = \min_{w \in \Gamma(v)}(M(v, w), M(w, v))$. STOP.

Sei o.B.d.A. $v \in L$, dann gibt es einen Knoten $w_0 \in R$. Für diesen Knoten wird in Schritt (2) $M(v, w_0)$ und $M(w_0, v)$ berechnet. Also haben wir zwei Knoten auf verschiedenen Seiten des minimalen Schnitts gefunden und mit Lemma 4.1 folgt die Korrektheit des Algorithmus.

Wir benötigen hier 2($n-1$) Aufrufe des MFMC-Algorithmus. Mit der Wahl eines festen Knotens $v$ und der Suche eines Knotens auf der entgegengesetzten Seite von $v$ kommen wir nicht weiter. Mit dem Lemma 4.1 sehen wir auch, daß der Knoten $v$ nicht von zentraler Bedeutung ist. Wir brauchen nur die Berechnung von $M(u, v)$ und $M(x, y)$, mit $u, y \in L \ v, x \in R$.

**Lemma 4.2 ([SCHNORR79])** Sei $v_0, v_1, \ldots, v_p$ eine geschlossene Kette (d.h. $v_p = v_0$) von Knoten aus $V$, so daß mindestens ein $v_i \in L$ und mindestens ein $v_j \in R$. Dann gilt $\lambda_G = \min(M(v_i, v_{i+1}) | i = 0, 1, \ldots, p - 1)$.

Beweis: Mit den Vorbemerkungen ist der Beweis trivial. \hfill \Box

Eine geschlossene Kette mit allen Knoten aus $V$ genügt der Bedingung aus dem Lemma und zeigt die Korrektheit des folgenden Algorithmus.
Algorithmus zur Kantenzusammenhangszahl III

(1) Sei $V = \{v_0, v_1, \ldots, v_{n-1}\}$. Betrachte die geschlossene Kette $U = \{v_0, v_1, \ldots, v_n\} (v_n = v_0)$.
Setze $\lambda_G = \min(M(v_i, v_{i+1}) | i = 0, 1, \ldots, n-1)$. STOP.
4.2 Berechnung der Kantenzusammenhangszahl nach Esfahanian und Hakimi

Wir wollen in diesem Abschnitt eine Menge $T$ mit $|T| \leq n/2$ angeben, die einen Knoten aus $L$ und einen Knoten aus $R$ enthält. Dazu benötigen wir etwas Vorbereitung.

Sei $\delta_G = \min_{v \in V} (\delta(v), \varrho(v))$. Weiterhin sei $(L, R)$ der minimale Schnitt des Graphen.

**Lemma 4.3** Sei $\lambda_G < \delta_G$. Dann gilt $|L| > \delta_G$ und $|R| > \delta_G$.

Beweis: Es gilt $\delta(v) \geq \delta_G$ und damit $\sum_{v \in L} \delta(v) \geq |L| \delta_G$. Weiter gilt $\sum_{v \in L} \delta(v) = E(L) + |(L, R)|$. (Dekomposition der Kantenmenge in einen Teil, der nur zwischen Knoten aus $L$ verläuft und einen Teil zwischen Knoten aus $L$ und Knoten aus $R$.) Damit gilt:

$$|L| \delta_G \leq \sum_{v \in L} \delta(v) = |E(L)| + |(L, R)| < |L|(|L| - 1) + \delta_G$$

Damit folgt $(|L| - 1) \delta_G < |L|(|L| - 1)$. Wegen $\lambda_G < \delta_G$ gilt $|L| > 1$ und damit $L | > \delta_G$.

Der Beweis verläuft für $R$ analog. □

Für jeden Knoten $u \in L$ sei $x(u)$ die Anzahl der Kanten aus $E(L)$, die in $u$ beginnen. Sei $y(u)$ die Anzahl der Kanten aus $(L, R)$, die nicht in $u$ beginnen.

**Lemma 4.4** Sei $\lambda_G < \delta_G$. Dann gilt für alle $u \in L$ $x(u) > y(u)$.

Beweis: Sei für $u \in L$ $z(u) = \{uv \in E|v \in R\}$. Es gilt $z(u) + y(u) = |(L, R)|$. Nach Voraussetzung gilt aber $|(L, R)| = \lambda_G < \delta_G \leq \delta(u) = x(u) + z(u) \rightarrow y(u) < x(u)$. □

**Bemerkung 4.2** Für Knoten aus $R$ gilt das Lemma nicht.

**Lemma 4.5** Sei $\lambda_G < \delta_G$. Sei $B$ ein durch Baumkanten, die beim Durchlaufen des DFS-Verfahrens durch den Graphen $G$ entstanden sind, erzeugter gerichteter Baum. Dann enthält sowohle $E(L)$ als auch $E(R)$ eine Kante aus $B$.

Beweis: (a) Annahme: Es existiert keine Baumkante in $E(R)$. Da jeder Knoten vom DFS-Verfahren besucht wurde, heißt das, daß alle Knoten $v \in R$ von einem Knoten $u \in L$ aus besucht wurden. Also liegt für jeden Knoten $v \in R$ eine Kante $uv$ mit $u \in L$ im Schnitt $(L, R)$. D.h. $\lambda_G = |(L, R)| \geq |R| > \delta_G$. Das letzte Ungleichheitszeichen gilt nach Lemma 4.3. Das ist ein Widerspruch zur Voraussetzung.

(b) Annahme: Es existiert keine Baumkante in $E(L)$. Sei $w \in L$ und sei $w$ das erste Blatt von $B$, das beim Besuch der Knoten auftaucht. (Ein Knoten $w$ heißt
Blatt in einem Baum, wenn er keine Nachfahren im Baum mehr hat.) Sei \( X = \{ v \in L | vv \in E(L) \} \). Da \( w \) ein Blatt in \( B \) ist, sind alle Knoten aus \( X \) schon besucht worden. Da \( w \) das erste Blatt ist, müssen alle Knoten aus \( X \) einen Nachfahren haben. Laut Voraussetzung liegt kein direkter Nachfahre in \( L \). D.h. \( y(w) \geq |X| = x(w) \). Widerspruch zu Lemma 4.4.

\[
\begin{array}{c}
\text{l} \\
\text{\textit{L}} \\
\text{\textit{x}_1} \\
\text{\textit{x}_2} \\
\text{\textit{w}} \\
\text{\textit{x}_3} \\
\end{array}
\quad
\begin{array}{c}
\text{r} \\
\text{\textit{R}} \\
\text{\textit{x}_4} \\
\text{\textit{x}_5} \\
\text{\textit{x}_6} \\
\end{array}
\]

\[\rightarrow\]

Baumkanten

Abb. 4.1

**Satz 4.6 ([ESFAHANIAN84])** Sei \( \lambda_G < \delta_G \). Die Menge, die aus den Knoten auf den Ebenen von \( B \) mit gerader (ungerader) Ebenenzahl besteht, enthält mindestens zwei Knoten. Mindestens einer dieser Knoten ist aus \( L \) und einer aus \( R \).

Beweis: Die Terminalknoten einer Baumkante liegen auf Ebenen verschiedener Parität. Es gibt mindestens eine Kante zwischen Knoten aus \( L \) und eine Kante zwischen Knoten aus \( R \). Also liegen mindestens zwei Knoten auf den entsprechenden Ebenen.

Sei \( EVEN(B) \) (\( ODD(B) \)) die Menge der Knoten auf Ebenen mit gerader (ungerader) Ebenenzahl. Sei \( v_1w_1 \in E(L) \) und \( v_2w_2 \in E(R) \). Sei \( v_1, v_2 \in EVEN(B) \) und \( w_1, w_2 \in ODD(B) \). Wir sehen, daß jeweils in \( EVEN(B) \) und \( ODD(B) \) mindestens ein Knoten aus \( L \) und \( R \) enthalten ist. Auch die drei restlichen Kombinationen liefern das gleiche Ergebnis.

Bilden wir mit den Knoten aus \( EVEN(B) \) bzw. \( ODD(B) \) eine geschlossene Kette, so folgt mit dem Lemma 4.2 die Korrektheit des nachstehenden Algorithmus.

**Algorithmus zur Kantenzusammenhangszahl IV**

1. **Bestimme einen Baum \( B \) mit dem DFS-Verfahren.**
(2) Sei $EVEN(B) = \{ v \in V \mid v \text{ liegt auf einer Ebene von } B \text{ mit gerader Ebenenzahl } \}$. Sei $ODD(B) = \{ v \in V \mid v \text{ liegt auf einer Ebene von } B \text{ mit ungerader Ebenenzahl } \}$. Wähle aus diesen beiden Mengen die Menge mit der kleineren Anzahl von Knoten. Sie habe die Mächtigkeit $p$. (Es gilt $p \leq n/2$.)

(3) Bilde eine geschlossene Kette mit den Knoten aus der in (2) gewählten Menge: $U = \{ u_0, u_1, \ldots, u_p \} \ (u_p = u_0)$. Berechne $\varphi = \min(M(u_i, u_{i+1}) \mid i = 0, 1, \ldots, p)$.

(4) Setze $\lambda_G = \min(\varphi, \delta_G)$.

Der Schritt (4) ist notwendig, da wir in den oben genannten Aussagen stets mit der Voraussetzung $\delta_G > \lambda_G$ arbeiten. $\delta_G$ läßt sich in $O(m)$ berechnen. Den Baum $B$ können wir mit einer Komplexität von $O(n + m)$ konstruieren (siehe DFS-Verfahren Kapitel 3).

Wie schon im Algorithmus angemerkt, wird der MFM-C-Algorithmus höchstens $n/2$ mal aufgerufen. Wir erzielen damit keine Verbesserung der Komplexität gegenüber den Algorithmen II und III. Die Komplexität der Algorithmen II–IV ist $O(n \ast MFM(n, m)).(MFM(n, m) = \text{Komplexität eines maximalen Fluß-Algorithmus.})$

**Bemerkung 4.3** Mit dem preflow-push-Algorithmus aus Kapitel 3 folgt eine Komplexität von $O(n^4)$. 
Kapitel 5

Einführung in die Kantenzusammenhangsaugmentierung

5.1 Problemformulierung

Viele Probleme, die sich mit Zusammenhangseigenschaften beschäftigen, lauten: Wieviele Kanten muß man aus einem Graphen entfernen, damit er einer gewissen Eigenschaft genügt? Nehmen wir als Eigenschaft: 'Der (ungerichtete) Graph ist unzusammenhängend'. Lautet die Antwort 0, so ist der (ungerichtete) Graph unzusammenhängend. Ist die Antwort größer als 0, so ist der (ungerichtete) Graph zusammenhängend. Diese Zahl führt zum Begriff des Kantenzusammenhangs für (ungerichtete) Graphen. (Ähnliche Betrachtungen können wir für gerichtete Graphen durchführen.)


Definition 5.1 Sei $C$ eine Eigenschaft für (ungerichtete) gerichtete Graphen. Sei $G = (V, E)$ ein Graph. $C$ heißt monoton wachsend, wenn folgende Implikation gilt: Gilt für $G$ die Eigenschaft $C$, so gilt $C$ für alle Obergraphen $G' = (V, E \cup E')$.

Beispiele für solche Eigenschaften sind 'G ist zusammenhängend', 'G ist nicht planar', 'G enthält einen Kreis'. (Ist $C$ eine monoton wachsende Eigenschaften, so heißt deren Negation monoton fallend.)

Wir können das Augmentierungsproblem noch verallgemeinern, indem wir nicht nur nach der Anzahl von Kanten fragen, sondern nach einer Kantenmenge mit minimalem Gewicht. Dazu führen wir eine Kostenfunktion $a : V^2 \rightarrow R \cup \infty$ ein, die den möglichen Kanten ein Gewicht zuweist. Wir werden uns im weiteren mit Augmentierungsproblemen bezüglich einer monoton wachsenden Eigenschaft und bzgl. einer Gewichtsfunktion $a$ beschäftigen. Also können wir oBdA $a \geq 0$ annehmen. (Gibt es
Kanten mit negativen Gewichten, so können wir sie von vornherein zum Graphen hinzufügen. Sie verringern das Gewicht der gesuchten Menge und da die Eigenschaft monoton wachsend ist, können nie zu viele Kanten vorhanden sein.)

**Formulierung der Augmentierungsprobleme**

Sei $V$ eine Menge von Knoten. Sei $a : V^2 \rightarrow R^+_0 \cup \infty$ eine Gewichtsfunktion. Sei $C$ eine monoton wachsende Eigenschaft. Dann kann man die folgenden Probleme betrachten:

1. Finde eine minimal gewichtete Menge $E$, so daß $G = (V, E)$ die Eigenschaft $C$ erfüllt.
2. Sei $G_0 = (V, E_0)$ ein Graph. Finde eine minimal gewichtete Menge $E$, so daß $G = (V, E_0 \cup E)$ der Eigenschaft $C$ genügt.
3. Sei $E^* = \{vw | a(vw) < \infty\}$. Gilt $C$ für $G^* = (V, E^*)$, so finde einen minimal gewichteten aufspannenden Teilgraphen von $G^*$, der die Eigenschaft $C$ erfüllt. Ist $C$ nicht für $G^*$ erfüllt, so hat jeder Graph mit Knotenmenge $V$ unendliches Gewicht.

Ist $a$ eine 0-1-wertige Funktion so heißt das Augmentierungsproblem ungewichtet. Die drei Problemformulierungen lassen sich leicht ineinander überführen.

Wir wollen als nächstes einige Probleme als Augmentierungsprobleme formulieren.

In ungerichteten Graphen:

1. $v, w \in V$. $C =$ „Es gibt einen Weg zwischen $v$ und $w$“. Das Augmentierungsproblem sucht nach einem kürzesten Weg zwischen $v$ und $w$ in $G^*$.
2. $C =$ „$G$ ist zusammenhängend“. Das Augmentierungsproblem sucht nach einem minimalen aufspannenden Baum in $G^*$.
3. $C =$ „Es gibt einen Hamiltonschen Kreis“. Ausgehend von einem beliebigen Graphen $G_0 = (V, E_0)$ ist das Problem NP-vollständig ([GAREY79] HAMILTONIAN COMPLETION (GT34)). Gehen wir aber von einem Baum aus, so kann das Problem in polynomialer Zeit gelöst werden.
4. $R \in N^{[V \times V]}$ symmetrische Matrix. $C =$ „$v, w$ sind durch $R(v, w)$ knotendisjunkte Wege verbunden“. Das Augmentierungsproblem sucht nach einer minimalen Kantenmenge $E$, mit deren Hinzunahme der Graph $G_0 = (V, E_0)$ den Zusammenhangsanforderungen aus $R$ genügt.
5. $R \in N^{[V \times V]}$ symmetrische Matrix. $C =$ „$v, w$ sind durch $R(v, w)$ kantendisjunkte Wege verbunden“. Das Augmentierungsproblem sucht nach einer minimalen Kantenmenge $E$, mit deren Hinzunahme der Graph $G_0 = (V, E_0)$ den Zusammenhangsanforderungen aus $R$ genügt.
6) Sei $S \subseteq V$. $C = \text{"Es gibt einen Weg zwischen zwei Knoten aus } S\text{". Das Augmentierungsproblem heißt } \textit{Steiner tree Problem}. Es sucht nach einem minimalen Baum in $G^*$, der alle Knoten aus $S$ enthält.

In gerichteten Graphen

7) $C = \text{"Alle Knoten sind von einem Knoten erreichbar". Das Augmentierungsproblem sucht nach einer minimalen Arboreszenz in } G^*$.

8) $C = \text{"Alle Kanten sind in einem Kreis enthalten". Das Augmentierungsproblem sucht nach einer minimalen Kantenmenge, die zu einem Graphen } G_0 = (V, E_0) \text{ hinzugefügt werden muß, damit er einen Eulerschen Kreis enthält.}$

9) $R \in N^{\lfloor V \times V \rfloor}$. $C = \text{"Es gibt } R(u, v) \text{ kantendisjunkte Wege von } u \text{ nach } v." \text{ Das Augmentierungsproblem sucht nach einer minimalen Menge } E, \text{ mit deren Hinzunahme der Graph } G_0 = (V, E_0) \text{ den Zusammenhangsanforderungen aus } R \text{ genügt.}$

10) Wie Problem 9, nur mit $R \equiv k \in N$. Dieses Problem heißt Hauptproblem. Wir werden in Kapitel 7 für das Hauptproblem einen polynomialen Algorithmus angeben.

Literaturverweise zu den Problemen 1,2,3,6,7 befinden sich in [ESWARAN76]. Die restlichen Probleme werden anschließend behandelt oder sind in [FRANK90] nachzulesen.
5.2 NP-vollständige Augmentierungsprobleme


Dies gilt nicht für das Steiner tree Problem (s. [GAREY79] STEINER TREE IN GRAPHS (ND12)).

Lemma 5.1 Die gewichteten Probleme 4,5,10 (und damit auch 9) sind NP-vollständig.

Bemerkung 5.1 Die Klasse NP ist nur für Entscheidungsprobleme definiert. Fassen wir also die oben erwähnten Probleme als Entscheidungsprobleme auf: Gibt es eine Kantenmenge $E$ mit $a(E) \leq \alpha \in \mathbb{R}$, mit deren Hinzunahme der Graph $G_0 = (V, E_0)$ den Zusammenhangsanforderungen aus $R$ genügt?

Der Beweis der NP-Vollständigkeit besteht aus zwei Teilen. Im ersten Teil wird gezeigt, daß das Problem aus NP ist. Im zweiten Teil wird bewiesen, daß alle Probleme aus NP auf das betrachtete Problem mittels einer polynomialen Transformation zurückgeführt werden können. Dieses geschieht i. allg. dadurch, daß das betrachtete Problem auf ein bekanntes NP-vollständiges Problem zurückgeführt wird.

Beweis: Wir beschränken die Probleme, indem wir in den Fällen 4,5 $R \equiv k = 2$ und in Fall 10 $k = 1$ wählen.

(i) Es lassen sich für die verschiedenen Probleme leicht Rate- und Testalgorithmen angeben. Damit sind die Probleme aus NP.

(ii) Zum Nachweis der NP-Vollständigkeit führen wir die Probleme auf das (ungerichtete) gerichtete Hamiltonsche Kreis Problem zurück, welches bekanntlich NP-vollständig ist (s. [GAREY79] HAMILTONIAN CIRCUIT (GT37), DIRECTED HAMILTONIAN CIRCUIT (GT38)). Sei $G = (V, E)$ ein (ungerichteter) gerichteter Graph. Sei $a : V^2 \rightarrow \{1, 2\}$, mit $a(vw) = 1$ für $vw \in E$, $a(vw) = 2$ für $vw \notin E$. Löse das Entscheidungsproblem: “Gibt es eine Kantenmenge $E'$ mit $a(E') \leq |V|$, so daß $G'=(V,E')$ der Zusammenhangsanforderungen genügt?” Ist $a(E') \leq |V|$, so enthält $G$ einen Hamiltonschen Kreis, denn um den Zusammenhangsanforderungen zu genügen, müssen wenigstens $|V|$ Kanten im Graphen vorhanden sein. Da $a(E') \leq |V|$ und $|E'| \geq |V|$ wurden nur Kanten aus $E$ benutzt. Also ist $G'$ der gesuchte Hamiltonsche Kreis. Wir haben das Hamiltonsche Kreis Problem auf jedes der gewichteten Probleme 4,5,9 polynomial zurückgeführt. □

Das ungewichtete Problem 5 wird in [FRANK90] in stark polynomialer Zeit gelöst. Das Problem 10 wird in Kapitel 6 für $k = 1$ und in Kapitel 7 für allgemeines $k$ gelöst.
Die Hoffnung auf polynomiale Lösbarkeit des ungewichteten Problems 9 wird
durch folgenden Satz zerschlagen.

*Satz 5.2 ([FRANK92]) (A) Sei \( G = (V, E) \) ein gerichteter Graph, \( s \) ein ausge-
zeichneten Knoten von \( G \), \( T \subseteq V \) eine ausgezeichnete Menge von Knoten und \( \gamma > 0 \)
eine ganze Zahl. Ist es möglich durch Hinzunahme von höchstens \( \gamma \) Kanten zu ga-
rantieren, daß es einen Weg von \( s \) zu jedem Knoten aus \( T \) gibt?
(B) Sei \( G' = (V, E') \) gerichteter Graph, \( U \subseteq V \) eine ausgezeichnete Menge von Kno-
ten und \( \gamma > 0 \) eine ganze Zahl. Ist es möglich durch Hinzunahme von höchstens \( \gamma \)
Kanten zu garantieren, daß es einen Weg von jedem Knoten aus \( U \) zu jedem Knoten
aus \( U \) gibt?

*Beide Probleme (A) und (B) sind NP-vollständig.*

Beweis: (i) Es ist leicht zu sehen, daß beide Probleme aus NP sind.

(ii) Wir führen Problem (A) auf das minimale Überdeckungsproblem zurück, welches
NP-vollständig ist (s. [GAREY79] Hitting SET (SP8)) und das wie folgt lautet:
Sei \( F = \{X_1, ..., X_k\} \) eine Familie von Teilmengen einer Menge \( V \) und \( \gamma \) eine ganze
Zahl. Gibt es eine Menge \( Z \subseteq V \), mit \(|Z| \leq \gamma\), so daß \( Z \) mindestens ein Element
aus jeder Menge \( X_i \) enthält?

Sei \( X_1 \cup \ldots \cup X_k, \gamma \) eine Instanz für das Hitting SET Problem. Sei \( S =
X_1 \cup \ldots \cup X_k \). Für jedes \( X_i \) sei \( t_i \) ein neues Element (d.h. \( t_i \notin S, t_i \neq t_j \) für \( i \neq j \)),
\( T = \{t_1, ..., t_k\} \). Sei \( s \) ein Element nicht aus \( T \cup S \) und sei \( V = S \cup T \cup s \). Sei
\( G = (V, E) \) ein gerichteter Graph mit \( E = \bigcup_{i=1}^{k} \{vt_i | v \in X_i\} \).

Wir haben eine Instanz des Hitting SET Problems polynomial auf eine Instanz
des Problems (A) zurückgeführt. Nun lösen wir Problem (A). Hat Problem (A) eine
Lösung, so gibt es auch eine Lösung, die nur aus Kanten \( sv, v \in S \) besteht. Dazu
das folgende Bild.
Die Endknoten der neuen Kanten formen die gesuchte Menge $Z$ mit $|Z| \leq \gamma$, da es von $s$ zu jedem $t_i$ einen Weg gibt. Alle Wege benutzen höchstens $\gamma$ Knoten aus $S$. Aber jede Menge $X_i$ wird besucht, denn nur von $X_i$ gehen auch Kanten zu $t_i$.

