Effiziente Algorithmen und Datenstrukturen II

Prof. Dr. Christian Scheideler

Technische Universität München, 25. April 2006

1 Algorithmen für maximale Flüsse

1.1 Flüsse

Ein *Flussnetzwerk* $G = (V, E)$ ist ein gerichteter Graph, in dem jede Kante $(v, w) \in E$ eine *Kapazität* $c(v, w) > 0$ besitzt. (Für alle Kanten $(v, w) \notin E$ nehmen wir an, dass ihre Kapazität 0 ist. Dazu gehören alle Kanten der Form (v, v).) Es gibt in G zwei ausgezeichnete Knoten, die Quelle s und die Senke t.

Ein (s, t)-*Fluss* (oder einfach nur *Fluss* wenn s und t klar sind) ist eine Funktion $f : E \rightarrow \mathbb{R}_0^+$, die folgende Bedingungen erfüllt:

1. *Flusserhaltung*: für alle $v \in V \setminus \{s, t\}$ gilt $\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$
2. *Kapazitätseinhaltung*: für alle $v, w \in V$ gilt $f(v, w) \leq c(v, w)$

Der *Flusswert* von f ist dann definiert als

$$|f| = \sum_{v \in V} f(s, v) - \sum_{u \in V} f(u, s)$$

Siehe Abb. 1 für einen legalen Fluss.

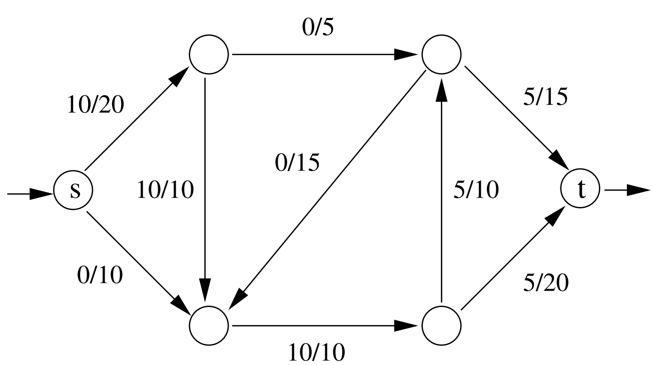

Figure 1: Ein (s, t)-Fluss mit Wert 10. Jede Kante ist bezeichnet mit Fluss/Kapazität.

Es ist nicht hart zu zeigen, dass $|f|$ auch gleich dem überschüssigen Fluss in die Senke t ist. Zunächst stellen wir dazu fest, dass

$$\begin{aligned}\sum_{v \in V}\left(\sum_{w \in V} f(v,w) - \sum_{u \in V} f(u,v)\right) &= \sum_{v \in V}\sum_{w \in V} f(v,w) - \sum_{v \in V}\sum_{u \in V} f(u,v) \\ &= \sum_{v \in V}\sum_{w \in V} f(v,w) - \sum_{u \in V}\sum_{v \in V} f(u,v) = 0\end{aligned}$$

ist, da beide Summen den Fluss über alle Kanten aufsummieren. Auf der anderen Seite gilt aufgrund des Flusserhaltungsprinzips, dass

$$\begin{aligned}&\sum_{v \in V}\left(\sum_{w \in V} f(v,w) - \sum_{u \in V} f(u,v)\right) \\ &\quad= \left(\sum_{w \in V} f(s,w) - \sum_{u \in V} f(u,s)\right) + \left(\sum_{w \in V} f(t,w) - \sum_{u \in V} f(u,t)\right) \\ &\quad= |f| + \left(\sum_{w \in V} f(t,w) - \sum_{u \in V} f(u,t)\right)\end{aligned}$$

ist. Daraus folgt, dass

$$|f| = \sum_{u \in V} f(u,t) - \sum_{w \in V} f(t,w)$$

Für einen gegebenen Fluss f und eine Kante e sagen wir, dass f Kante e *saturiert* wenn $f(e) = c(e)$ ist und f Kante e *meidet* wenn $f(e) = 0$ ist.

1.2 Schnitte

Ein (s,t)-*Schnitt* (S,T) (oder einfach nur *Schnitt* wenn s und t klar sind) ist eine Partition der Knoten in zwei disjunkte Teilmengen S und T (d.h. $S \cup T = V$ und $S \cap T = \emptyset$) mit der Eigenschaft, dass $s \in S$ und $t \in T$. Die *Kosten* eines Schnitts ist die Summe der Kapazitäten der Kanten von S nach T:

$$|(S,T)| = \sum_{v \in S}\sum_{w \in T} c(v,w)$$

Ein minimaler Schnitt ist ein Schnitt mit minimalen Kosten. Ein Beispielschnitt ist in Abb. 2 gegeben.

Intuitiv sollte der minimale Schnitt die einfachste Art sein, jeglichen Fluss von s nach t zu unterbinden. In der Tat ist es nicht schwer zu zeigen, dass der Wert eines jeden (s,t)-Flusses durch die Kosten eines jeden (s,t)-Schnittes nach oben hin beschränkt ist. Dazu betrachten wir einen beliebigen Fluss f und einen beliebigen Schnitt (S,T). Es gilt:

$$\begin{aligned}|f| &= \sum_{w \in V} f(s,w) - \sum_{u \in V} f(u,s) && \text{per Definition} \\ &= \sum_{v \in S}\left(\sum_{w \in V} f(v,w) - \sum_{u \in V} f(u,v)\right) && \text{aufgrund der Flusserhaltung} \\ &= \sum_{v \in S}\left[\left(\sum_{w \in S} f(v,w) - \sum_{u \in S} f(u,v)\right) + \left(\sum_{w \in T} f(v,w) - \sum_{u \in T} f(u,v)\right)\right]\end{aligned}$$

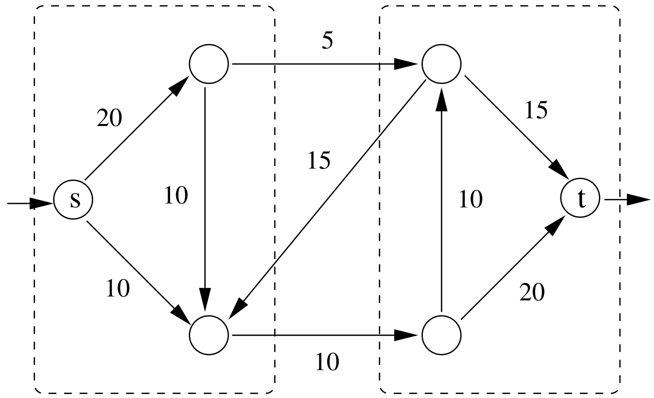

Figure 2: Ein (s,t)-Schnitt mit Kosten 15. Jede Kante ist bezeichnet mit ihrer Kapazität.

$$
\begin{aligned}
&= \sum_{v\in S}\left(\sum_{w\in T} f(v,w) - \sum_{u\in T} f(u,v)\right) \qquad \text{da sich Kanten in } S \text{ aufheben}\\
&\leq \sum_{v\in S}\sum_{w\in T} f(v,w) \qquad \text{da } f(u,v)\geq 0\\
&\leq \sum_{v\in S}\sum_{w\in T} c(v,w) = |(S,T)|
\end{aligned}
$$

1.3 Das Maxflow-Mincut Theorem

Im folgenden nehmen wir an, dass G ein einfacher gerichteter Graph ist, d.h. für jedes Knotenpaar $v,w\in V$ ist höchstens eine der Kanten (v,w) und (w,v) in E. Das ist keine Einschränkung, denn für den Fall, dass (v,w) und (w,v) für ein Paar $v,w\in V$ in E sind, können wir einen künstlichen Knoten x einführen und (w,v) ersetzen mit (w,x) und (x,v) mit Kapazitäten $c(w,v)$ (siehe Abb. 3). Wie leicht zu überprüfen ist, wird dadurch nicht der maximale Fluss oder minimale Schnitt verändert.

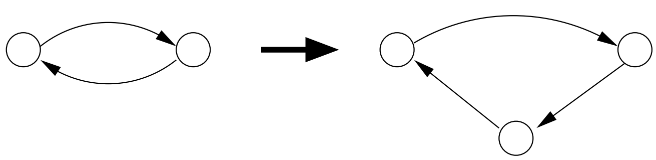

Figure 3: Transformation zur Erzeugung eines einfach gerichteten Graphen.