Als nächstes führen wir Problem (B) auf Problem (A) zurück. Sei also $\{G = (V, E), s, T, \gamma\}$ eine Instanz von Problem (A). Dann ist $\{G' = (V, E'), U, \gamma\}$ mit $E' = E \cup \{vs|v \in T\}$ und $U = T \cup s$ eine Instanz für Problem (B). Sie wurde polynomial von der Instanz aus (A) transformiert. Nun löse Problem (B). Hat die so hergeleitete Instanz des Problems (B) eine Lösung mit nicht mehr als $\gamma$ neuen Kanten, so hat auch die Instanz von Problem (A) eine Lösung mit nicht mehr als $\gamma$ neuen Kanten. Die Kantenmenge überträgt sich exakt. \(\square\)

Wir wir leicht sehen ist Problem (B) eine Einschränkung von Problem 9: Sei $U \subseteq V$. Setze die Kantenzusammenhangsanforderung in $R$ wie folgt:

$$r_{i,j} = \begin{cases} 1 & : i, j \in U \\ 0 & : \text{sonst} \end{cases}$$

Schon dieses einfache 0–1 Teilproblem von Problem 9 ist NP-vollständig.
Kapitel 6

Augmentierung zu stark zusammenhängenden Graphen

Im vorherigen Kapitel haben wir das “starke Zusammenhangs Augmentierungs-
Problem” kennen gelernt und gesehen, daß die allgemeine gewichtete Version NP-
vollständig ist. Wir werden in diesem Kapitel einen effizienten Algorithmus zur
Lösung des ungewichteten Problems angeben.

Ausgehend von einem gerichteten Graphen \( G = (V, E) \) suchen wir eine Kanten-
menge \( E_A \) mit minimaler Mächtigkeit, so daß \( G' = (V, E \cup E_A) \) ein stark zusam-
menhängender Graph ist.

Wir werden sehen, daß wir unsere Betrachtungen auf den Kondensationsgraphen
\( \tilde{G} = (\tilde{V}, \tilde{E}) \) von \( G \) reduzieren können (zur Definition des Kondensationsgraphen
siehe Kapitel 3). Für alle \( v \in V \) sei \( \alpha(v) \in \tilde{V} \) die Komponente von \( G \) (Knoten
von \( \tilde{G} \)), in der \( v \) liegt. Für alle \( \tilde{v} \in \tilde{V} \) (in Kapitel 3 waren die Knoten von \( \tilde{G} \) mit
\( C_0, \ldots, C_k \) durchnummeriert) sei \( \beta(\tilde{v}) \in V \) ein Repräsentant der Komponente. Es
gilt \( \alpha(\beta(\tilde{v})) = \tilde{v} \quad \forall \tilde{v} \in \tilde{V} \) (beachte: i.allg. \( \beta(\alpha(v)) \neq v \)).

Lemma 6.1  (i) Sei \( E_A \) die gesuchte Kantenmenge, die \( G \) zu einem stark zusam-
menhängenden Graphen macht. Dann ist auch \( \tilde{G}' = (\tilde{V}, \tilde{E} \cup \alpha(E_A)) \) mit \( \alpha(E_A) = \{ \alpha(v)\alpha(w) | vw \in E_A, \alpha(v) \neq \alpha(w) \} \) ein stark zusammenhängender Graph.
(ii) Sei \( \tilde{E}_A \) eine Kantenmenge, die \( \tilde{G} \) zu einem stark zusammenhängenden Graphen
macht. Dann ist auch \( G' = (V, E \cup \beta(\tilde{E}_A)) \) mit \( \beta(\tilde{E}_A) = \{ \beta(\tilde{v})\beta(\tilde{w}) | \tilde{v}\tilde{w} \in \tilde{E}_A \} \) ein
stark zusammenhängender Graph.

Beweis: Betrachte \( \tilde{v}, \tilde{w} \in \tilde{V} \). Suche einen Weg in \( \tilde{G} \) von \( \tilde{v} \) nach \( \tilde{w} \) und umkehrt.
Sei \( v = \beta(\tilde{v}) \) und \( w = \beta(\tilde{w}) \). Wir wissen, daß es einen Weg zwischen \( v \) und \( w \) und
umkehrt gibt. Sei \( v = x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_{k-1} \rightarrow x_k = w \) solch ein Weg. Seien
\( x_{i_1} \rightarrow x_{i_1+1}, x_{i_2} \rightarrow x_{i_2+1}, \ldots, x_{i_t} \rightarrow x_{i_t+1} \) die Kanten, die zwischen den Zusammen-
hängskomponenten von \( G \) verlaufen. Andere den Weg so ab, daß von \( x_{i_j} \) entlang
eines Weges der Knoten \( x = \beta(\alpha(x_{i_j})) \) besucht wird. Dann betrete mit der Kante

39
vw mit $y = \beta(\alpha(x_{i+1}))$ die andere Komponente und gehe entlang eines Weges zu
$x_{i+1}$ zurück. Setze den Weg fort und wiederhole eventuell die oben beschriebene
Prozedur. Betrachte die Folge $\alpha(v) = \alpha(x_0), \alpha(x_1), \ldots, \alpha(x_k) = \alpha(w)$. Kommt ein
Knoten mehrfach vor, so streiche ihn so oft, daß er nur noch einmal vorkommt. Die-
se Folge entspricht einem Weg in $\tilde{G}$. (Bemerkung: Die Kanten aus $E_A$ innerhalb einer
Komponenten spielen keine Rolle.) Den Weg von $w$ nach $v$ finden wir anlog.

Der Beweis der zweiten Aussage verläuft analog.

Beschränken wir unsere Anstrengungen also auf den Kondensationsgraphen $\tilde{G}$. Wie
schon in Kapitel 3 angemerkt, ist $\tilde{G}$ azykличisch. Sei $\#s$ die Anzahl der Quellen,
$\#t$ die Anzahl der Senken und $\#q$ die Anzahl der isolierten Knoten in $\tilde{G}$.

**Lemma 6.2** Es gilt $\#s + \#t + \#q > 0$.

Beweis: Ist $\#q > 0$, so ist nichts zu zeigen. Sei $\#q = 0$. Annahme: Es gibt
keine Senke in $\tilde{G}$. Wähle einen Knoten $\tilde{w}_0 \in \tilde{V}$. Ist $\tilde{w}_0$ keine Senke, so gibt es
keinen Nachbarn $\tilde{w}_1$. Ist dieser wieder keine Senke, so gibt es einen Nachbarn $\tilde{w}_0$. Da es keine Senke gibt, können wir unendlich lange so fortfahren. Nach $|\tilde{V}|$ Schritten
treten wir in einen Knoten ein, der schon einmal besucht wurde und haben einen
Kreis gefunden. Widerspruch.

**Lemma 6.3** Sei $\tilde{G}$ ein kriegsfreier Graph. $\#s, \#t, \#q$ seien wie oben definiert. Dann
benötigt man mindestens $\max(\#s, \#t) + \#q$ Kanten, um $\tilde{G}$ zu einem stark zusam-
menhängenden Graphen zu augmentieren.

Beweis: Nach dem Augmentieren muß für jeden isolierten Knoten und jede Sen-
ke $v$ $\delta(v) \geq 1$ gelten. Wir müssen also mindestens $\#q + \#s$ Kanten hinzufügen.
Genauo muß nach dem Augmentieren für jeden isolierten Knoten und jede Quelle
$v$ $\rho(v) \geq 1$ gelten. Damit folgt die Behauptung.

Wir werden nun eine Menge $E_A$ mit $|E_A| = \max(\#s, \#t) + \#q$ konstruieren, die
das gewünschte leistet. OBdA sei $\#t \geq \#s$. (Gilt $\#t < \#s$, so orientiere alle Kanten
im Kondensationsgraphen um. Nach der Augmentierung orientiere die Kanten aus
$E_A$ erneut um.)

Seien $s(1), \ldots, s(p)$ Quellen und $t(1), \ldots, t(p)$ Senken, so daß folgendes gilt:

i) Es gibt einen Weg von $s(i)$ nach $t(i)$ für $1 \leq i \leq p$.

ii) Für jede Quelle $s$ gibt es einen Weg zu einem $t(i_0)$ mit $1 \leq i_0 \leq p$.

iii) Für jede Senke $t$ gibt es einen Weg von einem $s(i_0)$ zu $t$ mit $1 \leq i_0 \leq p$.

OBdA sei $p > 0$. Denn gibt es überhaupt eine Quelle und Senke so werden die
Bedingungen i)–iii) von mindestens einem Paar erfüllt. Besteht der Kondensations-
graph nur aus isolierten Knoten, so ist die Augmentierung trivial. Sei im folgenden
$\#z := \#q + \#t - \#s$. Seien $s(p+1), \ldots, s(\#s)$ die restlichen Quellen, $t(p+1), \ldots, t(\#s)$
weitere Senken und $x(1), \ldots, x(\#z)$ die übrigen Senken und isolierten Knoten.
Satz 6.4 ([ESWARAN76]) Der Graph $\tilde{G}' = (\tilde{V}, \tilde{E} \cup \tilde{E}_A)$ mit

$$\tilde{E}_A = \{t(i)s(i + 1) | 1 \leq i < p\} \cup \{t(i)s(i) | p + 1 \leq i \leq \#s\} \cup \tilde{E}_{A_1}$$

$$\tilde{E}_{A_1} = \begin{cases} 
  t(p)s(1) & : \# z = 0 \\
  \{t(p)x(1), x(\#z)s(1)\} \cup \{x(i)x(i + 1) | 1 \leq i < \#z\} & : \# z > 0
\end{cases}$$

ist stark zusammenhängend.

![Diagram](image)

Abb. 6.1

Beweis: Offensichtlich existiert ein Kreis, der die Knoten $s(i), t(i) \ 1 \leq i \leq p$ und $x(i) \ 1 \leq i \leq \#z$ enthält (s. Abbildung). Betrachte eine Quelle $s(j) \ p + 1 \leq j \leq \#s$. Mit der Bedingung ii) gibt es einen Weg zu einer Senke $t(i)$ auf dem Kreis und damit zu allen Knoten auf dem Kreis. $s(j)$ kann durch eine neue Kante vom Knoten $t(j)$ erreicht werden. Mit der Bedingung iii) erreichen wir den Knoten $t(j)$ von einer Quelle auf dem Kreis. Damit ist $s(j)$ von jedem Knoten erreichbar. Genauso erreicht $t(j) \ p + 1 \leq j \leq \#s$ einen Knoten auf dem Kreis. Damit gilt die Behauptung. \qed

Bemerkung 6.1 Die Menge $\tilde{E}_A$ enthält genau $\#t + \#q$ Kanten.

Die Augmentierung des Graphen bereitet also keine Probleme. Die Generierung der Menge $\tilde{E}_A$ ist linear abhängig von $|\tilde{V}|$ und damit von $n$. Auch die Bestimmung des Kondensationraphen läßt sich effizient mit der Variante des DFS-Verfahrens aus Kapitel 3 bestimmen. Bleibt also nur noch die Bestimmung einer Menge von Quellen und Senken, die den Bedingungen i) bis iii) genügt.

Die Idee besteht darin, die schwachen Zusammenhangskomponenten von $\tilde{G}$ zu betrachten. (Die schwachen Zusammenhangskomponenten sind die Zusammenhangskomponenten des zugrundeliegenden ungerichteten Graphen.) Besteht eine solche Komponente nicht nur aus einem Knoten, so gibt es nach dem Beweis von Lemma 6.2 eine Quelle und eine Senke in der Komponenten. Wir versuchen zuerst von allen
möglichen Quellen $s$ zu den Senken zu gelangen, treffen wir in eine noch nicht be-
suchte Senke $t$ ein, so nehmen wir das Paar $s, t$ in die gesuchte Menge $ST$ auf. Nach
 diesem Schritt ist es also möglich, von jeder Quelle zu einer Senke aus $ST$ zu kom-
 men (d.h. Bedingung ii) ist erfüllt). In einem zweiten Schritt werden wir ausgehend
 von den Senken $t$, die sich noch nicht in $ST$ befinden, einen Weg zu den Quellen
 suchen. Finden wir eine Quelle $s$, die schon in $ST$ enthalten ist, so gibt es einen Weg
 von $s \in ST$ zu der Senke $t$ und damit gilt iii). Finden wir eine Quelle $s$, die noch
 nicht in $ST$ liegt, so nehmen wir die Senke $t$ und die fundene Quelle $s$ in $ST$ auf.
 Somit haben wir eine Quelle $s \in ST$, von der es einen Weg zu $t$ gibt.

Das Verfahren, mit dem wir Quellen/Senken im Graphen suchen, ist dem DFS-
 Verfahren sehr verwandt. Der Unterschied besteht darin, daß der Rückschritt (der
 Schritt, wenn alle inzidenten Kanten eines Knotens benutzt sind, und das Zentrum
 der Aktivität in den Vorgängerknoten verschoben wird) hier fehlt.

**Algorithmus $ST$-Menge**

1. **TEIL 1**  
   $k(v) = 0 \quad \forall v \in V, \ i = 0, j = 0$.

2. Gibt es eine Quelle $s$ mit $k(s) = 0$, dann setze $v = s$.  
   Sonst gehe nach (8).

3. Setze $i = i + 1$, $k(v) = i$.

4. Ist $v$ eine Senke, dann gehe nach (7).

5. Wähle eine Kante $vw$.

6. Ist $k(w) \neq 0$, dann gehe nach (2).  
   Sonst setze $v = w$ und gehe nach (3).

7. Setze $j = j + 1$, $s(j) = s$, $t(j) = v$. Gehe nach (2).

8. **TEIL 2**  
   $k(v) = 0 \quad \forall v \in V, \ i = 0$.

9. Gibt es eine Senke $t$, die nicht aus $\{t(i)|1 \leq i \leq j\}$, und $k(t) = 0$,  
   dann setze $v = t$.  
   Sonst gehe nach (15).

10. Setze $i = i + 1$, $k(v) = i$.

11. Ist $v$ eine Quelle in $\{s(i)|1 \leq i \leq j\}$, dann gehe nach (9).  
    Ist $v$ eine Quelle aber nicht in $\{s(i)|1 \leq i \leq j\}$, dann gehe nach (14).

12. Wähle eine unbentutzte Kante $uv$ und erkläre sie für benutzt.
(13) Ist $k(w) \neq 0$, dann gehe nach (9).
Sonst gehe nach (10).

(14) Setze $j = j + 1$, $s(j) = v$, $t(j) = t$. Gehe nach (9).

(15) Setze $p = j$, STOP.

Lemma 6.5 Die Knoten aus $S = \{s(i) | 1 \leq i \leq p\}$ und $T = \{t(i) | 1 \leq i \leq p\}$
erfüllen die Bedingungen i)–iii).

Beweis: Zu i): Der Algorithmus konstruiert einen Weg von $s(i)$ nach $t(i)$ direkt
in Schritt (5) und (6) bzw. (12) und (13).

Zu ii): Sei $s$ eine Quelle, dann wird $s$ irgendwann in (2) gewählt. Wird im Verlauf
der Bearbeitung von $s$ der Schritt (4) bejaht, so ist eine neue Senke $t$ gefunden. $s$
und $t$ werden in $S$ und $T$ eingetragen und nach i) gibt es einen Weg zwischen ihnen.
Wird Schritt (4) immer verneint, so beendet der Algorithmus die Bearbeitung von
$s$ in Schritt (6). Wir treffen auf einen bereits markierten Knoten. Da der Graph $\tilde{G}$
kreisfrei ist, wurde dieser Knoten in einem vorherigen Schritt markiert. Von ihm
aus kommen wir zu einer Senke, die in einem vorherigen Schritt in die Menge $T$
aufgenommen wurde.

Zu iii): Sei $t$ eine Senke. Wurde $t$ im ersten Teil des Algorithmus markiert, so
finden wir direkt mit Schritt (5) und (6) (rückwärts) den Weg zu einer Quelle aus
$S$. Bleibt $t$ in Teil 1 unmarkiert, so wird $t$ in Schritt (9) gewählt. Von dort konstruieren wir einen Weg (rückwärts) zu einer Quelle $s$. Ist $s$ aus $S$, so haben wir einen
gewünschten Weg gefunden. Andernfalls fügen wir $s$ und $t$ zu $S$ und $T$ hinzu. Nach
i) gibt es einen Weg von $s$ nach $t$. \hfill \Box

Nach dem Lemma ist der Algorithmus korrekt. Da wir zweimal eine Variante des
DFS-Verfahrens anwenden, benötigt der Algorithmus $ST$–Menge die Komplexität
von $O(n + m)$.

Mit dem folgenden Algorithmus augmentieren wir einen Graphen $G$ zu einem
stark zusammenhängenden Graphen $G'$.

**STRONGAUGMENT**

(1) Erzeuge mit Hilfe einer DFS–Variante den Kondensationsgraphen $\tilde{G}$
von $G$.

(2) Wende den Algorithmus $ST$–Menge zur Bestimmung der Quellen und
Senken (mit den Eigenschaften i)–iii)) auf $\tilde{G}$ an.

(3) Konstruiere nach Satz 6.4 die augmentierende Kantenmenge $\tilde{E}_A$.

(4) Füge $\beta(\tilde{E}_A)$ zu $G$ hinzu.
(5) STOP.

Betrachten wir die Laufzeit des Algorithmus. Schritt (1) besitzt eine Komplexität von \(O(n + m)\) (s. Kapitel 3). Der Schritt (2) läuft ebenfalls in \(O(n + m)\) ab. Die Konstruktion der Kantenmenge der Mächtigkeit \(\text{max}(|s|, |t|) + |q|\) und das Hinzufügen der Kanten hängt linear von der Anzahl der Quellen, Senken und isolierten Knoten ab. Damit erhalten wir für die Schritte (3) und (4) eine Komplexität von \(O(n)\). Insgesamt ergibt das eine Komplexität von \(O(n + m)\).

Als nächstes wollen wir ein kleines Beispiel zur Verdeutlichung des Algorithmus bringen.

Betrachten wir folgenden Graphen:

![Graph](image)

Abb. 6.2

Setzen wir den Algorithmus ST auf Kondensationsgraphen an, so finden wir nur einen Quellknoten, den Knoten 1. Also ist $s(1) = 1$. Als Senke können wir sowohl Knoten 10, als auch Knoten 9 nehmen. Wählen wir Knoten 10 als $t(1)$, so wird Knoten 9 zu $x(1)$ und der isolierte Knoten 13 wird zu $x(2)$. Fügen wir nun die Kantenmenge $E_A$ (das sind hier die Kanten von 10 nach 9, von 9 nach 13 und von 13 nach 1) hinzu, so erhalten wir einen stark zusammenhängenden Graphen. Transformieren wir diese Kantenmenge mit Hilfe der Funktion $\beta$ in den Originalgraphen, so erhalten wir:
Kapitel 7

Augmentierung gerichteter Graphen

7.1 Der Satz von Frank

In Kapitel 5 haben wir das Hauptproblem kennengelernt. Formulieren wir es hier (in etwas anderer Form) erneut:

Hauptproblem: Sei $G = (V, E)$ ein gerichteter Graph. Gibt es eine Kantenmenge $F$ mit $|F| \leq \gamma$ für die $G' = (V, E \cup F)$ ein $k$-fach kantenzusammenhängender Graph ist.

Wir werden einen Satz, der eine Äquivalenz zwischen dem Hauptproblem und einer max-Formel aufzeigt, angeben. Der konstruktive Beweis zu diesem Satz führt zu einem stark polynomialen Algorithmus, dessen Implementation der letzte Abschnitt dieses Kapitels zum Inhalt hat.

Doch zuvor werden wir die Idee, die diesem Satz zugrunde liegt, näher erläutern. In Kapitel 6 haben wir den Kondensationsgraphen zu einem stark zusammenhängenden Graphen augmentiert. Wir haben die Quellen, Senken und isolierten Knoten geschickt angeordnet und den Defekt an ihnen behoben. Unter dem Defekt $d_{f_{\delta}(\delta)}(v)$ eines Knotens bezüglich einer gewünschten Zusammenhangszahl $k$ wollen wir die Anzahl der an $v$ fehlenden, eintreffenden (austretenden) Kanten verstehen, also $d_{f_{\delta}(\delta)}(v) = k - \rho(v)$ ($d_{f_{\delta}}(v)$ analog).

Bei der Augmentierung zu stark zusammenhängenden Graphen waren es gerade die Quellen (Senken) und isolierten Knoten im Kondensationsgraphen, die einen positiven $\rho$ ($\delta$)-Defekt hatten. Durch geschickte Hinzunahme von Kanten konnte der Defekt ausgeglichen werden. Dieses werden wir auch bei der Augmentierung zu $k$-fach kantenzusammenhängenden Graphen anwenden. Folgendes Beispiel belegt, daß wir nicht nur den Defekt von Knoten betrachten dürfen:
Der Defekt an den Knoten ist zwar überall 0, doch ist der Graph keinesfalls stark zusammenhängend. Im Kapitel 6 mußten wir zum Kondensationsgraphen übergehen, um die Augmentierung durchzuführen. Die Existenz von Kreisen erschwert die Suche nach den Mengen mit maximalem Defekt. Tatsächlich werden wir in Kapitel 8 sehen, daß bei kreisfreien Graphen die Betrachtung der Knotendefekte ausreicht. Im allgemeinen Fall muß jedoch der Defekt ganzer Knotenmengen betrachtet werden.

**Satz 7.1 ([FRANK90])** Sei \(G = (V, E)\) ein gerichteter Graph und sei \(k > 0\) eine ganze Zahl. \(G\) kann genau dann durch Hinzunahme von höchstens \(\gamma\) Kanten zu einem \(k\)-fach kanten zusammenhängenden Graphen gemacht werden, wenn für jede echte Teilpartition \(\{X_1, X_2, \ldots, X_t\}\) von \(V\) gilt:

\[
S_\phi(X_1, \ldots, X_t) := \sum_{i=1}^{t}(k - \phi(X_i)) \leq \gamma \quad \text{und}
\]

\[
S_\delta(X_1, \ldots, X_t) := \sum_{i=1}^{t}(k - \delta(X_i)) \leq \gamma
\]

**Bemerkung 7.1** Das Hauptproblem kann nun gelöst werden, indem entschieden wird ob

\[
S_{\max, \phi} := \max(S_\phi(X_1, \ldots, X_t)|\{X_1, \ldots, X_t\} \text{ echte Teilzerlegung von } V) \leq \gamma
\]

und

\[
S_{\max, \delta} := \max(S_\delta(X_1, \ldots, X_t)|\{X_1, \ldots, X_t\} \text{ echte Teilzerlegung von } V) \leq \gamma
\]

gilt. Suchen wir die minimale Anzahl \(\gamma\) von Kanten, die zum Graphen hinzugekommen werden muß, damit er \(k\)-fach kanten zusammenhängend wird, so erhalten wir folgende min–max–Formel:

\[
\gamma = \min(\gamma' | S_{\max, \phi} \leq \gamma' \text{ und } S_{\max, \delta} \leq \gamma')
\]
Beweis zu Satz 7.1: 1) Sei \( G' = (V, E \cup F) \) ein \( k \)-fach kantenzusammenhängender Obergraph von \( G \) wobei \( F \) die Menge der neuen Kanten bezeichnet. Sei \( \{X_1, \ldots, X_t\} \) eine beliebige echte Teilzerlegung von \( V \). Da \( G' \) \( k \)-fach kantenzusammenhängend ist, treten in jede Teilmenge \( X_i \) mindestens \( k - g(X_i) \) neue Kanten ein. Damit gilt (7.1). (7.2) folgt analog.