Für einen einfachen gerichteten Graphen G und einen Fluss f definieren wir die *residuale Kapazität* einer Kante (u,v) in G_f als

$$
c_f(u,v) = \begin{cases} c(u,v)-f(u,v) & \text{falls } (u,v)\in E\\ f(v,u) & \text{falls } (v,u)\in E\\ 0 & \text{sonst}\end{cases}
$$

Da $f(u,v)\geq 0$ und $f(u,v)\leq c(u,v)$ für alle Kanten (u,v), ist die residuale Kapazität immer größer gleich 0. Wir definieren das *residuale Netzwerk* $G_f=(V,E_f)$ als den Graphen, der alle Kanten (u,v) enthält mit $c_f(u,v)>0$. G_f ist im Allgemeinen kein Teilgraph von G. Auch kann es passieren, dass für ein Paar $u, v \in V$ gilt $c_f(u,v) > 0$ und $c_f(v,u) > 0$. Ein *augmentierender Pfad* p in G_f ist ein einfacher Pfad (d.h. kein Knoten wird doppelt besucht), der in s startet und in t endet. Die *residuale Kapazität* eines augmentierenden Pfades $p = (s = v_1, v_2, \ldots, v_\ell = t)$ ist

$$c_f(p) = \min\{c_f(v_i, v_{i+1}) \mid i \in \{1, \ldots, \ell - 1\}\}$$

Der folgende Satz ist von zentraler Bedeutung für maximale Fluss Probleme:

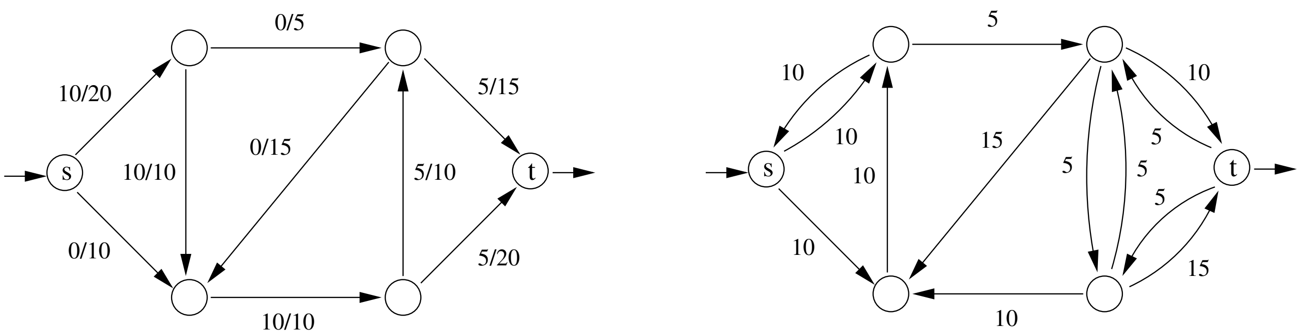

Figure 4: Ein Fluss f und sein residuales Netzwerk.

Satz 1.1 (Maxflow-Mincut Theorem) *Sei G ein Flussnetzwerk und f ein Fluss, so sind folgende Aussagen äquivalent:*

1. *f ist ein maximaler Fluss in G*
2. *G_f hat keinen augmentierenden Pfad*
3. *$|f| = |(S,T)|$ für einen Schnitt (S,T) in G*

Beweis. Wir zeigen zunächst $(1) \Rightarrow (2)$ (d.h. aus (1) folgt (2)), dann $(2) \Rightarrow (3)$, und dann $(3) \Rightarrow (1)$. Aus diesen drei Implikationen folgt der Satz.