2) Die andere Richtung ist wesentlich schwieriger zu beweisen. Daher teilen wir den Beweis auf und fügen die Ergebnisse später zusammen.

Lemma 7.2 Seien \( A, B \subseteq V \) mit \( g(A) = g(B) = k \leq \min(g(A \cup B), g(A \cap B)) \). Dann gilt: \( g(A \cup B) = g(A \cap B) = k \) und \( d(A, B) = 0 \).

Beweis: Nach Lemma (2.1) gilt: \( k + k = g(A) + g(B) = g(A \cup B) + g(A \cap B) + d(A, B) \geq k + k + d(A, B) \geq k + k \). Daraus folgt, daß \( d(A, B) = 0 \) und damit \( g(A \cup B) = g(A \cap B) = k \).

Wir arbeiten mit einem erweiterten Graphen \( G' \). Die Gradfunktionen \( g \) und \( \delta \) bezüglich \( G' \) heißen \( g' \) und \( \delta' \).

Lemma 7.3 \( G \) kann durch Hinzunahme eines neuen Knotens \( s, \gamma \) (aus Satz 7.1) neuen Kanten, die in \( s \) eintreten, und \( \gamma \) neuen Kanten, die \( s \) verlassen, zu einem Graphen \( G' = (V \cup s, E') \) gemacht werden, so daß für jede Teilmenge \( \emptyset \subset X \subset V \) folgendes gilt:

\[
\begin{align*}
(7.3) & \quad g'(X) \geq k \\
(7.4) & \quad \delta'(X) \geq k
\end{align*}
\]

Das Lemma garantiert den \( k \)-fachen Kantenzusammenhang zwischen Knoten aus \( V \). Die kantendisjunkten Wege können den Knoten \( s \) enthalten.

Definition 7.1 Eine Menge \( \emptyset \subset X \subset V \) heißt \( g'-kritisch \) (\( \delta'-kritisch \)) genau dann, wenn \( g'(X) = k \) (\( \delta'(X) = k \)) ist. Eine Menge \( \emptyset \subset X \subset V \) heißt kritisch genau dann, wenn sie \( g'\)-kritisch oder \( \delta'\)-kritisch ist.

Beweis von Lemma 7.3: Wir fügen eine Anzahl von Kanten zu \( G \) hinzu, so daß (7.3) erfüllt ist. Das können wir erreichen, indem wir \( k \) parallele Kanten von \( s \) zu jedem Knoten \( v \in V \) zu \( G \) hinzufügen. Nun entfernen wir, falls möglich, die neu hinzugefügten Kanten nacheinander, ohne die Bedingung (7.3) zu verletzen. Nach der Entfernung aller überflüssigen Kanten erhalten wir den Graphen \( G' = (V \cup s, E') \) und es gilt: \( \delta'(s) \leq \gamma \). Denn: Sei \( S = \{v \in V | sv \in E'\} \). Eine Kante \( sv \) kann nicht entfernt werden, wenn \( v \) in einer \( g'\)-kritischen Menge liegt. Da keine Kante mehr entfernt werden kann, treten alle Kanten, die \( s \) verlassen, in eine \( g'\)-kritische Menge ein. Wir können also die Menge \( S \) durch eine Familie \( \mathcal{F} = \{X_1, \ldots, X_t\} \) von \( g'\)-kritischen Mengen überdecken. Wir wählen eine Familie für die \( t \) minimal ist.
Fall 1) \( \mathcal{F} \) besteht nur aus disjunkten Mengen. Dann gilt:

\[
kt = \sum_{i=1}^{t} \delta'(X_i) = \sum_{i=1}^{t} \delta(X_i)
\]

Nach \( \delta'(s) \) aufgelöst und die Voraussetzung (7.1) eingesetzt ergibt:

\[
\delta'(s) = \sum_{i=1}^{t} (k - \delta'(X_i)) \leq \gamma
\]

Fall 2) Es gibt \( A, B \in \mathcal{F} \) mit \( A \cap B \neq \emptyset \) und \( A \cup B = V \). \( \{V - A, V - B\} \) ist eine Teilpartition von \( V \) also gilt (7.2):

\[
\gamma \geq k - \delta(V - A) + k - \delta(V - B)
= k - \delta(A) + k - \delta(B)
\geq k - \delta'(A) + k - \delta'(B) + \delta'(s)
= \delta'(s)
\]

Die letzte Ungleichung gilt, da die Kanten aus der Schnittmenge von \( A \) und \( B \) doppelt subtrahiert, aber nur einfach addiert werden.

Fall 3) Es gibt \( A, B \in \mathcal{F} \) mit \( A \cap B \neq \emptyset \) und \( A \cup B \neq V \). Nach Lemma 7.2 ist auch \( A \cup B \) \( \delta' \)-kritisch. Das Entfernen der Mengen \( A \) und \( B \) und das Hinzufügen der Menge \( A \cup B \) zu \( \mathcal{F} \) führt zu einer kleineren Familie. Dies steht im Widerspruch zur Minimalität von \( t \).

Wir haben gezeigt, daß durch Hinzunahme von \( \gamma \) Kanten, die \( s \) verlassen, die Bedingung (7.3) für alle \( \emptyset \subset X \subset V \) erfüllt ist. Orientieren wir alle Kanten aus \( G \) um, so erhalten wir einen Graphen \( \bar{G} \). Auch für diesen Graphen finden wir \( \gamma \) Kanten, so daß (7.3) gilt. Orientieren wir die Kanten des so erweiterten Graphen wieder um, so erhalten wir den Graphen \( \bar{G} \). Die aus \( s \) austretenden \( \gamma \) Kanten werden zu den in \( s \) eintretenden Kanten. Da für \( \bar{G} \) (7.3) galt, so gilt für \( \bar{G} \) (7.4). \( \bar{G} \) eingeschränkt auf \( V \) ist gerade unser Graph \( G \). Wir haben also \( \gamma \) in \( s \) eintreffende Kanten gefunden, so daß (7.4) gilt.

\[\square\]

**Definition 7.2** Sei \( G = (V, E) \) ein gerichteter Graph. Sei \( uv, vw \in E \) ein Kantennpaar. Das Entfernen der Kanten \( uv \) und \( vw \) aus \( E \) und das Hinzufügen der Kante \( uw \) zu \( E \) heißt Abspalten des Kantennpaares \( uv, vw \).

Wir werden diese Abspaltungsoperation auf die zu \( s \) inzidenten Kanten anwenden, bis \( s \) ein isolierter Knoten ist. Dazu müssen wir aber zuerst garantieren, daß \( \delta'(s) = \delta'(s) \) gilt. Sei oBdA \( \gamma \geq \delta'(s) > \delta'(s) \). Dann können wir zu \( G' \) \( \delta'(s) - \delta'(s) \)
Kanten von einem Knoten \( v_0 \in V \) zu \( s \) ziehen, ohne die Aussage des Lemmas zu verletzen.

Um die Aussage des Satzes 7.1 zu beweisen, müssen wir garantieren, daß wir alle mit \( s \) inzidenten Kanten abspalten können.

**Satz 7.4 ([MADER82])** Sei \( G' = (V+s, E') \) ein Graph, für den \( \delta'(s) = \varrho'(s) \) gilt. Weiterhin gelte (7.3) und (7.4) für alle \( \emptyset \subset X \subset V \). Dann gibt es ein Kantenpaar \( vs, st \), das abgespalten werden kann, ohne (7.3) und (7.4) zu verletzen.

Nach einem korrekten Abspalten sind die Voraussetzungen des Satzes 7.4 für den resultierenden Graphen weiterhin erfüllt. Wir können den Satz wiederholt anwenden. Nach höchstens \( \gamma \) Anwendungen von Satz 7.4 ist \( s \) ein isolierter Knoten. Entfernen wir \( s \) aus diesem Graphen, so erhalten wir einen Graphen \( G^* = (V, E \cup F) \). Für diesen Graphen gelten (7.3) und (7.4), d.h. der Graph ist \( k \)-fach kantenzusammenhängend. Damit ist der zweite Teil des Beweises von Satz 7.1 erledigt.

Für den Beweis von Satz 7.4 benötigen wir einige Vorbereitungen:

**Lemma 7.5** Sei \( G' = (V+s, E') \) ein Graph für den \( \delta'(s) = \varrho'(s) \) gilt. Weiterhin gelte (7.3) und (7.4) für alle \( \emptyset \subset X \subset V \). Seien \( X, Y \subset V \) zwei sich schneidende Mengen mit \( X \cap Y = s \) und \( \delta'(X) = \delta'(Y) = k \). Dann gilt: \( \delta'(X-Y) = \delta'(Y-X) = k \) und \( \tilde{d}(X,Y) = 0 \).

**Beweis:** Nach Lemma 2.1 gilt: \( k + k = \delta'(X) + \delta'(Y) = k \delta'(X-Y) + \delta(X-Y) + \delta'(Y-X) + \tilde{d}(X,Y) \geq k + k + \tilde{d}(A,B) \geq k + k \). Daraus folgt, daß \( \tilde{d}(X,Y) = 0 \) und damit \( \delta'(X-Y) = \delta'(Y-X) = k \).

**Lemma 7.6** Seien \( A, B \subset V \) zwei sich schneidende kritische Mengen. Dann gilt entweder (i) \( A \cup B \) ist kritisch oder (ii) \( B - A \) ist kritisch und \( \tilde{d}(A,B) = 0 \).

**Beweis:**

Fall 1) \( A, B \) \( \varrho' \)-kritisich und \( A \cup B \subset V \): Da (7.3) und (7.4) gelten, folgt \( k \geq \min(\varrho'(A \cap B), \varrho'(A \cup B)) \). Damit ist Lemma 7.2 anwendbar und wir erhalten die Alternative (i).

Fall 2) \( A, B \) \( \varrho' \)-kritisich und \( A \cup B \subset V \): Setze \( X = (V+s) - A \) und \( Y = (V+s) - B \). \( X \) und \( Y \) sind zwei sich schneidende \( \delta \)-kritische Mengen mit \( X \cap Y = s \). Damit ist Lemma 7.5 anwendbar. Wir erhalten: \( X - Y = B - A \) ist kritisch und

\[ \tilde{d}(X,Y) = \tilde{d}(A, (V+s) - B) = \tilde{d}(A \cup B, s) = \tilde{d}(A,B) = 0 \]

und somit die Alternative (ii).
Fall 3) \( A, B \delta'\)-kritisch: analog 1) und 2).

Fall 4) \( A \varrho'\)-kritisch, \( B \delta'\)-kritisch: Wende das Lemma 7.2 auf \( A, (V + s) - B \) an. Das Lemma besagt, daß \( A \cap ((V + s) - B) \varrho'\)-kritisch ist und \( \delta'(A, (V + s) - B) = \tilde{d}(A, B) = 0 \). Wenn \( A \cap ((V + s) - B) \varrho'\)-kritisch ist, dann ist \( B - A \delta'\)-kritisch. Somit gilt die Alternative (ii).

Fall 5) \( A \delta'\)-kritisch, \( B \varrho'\)-kritisch: analog 4). \( \square \)

Beweis von Satz 7.4: Ein Kantenspaar \( vs, st \) kann genau dann nicht abgespalten werden, wenn es eine kritische Menge gibt, die sowohl \( v \) als auch \( t \) enthält. Wir wählen die Kante \( st \) fest und finden eine Kante \( vs \), so daß das Paar \( vs, st \) abgespalten werden kann. Gibt es keine kritische Menge, die \( t \) enthält, so kann also jede beliebige Kante \( vs \) genommen werden. Gibt es kritische Mengen \( A, B \), die \( t \) enthalten, so ist nur die Alternative (i) aus Lemma 7.6 möglich, da \( \tilde{d}(A, B) \neq 0 \). Wir können also alle kritischen Mengen, die \( t \) enthalten, zu einer großen kritischen Menge \( M \) vereinigen. Gibt es nun eine Kante \( vs \) mit \( v \in V - M \), so ist die gesuchte Kante gefunden.

Annahme: Es gibt keine Kante \( vs \) mit \( v \in V - M \).

Fall 1) \( M \) ist \( \varrho' \)-kritisch: Es treten also genau \( k \) Kanten in \( M \) ein. Eine Kante kommt vom Knoten \( s \), also treten aus \( V - M \) höchstens \( k - 1 \) Kanten nach \( M \) ein. Da nach Voraussetzung keine Kante von \( V - M \) nach \( s \) geht, gilt folgende Ungleichung: \( \delta'(V - M) < \varrho'(M) = k \). Widerspruch zu (7.4).

Fall 2) \( M \) ist \( \delta' \)-kritisch: Betrachten wir die Zahl der in \( V - M \) eintreffenden Kanten. Alle in \( s \) eintreffenden Kanten kommen laut Annahme aus \( M \). Da \( M \delta'\)-kritisch ist, laufen genau \( k - \varrho'(s) \) Kanten von \( M \) nach \( V - M \). Da aber mindestens eine Kante von \( s \) nach \( M \) verläuft, gehen höchstens \( \delta'(s) - 1 = \varrho'(s) - 1 \) Kanten von \( s \) nach \( V - M \). Es gilt also \( \varrho'(V - M) < k \). Widerspruch zu (7.3). \( \square \)

Bemerkung 7.2 Die Sätze 7.1 und 7.4 sind zwar "nur" Existenzsätze, aber die zugehörigen Beweise zeigen, wie man den gesuchten Graphen konstruieren kann. Mit dem abgeleiteten Algorithmus und einigen Erweiterungen beschäftigen wir uns im nächsten Abschnitt.
7.2 Der Algorithmus von Frank


Seien $f_{\delta}(\delta) : V \rightarrow N$ und $g_{\delta}(\delta) : V \rightarrow N \cup \infty$, mit $f_{\delta}(\delta) \leq g_{\delta}(\delta)$. (Zur Erinnerung: $f_{\delta}(X) = \sum_{v \in X} f_{\delta}(v)$.)

**Satz 7.7 ([FRANK90])** Sei $G = (V, E)$ ein gerichteter Graph und sei $k > 0$ eine ganze Zahl. $G$ kann genau dann durch Hinzunahme einer Kantenmenge $F$ von genau $\gamma$ Kanten zu einem $k$-fach kantenzusammenhängenden Graphen gemacht werden, so daß

\[
\begin{align*}
  f_{\delta}(v) &\leq g_{\delta}(v) \\
  f_{\delta}(v) &\leq \delta_F(v) \leq g_{\delta}(v)
\end{align*}
\]

gilt, wenn

\[
\begin{align*}
  k - \varrho(X) &\leq g_{\delta}(X) \\
  k - \delta(X) &\leq g_{\delta}(X)
\end{align*}
\]

für jedes $\emptyset \subset X \subset V$ und

\[
\begin{align*}
  \sum_{i=1}^{t} (k - \varrho(X_i)) + f_{\delta}(X_0) &\leq \gamma \\
  \sum_{i=1}^{t} (k - \delta(X_i)) + f_{\delta}(X_0) &\leq \gamma
\end{align*}
\]

für jede echte Teilpartition $\{X_0, X_1, \ldots, X_t\}$ von $V$, wobei $X_0$ leer sein kann, gilt.

**Bemerkung 7.3** Um den Anforderungen eines Knotens zu genügen, d.h. um mindestens $f_{\delta}(\delta)(v)$ neue Kanten zum Graphen hinzuzufügen, kann es erforderlich sein, Schleifen (also Knoten mit Anfangs- und Endknoten $v$) zum Graphen hinzuzufügen.

Beweis: 1) Sei $F$ die gewünschte Menge der neuen Kanten. Da $G = (V, E \cup F)$ $k$-fach kantenzusammenhängend ist, gilt:

\[
  k \leq g(\delta)(X) + g(\delta)_F(X) \leq g(\delta)(X) + \sum_{v \in X} g(\delta)_F(v) \leq g(\delta)(X) + g_{\delta}(\delta)(X)
\]
Damit folgt (7.6). Aus dem gleichen Grund und wegen \( g(\delta)_F(v) \geq f_{g(\delta)}(v) \) folgt für jede echte Teilpartition \( \{X_0, X_1, \ldots, X_t\} \) von \( V \), wobei \( X_0 \) leer sein kann:

\[
\gamma \geq \sum_{i=0}^{t} g(\delta)_F(X_i) \geq \sum_{i=1}^{t} (k - g(\delta)(X_i)) + f_{g(\delta)}(X_0)
\]

Damit folgt (7.7).

2) Der Beweis verläuft analog zum zweiten Teil des Beweises von Satz 7.1. Wir fügen einen neuen Knoten \( s \) zu \( V \) hinzu. Dann fügen wir sowie zu \( s \) inzidente Kanten hinzu, daß (7.3) und (7.4) gelten. (Da diese möglich ist, sichert uns die Bedingung (7.6).) Danach löschen wir alle "überflüssigen" Kanten, ohne den Kantenanzahlvergleich zwischen Knoten aus \( V \) zu stören. Wir erhalten den Graphen \( G' = (V + s, E') \). Ist nach dem Löschen \( \gamma \geq \delta'(s) = g'(s) \), so können wir den Satz 7.4 wiederholt anwenden und erhalten den gewünschten Graphen.

Probleme bekommen wir dann, wenn \( g'(s) > \gamma \) oder \( \delta'(s) > \gamma \) gilt. Nehmen wir also an, daß \( \delta'(s) > \gamma \) gilt. (Der andere Fall verläuft analog.) Eine Kante \( sv \) kann nicht entfernt werden, wenn sie entweder in eine \( g' \)-kritische Menge eintritt oder \( f_{\delta}(v) = g_{F'}(v) \) gilt. Das heißt also, daß alle Knoten, für die \( f_{\delta}(v) < g_{F'}(v) \) gilt, in einer \( g' \)-kritischen Menge liegen. Wir finden eine Familie \( \mathcal{F} = \{X_1, \ldots, X_t\} \) von \( g' \)-kritischen Mengen, die all diese Knoten überdeckt. Sei \( \mathcal{F} \) eine solche Familie für die \( t \) minimal ist.

Fall 1) \( \mathcal{F} \) besteht nur aus disjunkten Mengen. Dann gilt nach (7.7) (mit \( X_0 = \bigcup_{i=1}^{t} X_i \)) \( \gamma \geq \sum_{i=1}^{t} (k - g(X_i)) + f_{\delta}(X_0) \geq \delta'(s) > \gamma \). Widerspruch.

Fall 2) Es gibt \( A, B \in \mathcal{F} \) mit \( A \cap B \neq \emptyset \) und \( A \cup B = V \). \( \{A \cap B, V - A, V - B\} \) ist eine Teilpartition von \( V \) also gilt (7.7):

\[
\gamma \geq k - \delta(V - A) + k - \delta(V - B) + f_{\delta}(A \cap B)
\]

\[
= k - g(A) + k - g(B) + f_{\delta}(A \cap B)
\]

\[
\geq k - g'(A) + k - g'(B) + \delta'(s)
\]

\[
= \delta'(s) > \gamma
\]

Die letzte Ungleichung gilt, da die Kanten aus der Schnittmenge von \( A \) und \( B \) doppelt subtrahiert werden, aber nur einfach addiert werden. Auch in diesem Fall erhalten wir einen Widerspruch.

Fall 3) Es gibt \( A, B \in \mathcal{F} \) mit \( A \cap B \neq \emptyset \) und \( A \cup B \neq V \). Nach Lemma 7.2 ist auch \( A \cup B \) \( g' \)-kritisch. Das Entfernen der Mengen \( A \) und \( B \) und das Hinzufügen der Menge \( A \cup B \) zu \( \mathcal{F} \) führt zu einer kleineren Familie. Dies steht im Widerspruch zur Minimalität von \( t \).
Wir haben in Kapitel 5 gesehen, daß die Version des Augmentierungsproblems mit Gewichten für die Kanten NP-vollständig war. Dies galt sogar für den Fall \( k = 2 \).
Wir können jedoch auch den Knoten ein Gewicht zuordnen. Sei \( a_g : V \rightarrow R^+_0 \) eine Kostenfunktion. Wir suchen nun eine Kantenmenge \( F \), mit \( |F| \leq \gamma \), die zu einem gegebenen Graphen \( G = (V, E) \) hinzugefügt, ihn \( k \)-fach kanten zusammenhängend macht und \( \sum_{v \in V} (\varphi_F(v) a_g(v) + \delta_F(v) a_\delta(v)) \) minimiert. Wir werden sehen, daß das Entfernen der jeweils teuersten überflüssigen Kante (im Beweis von Satz 7.1) zu einer optimalen Lösung führt. Dies entspricht einem worst–out–Greedy–Algorithmus [KORTE84].

**Algorithmus von Frank**

1. Füge einen Knoten \( s \) zu \( V \) hinzu. Füge für alle \( v \in V \) \( k \) parallele Kanten \( sv \) zum Graphen hinzu. Sortiere die neuen Kanten nach absteigenden Knotenkosten des Endknoten. Gehe in dieser Reihenfolge durch die neuen Kanten und streiche eine Kante heraus, falls durch ihr Streichen die Bedingung (7.3) nicht verletzt wird. \( \gamma_1 \) bezeichne die Zahl der restlichen Kanten.

2. Füge für alle \( v \in V \) \( k \) parallele Kanten \( vs \) zum Graphen hinzu. Sortiere die neuen Kanten nach absteigenden Knotenkosten des Anfangsknoten. Gehe in dieser Reihenfolge durch die neuen Kanten und streiche eine Kante heraus, falls durch ihr Streichen die Bedingung (7.4) nicht verletzt wird. \( \gamma_2 \) bezeichne die Zahl der restlichen Kanten.

3. Sei \( \gamma = \max(\gamma_1, \gamma_2) \). Ist \( \gamma_2 < \gamma_1 \), so füge \( \gamma_1 - \gamma_2 \) parallele Kanten von \( u \in V \) nach \( s \) hinzu. \( u \) ist der Knoten, für den \( a_\delta \) das Minimum annimmt. Ist \( \gamma_2 > \gamma_1 \), so füge \( \gamma_2 - \gamma_1 \) parallele Kanten von \( s \) nach \( u \in V \) hinzu. \( u \) ist der Knoten, für den \( a_\delta \) das Minimum annimmt.

4. Bezeichne \( G' \) den resultierenden Graph. In \( G' \) gilt \( \delta'(s) = \varphi'(s) \) und (7.3) und (7.4) gelten. Wir können die mit \( s \) inzidenten Kanten nach Satz 7.4 abspalten. Wir erhalten den Graphen \( G_{opt} = (V, E \cup F) \).

**Satz 7.8** Der im Algorithmus von Frank erzeugte Graph \( G_{opt} \) ist \( k \)-fach kanten zusammenhängend und Knoten–kosten–minimal.

7.3 Greedy–Algorithmus und Kontrapolymatroiden

In diesem Kapitel werden wir die Kontrapolymatroiden einführen und zeigen, daß der Greedy–Algorithmus eine Minimierungsaufgabe über diesen Kontrapolymatroiden korrekt löst. Im zweiten Teil werden wir zeigen, daß die Gradvektoren \( z_S, z_v \in \mathbb{R}^n \) aller Kantenmenge \( F (z_S[v] = \delta_F(v), z_v[v] = \varrho_F(v)) \), die einen Graphen \( G = (V, E) \) zu einem \( k \)-fach kantenzusammenhängenden Graphen machen, ein Kontrapolymatroid aufspannen. Da der Algorithmus von Frank nichts anderes ist als ein Greedy–Algorithmus, folgt sofort der Satz 7.8.

Sei im folgenden \( a : V \rightarrow R^+_n \) stets eine Kostenfunktion. Wir stellen uns die Menge \( V \) angeordnet vor und zwar so, daß gilt: \( a(v_1) \geq a(v_2) \geq \ldots \geq a(v_n) \). Unser Ziel ist es, die Funktion \( a^T x \), mit \( x \in \mathbb{R}^n \), über einem Kontrapolymatroid zu minimieren.

**Definition 7.3** Sei \( p : 2^V \rightarrow Z \) eine supermodulare Funktion mit \( p(\emptyset) = 0 \). Das Polyeder \( C(p) = \{ x \in \mathbb{R}^n | x(A) \geq p(A) \ \forall A \subseteq V \} \) heißt Kontrapolymatroid.

**Bemerkung 7.4** Die Notation \( x(A) \) ist allgemein üblich und bedeutet \( \chi(A)^T x \), \( \chi(A) \in R^n \) ist der Indikationsvektor von \( A \), d.h.

\[
(\chi(A))[i] = \begin{cases} 
1 & : v_i \in A \\
0 & : \text{sonst}
\end{cases}
\]

**Satz 7.9** Sei \( C(p) \) ein Kontrapolymatroid. Sei \( X_0 = \emptyset \) und \( X_i = \{v_1, \ldots, v_i \} \) für \( i = 1, \ldots, n \). Der Vektor \( z \in \mathbb{R}^n \) mit \( z[i] = p(X_i) - p(X_{i-1}) \) für \( i = 1, \ldots, n \) ist eine optimale Lösung zu \( \min(a^T x | x \in C(p)) \).

Beweis: 1) \( z \in C(p) \). Wir müssen zeigen, daß \( z(A) \geq p(A) \) gilt. Wir führen einen Induktionsbeweis über die Mächtigkeit von \( A \). Für \( A = \emptyset \) gilt nach Definition von \( p: p(\emptyset) = 0 = z(\emptyset) \). Sei nun die Behauptung für alle Mengen \( A' \) mit \( |A'| < j \) richtig. Sei \( A \) beliebig mit \( |A| = j \). Es gilt: \( z(X_i) = \sum_{i=1}^n p(X_i) - p(X_{i-1}) = p(X_i) \) für \( i = 1, \ldots, n \). Sei \( i \) der größte Index für den \( v_i \in A \) gilt.

\[
p(A) + p(X_{i-1}) \leq^1 p(A \cap X_{i-1}) + p(A \cup X_{i-1})
= p(A - v_i) + p(X_i)
\leq^2 z(A - v_i) + z(X_i)
= z(A) + z(X_{i-1})
\]

Das ist äquivalent mit \( p(A) \leq z(A) \). Ungleiche 1 folgt wegen der Supermodularität und wegen der Induktionsannahme und der Vorüberlegung \( p(X_i) = z(X_i) \) folgt Ungleiche 2.

2) Zu zeigen bleibt, daß der Vektor \( z \) die lineare Optimierungsaufgabe

\[
\min(a^T x | x(A) \geq p(A) \ \forall A \subseteq V)
\]
löst. Dazu betrachten wir die duale Optimierungsaufgabe:

$$\max(\mathbf{y}^T \mathbf{p} | \mathbf{y} \in \mathbb{R}^n_+, \sum_{A \in 2^V} (\chi(A))[i]y[A] = a[i] \text{ für } i = 1, \ldots, n, \ y \geq 0)$$

Für den Vektor $y^*$ mit

$$y^*[A] = \begin{cases} a(v_n) & : A = V \\ a(v_i) - a(v_{i-1}) & : A = X_i, \ i = 1, 2, \ldots, n - 1 \\ 0 & : \text{sonst} \end{cases}$$

ist $\sum_{A \in 2^V} (\chi(A))[i]y^*[A] = \sum_{i=1}^n y^*[X_j] = a(v_i) = a[i]$. Wir sehen, daß der Vektor $y^*$ dual zulässig ist. Weiter gilt:

$$a^Tz = \sum_{i=1}^n a(v_i)(p(X_i) - p(X_{i-1})),$$

$$= \sum_{i=1}^{n-1} p(X_i)(a(v_i) - a(v_{i+1})) + p(X_n)a(v_n),$$

$$= y^*^T \mathbf{p}.$$  

Wir haben eine primal zulässige Lösung $z$ und eine dual zulässige Lösung $y^*$. Weiterhin gilt $a^Tz = y^*^T \mathbf{p}$. Damit sind die Voraussetzungen für den schwachen Dualitätssatz (der linearen Optimierung) [CHVATAL83] erfüllt und wir erhalten, daß $z$ und $y^*$ jeweils optimale Lösungen der betreffenden Optimierungsaufgabe sind. □

Kennen wir die supermodulare Funktion $p$ und können wir sie effizient berechnen, so läßt sich der optimale Vektor leicht bestimmen. Es gelingt uns mit Hilfe des Satzes 7.9 eine Bestimmung der optimalen Lösung zu finden, ohne $p$ zu kennen:

**Korollar 7.10** Sei $C(p)$ ein Kontrapolymatroid. Der rekursiv definierte Vektor $z' \in \mathbb{R}^n$ mit

$$z'[i] = \min \{x[i] | x \in C(p) \text{ und } x[j] = z'[j] \text{ für } j = 1, \ldots, i - 1\}$$

ist eine optimale Lösung zu $\min(a^Tz | x \in C(p))$.

Beweis: Wir zeigen die Gleichheit des in Satz 7.9 bestimmten Vektors $z$ und des Vektors $z'$ durch vollständige Induktion über $i$. Die Definition der Mengen $X_i$ übernehmen wir aus Satz 7.9.

**Induktionsanfang**: $z'[1] = \min \{x[1] | x \in C\} \leq z[1]$, da $z \in C$. Umgekehrt gilt: $z[1] = p(X_1) \leq z'(X_1) = z'[1]$. Damit gilt $z[1] = z'[1]$.
Induktionsvoraussetzung: $z[j] = z'[j]$ für alle $1 \leq j < i$
Induktionsschluß:

\[
z'[i] = \min(x[i]|x \in C(p) \text{ und } x[j] = z'[j] \text{ für } j = 1, \ldots, i - 1) \\
= \min(x[i]|x \in C(p) \text{ und } x[j] = z[j] \text{ für } j = 1, \ldots, i - 1) \\
\leq z[i]
\]

Umgekehrt gilt:

\[
z[i] = p(X_1) - p(X_{i-1}) \leq z'(X_i) - p(X_{i-1}) = z'[i] + z'(X_{i-1}) - p(X_{i-1}) \\
= z'[i] + z(X_{i-1}) - p(X_{i-1}) = z'[i]
\]

Damit gilt $z[i] = z'[i]$.

Sei $C = \{x \in \mathbb{R}^n|x(A) \geq q(A)\}$ mit $q : 2^V \rightarrow N$ ein Kontrapolymatroid (d.h. es existiert eine supermodulare Funktion $p$, für die $C(p) = C$ gilt). Sei $k = \max(q(X)|X \subseteq V)$. Dann liegt der Vektor $(k, k, \ldots, k)$ in $C$. Aus Korollar 7.10 können wir das folgende Korollar ableiten:

**Korollar 7.11** Sei $C$ ein Kontrapolymatroid (wie oben definiert). Der rekursiv definierte Vektor $z^* \in \mathbb{R}^n$ mit

\[
z^*[i] = \min(x[i]|x \in C, x[j] = z^*[j] \text{ für } j = 1, \ldots, i-1, x[l] = k \text{ für } l = i+1, \ldots, n)
\]

ist eine optimale Lösung zu $\min(a^T x|x \in C)$.

Beweis: Wir zeigen $z^* = z'$ (aus Korollar 7.10). Wir benutzen die Mengen $X_i$ aus Satz 7.9.

Annahme: $z^* \neq z'$, dann gibt es eine Komponente mit dem kleinsten Index $t$, für die $z^*[t] \neq z'[t]$ gilt. Nach Definition von $z'$ gilt offensichtlich:

\[
z'[t] < z^*[t]
\]

Da $z^*[t]$ nicht weiter reduziert werden kann, gibt es eine Menge $A$ mit $v_t \in A$ und $q(A) = z^*(A) = z^*(X_t \cap A) + k|A - X_t|$

Fall 1) $A \subseteq X_t$: Dann gilt: $q(A) = z^*(A) > z'(A)$. Das widerspricht $z' \in C$.

Fall 2) $A \not\subseteq X_t$: Dann gilt $q(A) \geq z^*[t] + k \geq q(A)$, da $k = \max(q(X)|X \subseteq V)$ ist. Damit gilt: $0 = z^*[t] > z'[t] \geq 0$. Widerspruch.


Sei $G = (V, E)$ ein Graph, der mit $\gamma$ neuen Kanten zu einem $k$–fach kantenzusammenhängenden Graphen gemacht werden kann.
Satz 7.12 ([FRANK92]) \( C_q = \{ z \in \mathbb{R}^n | z(X) \geq q(X) \ \forall X \subseteq V \} \) mit
\[
q(X) := \begin{cases} 
\max(0, k - q(X)) : X \subset V \\
0 : X = \emptyset \\
\gamma : X = V
\end{cases}
\]
ist ein Kontrapolymatroid \( C(p_\varrho) \) mit
\[
p_\varrho(A) = \max\left( \sum_{i=1}^{t} q(A_i) \right) | \{ A_1, \ldots, A_t \} \text{ ist Teilpartition von } A
\]

Beweis: Wir betrachten eine zu \( q \) sehr ähnliche Funktion \( q' \):
\[
q'(X) := \begin{cases} 
k - q(X) : \emptyset \subset X \subset V \\
0 : \text{sonst}
\end{cases}
\]

Zwischen \( q \) und \( q' \) gilt folgender Zusammenhang: Sei \( \{ X_1, \ldots, X_t \} \) eine echte Teilpartition von \( X \subseteq V \), dann gibt es eine Teilpartition \( \{ X'_1, \ldots, X'_t \} \) von \( X \) mit \( \sum_{i=1}^{t} q(X_i) = \sum_{i=1}^{t} q'(X'_i) \). \( \{ X'_1, \ldots, X'_t \} \) entsteht durch Streichen der \( X_i \) mit \( k - q(X_i) < 0 \) aus \( \{ X_1, \ldots, X_t \} \). Das Gleiche gilt auch umgekehrt. Im folgenden benutzen wir \( p \) und \( C \) statt \( p_\varrho \) und \( C_\varrho \). Es treten im folgenden häufig Summen über Teilpartitionen auf. Der Buchstabe \( X \) steht für eine Teilpartition \( \{ X_1, \ldots, X_t \} \).

1) Es gilt \( p(V) = \gamma \). Denn:
\[
p(V) = \max\left( \sum_{i=1}^{t} q(X_i) \right) | X \text{ Teilpartition von } V \geq q(V) = \gamma
\]

Annahme: \( p(V) > \gamma \). Das Maximum aus (7.8) wird dann von einer echten Teilpartition \( X \) angenommen. Dann folgt mit dem Satz 7.1:
\[
\gamma < p(V) = \sum_{i=1}^{t} q(X_i) = \sum_{i=1}^{t} q'(X'_i) \leq \gamma
\]
Widerspruch. Wir bemerken noch, daß \( p(\emptyset) = 0 \) gilt.

2) Es gilt \( C(p) = C \): a) \( C(p) \subseteq C \): Sei \( x \in C(p) \), dann gilt \( x(A) \geq p(A) \). Wenn wir zeigen, daß \( p(A) \geq q(A) \) gilt, so ist auch \( x \in C \). Ist \( A = V \) oder \( A = \emptyset \), so folgt sofort \( p(A) = q(A) \). Sei \( \emptyset \subset A \subset V \), dann ist \( q(A) = \max(0, k - q(A)) \). Wird das Maximum von 0 angenommen, so gilt: \( q(A) = 0 \leq p(A) \). Im anderen Fall gilt:
\[
p(A) = \max(\sum q(A_i) | A \text{ Teilpartition von } A) \geq k - q(A) = q(A)
\]
b) \( C \subseteq C(p) \): Sei \( x \in C \), dann ist \( p(A) = \sum q(A_i) \) für eine Teilpartition \( A \). Es gilt
\[
z(A) = \sum z(A_i) + z(A - \bigcup A_i) \geq \sum q(A_i) = p(A)
\]

Um die Supermodularität von $p$ zu zeigen, nutzen wir die Supermodularität von $q'$ aus, die wir für sich kreuzende Mengen zeigen. Seien also $A, B \subset V$ zwei sich kreuzende Mengen (d.h. $A$ und $B$ sind zwei sich schneidende Mengen und $A \cap B \neq \emptyset$). $q'(A) + q'(B) = k - q(A) + k - q(B) = 2k - (q(A) + q(B)) \leq 2k - (q(A \cap B) + q(A \cup B)) = q'(A \cap B) + q'(A \cup B)$, da $q$ nach Lemma 2.1 submodular ist.

Seien $\emptyset \subset A, B \subset V$ zwei beliebige Mengen. (Die restlichen Möglichkeiten führen zu trivialen Fällen.) $p(A) = \sum q(A_i) = \sum q'(A'_i)$ und $p(B) = \sum q(B_i) = \sum q'(B'_i)$ für Teilpartitionen $A, A', B, B'$ von $A$ und $B$. Sei $F = \{A'_1, \ldots, A'_{\lambda}, B'_1, \ldots, B'_{\mu}\}$ und $q'(F) = \sum_{X \subseteq F} q'(X)$. Es gilt:

\begin{align*}
(7.9) & \quad \forall \; v \in A \cap B \text{ gibt es höchstens 2 Mengen aus } F \text{ die } v \text{ enthalten.} \\
& \quad \forall \; v \in (A \cup B) - (A \cap B) \text{ gibt es höchstens eine Menge aus } F \text{ die } v \text{ enthält.}
\end{align*}

Gibt es in $F$ zwei sich kreuzende Mengen $A'_i, B'_j$, so ersetze sie durch $A'_{i,j} \subseteq A'_{i} \cap B'_{j}$. Es entsteht die Familie $F_1$. Da $q'$ supermodular ist, gilt: $q'(F) \leq q'(F_1)$. Wende diese Operation an, bis es keine sich kreuzenden Mengen mehr gibt. Bei jeder Operation wächst $\sum_{X \subseteq F} |X|^2$ echt, somit endet diese Operation nach endlich vielen Schritten mit der Familie $F_0$.

a) $F_0$ enthält keine sich schneidenden Mengen: $P_\cap$ bestehend aus den minimalen Mengen aus $F$, die ganz in $A \cap B$ enthalten sind. (Gibt es zwei Mengen $A'_{i,j} = B'_{i,j} \subseteq A \cap B$, so nimm eine der Mengen zu $P_\cap$ hinzu.) $F_0 = F_0 - P_\cap$. $P_\cap (P_\cup)$ ist Teilpartition von $A \cap B (A \cup B)$. Wegen der Definition von $p$ gilt:

\[ p(A \cap B) \geq q(P_\cap) = q'(P_\cap) \quad \text{und} \quad p(A \cup B) \geq q(P_\cup) = q'(P_\cup). \]

Es gilt also abschließend:

\[ p(A) + p(B) = q'(F) \leq q'(F_0) = q'(P_\cap) + q'(P_\cup) \leq p(A \cap B) + p(A \cup B) \]

b) $F_0$ enthält zwei sich schneidende Mengen $X$ und $Y$: Da es keine sich kreuzende Mengen sind, gilt: $X \cup Y = V$. Mit (7.9) folgt, daß die anderen Mengen aus $F_0$ disjunkte Teilmenge von $A \cap B$ sind. Wegen der Definition von $p$ gilt:

\[ p(A \cap B) \geq q(F_0) - q(X) - q(Y) = q'(F_0) - q'(X) - q'(Y) \]

Weiter gilt nach Satz 7.1:

\[ q'(X) - q'(Y) = k - q(X) + k - q(Y) = k - \delta(V - X) + k - \delta(V - Y) \leq \gamma \]

Beides zusammen ergibt:

\begin{align*}
p(A) + p(B) & = q'(F) \leq q'(F_0) \\
& \leq p(A \cap B) + q'(X) + q'(Y) \\
& \leq p(A \cap B) + \gamma \\
& = p(A \cap B) + p(A \cup B) \quad \square
\end{align*}
Tauschen wir in Satz 7.12 ρ durch δ aus, so erhalten wir folgenden Satz:

**Satz 7.13** \( C_δ = \{ z \in R^n | z(X) ≥ q(X) \, \forall X \subseteq V \} \) mit

\[
q(X) := \begin{cases} 
\max(0, k - δ(X)) & : \ X \subseteq V \\
0 & : \ X = \emptyset \\
γ & : \ X = V 
\end{cases}
\]

ist ein Kontrapolymatriod \( C(p_δ) \) mit

\[
p_δ(A) := \max\{ \sum_{i=1}^{t} q(A_i) | \{A_1, \ldots, A_t\} \text{ ist Teilpartition von } A \}
\]

Der Beweis verläuft analog zum Beweis von Satz 7.12.

Der Zusammenhang zwischen Kontrapolymatroiden und \( k \)-fach kantenzusammenhängenden Graphen zeigt sich in dem folgenden Korollar:

**Korollar 7.14** (i) Ist \( F \) eine Menge von γ Kanten, die \( G^* = (V, E \cup F) \) zu einem \( k \)-fach kantenzusammenhängenden Graphen macht, so gilt \( z \in C_δ(z \in C_δ) \) mit \( z \in N^n \) und \( z|v| = q_F(v) \) \((z|v| = δ_F(v))\) für alle \( v \in V \).

(ii) Sind \( z_δ \in C_δ \) und \( z_δ \in C_δ \) ganzzahlige Vektoren mit \( γ = z_δ(V) = z_δ(V) \), dann gibt es eine Kantenmenge \( F \) mit γ neuen Kanten, für die \( G^* = (V, E \cup F) \) ein \( k \)-fach kantenzusammenhängender Graph ist und \( z_δ(v) = δ_F(v) \) und \( z_ρ(v) = q_F(v) \) für alle \( v \in V \) gilt.

Beweis: Der erste Teil folgt sofort aus der Definition von \( C_δ(C_δ) \). Zum Beweis des zweiten Teils seien \( z_ρ \) und \( z_δ \) Vektoren mit den gewünschten Eigenschaften. Wir nehmen zum Graphen \( G \) einen neuen Knoten \( s \) und ziehen \( z_δ(v) \) parallele Kanten von \( s \) nach \( v \) und \( z_δ(v) \) parallele Kanten von \( v \) nach \( s \). Damit sind die Voraussetzung von Satz 7.4 erfüllt, der besagt, daß eine wie oben beschriebene Kantenmenge \( F \) existiert. \( \square \)

Der Lohn der Vorbereitungen liegt nun in einem sehr einfachen Beweis des Satzes 7.8:

Beweis von Satz 7.8: Die Situation zu Beginn von Schritt 1 im Algorithmus von Frank entspricht einem Vektor \((k, k, \ldots, k)\) in \( C_δ \) (nach Satz 7.12). In Schritt 1 geht der Algorithmus genauso vor wie es in Korollar 7.11 beschrieben ist. Wir erhalten einen optimalen Vektor \( z_δ^* \). (Analog bekommen wir den Vektor \( z_δ^* \).) Im Schritt 3 erreichen wir die Voraussetzung von Korollar 7.14 durch eine kleinste mögliche Erhöhung von \( a_{ε} z_δ^* + a_{ε} z_δ^* \). Das Korollar sagt uns, daß die zu den optimalen Vektoren gehörende Kantenmenge den Graphen \( k \)-fach kantenzusammenhängend macht. \( \square \)
7.4 Implementation des Algorithmus von Frank

Wir wollen bei der Implementation des Algorithmus von Frank eine Version mit Kantenkapazitäten betrachten. Die Version des Hauptproblems mit Kantenkapazitäten lautet:

Sei $c : V^2 \rightarrow R_0^+$ eine Kapazitätsfunktion auf den möglichen Kanten zwischen Knoten aus $V$. Sei $k \in R_0^+$ die Mindestanforderung an den Wert eines minimalen $u, v$–Schnitts für alle $u, v \in V$. Das Problem besteht darin, die Kapazitäten so zu erhöhen, daß in dem resultierenden Netzwerk der Wert eines minimalen $u, v$–Schnitts für alle $u, v \in V$ mindestens $k$ beträgt. Dabei soll die Summe der Kapazitätserhöhungen minimal sein.

Für ganzzahlige Daten erhalten wir folgende Äquivalenzen: Sei $G = (V, E)$ ein gerichteter Graph, $c : V^2 \rightarrow N_0$ eine Kapazitätsfunktion und sei $k \in N$, dann sind folgende Aussagen äquivalent:

(i): $G$ kann durch Erhöhung der Kapazitäten um $\gamma$ Einheiten zu einem Graphen erweitert werden, für den der Wert eines minimalen $u, v$–Schnitts für alle $u, v \in V$ größer als $k$ ist.

(ii): $G' = (V, E')$ mit $E' = \sum_{u,v \in V} \sum_{i=1}^{c(uv)} uv$ kann durch Erweiterung um $\gamma$ neue Kanten zu einem $k$-fach-kanten zusammenhängenden Graphen gemacht werden.

(iii): Für jede echte Teilzerlegung $\{X_1, \ldots, X_t\}$ von $V$ gilt:

\[ \sum_{i=1}^{t} (k - \rho_{E'}(X_i)) \leq \gamma \quad \text{und} \quad \sum_{i=1}^{t} (k - \delta_{E'}(X_i)) \leq \gamma \]

(iv): Für jede echte Teilzerlegung $\{X_1, \ldots, X_t\}$ von $V$ gilt:

\[ \sum_{i=1}^{t} (k - \rho_{E}(X_i)) \leq \gamma \quad \text{und} \quad \sum_{i=1}^{t} (k - \delta_{E}(X_i)) \leq \gamma \]

mit $\rho_{E}(X) := \sum_{uv \in \mathcal{N}^+(X)} c(uv), \delta_{E}(X) := \sum_{uv \in \mathcal{N}^-(X)} c(uv)$

Die Äquivalenz (i) $\Leftrightarrow$ (ii) folgt aus der Beobachtung, daß $\lambda(u, v)$ als Wert eines minimalen $u, v$–Schnitts aufgefaßt werden kann (die Kapazitäten auf den Kanten sind 1). (ii) $\Leftrightarrow$ (iii) ist die Aussage des Satzes 7.1. (iii) $\Leftrightarrow$ (iv) folgt sofort mit der Definition von $\rho$ und $\delta$.