$(1) \Rightarrow (2)$: Wir zeigen statt dessen die äquivalente Aussage $\neg(2) \Rightarrow \neg(1)$, d.h. wenn G_f einen augmentierenden Pfad hat, dann ist f kein maximaler Fluss in G. Nehmen wir also an, für einen Fluss f habe G_f einen augmentierenden Pfad $p = (v_1 = s, v_2, \ldots, v_\ell = t)$. Sei $x = \min_i c_f(v_i, v_{i+1})$. Da alle Kanten in G_f eine positive residuale Kapazität besitzen, gilt $x > 0$. Wir definieren den neuen Fluss $f' : E \to \mathbb{R}_0^+$ wie folgt für alle Kanten (u,v):

$$f'(u,v) = \begin{cases} f(u,v) + x & \text{falls } (u,v) \in p \\ f(u,v) - x & \text{falls } (v,u) \in p \\ f(u,v) & \text{sonst} \end{cases}$$

Es ist einfach zu sehen, dass f' die Flusserhaltungsbedingung erfüllt, da p für jede eingehende Kante in einen Knoten $v \notin \{s,t\}$ auch eine ausgehende Kante haben muss. Wir müssen noch zeigen,

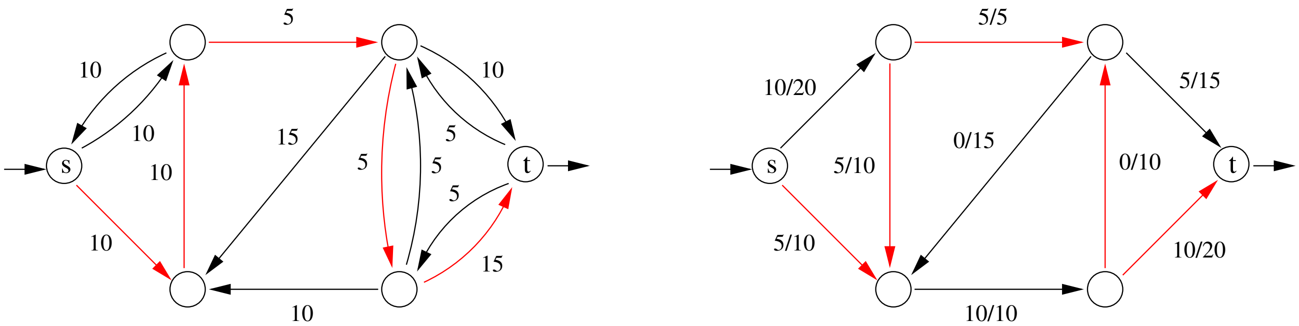

Figure 5: Ein augmentierender Pfad in G_f mit Wert 5 und der augmentierte Fluss f'.

dass f' auch die Kapazitätsbedingungen erfüllt. Betrachte dazu eine beliebige Kante (u, v) in G. Falls $(u, v) \in p$, dann gilt $f'(u, v) > f(u, v) \geq 0$ und

$$\begin{aligned} f'(u,v) &= f(u,v) + x && \text{per Definition von } f' \\ &\leq f(u,v) + c_f(u,v) && \text{per Definition von } x \\ &= f(u,v) + c(u,v) - f(u,v) && \text{per Definition von } c_f \\ &= c(u,v) \end{aligned}$$

Falls $(v, u) \in p$, dann gilt $f'(u, v) < f(u, v) \leq c(u, v)$ und

$$\begin{aligned} f'(u,v) &= f(u,v) - x && \text{per Definition von } f' \\ &\geq f(u,v) - c_f(v,u) && \text{per Definition von } x \\ &= f(u,v) - f(u,v) && \text{per Definition von } c_f \\ &= 0 \end{aligned}$$

Also sind die Kapazitätsbedingungen erfüllt. Desweiteren gilt, dass $|f'| = |f| + x > |f|$, f kann also kein maximaler Fluss sein.

(2) $\Rightarrow$ (3): Angenommen, es gibt keinen augmentierenden Pfad von s nach t in G_f. Sei S die Menge aller Knoten, die von s in G_f erreichbar sind, und setze $T = V \setminus S$. Die Partition (S, T) ist offensichtlich ein (s, t)-Schnitt. Betrachte ein beliebiges Knotenpaar (u, v) mit $u \in S$ und $v \in T$. Falls $(u, v) \in E$, dann ist $c_f(u, v) = c(u, v) - f(u, v) = 0$ und damit $f(u, v) = c(u, v)$, und falls $(v, u) \in E$, dann ist $c_f(u, v) = f(v, u) = 0$ und damit $f(v, u) = 0$. Mit anderen Worten, unser Fluss f saturiert jede Kante von S nach T in G und meidet jede Kante von T nach S in G. Folglich ist $|f| = |(S, T)|$.