Diese theoretische Äquivalenz gestattet uns nun auch das Hauptproblem mit Hilfe der Version mit Kantenkapazitäten zu lösen. OBD\textsubscript{A} können wir annehmen, daß keine parallelen Kanten im Ausgangsgraphen enthalten sind.
Definition 7.4 Sei $0 \leq z \leq \min(c(us), c(sv))$. Folgende Aktion heißt gewichtetes Abspalten des Paares $\{us, sv\}$ um den Wert $z$ : Reduziere $c(us)$ und $c(sv)$ jeweils um $z$ und erhöhe $c(uv)$ um $z$. Ein gewichtetes Abspalten heißt zulässig, wenn nach der Aktion die Flußanforderungen für jedes Knotenpaar aus $V$ erfüllt bleibt.

Algorithmus von Frank für Netzwerke

1. Füge einen Knoten $s$ zu $V$ hinzu. Füge für alle $v \in V$ die Kanten $sv$ mit Kapazität $k$ zum Graphen hinzu. Sortiere die neuen Kanten nach absteigenden Knotenkosten des Endknotens. Gehe in dieser Reihenfolge durch die neuen Kanten und reduziere die Kapazität um den größten möglichen Wert ohne die Schnittanforderung zu verletzen. $\gamma_1$ bezeichne die Summe der restlichen Kapazitäten der Kanten $sv$ für $v \in V$.

2. Füge für alle $v \in V$ die Kanten $vs$ mit Kapazität $k$ zum Graphen hinzu. Sortiere die neuen Kanten nach absteigenden Knotenkosten des Anfangsknotens. Gehe in dieser Reihenfolge durch die neuen Kanten und reduziere die Kapazität um den größten möglichen Wert ohne die Schnittanforderung zu verletzen. $\gamma_2$ bezeichne die Summe der restlichen Kapazitäten der Kanten $vs$ für $v \in V$.

3. Sei $\gamma = \max(\gamma_1, \gamma_2)$. Ist $\gamma_2 < \gamma_1$, so erhöhe die Kapazität der Kante $us$, $u \in V$ um $\gamma_1 - \gamma_2$. $u$ ist der Knoten, für den $a_\delta$ das Minimum annimmt. Ist $\gamma_2 > \gamma_1$, so erhöhe die Kapazität der Kante $sv$, $v \in V$ um $\gamma_2 - \gamma_1$. $v$ ist der Knoten, für den $a_\delta$ das Minimum annimmt.

4. Bezeichne mit $G'$ den resultierenden Graphen. In $G'$ gilt $\delta_e(s) = g_{\sigma'}(s)$ und die kapazitären Versionen von (7.3) und (7.4) gelten. Damit können wir die mit $s$ inzidenten Kanten (nach Satz 7.4) abspalten. Wir erhalten den Graphen $G_{opt} = (V, E \cup F)$.

Der Anfang von Schritt 1 und 2 bereitet uns weiter keine Schwierigkeiten. Das Einfügen des neuen Knotens $s$ ist in konstanter Zeit zu bewerkstelligen. Die mit $s$ inzidenten Kanten mit den entsprechenden Kapazitäten können in $n$ Schritten (da jeder Knoten besucht werden muß) mit jeweils konstantem Zeitaufwand eingefügt werden. Also erledigen wir diese Operation mit einer (Zeit)-Komplexität von $O(n)$. Der Speicherplatzbedarf für die neuen Kanten ist ebenfalls lineare in $n$ abhängig, denn es werden $2n$ Kanten in den Graphen eingefügt.

Hier sieht man zum erstenmal den Vorteil der Version mit Kantenkapazitäten. Würde man die diskrete Version implementieren, so wäre schon diese Operation nicht mehr in streng polynomialer Zeit (und Speicher) zu lösen. Die Anzahl der
zu behandlenden Kanten ist $2kn$. Also wäre die Komplexität (sowohl Zeit als auch Speicher) hier $O(kn)$.

Sei nun $G' = (V \cup s, E')$ mit $E' = E \cup \{su|u \in V\} \cup \{us|u \in V\}$ und $c'$ die Kapazitätsfunktion mit $c'_\{V\} = c$, $c'(su) = k = c'(us) \ \forall u \in V$. Bezeichne mit $\maxflow_{G',c}(u,v)$ den Wert eines maximalen Flusses in $G$ mit der Kapazitätsfunktion $c$ von $u$ nach $v$.

Betrachten wir nun den zweiten Teil von Schritt 1. Wir wollen die Kapazitäten der aus $s$ austretenden Kanten nacheinander reduzieren, ohne die folgende Bedingung zu verletzen:

$$\min(\delta c'(T)|(T, \overline{T}) \quad u,v-\text{Schnitt}) \geq k \quad \forall u, v \in V \quad u \neq v$$

Diese Bedingung ist nach dem Satz 2.3 äquivalent mit

$$\maxflow_{G',c}(u,v) \geq k \quad \forall u, v \in V \quad u \neq v$$

Wir müssen einen Wert $z$ berechnen, auf den wir $c'(su)$ setzen können ohne (7.12) zu verletzen.

Betrachten wir die Kante $su$. Vor dem Absenken ist $c'(su) = k$ (aus Schritt 1). Wir suchen nun den Knoten $v$, für den gilt:

$$\min(\delta c'(T_0)|(T_0, \overline{T}_0) \quad \{v,s\}, u-\text{Schnitt}) =$$

$$\min(\min(\delta c'(T)|(T, \overline{T}) \quad \{w, s\}, u-\text{Schnitt}|w \in V - u, \quad v \in V - u)$$

Sei $v$ dieser Knoten und sei $k + \epsilon = |(T_0, \overline{T}_0)|_c$ (*). Es gilt $\epsilon \geq 0$, da die Kante $su$ im Schnitt liegt und $c'(su) = k$. (*) bedeutet, daß der Wert eines maximalen Flusses von $s$ und $v$ nach $u$ gerade $k + \epsilon$ beträgt. Setzen wir $c'(su) := \max(0, k - \epsilon)$ so gilt weiterhin (7.12).

Beweis: An einem maximalen Fluß von $u$ nach $w \in V - u$ ändert sich nichts, wenn man $c'(su)$ absenkt. Die Einheiten auf dieser Kante treffen in $u$ ein, d.h. diese Einheiten wurden auf einem Kreis verschickt, der zum Wert des maximalen Flusses nicht beiträgt. Weiterhin gilt, daß der Wert eines maximalen Flusses von $v$ nach $u$ mindestens $k$ beträgt, denn so wurde der Knoten $v$ ja gerade bestimmt. Betrachten wir nun den maximalen Fluß von $w \in V - \{u,v\}$ nach $u$. Annahme: Es existiert ein Knoten $x \in V - \{u,v\}$ für den nach dem Absenken der Kapazität $c'(su)$ $\maxflow_{G',c}(x,u) < k$ gilt. Sei $(T_1, \overline{T}_1)$ der zugehörige minimale Schnitt.

Fall 1: $s \in \overline{T}_1$, dann liegt die Kante $su$ nicht im Schnitt und so galt auch vor dem Absenken von $c'(su)$, daß $|(T_1, \overline{T}_1)|_c < k$. Das steht im Widerspruch zu (7.11).

Fall 2: $s \in T_0$, dann war der Schnitt $(T_1, \overline{T}_1)$ an der Minimumbildung in (7.13) beteiligt. Es galt

$$|(T_1, \overline{T}_1)|_c \geq |(T_0, \overline{T}_0)|_c = k + \epsilon$$
vor der Absenkung von $c'(su)$. Wir haben die Kapazität der Kante $su$ um $\epsilon' = \min(k, \epsilon)$ reduziert. Die Kante $su$ liegt sowohl in $(T_0, \bar{T}_0)$ als auch in $(T_1, \bar{T}_1)$. Nach der Absenkung gilt also

$$k \leq k + \epsilon - \epsilon' = |(T_0, \bar{T}_0)|_{c'} - \epsilon' \leq |(T_1, \bar{T}_1)|_{c'} - \epsilon'.$$

Widerspruch.

Es gilt

$$\maxflow_{G,c}(w, u) \geq k \quad \forall w \in V - u$$
$$\maxflow_{G,c}(u, w) \geq k \quad \forall w \in V - u$$

Mit dem folgenden Lemma gilt:

$$\maxflow_{G,c}(u, v) \geq k \quad \forall u, v \in V \quad u \neq v$$

**Lemma 7.15** Sei $\maxflow_{G,c}(u, v) \geq k$ und $\maxflow_{G,c}(v, w) \geq k$, dann gilt $\maxflow_{G,c}(u, w) \geq k$.

**Beweis:** Annahme: $\maxflow_{G,c}(u, w) < k$. Dann gibt es einen $u, w$–Schnitt $(T, \bar{T})$ mit $|(T, \bar{T})|_{c} < k$.

Fall 1): $v \in T$, dann ist $(T, \bar{T})$ ein $v, w$–Schnitt und damit gilt $|(T, \bar{T})|_{c} \geq k$, denn $\maxflow_{G,c}(v, w) \geq k$. Widerspruch.

Fall 2): $v \in \bar{T}$, dann ist $(T, \bar{T})$ ein $u, v$–Schnitt und damit gilt $|(T, \bar{T})|_{c} \geq k$, denn $\maxflow_{G,c}(u, v) \geq k$. Widerspruch. $\square$

Der zweite Teil von Schritt 1 läßt sich also durch Bestimmung minimaler $\{v, s\}$, $u$–Schnitte lösen. Nach dem Satz 2.3 heißt das, daß wir maximale Flüsse von $v$ und $s$ nach $u$ berechnen müssen. Für jede Kante $su$ bestimme:

(7.14) \hspace{1cm} k + \epsilon = \min(\delta^c(T_0)(T_0, \bar{T}_0) \quad \{v, s\}, u$–Schnitt) =

$$\min(\min(\delta^c(T)(T, \bar{T})) \quad \{w, s\}, u$–Schnitt)|w \in V - u$$

Wie oben erwähnt kann man einen minimalen $\{v, s\}, u$–Schnitt mit Hilfe einer maximalen Flußbestimmung (kurz: MFMC-Berechnung, MFMC = max-flow-min-cut) von $s$ und $v$ nach $u$ berechnen. Wir brauchen also zur Bestimmung des Schnitts $(T_0, \bar{T}_0) \quad (n - 1)$ dieser MFMC-Berechnungen. Das bedeutet, daß wir für alle von $s$ ausgehenden Kanten $n(n - 1)$ MFMC-Berechnungen anstellen müssen.

Im folgenden wollen wir zeigen, daß sich diese Zahl auf $2(n - 1)$ MFMC-Berechnungen reduzieren läßt.

Wir suchen den Schnitt $(T_0, \bar{T}_0)$ aus (7.14). Dazu bestimmen wir einen minimalen $s, u$–Schnitt $(T_1, \bar{T}_1)$ mit $|(T_1, \bar{T}_1)|_{c} = k + \eta$. Wir führen folgende Fallunterscheidung durch:
Fall A: $\eta \geq k$ : Beh. $|(T_0, \bar{T}_0)|_{\epsilon} \geq 2k$ (und damit $\epsilon \geq k$).
Beweis: $|(T_0, \bar{T}_0)|_{\epsilon} \geq \frac{1}{2} |(T_1, \bar{T}_1)|_{\epsilon} = k + \eta \geq 2k$. Die Ungleichung 1 gilt, da $(T_0, \bar{T}_0)$ $s,u$-Schnitt ist, und $(T_1, \bar{T}_1)$ minimaler $s,u$-Schnitt ist. $c'(su)$ wird auf max$(0, k - \epsilon) = 0$ gesetzt.

Fall B: $\eta < k$ :
(i) $T_1 - s \neq \emptyset$, d.h. es gibt einen Knoten $v \in T_1$. Behauptung: $|(T_1, \bar{T}_1)|_{\epsilon} \leq |(T_0, \bar{T}_0)|_{\epsilon}$ (und damit $|(T_1, \bar{T}_1)|_{\epsilon} = |(T_0, \bar{T}_0)|_{\epsilon}$ und $\epsilon = \eta$ aus (7.14)).
Beweis: $(T_1, \bar{T}_1)$ taucht bei der Minimumsbildung in (7.14) auf, da $v \in T_1$. Annahme: Es existiert ein $w$ mit $(T_2, \bar{T}_2) \{w, s\}$, $u$-Schnitt und $|(T_2, \bar{T}_2)|_{\epsilon} < |(T_1, \bar{T}_1)|_{\epsilon}$. Da $(T_1, \bar{T}_1)$ minimaler $s,u$-Schnitt und auch $(T_2, \bar{T}_2)$ ein $s,u$-Schnitt ist, folgt $|(T_1, \bar{T}_1)|_{\epsilon} \leq |(T_2, \bar{T}_2)|_{\epsilon}$. Widerspruch.
(ii) $T_1 - s = \emptyset$ d.h. $T_1 = s$. Hier ist keine Zurückführung von $(T_1, \bar{T}_1)$ auf $(T_0, \bar{T}_0)$ möglich.

Wir sehen, daß wir in den Fällen A und B(i) mit einer MFMC-Berechnung die neue Kapazität für $su$ bestimmen können. Der einzige Fall, in dem weiterhin $(n - 1)$ (mit der Bestimmung von $(T_1, \bar{T}_1)$ sogar $n$) MFMC Berechnungen nötig sind, ist B(ii). Es läßt sich zeigen:

**Lemma 7.16** Der Fall B(ii) tritt höchstens einmal bei der Reduzierung der von (in) $s$ ausgehenden (eintreffenden) Kanten auf.

Beweis: Betrachten wir nur den Fall der ausgehenden Kanten. Der Beweis für die eintreffenden Kanten läuft analog.

Sei $su_1, su_2, \ldots, su_n$ die Reihenfolge der Kanten in der die Kapazitäten abgesenkt werden sollen. Betrachte die Kante $su_i$, $1 \leq i < n$: Annahme: Für den minimalen $s,u_i$-Schnitt $(T_1, \bar{T}_1)$ mit $|(T_1, \bar{T}_1)|_{\epsilon} = k + \eta$ gilt: $T_1 = s$ und $\eta < k$ (d.h Fall B(ii) tritt ein). Es gilt:

$|(T_1, \bar{T}_1)|_{\epsilon} = \delta'(s) \geq \frac{1}{2} k(n - (i - 1)) \geq 2k$

so folgt $\eta \geq k$. Widerspruch. Die Ungleichung 1 folgt, da die Kanten $su_i, \ldots, su_n$ noch die Kapazität $k$ haben. □

**Bemerkung 7.5** Der Fall B(ii) tritt bei der Kante $su_n$ auch nur dann auf, wenn $\gamma$ (Summe der Kapazitäten erhöhung) $< k$.

Wir lösen die betreffenden Teile von Schritt 1 und 2 also mit $2(2n - 1)$ MFMC-Berechnungen. Alle anderen Berechnungen liegen unter dieser Zeitspanne. Wir erhalten für diesen Teil des Algorithmus eine Komplexität von $O(n \times MFMC)$.

$MFMC = Komplexität für einen maximalen Fluß-Algorithmus.$
Kapitel 7. Augmentierungsgerichteter Graphen

Kommen wir nun zum Schritt 3. Wir suchen eine Folge von zulässigen gewichteten Abspaltungen, nach der s ein isolierter Knoten in $G'$ ist.

Um zu garantieren, daß eine solche Folge existiert, brauchen wir die Voraussetzung $\delta^c(s) = \rho^c(s)$. Nach Schritt 2 muß diese Forderung noch nicht erfüllt sein. Sei oBdA $\delta^c(s) < \rho^c(s)$. Dann erhöhe die Kapazität von $su_n$ um $\rho^c(s) - \delta^c(s)$. Somit ist die oben genannte Voraussetzung erfüllt. Diese Operation ist in konstanter Zeit durchzuführen.

Schwierigkeiten bereitet uns nun noch der Schritt 4. In $G'$ gibt es maximal $n^2$ verschiedene Kantenpaare. Führen wir nun sukzessiv zulässige, maximal gewichtete Abspaltungen durch, so ist nach höchstens $n^2$ Abspaltungen s ein isolierter Knoten, denn nach Satz 7.4 existiert immer ein solches Paar. Durch die Zulässigkeit der Abspaltung sind die Voraussetzungen für den Satz weiter erfüllt, und durch die Maximalität des Abspaltens kann ein Paar nur einmal abgespalten werden. Das heißt wir sind nach $n^2$ zulässigen maximal gewichteten Abspaltungen mit Schritt 4 fertig. Mit Hilfe einer Familie $\mathcal{F}$ aus disjunkten kritischen Mengen können wir diese Zahl auf $3n - 1$ drücken.

Doch beschäftigen wir uns zuerst mit dem eigentlichen Abspalten. Wir wollen für das Paar $\{us, sv\}$ eine zulässige maximal gewichtete Abspaltung durchführen, d.h. wir wollen die Kapazitäten der Kanten $us, sv$ um den größtmöglichen Wert $\epsilon$ reduzieren und $c'(uv)$ um $\epsilon$ erhöhen ohne (7.12) zu verletzen.

Wenn wir die Kapazität der Kante $us$ ($sv$) verkleinern, ist es möglich, daß $\text{maxflow}_{G', \epsilon}(u, w)$ ($\text{maxflow}_{G', \epsilon}(w, v)$) für ein $w \in V - \{u, v\}$ unter den Wert $k$ fällt.

Als erstes stellen wir trivialerweise fest, daß auch nach dem Abspalten $\text{maxflow}_{G', \epsilon}(u, v) \geq k$ (♯) gilt.

Wir wollen uns nur mit $\text{maxflow}_{G', \epsilon}(u, w)$ beschäftigen. Für $\text{maxflow}_{G', \epsilon}(w, v)$ gelten folgende Überlegungen analog. Bestimme

$$k + \epsilon = \min(\delta^c(T_0)|(T_0, \bar{T}_0) = \{u, v\}, \{s, w\} - \text{Schnitt}) = \min(\delta^c(T)|(T, \bar{T}) = \{u, v\}, \{x, s\} - \text{Schnitt}) x \in V - \{u, v\})$$

Behauptung: Nach dem Absenken von $c'(us)$ um $\epsilon$ und Anheben von $c'(uv)$ um $\epsilon$ gilt $\text{maxflow}_{G', \epsilon}(u, w) \geq k$.

Beweis: Sei $(T, \bar{T})$ ein minimaler $u, w$–Schnitt.

Fall 1: $v \in T, s \in \bar{T} : (T, \bar{T})$ ist $\{u, v\}, \{w, s\} - \text{Schnitt}$, darum gilt $|(T, \bar{T})|_{\epsilon} \geq k$.

Fall 2: $v \in T, s \in T :$ Annahme: $|(T, \bar{T})|_{\epsilon} < k$. Da an den Kapazitäten der Kanten im Schnitt nichts geändert wurde und $(T, \bar{T})$ auch vor der Änderung ein $u, w$–Schnitt war, folgt, daß schon vor der Änderung (7.12) nicht mehr erfüllt war. Widerspruch.

Fall 3: $v \in \bar{T} : (T, \bar{T})$ ist $u, v$–Schnitt, darum gilt mit (♯) $|(T, \bar{T})|_{\epsilon} \geq k$. □
Behauptung: Die Absenkung um $\epsilon$ ist maximal.
Beweis: Eine Absenkung um $\nu > \epsilon$ führt zu $\delta^e(T_0) < k$, welches (7.12) verletzt. $\square$

Auf die gleiche Weise konstruieren wir

$$k + \eta = \min(\delta^e(T_0)|(T_0, \bar{T}_0) \{s, w\}, \{u, v\} - \text{Schnitt}) =$$

$$\min(\min(\delta^e(T)|(T, \bar{T}) \{x, s\}, \{u, v\} - \text{Schnitt})|x \in V - \{u, v\})$$

Führe das folgende zulässige maximal gewichtete Abspalten durch: Sei $z := \min(c'(us), c'(sv), \epsilon, \eta)$. Senke $c'(us)$ und $c'(sv)$ um $z$ ab. Hebe $c'(uv)$ um $z$ an.

**Bemerkung 7.6** Solange $\delta^e(s) = \varphi^e(s) \geq k$, lohnt es sich u.U. die gleiche Methode wie in Schritt 1 und 2 anzuwenden. Berechne erst einen minimalen $\{u, v\}, s-$Schnitt $(T_0, \bar{T}_0)$. Gilt $k + \nu = |(T_0, \bar{T}_0)|_{s} \geq k + c'(us)$, so folgt $\epsilon \geq c'(us)$. Gilt $k + \nu < k + c'(us)$ und $T_0 - s \neq \emptyset$, so folgt $\epsilon = \nu$. Die Häufigkeit des Auftretens des ungünstigen Falles $\bar{T}_0 = s$ läßt sich leider nicht geeignet beschränken wie in Schritt 1 und 2.

Wie bereits erwähnt, können wir mit Hilfe einer Mengenfamilie $\mathcal{F}$ von kritischen disjunkten Mengen die Zahl der oben beschriebenen Abspaltungen von $n^2$ auf $3n - 1$ senken.

Zu Beginn von Schritt 4 sei $\mathcal{F}$ leer. Wähle nun ein Knotenpaar $u, v$ so, daß $c'(us) > 0, c'(sv) > 0$ und $u$ und $v$ nicht zum selben Element von $\mathcal{F}$ gehören. Solch ein Knotenpaar läßt sich immer finden, denn sei oBdA $Y$ eine $\delta^e$-kritische Menge (d.h. es gibt $x \in Y$ und $y \in V - Y$ mit $\maxflow_{G',e}(x, y) = k$), mit $u \in Y$ und $c'(us) > 0$. Annahme: Es existiert kein $v \in V - Y$ mit $c'(sv) > 0$. Dann folgt $\varphi^e(V - Y) \leq \delta^e(V) - c'(us) = k - c'(us) < k$. Widerspruch zu (7.12).

Beim gewichteten Abspalten des Paares $\{us, sv\}$ können zwei Fälle auftreten.

**Fall 1:** Die Kapazität mindestens einer Kante $us, sv$ kann auf 0 abgesenkt werden, ohne daß eine Menge $T \subset V$ entsteht, die kritisch wird.

**Fall 2:** Die Kapazitäten der Kanten $us, sv$ können um den Wert $z, \ 0 \leq z \leq \min(c'(us), c'(sv))$, abgesenkt werden und eine kritische Menge $T \subset V$ entsteht.

Tritt der Fall 2 ein so erhalten wir eine kritische Menge $T$. Wir wollen $T$ geeignet in $\mathcal{F}$ eintragen. Führe also folgende Aktion durch:

Seien $F_1, \ldots, F_l$ die Mengen aus $\mathcal{F}$, für die gilt $F_i \cap T \neq \emptyset$ ($i = 1, \ldots, l$). Nach Lemma 7.6 tritt immer einer der beiden Fälle auf:

**Fall A:** $T \cup F_i$ ist kritisch, dann ersetze $T$ durch $F_i \cup T$ und entferne $F_i$ aus $\mathcal{F}$ (*).
Fall B: $F_i - T$ ist kritisch, dann ersetze $F_i$ durch $F_i - T$. (**) 

Nach dem Ausführen der Operationen (*) und (**) für die Mengen $F_1, ..., F_i$, füge die Menge $T$ zu $\mathcal{F}$ hinzu. (***)

Wir erhalten die Familie $\mathcal{F}'$, die weiterhin aus disjunkten kritischen Mengen besteht.

**Bemerkung 7.7** $\mathcal{F}$ enthält höchstens $n$ Mengen. Alle Mengen aus $\mathcal{F}$ enthalten zusammen höchstens $n$ Knoten. Darum lassen sich die Menge $F_1, ..., F_i$ in einer Komplexität von $O(n)$ berechnen. Ähnlich verwandelt sich mit dem Prüfen der Bedingungen $(F_i \cup T$, bzw. $F_i - T$ kritisch ? und dem Aktualisieren der Familie $\mathcal{F}$ und $T$. Genaueres entnehmen man dem Kapitel 9.

**Lemma 7.17** Sei $\mathcal{F}, (\mathcal{F}')$ die Mengenfamilie vor (nach) dem Einfügen von $T$. Bestand $\mathcal{F}$ aus disjunkten, kritischen Mengen, so auch $\mathcal{F}'$. Weiterhin gilt:

$$ |\bigcup \mathcal{F}'| - |\mathcal{F}'| > |\bigcup \mathcal{F} | - |\mathcal{F}| $$

mit $\bigcup \mathcal{F} := \bigcup_{X \in \mathcal{F}} X $.

\begin{center}
\begin{tabular}{ccc}
$\mathcal{F}$ & $\mathcal{F}$ & $\mathcal{F}$ \\
\text{Fall A} & $\mathcal{F}$ & \text{Fall B(i)} \\
\text{Fall B(ii)} & $\mathcal{F}$ & \text{Fall B(iii)} \\
\end{tabular}
\end{center}

Abb. 7.2

Beweis: Die Aussage über die disjunkten, kritischen Mengen folgt aus der oben beschriebenen Aktion und dem Lemma 7.6. Für die Aussage (7.15) führen wir eine Fallunterscheidung durch.

Fall A: $T \cap \bigcup \mathcal{F} = \emptyset$ : Dann gilt $\mathcal{F}' = \mathcal{F} \cup \{T\}$ mit $|T| \geq 2$, da $u, v \in T$. Es gilt $|\bigcup \mathcal{F}'| - |\mathcal{F}'| \geq |\bigcup \mathcal{F}| - 2 - (|\mathcal{F}| + 1) = |\bigcup \mathcal{F}| - |\mathcal{F}| + 1 > |\bigcup \mathcal{F} | - |\mathcal{F}|$.

Fall B: $T \cap \bigcup \mathcal{F} \neq \emptyset$: Dann gilt $\bigcup \mathcal{F}' \supseteq \bigcup \mathcal{F}$.

(i) $u, v \in T - \bigcup \mathcal{F}$. Dann gilt die gleiche Formel wie in Fall A.

(ii) oBdA $u \in F \in \mathcal{F}$ und $v \in T - \bigcup \mathcal{F}$: (1) Wenn $T \cup F$ kritisch ist, dann wird in (*) $F$ aus $\mathcal{F}$ entfernt und in (***) $T$ in $\mathcal{F}$ eingetragen. (2) Ist $F - T$ kritisch und nach Lemma 7.6 $d'(F \cap T) = 0$. Es gilt aber $u \in T \cap F, s \notin T \cap F$ und $c'(us) > 0$. Widerspruch.

Bei den Schnitten mit den anderen kritischen Mengen $F_i \in \mathcal{F}$ kann die Zahl der Elemente aus $\mathcal{F}$ nicht steigen, also folgt $|\mathcal{F}'| \leq |\mathcal{F}|$ und damit $|\bigcup \mathcal{F}'| - |\mathcal{F}'| \geq |\bigcup \mathcal{F}| - 1 - |\mathcal{F}| > |\bigcup \mathcal{F} | - |\mathcal{F}|$.

(iii) oBdA $u \in F_1, v \in F_2$: (1) $T \cup F_1$ und $T \cup F_2$ kritisch, dann werden in (*) $F_1$ und $F_2$ aus $\mathcal{F}$ entfernt und in (***) $T$ in $\mathcal{F}$ eingetragen. (2) Mindestens
eines (oBdA $T \cup F_1$) ist nicht kritisch. Dann gilt $d'(F_1 \cap T) = 0$. Wie in (ii.2) läßt sich ein Widerspruch konstruieren. Wie bei (ii) wird bei dem Schnitt mit anderen kritischen Mengen die Zahl der Elemente aus $\mathcal{F}$ nicht vergrößert, also folgt $|\mathcal{F}'| < |\mathcal{F}|$ und damit $|\bigcup \mathcal{F}'| - |\mathcal{F}'| > |\bigcup \mathcal{F}| - |\mathcal{F}|$. □

**Satz 7.18** Der Schritt 4 kann mit $3n - 1$ gewichteten Abspaltungen gelöst werden.

Beweis: Der Fall 1 (beim gewichteten Abspalten siehe Seite 67) kann höchstens $2n$-mal auftreten, da es nur $2n$ Kanten gibt, die mit $s$ inzident sind. Der Fall 2 tritt höchstens $n - 1$ mal auf. Denn die (ganze) Zahl $|\bigcup \mathcal{F}| - |\mathcal{F}|$ wächst echt bei jedem Auftreten von Fall 2. Sie ist nach oben beschränkt durch $n - 1$. Wir beginnen mit $\mathcal{F} = \emptyset$. Also sind wir nach maximal $(n - 1)$-maligem Auftreten von Fall 2 fertig. □

Da wir eine zulässige, maximal gewichtete Abspaltung in $2n$-MFMC Berechnungen bestimmen können, lösen wir Schritt 4 mit einer Komplexität von $O(n^2 \cdot \text{MFMC})$.

**Bemerkung 7.8** Benutzen wir zur Berechnung eines maximalen Flusses den Preflow-Push-MFMC-Algorithmus von Goldberg-Tarjan 3, welcher bei geeigneter Implementation die Komplexität $O(n^3)$ besitzt, so lösen wir Schritt 1 und 2 in $O(n^4)$ und Schritt 4 in $O(n^5)$. 


7.5 Einfache Beispiele

Wir werden das Kapitel 7 mit einigen einfachen Beispielen abschließen. Die folgenden Beispiele können entweder als ungewichtete oder Beispiele mit Kapazität $c \equiv 1$ angesehen werden. (Im zweiten Beispiel muß man dann die parallelen Kanten zusammenfassen.)

Die Darstellung eines Beispiels besteht aus 6 (bzw. 7) Bildern. Das erste Bild zeigt den Ausgangsgraphen $G = (V, E)$. Das jeweils zweite Bild entspricht dem erweiterten Graphen in Schritt 1. Wir versuchen die Kanten $sv$ für alle $v \in V$ aus $G$ zu entfernen. Die Knoten sind so angeordnet, daß $a_\delta(1) \geq \ldots \geq a_\delta(n)$ und $a_\delta(n) \geq \ldots \geq a_\delta(1)$ gilt. Das heißt: Beim Entfernen der Kanten $sv$ versuchen wir erst die Kante $s1$, dann die Kante $s2$, usw. zu entfernen. Bei den Kanten $vs$ ist es umgekehrt: Wir versuchen zuerst die Kante $ns$, dann die Kante $(n-1)s$, usw. zu entfernen. Den Graphen des dritten Bildes erhalten wir nach Schritt 1. Der Graph des vierten Bildes ist um die Kanten $vs$ für alle $v \in V$ erweitert. Das fünfte Bild zeigt den Graphen nach Schritt 2. Anschließend (außer dem dritten Beispiel) folgt als 6. Bild, der resultierende Graph $G_{opt} = (V, E \cup F)$. Im dritten Beispiel ist die Voraussetzung $q(s) = \delta(s)$ nach Schritt 2 nicht erfüllt. Wir müssen im Schritt 3 den Unterschied ausgleichen. Die fett gedruckten Kanten im resultierenden Graph (Bild 6 oder 7) sind die neuen Kanten. Wir werden das erste Beispiel sehr ausführlich besprechen und bei den letzten beiden nur noch auf Besonderheiten hinweisen.

**Beispiel 1: $k = 1$**

![Abb. 7.3](image)

Betrachten wir das erste Bild von Abb. 7.3. Um zu entscheiden, ob die Kante $s1$ gelöscht werden kann, müssen wir als erstes den maximalen Fluß von $s$ nach 1 bestimmen: Mit einem Blick erkennen wir, daß wir eine Einheit direkt von $s$ nach 1 und eine Einheit von $s$ über 2 nach 1 schicken können. Der Wert des maximalen Flusses von $s$ nach 1 beträgt 2. Nach Fall A auf Seite 64 können wir die Kapazität von $s1$ auf 0 setzen, d.h. wir können sie löschen. Anders verhält es sich bei der Kante $s2$. Wir können nur eine Einheit von $s$ nach 2 verschicken. (Der Weg über 1 ist durch Löschen der Kante $s1$ zerstört). Als minimalen Schnitt finden wir die Knotenmenge $\{s, 3, 4, 5\}, \{1, 2\}$. Nach Fall B(i) dürfen wir die Kapazität der Kante
s2 nicht verändern. Genauso verhält es sich bei der Kante s3, der minimale s, 3-
Schnitt ist {s, 1, 2, 4, 5}, 3 mit dem Wert 1. Die Kanten s4 und s5 können wegen des
Wertes der maximalen Flüsse (s → 4 eine Einheit und s → 2 → 1 → 4 noch eine
Einheit; s → 5 eine Einheit und s → 2 → 5 noch eine Einheit) gelöscht werden.

Wir erhalten den Graphen des ersten Bildes:

Abb. 7.4

Fügen wir nun die Kanten vs für alle v ∈ V zum Graphen hinzu, so erhalten wir den Graphen des zweiten Bildes. Hier geht es nun anders herum. Wir wollen die
neue Kapazität der Kante 5s bestimmen. Wir müssen den Wert eines maximalen
Flusses von 5 nach s bestimmen. Wir können nur eine Einheit von 5 nach s senden.
Als minimalen Schnitt erhalten wir 5, {s, 1, 2, 3, 4}, es tritt also Fall B(i) ein. Die
Kante 5s bleibt unverändert. Der Wert eines maximalen Flusses von 4 nach s beträgt
2 (Eine Einheit direkt von 4 nach s und eines Einheit von 4 über 5 nach s). Die
Kante 4s kann gelöscht werden. Kante 3s muß bleiben, denn es gibt den minimalen
Schnitt 3, {s, 1, 2, 4, 5} mit Wert 1 (Fall B(i)). Die Kanten 2s und 1s können gelöscht
werden. Flußeinheiten von 2 nach s: 2 → s, 2 → 5 → s. Flußeinheiten von 1 nach s:
1 → s, 1 → 2 → 5 → s.

Wir erhalten den Graphen des ersten Bildes.

Abb. 7.5

Als letztes können wir das Abspalten durchführen. Das Kantenpaar s3, 3s kann
nicht abgespalten werden, da sonst ϕ(3) = δ(3) = 0 < k = 1 gelten würde. Der
Satz 7.4 garantiert uns eine Folge von zulässigen Abspaltungen. Es bleiben nur noch
die möglichen Paare 5s, s3 und 3s, s2. Den resultierenden Graphen sehen wir im
zweiten Bild der Abbildung 7.5.
Bespiel 2: $k = 2$

Abb. 7.6

Wir führen die gleichen Schritte durch wie in Beispiel 1, d.h. wir versuchen die Kanten $sv$ für alle $v \in V$ zu löschen ohne den Kantenzusammenhang innerhalb von $V$ zu zerstören. Betrachten wir die beiden Kanten von $s$ nach $1$. Ein maximaler Fluß von $s$ nach $1$ hat den Wert 4: 2 Einheiten direkt von $s$ nach 1 und 2 Einheiten von $s$ über 2 nach 1. Wir können die Kanten von $s$ nach 1 löschen. Das oben erläuterte Verfahren entspricht dem der kapazitären Version. Im ungewichteten Fall müßte man eigentlich jede Kante für sich untersuchen.

Die Kanten von $s$ nach 2 müssen beide bleiben, da es einen minimalen $s, 2$-Schnitt $\{s, 3\}, \{1, 2\}$ mit Wert 2 gibt. Bei den Kanten von $s$ nach 3 erhalten wir folgenden maximalen Fluß: 2 Einheiten von $s$ nach 3 und eine Einheit von $s$ über 2 nach 3. Wir können also eine der Kanten von $s$ nach 3 löschen. Wir erhalten den Graphen des ersten Bildes:

Abb. 7.7
Nach dem Löschen der Kanten \( vs \) für alle \( v \in V \) erhalten wir den folgenden Graphen:

![Graph](image1)

Abb. 7.8

Wir müssen nun das Abspalten durchführen. Nehmen wir uns das Kantenpaar 1s, s2 heraus und betrachten die Menge \( T = \{1, 2\} \). Es gilt \( \rho(T) = \delta(T) = 1 < k = 2 \). Das Abspalten dieses Paares ist sicher nicht zulässig. Genauso verhält es sich mit dem Paar 3s, s3 (\( \rho(3) = \delta(3) = 1 < k = 2 \)). Alle restlichen Abspaltungen sind zulässig. Wir erhalten den zweiten Graph aus der Abbildung 7.8.

**Beispiel 3: \( k = 1 \)**

Als letztes wollen wir die Ausführung von Schritt 3 zeigen. Die ersten fünf Graphen zeigen uns nichts Neues. Wir löschen die Kanten \( sv \) und \( vs \) falls möglich und erhalten den ersten Graphen der Abbildung 7.11.

![Graph](image2)

Abb. 7.9
Hier stellen wir fest, daß $1 = \delta(s) < \varrho(s) = 2$ gilt. Um die Voraussetzung zum Satz 7.4 zu erfüllen, müssen wir eine Kante $su$ zum Graphen hinzunehmen. Wir wählen eine Kante des billigsten Knotens. Das ist in diesem Fall die Kante $s3$. Wir erhalten den zweiten Graphen von Abbildung 7.11. Nach dem Abspalten sieht der Graph wie folgt aus:
Kapitel 8

Augmentierung von kreisfreien Graphen


Wie in Kapitel 7 schon erwähnt, kann man leicht eine untere Schranke für die Zahl von Kanten finden, die zu einem Graphen hinzugenommen werden müssen, um ihn $k$-fach kantenzusammenhängend zu machen.

In einem $k$-fach kantenzusammenhängenden Graphen $G^* = (V, E^*)$ muß für jeden Knoten $v \in V$ $\delta^*(v) \geq k$ und $\varrho^*(v) \geq k$ gelten. Sei $G = (V, E)$ ein beliebiger Graph, dann müssen mindestens $\gamma = \max(\gamma_\varrho, \gamma_\delta)$ mit

$$
\gamma_\varrho = \sum_{v \in V} \max(k - \varrho(v), 0), \quad \gamma_\delta = \sum_{v \in V} \max(k - \delta(v), 0),
$$

Kanten zu $E$ hinzugenommen werden, damit $G$ $k$-fach kantenzusammenhängend wird.

Lemma 8.1 Sei $G = (V, E)$ ein kreisfreier Graph. Dann gilt:

$$
\gamma_\varnothing := \sum_{v \in V} \max(k - \varnothing(v), 0)
$$

$$
= \max\left(\sum_{i=1}^{t} k - \varnothing(X_i)\right)\{X_1, ..., X_t\ \text{ist echte Teilpartition von } V\}
$$

und

$$
\gamma_\delta := \sum_{v \in V} \max(k - \delta(v), 0)
$$

$$
= \max\left(\sum_{i=1}^{t} k - \delta(X_i)\right)\{X_1, ..., X_t\ \text{ist echte Teilpartition von } V\}
$$

Das Lemma besagt, daß das Defizit an den Knoten das maximal mögliche Defizit im Graphen ist. Zusammen mit dem Satz 7.1 erhalten wir sofort folgendes Korollar:

Korollar 8.2 Sei $G = (V, E)$ ein gerichteter kreisfreier Graph und $k > 0$ eine ganze Zahl. $G$ kann durch Hinzunahme von genau $\gamma = \max(\gamma_\varnothing, \gamma_\delta)$ ($\gamma_\varnothing, \gamma_\delta$ aus Lemma 8.1) Kanten zu einem $k$-fach kantenzusammenhängenden Graphen gemacht werden.

Beweis von Lemma 8.1: Wir zeigen nur die Gleichung für $\gamma_\varnothing$. Der Beweis für $\gamma_\delta$ verläuft analog.

"$\leq$":

$$
\sum_{v \in V} \max(k - \varnothing(v), 0) = \sum_{v \in V - \{x \in V|\varnothing(x) > k\}} k - \varnothing(v)
$$

$$
\leq \max\left(\sum_{i=1}^{t} k - \varnothing(X_i)\right)\{X_1, ..., X_t\ \text{ist echte Teilpartition von } V\}
$$

da die Knoten aus $V - \{x \in V|\varnothing(x) \geq k\}$ eine echte Teilzerlegung von $V$ bilden.

"$\geq$": Annahme: Es gilt "$<$". Sei $\{X_1, ..., X_t\}$ eine Teilzerlegung für die das Maximum angenommen wird und weiterhin $\sum_{i=1}^{t} |X_i|$ minimal ist. Da laut Annahme das Maximum nicht von einer Teilzerlegung aus einzelnen Knoten angenommen wird, gibt es eine Menge $X_{in}$ mit $|X_{in}| > 1$.

Fall 1) $k - \varnothing(X_{in}) > k - \varnothing(x)$ $\forall x \in X_{in}$: Das ist äquivalent mit $\varnothing(X_{in}) < \varnothing(x)$ $\forall x \in X_{in}$. Das heißt, daß in jeden Knoten $x \in X_{in}$ eine Kante $yx$ mit $y \in X_{in}$ eintritt. Gehen wir diese Kanten rückschrittlich entlang. Da $X_{in}$ nur aus endlich vielen Knoten besteht, kommen wir nach höchstens $|X_{in}|$ Schritten zu einem Knoten, der bereits auf unserer Wanderung besucht wurde. Wir haben einen Kreis in $X_{in}$ gefunden. Widerspruch.

Fall 2) $k - \varnothing(X_{in}) \geq k - \varnothing(x)$ $\exists x \in X_{in}$: Da $\varnothing(x) \geq k$ für einen Knoten $x \in X_{in}$ gilt, ist $X_{in}$ nicht ein Kreis. Wir betrachten daher $X_{in} - \{x\}$ für einen solchen Knoten $x \in X_{in}$. Da $X_{in}$ nicht ein Kreis ist, gibt es einen Knoten $y \in X_{in}$, der nicht mit $x$ eine Kante hat. Wir betrachten daher $X_{in} - \{x, y\}$ und wiederholen den Beweis für diesen Graphen. Wenn wir keinen Knoten $x \in X_{in}$ finden, der nicht mit einer anderen Knoten $y \in X_{in}$ eine Kante hat, dann ist $X_{in}$ ein Kreis. In diesem Fall kann $X_{in}$ nicht der maximale Knoten sein, da es in $X_{in}$ einen Knoten gibt, der nicht mit anderen Knoten in $X_{in}$ eine Kante hat. Widerspruch.

Somit ist die Annahme "$<$" nicht möglich und daher gilt $\gamma = \max(\gamma_\varnothing, \gamma_\delta)$.
Fall 2) \( k - \varrho(X_{i_0}) \leq k - \varrho(x_0) \) für mindestens ein \( x_0 \in X_{i_0} \): Wir konstruieren eine neue Teilzerlegung von \( V \). \( X'_i = X_i \) für \( i \neq i_0 \) und \( X'_{i_0} = x_0 \). Gilt die echte Ungleichung so folgt:

\[
\sum_{i=1}^{t} k - \varrho(X_i) < \sum_{i=1}^{t} k - \varrho(X'_i)
\]

Das widerspricht der Maximalität der Teilpartition \( \{X_1, ..., X_t\} \). Gilt Gleichheit so folgt:

\[
\sum_{i=1}^{t} |X_i| = \sum_{i=1,i\neq i_0}^{t} |X_i| + |X_{i_0}| > \sum_{i=1,i\neq i_0}^{t} |X_i| + 1 = \sum_{i=1}^{t} |X'_i|
\]

Das widerspricht der Minimalität von \( \sum_{i=1}^{t} |X_i| \) \( \square \)

Mit Hilfe des Korollars 8.2 können wir die Schritte 1 und 2 des Algorithmus wesentlich einfacher gestalten:

**Algorithmus von Frank für kreisfreie Graphen**

1. Füge einen Knoten \( s \) zu \( V \) hinzu.
   Füge für alle \( v \in V \) \( \max(k - \varrho(v), 0) \) parallele Kanten \( sv \) zum Graphen hinzu. \( \gamma_\rho \) bezeichnet die Zahl der neu hinzugenommenen Kanten.

2. Füge für alle \( v \in V \) \( \max(k - \delta(v), 0) \) parallele Kanten \( vs \) zum Graphen hinzu. \( \gamma_\delta \) bezeichnet die Zahl der neu hinzugenommenen Kanten.

3. Sei \( \gamma = \max(\gamma_\rho, \gamma_\delta) \). Ist \( \gamma_\delta < \gamma_\rho \), so füge \( \gamma_\rho - \gamma_\delta \) parallele Kanten von \( u \in V \) nach \( s \) hinzu. \( u \) ist der Knoten, für den \( a_\delta \) das Minimum annimmt. Ist \( \gamma_\delta > \gamma_\rho \), so füge \( \gamma_\delta - \gamma_\rho \) parallele Kanten von \( s \) nach \( u \in V \) hinzu. \( u \) ist der Knoten, für den \( a_\rho \) das Minimum annimmt.

4. Bezeichne \( G' \) den resultierenden Graph. In \( G' \) gilt \( \delta'(s) = \varrho'(s) \) und (7.3) und (7.4) geltend. Damit können wir die mit \( s \) inzidenten Knoten nach Satz 7.4 abspalten. Wir erhalten den Graphen \( G_{opt} = (V, E \cup F) \).

Wir wissen, daß wir höchstens \( \gamma \) Kanten zum Graphen hinzunehmen müssen, und wir kennen die Stellen, an denen mindestens \( \gamma \) Kanten nötig sind. Die Schritte 1 und 2, die bei der Implementation für allgemeine gerichtete Graphen eine Komplexität von \( O(n \ast MFMC) \) hatten, können nun in \( O(n) \) gelöst werden.