(3) $\Rightarrow$ (1): Angenommen, es gelte $|f| = |(S, T)|$ für einen Fluss f und einen Schnitt (S, T) in G. Dann kann es keinen Fluss f' geben mit $|f'| > |f|$, da wir in Kapitel 1.2 gesehen haben, dass $|f'| \leq |(S, T)|$ für jeden Fluss f' gelten muss. $\square$

1.4 Ford-Fulkerson Algorithmus

Das Maxflow-Mincut Theorem kann man direkt in einen Algorithmus umsetzen, wie das schon Ford und Fulkerson in den 50er Jahren getan haben: Angefangen mit einem leeren Fluss f, suche wiederholt einen augmentierenden Pfad p in G_f und füge p zu f hinzu (wie für f' oben), bis es keinen augmentierenden Pfad in G gibt.

Wenn jede Kantenkapazität eine natürliche Zahl ist, dann erhöht jeder Augmentierungsschritt den Wert des Flusses um mindestens 1. Der Algorithmus hält also nach höchstens $|f^*|$ Runden, wobei f^* ein maximaler Fluss ist. Jede Iteration benötigt $|E|$ Schritte, indem man auf G_f eine Tiefen- oder Breitensuche durchführt, um einen augmentierenden Pfad zu finden. Im worst case hat der Ford-Fulkerson Algorithmus also eine Laufzeit von $O(|f^*| \cdot |E|)$.

1.5 Probleme mit irrationalen Kapazitäten

Wenn wir alle Kapazitäten mit der gleichen positiven Konstanten multiplizieren, erhöht sich der maximale Fluss um denselben Faktor. Daraus folgt, dass wenn alle Kantenkapazitäten rational sind, der Ford-Fulkerson Algorithmus in endlicher Zeit hält. Wenn wir allerdings irrationale Kapazitäten haben, kann der Algorithmus unendlich lange laufen dadurch, dass er in jeder Runde zunehmend kleinere augmentierende Pfade findet. Schlimmer noch, diese unendliche Folge von Augmentierungen muss noch nicht einmal gegen den maximalen Fluss konvergieren! Eines der einfachsten Bespiele für diesen Effekt ist von Uri Zwick gefunden worden.

Betrachte den Graphen in Abb. 6 mit sechs Knoten und neun Kanten. Sechs der Kanten haben eine große ganzzahlige Kapazität X, zwei haben Kapazität 1, und eine hat Kapazität $\phi = (\sqrt{5}-1)/2 \approx 0,618034$, die so gewählt ist, dass $1 - \phi = \phi^2$. Um zu zeigen, dass der Ford-Fulkerson Algorithmus hängen bleibt, betrachten wir die residualen Kapazitäten der drei horizontalen Kanten während der Algorithmus voranschreitet. (Die residualen Kapazitäten der anderen Kanten sind immer mindestens $X - 3$.)

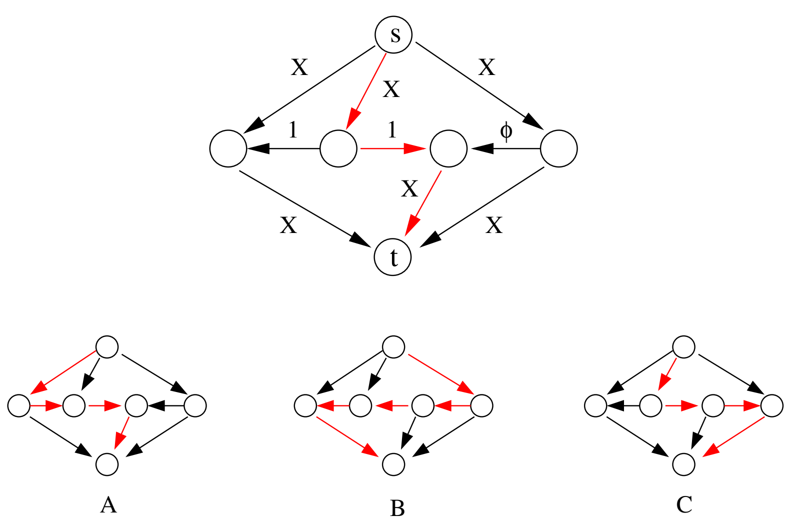

Figure 6: Uri Zwicks Beispiel.

Der Ford-Fulkerson Algorithmus startet mit dem augmentierenden Pfad in der oberen, großen Abbildung des Graphen in Abb. 6. Die drei horizontalen Kanten haben danach residuale Kapazitäten 1, 0 und ϕ. Angenommen, die horizontalen residualen Kapazitäten seien ϕ^{k-1}, 0 und ϕ^k für ein ganzzahliges k.

1. Augmentiere entlang B, was ϕ^k zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, ϕ^k und 0.

2. Augmentiere entlang C, was ϕ^k zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, 0 und ϕ^k.

3. Augmentiere entlang B, was ϕ^{k+1} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun 0, ϕ^{k+1} und ϕ^{k+2}.

4. Augmentiere entlang A, was ϕ^{k+1} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, 0 und ϕ^{k+2}.

D.h., nach $4n+1$ Augmentierungen sind die residualen Kapazitäten ϕ^{2n-2}, 0 und ϕ^{2n-1}. Wenn die Anzahl der Augmentierungen gegen Unendlich strebt, konvergiert der Wert des Flusses gegen

$$1 + 2\sum_{i=1}^{\infty} \phi^i = 1 + \frac{2}{1-\phi} = 4 + \sqrt{5} < 7$$

obwohl der maximale Flusswert $2X+1$ ist.

Aufmerksame Studenten mögen sich an diesem Punkt fragen, ob wir uns über irrationale Kapazitäten wirklich Sorgen machen sollten, da ein Computer ja nur rationale Zahlen darstellen kann. Nichts desto trotz kann ein Computer sehr lange rationale Zahlen darstellen, so dass das Ergebnis oben andeutet, dass der Ford-Fulkerson Algorithmus eventuell *sehr* lange brauchen kann um zu terminieren. Im schlimmsten Fall könnte der Algorithmus durch Rundungsfehler sogar in eine Endlosschleife laufen!

1.6 Edmonds-Karp Algorithmen

Der Ford-Fulkerson Algorithmus legt nicht fest, welcher augmentierende Pfad gewählt werden sollte, wenn es mehr als einen gibt. 1972 haben Edmonds und Karp zwei natürliche Heuristiken analysiert, um einen augmentierenden Pfad auszuwählen. Die erste ist im wesentlichen ein Greedy Algorithmus:

Heuristik 1: Wähle den augmentierenden Pfad mit dem größten Wert.

Es ist relativ einfach zu zeigen, dass der augmentierende Pfad mit maximalem Wert in $O(|E| \log |V|)$ Zeit mit Hilfe eines minimalen-Spannbaum-Algorithmus oder Dijkstras kürzeste-Wege-Algorithmus berechnet werden kann. Lass einfach einen Spannbaum T von s aus wachsen. Nimm jedesmal die Kante maximaler Kapazität, die T verlässt, und füge sie zu T hinzu, bis T einen Weg von s nach t besitzt (oder keine weitere Kante existiert).

Wir analysieren nun Heuristik 1 in Abhängigkeit zum Wert des maximalen Flusses f^*. Sei f ein beliebiger Fluss in G und sei f' ein maximaler Fluss im aktuellen residualen Graphen G_f. (Am Anfang des Algorithmus gilt $G_f = G$ und damit $|f'| = |f^*|$.) Sei e eine Engpasskante (d.h. die Kante mit kleinster Kapazität) im nächsten augmentierenden Pfad. Sei S die Menge der Knoten, die von s entlang Kanten mit Kapazität größer als $c(e)$ erreichbar sind, und sei $T = V \setminus S$. Nach der Konstruktion ist T nicht leer, und jede Kante von s nach T hat eine Kapazität von höchstens $c(e)$. Folglich ist der Wert des Schnittes (S,T) höchstens $c(e) \cdot |E|$. Auf der anderen Seite gilt $|(S,T)| \geq |f|$, woraus folgt, dass $c(e) \geq |f|/|E|$ ist.