**Bemerkung 8.1** Das Entscheidungsproblem zum Hauptproblem aus Kapitel 5 ist für kreisfreie gerichtete Graphen in \( O(n) \) lösbar. Das gleiche Problem für allgemeine Graphen hat bei geeigneter Implementation eine Komplexität von \( O(n \ast MFMC) \). Um die Kantenmenge, die den Graph \( k \)-fach kantenabhängig macht, zu erhalten, benötigen wir auch bei den kreisfreien Graphen eine Komplexität von \( O(n^2 \ast MFMC) \).
Der Knoten $u$ im Schritt 3 des Algorithmus ist bei dem Spezialfall für gerichtete Bäume von zentraler Bedeutung. Sei $T = (V, E)$ ein gerichteter Baum, der durch die Menge $F$ zu einem $k$-fach kantenzusammenhängenden Graphen $T^* = (V, E \cup F)$ gemacht werden kann. Für alle Knoten $v \in V - u$ gilt:
\[
\varphi^*(v) = \varphi(v) + \max(k - \varphi(v), 0),
\]
\[
\delta^*(v) = \delta(v) + \max(k - \delta(v), 0)
\]

$P = \{ v \in V | \delta(v) + \varphi(v) = 1 \}$ sei die Menge der Blätter des Baumes $T$. $I = V - P$ sei die Menge der inneren Knoten. Gilt $|I| = 0$, so besteht der Baum $T$ aus zwei Knoten. Die Augmentierung eines solchen Baumes ist trivial.


Der Induktionsanfang besteht also darin, einen Baum mit genau einem inneren Knoten (wir wollen diesen Baum einen Stern nennen) mit $\gamma$ Kanten zu einem $k$-fach kantenzusammenhängenden Graphen zu augmentieren. Diesen Beweis verschieben wir auf später.

Gele die Aussage nun für alle Bäume $T'$ mit $|I| \leq j$. Sei $T_0 = (X_0, E_0)$ ein Baum mit $j+1$ inneren Knoten. Dann gibt es zwei Knoten $x, y \in I$ mit $e = xy \in E_0$ (oder $e = yx \in E_0$). Diese Kante spalten wir auf in $e_1 = xv_1$ und $e_2 = v_2y$ (oder $e_1 = yv_1$ und $e_2 = v_2x$) mit den zwei neuen Knoten $v_1, v_2 \notin X_0$, den sogenannten Mischknoten. Es entstehen die Bäume $T_1$ und $T_2$.

![Diagram](image)

Abb. 8.1

Diese Bäume können wir laut Induktionsvoraussetzung mit geeignet vielen Kanten augmentieren. Wir erhalten die Graphen $T_1^*$ und $T_2^*$, die wir durch geschicktes
zusammenmischen, der mit \(v_1\) und \(v_2\) inzidenten Kanten, zu einem \(k\)-fach kantenzusammenhängenden Graphen machen können, der ebenfalls höchstens \(\gamma\) neue Kanten enthält. (Der Begriff *mischen* wird weiter unten genauer erläutert.) Somit ist der Induktionsbeweis fertig.

Die Schwierigkeit liegt im Detail. Wir müssen Sterne zu \(k\)-fach kantenzusammenhängende Graphen augmentieren und solch augmentierte Graphen unter den vorgegebenen Bedingungen zusammenmischen.

Wie schon erwähnt, wird es für jeden Teilbaum \(T_i\) einen Knoten \(u_i\) geben, der überflüssige Kanten enthält. Wir geben nun einen Algorithmus an, der die gewünschte Augmentierung durchführt. Die Teile in Anführungsstrichen werden später genauer betrachtet.

**Bemerkung 8.2** Die Form dieses Algorithmus weicht von der der anderen Algorithmen ab. Es werden Konstruktionselemente (if .. then .. else .. fi, prozedur .. end) benutzt, die in ähnlicher Form in vielen Programmiersprachen auftauchen. Der Grund für das Abweichen ist, daß sich der Algorithmus sehr elegant durch Rekursion aufschreiben läßt, was die bisherige Form nicht unterstützt.

    prozedur augment\((T_0, u_0)\) 
    if \((T_0\) ist ein Stern) then
        "Füge die augmentierende Kantenmenge zu \(T_0\) hinzu";
    else
        "Teile \(T_0\) (s.o.) in \(T_1 = (X_1, E_1)\) und \(T_2 = (X_2, E_2)\)";
        "Wähle geeignet Knoten \(u_1 \in X_1\) und \(u_2 \in X_2\)";
        augment\((T_1, u_1)\);
        augment\((T_2, u_2)\);
        "Mische \(T_1\) und \(T_2\) zu \(T_0\) zusammen";
    fi
end

Mit dem Aufruf augment\((T, u)\) führen wir die Augmentierung des Baumes \(T\) durch.

Betrachten wir die direkte Augmentierung eines Sterns. Sei \(T_0 = (X_0, E_0)\) mit \(u_0 \in X_0\) ein Stern und sei \(v_0 \in X_0\) der innere Knoten. OBoDA können wir annehmen, daß gilt: \(\delta_0(v_0) \geq \sigma_0(v_0)\) (sonst orientiere alle Kanten aus \(T_0\) um; augmentiere nach dem gleichen Verfahren und drehe anschließend wieder alle Kanten um). Weiterhin sei \(P_0 = X_0 - v_0\) und \(P_{\sigma_0(\Delta)} = \{x \in P_0 | \delta(\sigma_0(x) > 0)\}\).

Wir werden im folgenden nur die augmentierende Menge angeben. Der Beweis, daß der erweiterte Graph wirklich \(k\)-fach kantenzusammenhängend ist, ist zwar nicht sonderlich schwer, doch verzweigt sich der Beweis in viele kleine Fallunterscheidungen und wird sehr lang. Wir möchten an dieser Stelle nochmal auf die Arbeit
von Kajitani und Ueno [KAJITANI86] hinweisen. Wir schreiben im folgenden statt
\( \delta_0, \varrho_0, v_0, w_0, P_0, P_0, P_0, P_0, P_0 \) abkürzend \( \delta, \varrho, v, w, P, P_\delta, P_\varrho \)

**Fall 1)** \( k \geq \delta(v) \geq \varrho(v), \quad \delta(v) + \varrho(v) \geq k, \quad \varrho(v) \geq 1: \)
Teile \( P_\delta \) auf in \( X \) und \( W \) mit \( |X| = k - \varrho(v) \) und \( |W| = \delta(v) + \varrho(v) - k. \)
Teile \( P_\varrho \) auf in \( Y \) und \( Z \) mit \( |Y| = k - \delta(v) \) und \( |Z| = \delta(v) + \varrho(v) - k. \)
\( E^*_1 = \{ xv | x \in X \} \cup \{ vy | y \in Y \} \cup \{ w_i z_i | w_i \in W, z_i \in Z \} \)
(Die 3. Menge beschreibt Matchingkanten von \( W \) nach \( Z \))
\( E^*_2 = (k - 1) \) parallele Kanten zwischen Knoten aus \( P \) (zyklische Reihenfolge)

![Diagram](attachment:diagram.png)

**Abb. 8.2**

**Fall 2)** \( k \geq \delta(v) \geq \varrho(v), \quad \delta(v) + \varrho(v) \leq k, \quad \varrho(v) \geq 1: \)
Sei \( P_\delta = \{ x_1, \ldots, x_{\delta(v)} \}, \quad P_\varrho = \{ x_{\delta(v)+1}, \ldots, x_{\delta(v)+\varrho(v)} \}. \)
\( E^*_1 = \{ xv | x \in P_\delta \} \cup \{ vy | y \in P_\varrho \} \)
\( E^*_2 = (k - 1) \) parallele Kanten von \( x_j \) nach \( x_{j+1} \) für \( j = 1, 2, \ldots, \delta(v) + \varrho(v) - 1, \)
\( \delta(v) + \varrho(v) - 1 \) parallele Kanten von \( x_{\delta(v)+\varrho(v)} \) nach \( x_1 \) und \( k - (\delta(v) + \varrho(v)) \)
parallele Kanten von \( x_{\delta(v)+\varrho(v)} \) nach \( v \) und von \( v \) nach \( x_1. \)

**Fall 3)** \( k \geq \delta(v) \geq \varrho(v), \quad \varrho(v) = 0: \)
Sei \( P = \{ x_1, \ldots, x_{\delta(v)+\varrho(v)} \}. \)
\( E^*_1 = \{ xv | x \in P \} \)
\( E^*_2 = (k - 1) \) parallele Kanten von \( x_j \) nach \( x_{j+1} \) für \( j = 1, 2, \ldots, \delta(v) + \varrho(v) - 1, \)
k - (\delta(v) + \varrho(v)) parallele Kanten von \( x_{\delta(v)+\varrho(v)} \) nach \( v \) und von \( v \) nach \( x_1, \)
und \( \delta(v) + \varrho(v) - 1 \) parallele Kanten von \( x_{\delta(v)+\varrho(v)} \) nach \( x_1. \)

**Fall 4)** \( \delta(v) \geq k \geq \varrho(v): \)
Teile \( P_\varrho \) in \( X, Y \) und \( W \) auf, mit \( |X| = k - \delta(v), \quad |Y| = \delta(v) \) und \( |W| =
\delta(v) - k \) und \( u \not\in W. \)
\( E^*_1 = \{ xv | x \in X \} \cup \{ wu | w \in W \} \cup \{ y_i z_i | y_i \in Y, z_i \in P_\varrho \} \)
(Die 3. Menge beschreibt Matchingkanten von \( Y \) nach \( P_\varrho \))
\( E^*_2 = (k - 1) \) parallele Kanten zwischen Knoten aus \( P \) (zyklische Reihenfolge)
Fall 5) \( \delta(v) \geq \gamma(v) \geq k: \)
Teile \( P_0 \) in \( X \) und \( Z \) mit \( |X| = \gamma(v) \) und \( |Z| = \delta(v) - \gamma(v) \) und \( u \notin Z. \)
\( E_1^* = \{ z u | z \in Z \} \cup \{ x_i y_i | x_i \in X, y_i \in P_0 \} \)
(Die 2. Menge beschreibt Matchingkanten von \( X \) nach \( P_0 \))
\( E_2^* = (k-1) \) parallele Kanten zwischen Knoten aus \( P \) (zyklische Reihenfolge)
\( T_0^* = (X_0, E_0 \cup E_1^* \cup E_2^*) \) ist \( k \)-fach kantenzusammenhängend und es gilt:
\[
|E_1^* \cup E_2^*| = \gamma_\delta
\]

Betrachten wir nun das Mischen: Seien \( T_1 = (X_1, E_1) \) mit \( u_1 \in X_1 \), \( T_2 = (X_2, E_2) \) mit \( u_2 \in X_2 \) zwei Bäume, die durch Aufteilen des Baumes \( T_0 = (X_0, E_0) \) mit \( u_0 \in X_0 \) entstanden sind (\( v_1, v_2 \) seien die zugehörigen Mischknoten).
Sei \( E' \subseteq N_{T_1^*}(v_i) \) und \( E'' \subseteq N_{T_2^*}(v_j) \) mit \( i \neq j \) und \( |E'| = |E''| \). Unter dem Mischen der Menge \( E' \) mit der Menge \( E'' \) zu einer Menge \( \hat{E} \) wollen wir folgendes verstehen:
Solange es noch Kanten in \( E' \) gibt, wähle eine Kante \( v_i x \) aus \( E' \) und eine Kante \( y v_j \) aus \( E'' \). Lösche die Kante aus den betreffenden Mengen und füge die Kante \( y x \) zu \( \hat{E} \) hinzu.

Im Algorithmus erfolgt eine Auswahl der speziellen Knoten \( u_1, u_2 \) vor der eigentlichen Augmentierung. In Abhängigkeit von \( u_0 \) und der Anzahl und Richtung der "überflüssigen Kanten" wählen wir die speziellen Knoten und beschreiben das Zusammenmischen der beiden Bäume. Wir definieren: \( \beta(T) := \gamma_\delta - \gamma_\epsilon. \)

Fall a) \( \beta(T_1) \geq 0, \ \beta(T_2) \geq 0: \)
OBdA sei \( u_0 \in X_1 \). Wähle \( u_1 = u_0 \) und \( u_2 = v_2. \) \( L_2^\delta \subset N_{T_1^*}^\delta \) mit \( |L_2^\delta| = \beta(T_2) \)
und \( e_2 \notin L_2^\delta. \) Mische die Kanten aus \( N_{T_1^*}^\delta(v_1) \) mit \( N_{T_2^*}^\delta(v_2) \) und \( N_{T_1^*}^\delta(v_1) \) mit \( N_{T_2^*}^\delta(v_2) \) so, daß \( e_1 \) und \( e_2 \) zu \( e \) zusammengemischt werden. Die Kanten aus \( L_2^\delta \) erhalten anstelle ihres alten Endknotens \( v_2 \) den Endknoten \( u_0. \)

Fall b) \( \beta(T_1) \leq 0, \ \beta(T_2) \leq 0: \)
Orientiere die Kanten um und betrachte Fall a). Anschließend drehe wieder alle Kanten um.

Fall c) \( \beta(T_1) + \beta(T_2) = 0: \)
Wähle \( u_1 = v_1 \) und \( u_2 = v_2. \) Mische die Kanten aus \( N_{T_1^*}^\delta(v_1) \) mit \( N_{T_2^*}^\delta(v_2) \)
und \( N_{T_1^*}^\delta(v_1) \) mit \( N_{T_2^*}^\delta(v_2) \) so, daß \( e_1 \) und \( e_2 \) zu \( e \) zusammengemischt werden.

Fall d) \( \beta(T_1) \geq 1, \ \beta(T_2) \leq -1, \ \beta(T_1) + \beta(T_2) \geq 1, \ \ u_0 \in X_2: \)
Wähle \( u_1 = v_1 \) und \( u_2 = v_2. \) \( L_1^\delta \subset N_{T_1^*}^\delta \) mit \( |L_1^\delta| = k - \beta(T_2) \)
und \( e_1 \in L_1^\delta \cup N_{T_1^*}^\delta(v_1) \) Mische \( L_1^\delta \) mit \( N_{T_2^*}^\delta(v_2) \) und \( N_{T_1^*}^\delta(v_1) \) mit \( N_{T_2^*}^\delta(v_2) \) so, daß \( e_1 \) und \( e_2 \) zu \( e \) zusammengemischt werden. Die Kanten \( N_{T_2^*}^\delta(v_2) - L_1^\delta \) erhalten anstelle ihres alten Endknotens \( v_1 \) den Endknoten \( u_0. \)
Fall e) $\beta(T_1) \geq 1$, $\beta(T_2) \leq -1$, $\beta(T_1) + \beta(T_2) \geq 1$, $u_0 \in X_1$:
Wähle $u_1 = v_1$ und $u_2 = v_2$. Finde eine Menge $R$, die aus kantendisjunkten Wegen von $u$ nach $v_1$ besteht, mit $|R| = k$. Sei $F \subseteq N_{T_1^*}^R(v_1)$ die Menge von Kanten, die an den Wegen aus $R$ beteiligt sind. Wähle $J \subseteq N_{T_1}^R(v_1) \cup N_{T_1}^R(v_1)$ mit $J \supseteq E_1 \cup F \cup N_{T_1^*}^R(v_1)$ und $|J| = 2k - \beta(T_2)$. Sei $K = (N_{T_1}^R(v_1) \cup N_{T_1}^R(v_1)) - J$. Es gilt: $K \subseteq N_{T_1}^R(v_1)$ und $|K| = \beta(T_1) + \beta(T_2)$. Mische $J$ mit $N_{T_1}^R(v_1) \cup N_{T_1}^R(v_1)$ so, daß $e_1$ und $e_2$ zu $e$ zusammengemischt werden. Die Kanten $K$ erhalten anstelle ihres alten Endknotens $v_1$ den Endknoten den Endknoten $u_0$.

Abb. 8.3

Fall f) $\beta(T_1) \leq -1$, $\beta(T_2) \geq 1$, $\beta(T_1) + \beta(T_2) \geq 1$:
Durch Tausch von $T_1$ und $T_2$ erhalten wir Fall d) und e).

Fall g) $\beta(T_1) \geq 1$, $\beta(T_2) \leq -1$, $\beta(T_1) + \beta(T_2) \leq -1$:
Durch Umorientieren der Kanten in $T_1$ und $T_2$ erhalten wir Fall f).

Fall h) $\beta(T_1) \leq -1$, $\beta(T_2) \geq 1$, $\beta(T_1) + \beta(T_2) \leq -1$:
Durch Umorientieren der Kanten in $T_1$ und $T_2$ erhalten wir Fall d) bzw. e).


Beim Testen wird eine Kante während des gesamten Verfahrens nur einmal untersucht. Wir erhalten deshalb für diese Operation eine Komplexität von $O(m) = O(n)$, da $m = n - 1$.

Wie oben bereits erwähnt, kann ein Baum in höchstens $n - 2$ Sterne zerlegt werden. Wir müssen diese Sterne wieder zusammenmischen, d.h. wir werden höchstens $n - 2$ Mischoperationen ausführen müssen. Der ungünstigste Fall tritt auf, wenn wir innerhalb eines $k$-fach kantenzusammenhängenden Teilgraphen $T_i^* = (X_i, E_i)$ $k$ kantendisjunkte Wege finden müssen. In Kapitel 3 haben wir einen Algorithmus kennen gelernt, der dies in $O(k \cdot |X_i|^2)$ bewerkstellt. Schwierigkeiten bereitet uns die Mächtigkeit der Menge $X_i$. Betrachten wir dazu folgendes Bild:

![Diagramm]

Abb. 8.4

Der eingekreiste Teil sei der Stern $T_1$. Es gilt: $|X_1| = O(n)$ und $#Stern = O(n)$. Seien die Kanten so orientiert, daß durch das Mischen von $T_1$ mit irgendeinem Stern ein Fall eintritt, der die Suche von $k$ kantendisjunkten Wegen nötig macht. Wir müssen also $O(n)$-mal in einem Graphen mit einer Knotenmenge der Mächtigkeit $O(n)$ einen Algorithmus der Komplexität $O(kO(n)^2)$ anwenden. Wir erhalten eine Gesamtkomplexität von $O(kn^3)$ für das Mischen.

**Bemerkung 8.3** Der Algorithmus ist nicht stark polynomial, denn das $k$ geht in die Komplexität ein. Wir können durch Kantenkapazitäten das $k$ aus der Komplexität
eliminieren, doch müßten wir die Suche nach den kantendisjunkten Wegen mit einem maximalen Fluß-Algorithmus lösen. Mit der Implementation aus Kapitel 3 erhalten wir eine Gesamtkomplexität von $O(n^4)$. 
Kapitel 9

Datenstrukturen und Rechenergebnisse


Wir haben als Programmiersprache die Sprache C gewählt. In C können wir den Graphen, oder vielmehr das Netzwerk, in eine Struktur zusammenfassen. Die Struktur graph enthält die Anzahl der Knoten n, d.h. Knoten werden durch natürliche Zahlen zwischen 1 und n repräsentiert. Weiterhin enthält graph die Anzahl der Kanten m und mehrere Kantenlisten s, t, x, u. s[e] enthält den Anfangsknoten der Kante e, t[e] enthält den Endknoten der Kante e, x[e] enthält den Flußwert auf der Kante e und u[e] enthält die obere Schranke für einen Flußwert auf der Kante e.

Im Algorithmus von Frank werden die Knoten mit Kosten belegt. Die Kosten für eine eintreffende Kante in Knoten v sind in c[v] abzulesen, die Kosten für eine auslaufende Kante in c[v+n]. Dieses Feld befindet sich ebenfalls in der Struktur graph.

Es ist oft notwendig die inzidenten Kanten eines Knotens v zu berechnen. Dafür haben wir eine verkettete Liste. Eine solche Liste besteht aus einem n–dimensionalen Feld first und einem m–dimensionalen Feld next. first[v] soll die erste mit v inzidente Kante e enthalten. Da e eine Zahl zwischen 1 und m ist, kann sie als Index für das Feld next verwendet werden. In next[e] soll also die nächste mit v inzidente Kante stehen. So können wir die Liste next durchlaufen, bis wir auf einen Eintrag next[next[...first[v]...]] = 0 kommen. Dieser signalisiert, daß es keine weitere mit v inzidente Kante gibt. Teilen wir die inzidenten Kanten auf, in die Kanten, die v verlassen und in die Kanten, die in v eintreffen, so erhalten wir zwei verkettete
Listen, bestehend aus den Listen \texttt{firstf, nextf, firstb und nextb}, die wir im folgenden auch \textit{Forward- und Backwardstar} nennen wollen. Ein kleines Beispiel soll die verwandte Datenstruktur verdeutlichen. Die Zahlenpaare $a|b$ an den Kanten stehen einmal für den Wert eines Flusses und für die obere Kapazität der Kante.

\begin{center}
\includegraphics[width=0.5\textwidth]{image.png}
\caption{Abb. 9.1}
\end{center}

\begin{center}
n = 4 \quad m = 6
\end{center}

\begin{tabular}{|c|c|c|c|c|}
\hline
s: & 1 & 1 & 2 & 3 & 1 & 4 \\
\hline
t: & 2 & 2 & 3 & 4 & 3 & 2 \\
\hline
x: & 0 & 3 & 4 & 4 & 1 & 0 \\
\hline
u: & 5 & 9 & 4 & 7 & 1 & 9 \\
\hline
\end{tabular}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\texttt{firstf:} & 1 & 3 & 4 & 6 \\
\hline
\texttt{nextf:} & 2 & 5 & 0 & 0 & 0 & 0 \\
\hline
\texttt{firstb:} & 0 & 6 & 5 & 4 \\
\hline
\texttt{nextb:} & 2 & 0 & 0 & 0 & 3 & 1 \\
\hline
\end{tabular}
\end{center}

Wir haben den Algorithmus zur Augmentierung zu $\lambda_G = 1$, den Algorithmus zur Berechnung der Kantenzusammenhangszahl $\lambda_G$ und den Algorithmus von Frank (in seinen verschiedenen Versionen) programmiert. Hier wollen wir nun einige benutzte Datenstrukturen vorstellen.

Datenstrukturen im Algorithmus zur Augmentierung zu $\lambda_G = 1$: Im ersten Teil des Verfahrens mußte der Kondensationsgraph mit Hilfe eines erweiterten DFS-Verfahrens bestimmt werden. Dazu benötigte das Verfahren einen \textit{Kellerspeicher} oder \textit{Stack}. Bei uns besteht der Stack aus einem Feld genügender Größe (es können höchstens $n$ Elemente auf dem Stack liegen) und einem Zeiger auf das oberste Element. Die Operationen für diesen Stack sind das Anlegen eines Stacks (\texttt{create}), das Ablegen eines Elements auf den Stack (\texttt{push}) und das Herunterholen des obersten Stackelements (\texttt{pop}). Mit Hilfe dieser komplexen Datenstruktur und dem Forwardstar läßt sich der Algorithmus zur Bestimmung des Kondensationsgraphen leicht programmieren.

Für den Kondensationsgraphen legen wir einen neuen Graphen $g_0$ an. Für ihn erzeugen wir ebenfalls einen For- und Backwardstar. Ein Knoten $v$ aus dem Orgi-
nalgraph liegt in der Komponente alpha[v]. Der Repräsentant der Komponente i ist der Knoten beta[i].