Daraus ergibt sich, dass die Augmentierung von f um einen augmentierenden Pfad mit maximalem Wert in G_f den maximalen Flusswert in G_f um den Faktor mindestens $1 - 1/|E|$ erniedrigt. Mit anderen Worten, der residuale Fluss vermindert sich exponentiell mit der Anzahl der Iterationen. Nach $|E| \cdot \ln |f^*|$ Durchläufen ist der maximale Flusswert in G_f höchstens

$$|f^*| \cdot (1 - 1/|E|)^{|E| \ln |f^*|} < |f^*| \cdot e^{-\ln |f^*|} = 1$$

(e ist hier die Eulersche Konstante.) Falls alle Kapazitäten ganzzahlig sind, dann ist der maximale Flusswert im residualen Netzwerk nach $|E| \cdot \ln |f^*|$ Iterationen gleich 0 und damit f ein maximaler Fluss. Wir folgern daraus, dass für Graphen mit ganzzahligen Kapazitäten Heuristik 1 eine Laufzeit von $O(|E|^2 \log |E| \log |f^*|)$ hat.

Die zweite Edmonds-Karp Heuristik wurde auch von Ford und Fulkerson in ihrem Originalpaper vorgeschlagen und zuerst vom russischen Mathematiker Dinic im Jahre 1970 analysiert. Edmonds und Karp veröffentlichten ihr Resultat unabhängig von Dinic's Resultat im Jahre 1972, und da deren Resultat eine wesentlich größere Verbreitung erfuhr, nennt fast jeder diese Heuristik Edmonds-Karp Algorithmus.

Heuristik 2: Wähle den augmentierenden Pfad mit der kleinsten Anzahl an Kanten.

Der kürzeste augmentierende Pfad kann in $O(|E|)$ Zeit durch Breitensuche gefunden werden. Interessanterweise benötigt diese Heuristik nur eine polynomielle Anzahl an Iterationen, unabhängig von den Kantenkapazitäten.

Der Beweis der oberen Schranke beruht auf zwei Beobachtungen über die Entwicklung des residualen Graphen. Sei G_i das residuale Netzwerk nach i Augmentierungsschritten, d.h. $G_0 = G$. Für einen Knoten v sei $dist_i(v)$ die Distanz zwischen s und v (d.h. die Länge eines kürzesten gerichteten Weges von s nach v) in G_i. Falls es keinen Weg von s nach v gibt, definieren wir $dist_i(v) = \infty$. Das folgende Lemma zeigt, dass jeder Knoten mit Distanz ∞ nie mehr wieder von s aus erreichbar sein wird.

Lemma 1.2 *Für jeden Knoten v mit $dist_i(v) = \infty$ gilt auch $dis_{i+1}(v) = \infty$.*

Beweis. Betrachte einen beliebigen Knoten $u \in V$ mit $dist_i(u) = \infty$. Sei U die Menge aller Knoten, die einen Weg zu u haben. Dann gilt auch für alle Knoten $u' \in U$, dass $dist_i(u') = \infty$. Angenommen, $dist_{i+1}(u) \neq \infty$. Dann muss in Runde i ein augmentierender Pfad ausgewählt worden sein, der durch einen Knoten in U führt. In diesem Fall muss es aber einen Weg von s zu einem Knoten in U in G_i gegeben haben, was zum Widerspruch führt. □

Als nächstes zeigen wir, dass die Distanzen über die Zeit nur wachsen können.