Im Kondensationsgraphen müssen geeignete Quellen und Senken gesucht werden. Dazu legen wir uns ein Feld st an, das für einen Knoten v den Typ des Knotens enthält, d.h. ob er Quelle, Senke, isolierter oder "normaler" Knoten ist. Die Repräsentation des Typs verläuft über eine natürliche Zahl. Wird eine Quelle oder eine Senke im Laufe des Algorithmus in die Menge der ausgezeichneten Knoten übernommen, so wird dies durch Negation dieser Zahl gekennzeichnet. Wir können für einen Knoten in konstanter Zeit entscheiden, ob er in der ausgezeichneten Menge ist, oder nicht.

Eine Menge, bei der selten oder nie getestet werden muß, ob sich ein bestimmtes Element in ihr befindet, wird durch ein Feld der Größe der Mächtigkeit der Menge repräsentiert. Die Mengenelemente stehen ungeordnet aufeinanderfolgend in diesem Feld (Typ1). Eine Menge, bei der häufig die Zugehörigkeit eines Elements getestet wird, wird durch ein Feld der Größe der Anzahl der möglichen Elemente repräsentiert. Ist ein Element v in der Menge, so enthält der v-te Eintrag des Feldes eine Zahl ungleich Null, sonst Null. Typischerweise gibt es hier meist Mengen von Knoten oder Kanten, also werden diese Mengen durch Felder der Dimension n bzw. m repräsentiert (Typ2). So sind z.B. die Mengen der ausgezeichneten Quellen, Senken und isolierten Knoten als Typ1-Mengen abgespeichert.

Datenstrukturen im Algorithmus zur Berechnung der Kantenzusammenhangszahl:

Haben wir alle Nachfahren des Startknotens besucht und gibt es trotzdem noch einen unbesuchten Knoten, so können wir mit dem Verfahren stoppen. Denn der Graph ist nicht einmal stark zusammenhängend. Also folgt λG = 0. Alle anderen Schritte des Verfahrens lassen sich direkt programmieren.

Datenstrukturen im Algorithmus zur Berechnung eines maximalen Flusses:
Diesen Algorithmus müssen wir besonders effizient programmieren, denn die Zahl der Aufrufe im Algorithmus von Frank (mit der Komplexität O(n^3)) steigt selbst bei relativ kleinen Netzwerken enorm. Betrachten wir ein Netzwerk mit 250 Knoten (dies ist die Knotenzahl des größten getesteten Netzwerks). Wir müssen im schlechtesten Fall (3 * 250 − 1) * (250 − 2) maximale Flüsse berechnen (nur Schritt
4). Nehmen wir für eine Berechnung eines maximalen Flusses in einem Netzwerk mit 251 Knoten (nur) eine halbe Sekunde an, so rechnen wir allein für die Auswertung der maximalen Flüsse über 25 Stunden. Dieses wird sich in der Zeitanalyse des Algorithmus von Frank wiederspiegeln. Beim maximalen Fluss-Algorithmus von Goldberg–Tarjan haben wir zwei Freiheitsgrade (s. Kapitel 3): Zum einen die Bestimmung eines zulässigen Labelings und zum anderen die Auswahl der aktiven Knoten. In der Arbeit von Derigs und Meier [DERIGS89] sind verschiedene Varianten getestet worden. Für unsere Testnetzwerke sind das exakte Labeling und die highest-first-Auswahl am geeignetsten. Wie in Kapitel 3 beschrieben, können wir das exakte Labeling mit Hilfe des BFS-Verfahrens berechnen. Wir legen den Startknoten (das ist in diesem Fall die Senke) in einer Schlanke ab. Eine Schlanke besteht aus einem Feld \( Q \) genügender Größe (in unserem Fall reicht ein Feld der Dimension \( n \)) und zwei Zeiger, einer zeigt auf den Anfang (\( qa \)) und der andere auf das Ende (\( qe \)) der Schlanke. Die Elemente zwischen \( qa \) und \( qe \) sind die Elemente der Schlanke. (Es kann \( qa > qe \) gelten. Dann sind alle Elemente von \( qa \) bis \( n \) und von \( 1 \) bis \( qe \) gemeint.) Wir legen im BFS-Verfahren die neu besuchten Knoten immer am Ende der Schlanke ab. Wählen wir den Knoten, dessen Nachbarn wir als nächstes besuchen wollen, immer am Anfang der Schlanke, so gewährleisten wir den geforderten Durchlauf durch den Graphen.

Bei der Verwaltung der aktiven Knoten müssen wir folgende Operationen effizient durchführen: Wir müssen einen aktiven Knoten deaktivieren und einen inaktiven Knoten aktivieren. Weiterhin muß die Suche nach dem aktiven Knoten mit größtem Labeling möglichst einfach sein. Um diese Operationen effizient durchzuführen, werden wir eine verkettete Liste ähnlich einem Forward- bzw. Backwardstar anlegen. Wir verwenden ein Feld \( Q \) und ein Feld \( nQ \) jeweils mit der Dimension \( n \). In \( Q[i] \) steht ein aktiver Knoten mit Label \( i \); in \( nQ[Q[i]] \) steht der nächste und in \( nQ[nQ[Q[i]]] \) ein weiterer. Wir finden solange aktive Knoten mit Label \( i \) bis \( nQ[nQ[i]] = 0 \) gilt. Wählen wir im Verfahren den Knoten mit dem größten Label aus dem Feld \( Q \), so ist das Entfernen dieses Knotens in konstanter Zeit zu erledigen. Genauso verhält es sich mit dem Einfügen von neuen aktiven Knoten. Um den Knoten mit dem zur Zeit größten Label zu finden, führen wir einen Zeiger \( highactive \) ein, welcher bei Veränderung der aktiven Knoten geeignet umgesetzt werden muß. Wird ein Knoten ausgewählt, so ist es der mit dem größten Label. Im Iterationsschritt werden nur Knoten mit kleinerem Label aktiv, da nur Flüßeinheiten zu Knoten mit kleinerem Labeling verschickt werden. Der Zeiger \( highactive \) muß also nur im Relabel-Schritt aktualisiert werden. Wird der ausgewählte Knoten \( v \) inaktiv, so müssen wir ausgehend vom Niveau des Knotens \( v \) absteigend im Feld \( Q \) suchen, bis wir auf einen Eintrag ungleich 0 gestoßen sind. Auf dieses Niveau muß der Zeiger \( highactive \) nun zeigen.

Eine weitere Operation muß sehr effizient durchgeführt werden: Das Auffinden von Gaps. Wie schon in Kapitel 3 angemerkt, können wir uns für jedes Niveau \( i \)

Datenstrukturen im Algorithmus von Frank:

Im Algorithmus haben wir unterschiedliche Kanten. Zum einen sind es die Originalkanten des Graphen, zum anderen die mit $s$ inzidenten Kanten und die Kanten, die beim Abspalten neu zum Graphen hinzukommen. Da wir in dem erweiterten Graphen einen maximalen Fluß berechnen wollen, müssen diese Kanten in den Kantenlisten $s, t$ und $u$ auftauchen. Weiterhin müssen wir dafür sorgen, daß gewisse Knotenpaare $u, v$ auf derselben Seite des Schnitts liegen. Dieses können wir erreichen, indem wir eine Kante $uv$ bzw. $vu$ in die Kantenlisten einfügen und ihr eine unendliche bzw. genügend große Kapazität geben. Die verwaltete Kantenliste können wir uns wie folgt vorstellen:

<table>
<thead>
<tr>
<th>Orginalkanten</th>
<th>$sv$</th>
<th>$vs$</th>
<th>$uv$</th>
<th>neue Kanten</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Abb. 9.2

Die ersten $m$ Kanten sind die Originalkanten des Graphen. Danach kommen die Kanten $sv$ ($n$ Stück) und $vs$ ($n$ Stück). Es folgt die Kante $uv$ mit genügend großer
Kapazität, die dafür sorgt, daß die Knoten \( u \) und \( v \) auf derselben Seite des Schnitts liegen. Nach dieser Kante tauchen die beim Abspalten erzeugten neuen Kanten auf. Wird ein Paar \( xs, sy \) abgespalten und existiert die Kante \( xy \) bereits im Graphen, so wird natürlich nur die Kapazität dieser Kante vergrößert.

Um die maximale Anzahl von möglichen Abspaltungen durch \( O(n) \) zu beschränken, müssen wir eine Mengenfamilie \( F \) von disjunkten, kritischen Mengen verwalten. Wir müssen in diese Mengenfamilie eine gefundene kritische Menge von Knoten eintragen und zwar so, daß weiterhin alle Mengen aus \( F \) disjunkt und kritisch sind. Dazu sind folgende Operationen nötig: die Vereinigung, die Subtraktion, das Eliminieren und das Hinzunehmen einer Menge zu \( F \). Durch disjunkte Bäume können wir eine Mengenfamilie, auf welcher nur eine Vereinigungsoperation anzuwenden ist, darstellen. Hier kommen aber weitere Operationen ins Spiel. Die Mengenfamilie \( F \) besteht bei uns aus einem Feld der Größe \( n \). Die Mengen in dieser Familie sind durch natürliche Zahlen von 1 bis \( \text{high} \) repräsentiert. Ist ein Knoten \( v \) nun in der Menge \( i \), so finden wir \( F[i] = i \) vor. Die Knoten \( u \), die noch nicht in einer Menge der Familie enthalten sind, haben \( F[u] = 0 \). Ist im Iterationsschritt eine kritische Menge \( T \) entdeckt worden, so müssen wir alle Mengen \( i \) aus \( F \) finden, die einen nicht leeren Schnitt mit \( T \) haben. Die Menge \( T \) ist vom Typ 2. Um die Differenz von \( i - T \) zu berechnen, durchlaufen wir das Feld \( F \) und prüfen, ob der Knoten sowohl in der Menge \( i \) (also \( F[i] = i \)) als auch in der Menge \( T \) (also \( T[v] \neq 0 \)) liegt. Für diese Knoten berechnen wir den Ein- und Ausgangsgrad und stellen so fest, ob die Differenz der Mengen kritisch ist. Ist dies der Fall, so müssen wir die Menge \( i \) durch \( i - T \) in \( F \) ersetzen. Das heißt, wir müssen erneut durch die Menge \( F \) laufen und die Einträge der betreffenden Knoten in \( F \) auf 0 setzen. Ist die Differenz nicht kritisch, so bilden wir die Vereinigung \( i \cup T \) und speichern sie in \( T \). Wiederum ist ein Durchlauf durch das Feld \( F \) nötig. Sind alle Mengen aus \( F \) abgearbeitet, so bleibt das Einfügen der (modifizierten) Menge \( T \). Eventuell sind bei der Differenzenbildung ganze Mengen aus \( F \) verschwunden. Wir wollen aber erreichen, daß jede Zahl zwischen 1 und \( \text{high} \) einer Menge aus \( F \) entspricht. Das heißt, daß wir u.U. umnumerieren müssen. Die Menge \( T \) wird als Menge 1 in \( F \) eingetragen. Die übrigen Mengen erhalten Nummern zwischen 2 und \( |F| = \text{high} \). \( |F| \) bezeichnet die Anzahl der Mengen in \( F \). Dieses Umnumerieren macht wieder einen Durchlauf durch \( F \) nötig. Wir sehen aber, daß die Bearbeitung der Mengenfamilie in einer Komplexität von höchstens \( O(n^2) \) zu erledigen ist. Sie spielt also bei der Gesamtkomplexität des Verfahrens eine untergeordnete Rolle.
9.1 Netzwerkgeneratoren


Der Netzwerkgenerator RMFGEN erhält die Parameter $a, b, c_1, c_2 (> c_1)$ und erzeugt Netzwerke folgender Gestalt:

![Abb. 9.5](image)

Der erzeugte Graph besteht aus $b$ Rahmen, bestehend aus je $a^2$ Knoten, die auf den Gitterpunkten eines Quadrats mit Seitenlänge $a$ liegen. Jeder Knoten eines Rahmens ist mit seinen Gitternachbarn durch je ein Paar antiparalleler Kanten (zwei Kanten $ab, ba$ heißen antiparallel) verbunden. In jedem Rahmen gibt es also $4a(a - 1)$ Kanten. Die Kapazität auf diesen Kanten beträgt $c_2 a^2$. Weiterhin gibt es $a^2$ Kanten von den Knoten eines Rahmens zu den zufällig permutierten Knoten des darauffolgenden Rahmens mit Kapazität zufällig gewählt zwischen $[c_1, c_2]$. Im gleichen Maße gibt es $a^2$ Kanten von einem Rahmen zum vorherigen Rahmen, ebenfalls mit zufälliger Kapazität aus $[c_1, c_2]$. Der erzeugte Graph besteht also aus $n = a^2 b$ Knoten und $m = 6a^2 b - 4ab - 2a^2$ Kanten. Ein Weg im Inkrementgraph von der Quelle zur Senke hat mindestens die Länge $b - 1$. Ein minimaler Schnitt, der Quelle und Senke trennt, liegt zwischen zwei Rahmen und enthält $a^2$ Kanten, denn eine Kante innerhalb eines Rahmens kann nicht im Schnitt liegen, da ihre Kapazität zu groß ist.

Die Dichte der Netzwerke, die vom RMFGEN-Generator erzeugt werden, ist relativ gering. Um Netzwerke mit größerer Dichte zu erzeugen, vergrößern wir die Anzahl der Kanten zwischen zwei Rahmen. Wir benötigen einen weiteren Parameter $d$, der die Anzahl der Kanten eines Knotens zum nächsten Rahmen und zum vorherigen...
Rahmen bestimmt. Wir müssen nur wenige Änderungen durchführen, um die Eigenschaften der RMFGEN-Netzwerke beizubehalten. Wir müssen garantieren, daß eine Gitterkante nie im Schnitt liegt. Erhöhen wir die Kapazität auf \( c_2a^2d \), so erhalten wir das Gewünschte. Der Graph besteht nun aus \( m = 2a^2b(2 + d) - 4ab - 2a^2d \) Kanten. Durch Vergrößern des Parameters \( d \) können wir Netzwerke mit größerer Dichte erzeugen.

Wir haben mit dem RMFGEN-Generator Netzwerke mit folgenden Werten für die Parameter \( a, b, c_1, c_2, d \) erzeugt: \( a = 2, 3, 4, 5, b = 2, 3, ..., 10, c_1 = 0, c_2 = 10, 100, 1000 \) und \( d = 1, 2, 4 \). Die kleinsten Netzwerke bestehen aus 8 Knoten und 24 Kanten (\( a = b = 2, d = 1 \)), die größten aus 250 Knoten und 2600 Kanten (\( a = 5, b = 10 \) und \( d = 4 \)). Bei allen möglichen Kombinationen kommen wir auf eine Anzahl von 324 Netzwerken. Die Dichte (Dichte von \( G = m/(n(n-1)) \)) der erzeugten Netzwerke liegt zwischen 0.86 und 0.02. Die Kantenzusammenhangszahl liegt zwischen 7 und 12256.

Die oben genannten Netzwerke eignen sich nicht für den Algorithmus zur Augmentierung zu \( \lambda_G = 1 \), da alle Graphen schon stark zusammenhängend sind. Hier werden Zufallsgraphen mit wenig Struktur benutzt. Der Generator erhält die Anzahl der Knoten \( n \) und die gewünschte Dichte \( d \) als Parameter und setzt eine Kante \( uv \) mit der Wahrscheinlichkeit \( d \).

9.2 Rechenergebnisse


Für den Algorithmus zur Bestimmung der Kantenzusammenhangszahl wollen wir die benötigte Rechenzeit und die Größe der Knotenmenge \( T \), zwischen deren Knoten (in zyklischer Reihenfolge) maximale Flüsse berechnet werden müssen, näher betrachten. Die Komplexität des Algorithmus beträgt \( O(n^4) \), deshalb tragen wir die vierte Wurzel der ermittelten Zeitwerte auf. Es sollte sich also eine Gerade ergeben.

Aus der Tabelle im Anhang ersehen wir, daß die Mächtigkeit von \( T \) sehr nah an \( \frac{n}{2} \) liegt. Dies geschieht, da bei einer Zusammenhangszahl größer oder gleich 1 der Baum, der vom DFS-Verfahren erzeugt wird, zu einer Liste degeneriert. Es werden alle Knoten erreicht, ohne daß oft das Zentrum der Aktivität in den Vorgängerknoten verschoben wird und von dort neue Knoten entdeckt werden.

Den Algorithmus von Frank haben wir in 4 verschiedenen Versionen programmiert. Die triviale \( O(n^6) \) Implementation werden wir in Zukunft Version 0 nennen. Die Version mit der Mengenfamilie von disjunkten kritischen Mengen heißt Version 1. In Schritt 1 und 2 des Algorithmus konnten wir die Information eines minimalen \( s, u \) bzw. \( u, s \)-Schnitts ausnutzen, um den reduzierten Wert einer Kante \( su \) bzw. \( us \) zu bestimmen. Ähnliches können wir u.U. bei Schritt 4 anwenden (diese Strategie führte nicht zur Verminderung der Komplexität). Diese Implementenation heißt Version 2. Die Kombination der Versionen 1 und 2 ergibt eine dritte Implementation, genannt Version 3.
Im Anhang befinden sich einige Tabellen zu diesen verschiedenen Versionen. Hier wollen wir die auffälligsten Ergebnisse graphisch aufzeigen. Dazu haben wir drei Graphen mit jeweils 48 Knoten benutzt. Die Kantenzusammenhangszahl \( \lambda_G \) lag zwischen 53 und 57. Um die Abhängigkeiten deutlicher zu machen, haben wir manchmal die aufeinanderfolgenden Werte eines festen Graphen mit Linien verbunden.

Zunächst wollen wir die Anzahl der versuchten Abspaltungen betrachten. Hier müssen wir zwischen erfolgreichem und erfolglosem Abspaltungen unterscheiden. Eine versuchte Abspaltung eines Paares \( u, v \) war erfolgreich, wenn die Kapazitäten auf den Kanten \( us \) und \( sv \) um eine Zahl \( z > 0 \) vermindert wurden und die Kapazität der Kante \( uv \) um \( z \) vergrößert wurde, sonst nennen wir die Abspaltung erfolglos.

Die Abbildung zeigt die Anzahl der versuchten Abspaltungen bei Version 0 bzw. 2, die durch \( n(n - 1) \) beschränkt ist. Das Einknicken der Kurve (in der Abbildung nicht mehr zu erkennen) erfolgt erst bei sehr großem \( k \). Bei \( k = 10000 \) sind es ca. 1000 versuchte Abspaltungen bei \( k = 20000 \) sind es nur 30 mehr.
Anders ist es bei den erfolgreichen Abspaltungen. Hier knickt die Kurve schon bei relativ kleinem \( k \) (ca. \( k = 1250 \)) ein, und bleibt dann (fast) konstant. Die Anzahl der wirklich notwendigen Abspaltungen ist (gerade für große \( k \)) wesentlich kleiner als die versuchten Abspaltungen. Bei der Version 1 (ohne Abbildung) ist die Situation viel besser. Nur sehr selten wird ein getestetes Paar nicht abgespalten. Der Aufwand zur Verwaltung der Mengenfamilie, der von der Komplexität zu vernachlässigen ist, geht aber in so starkem Maße in die Rechenzeit ein, daß die Version 2 bis zu 10-mal schneller ist als Version 1. (Die Kombination der beiden Strategien wird zu dem besten Ergebnis führen.)

Als nächstes wollen wir die Summe der Kapazitätserhöhung näher betrachten.
Für $k < \lambda_G$ muß natürlich an dem Graphen nichts geändert werden. Die Summe der Kapazitätserhöhung steigt mit wachsendem $k$ und nähert sich dem Wert $nk$ (Gerade in der Abbildung) an. Der Wert $nk$ ist die Summe der Kapazitätserhöhungen einer trivialen Augmentierung: Wir suchen einen Kreis in $G$, der alle Knoten enthält. (Falls so ein Kreis nicht existiert, nehmen wir neue Kanten hinzu.) Auf den Kanten des Kreises setzen wir die Kapazität auf $k$. Damit ist der Graph mindestens $k$-fach kantenzusammenhängend und die Kapazitätserhöhung beträgt $nk$. Für $k \geq \lambda_G$ ist diese triviale Augmentierung eine gute Approximation für die optimale Lösung.

Das Verhalten der Kapazitätserhöhung spiegelt sich ebenfalls in der Anzahl der neuen Kanten wieder.
Auch hier steigt die Zahl mit wachsendem $k$ und bleibt dann ab einem festen $k$ (fast) konstant. Wir können dies ebenfalls mit der trivialen Augmentierung (s. Kapazitätserhöhungen) erklären. Ab einem festen $k$ erhöht der Algorithmus die Kapazitäten eines (erzeugten) Kreises. Die Anzahl der fehlenden Kanten wird dann nicht mehr größer, wenn einmal ein Kreis mit allen Knoten erzeugt wurde.

Wenden wir uns nun dem Zeitverhalten der Schritte 1 und 2 zu. Wir müssen $O(n)$ maximale Flüsse berechnen. Im günstigsten Fall müssen wir $2n$, im ungünstigsten Fall $4n$ maximale Flüsse berechnen. Dieses hängt von dem Auftreten des Falles B2 (s. Kapitel 7) ab. Bestand beim Reduzieren der jeweils letzten Kante $su$ ($us$) eine Seite des minimalen $s,u$ ($u,s$)-Schnitts nur aus dem Knoten $s$, so mußten wir zusätzlich $n$ maximale Flüsse berechnen. Das Auftreten dieses Falles ist in der folgenden Abbildung zu sehen.

Betrachten wir gleichzeitig die Rechenzeit für die Schritte 1 und 2, so ist der theoretische Zusammenhang auch in den Beispielen wiederzuerkennen.
Wir geraten mit Sicherheit nicht in einen Fall B2, wenn die Kapazitätserhöhung größer als $k$ ist. Dieses stellt sich bei etwa $k = 130$ ein. Auch in den Tabellen zeigt sich, daß bei $k \approx 2\lambda_{G}$ kaum noch der Fall B2 auftritt.

Da die Komplexität des Algorithmus $O(n^4)$ beträgt, tragen wir die vierte Wurzel der ermittelten Zeitwerte auf. Es sollte sich also eine Gerade ergeben.

Betrachten wir als letztes das Zeitverhalten von Schritt 4. Wie wir oben gesehen haben, hängt die Rechenzeit von der Anzahl der versuchten Abspaltungen ab. Bei der Version 0 werden eventuell $n^2$ Abspaltungen getestet. Das heißt, daß $n^2$ maximale Flüsse berechnet werden müssen. In unseren Beispielen wird die Schranke $n^2$ nicht ganz erreicht. Es sind aber immerhin noch ca. $\frac{2n^2}{5}$ versuchte Abspaltungen ($k = 20000$). Dementsprechend ist die verbrauchte Rechenzeit für Schritt 4. Bis

Literaturverzeichnis


Hiermit versichere ich an Eides Statt, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel benutzt habe.