Lemma 1.3 *Für jeden Knoten $v \in V$ gilt $dist_{i+1}(v) \geq dist_i(v)$.*

Beweis. Das Lemma ist trivial für $v = s$, da $dist_i(s) = 0$ für alle i. Für die anderen Knoten beweisen wir das Lemma per Induktion über die Distanz eines Knotens zu s. Wähle einen beliebigen Knoten $v \neq s$, und sei $p = (s, \ldots, u, v)$ ein kürzester Weg von s nach v in G_{i+1}. (Falls es keinen solchen Weg gibt, dann sind wir gemäß Lemma 1.2 fertig.) Da dieser ein kürzester Weg ist, haben wir $dist_{i+1}(v) = dist_{i+1}(u) + 1$, und die Induktionsbehauptung impliziert, dass $dist_{i+1}(u) \geq dist_i(u)$ ist.

Wir betrachten nun zwei Fälle. Falls (u, v) auch eine Kante in G_i ist, dann ist $dist_i(v) \leq dist_i(u) + 1$. Auf der anderen Seite, falls (u, v) keine Kante in G_i ist, dann muss (v, u) eine Kante im $i$ten augmentierenden Pfad sein. Also muss (v, u) auf einem kürzesten Weg von s nach t in G_i liegen, woraus folgt, dass $dist_i(v) = dist_i(u) - 1 \leq dist_i(u) + 1$ ist. In beiden Fällen gilt also

$$dist_{i+1}(v) = dist_{i+1}(u) + 1 \geq dist_i(u) + 1 \geq dist_i(v)$$

□

Jedesmal, wenn wir den Fluss augmentieren, dann verschwindet die Engpasskante im augmentierenden Pfad vom residualen Netzwerk und eine andere Kante in der Umkehrung des augmentierenden Pfades mag (wieder) erscheinen. Unsere zweite Beobachtung ist, dass eine Kante nicht zu oft erscheinen und wieder verschwinden kann.

Lemma 1.4 *Während der Abarbeitung der Heuristik 2 kann jede Kante (u, v) höchstens $|V|/2$ mal vom residualen Graphen verschwinden.*

Beweis. Angenommen, (u, v) ist in zwei residualen Graphen G_i und G_{j+1}, aber nicht in jedem der Graphen $G_{i+1}, \ldots, G_j$ dazwischen. Dann muss (u, v) im $i$ten augmentierenden Pfad liegen, also $dist_i(v) = dist_i(u) + 1$ sein, und (v, u) muss im $j$ten augmentierenden Pfad liegen, also $dist_j(v) = dist_j(u) - 1$ sein. Aus dem vorherigen Lemma folgt

$$dist_j(u) = dist_j(v) + 1 \geq dist_i(v) + 1 = dist_i(u) + 2$$

Mit anderen Worten, die Distanz zwischen s und u erhöht sich um mindestens 2 zwischen dem Erscheinen und Verschwinden von (u, v). Da jeder Dist entweder kleiner als $|V|$ oder unendlich ist, kann die Anzahl der Durchläufe, in denen (u, v) verschwindet, höchstens $|V|/2$ sein. □

Jetzt können wir eine obere Schranke für die Anzahl der Iterationen herleiten. Da jede Kante höchstens $|V|/2$ oft verschwinden kann, gibt es insgesamt höchstens $|E| \cdot |V|/2$ Ereignisse, in denen eine Kante verschwindet. Aber mindestens eine Kante verschwindet in jeder Iteration, so dass der Algorithmus nach höchstens $|E| \cdot |V|/2$ Iterationen halten muss. Daraus ergibt sich eine Laufzeit von $O(|V| \cdot |E|^2)$.

1.7 Goldbergs Algorithmus

Bisher haben wir nur Algorithmen betrachtet, die auf Flüssen aufbauen. Jetzt werden wir einen Algorithmus vorstellen, der sogenannte Präflüsse benutzt. Ein *Präfluss* ist eine Funktion $f : E \rightarrow \mathbb{R}_0^+$, die die folgenden Bedingungen erfüllt:

- *Präflussbedingung:* für alle $v \in V \setminus \{s\}$ gilt $\sum_{u \in V} f(u, v) \geq \sum_{w \in V} f(v, w)$, d.h. der eingehende Fluss ist mindestens so groß wie der ausgehende Fluss
- *Kapazitätseinhaltung:* für alle $v, w \in V$ gilt $f(v, w) \leq c(v, w)$

Der *Überschuss* eines Knotens v ist definiert als $e_f(v) = \sum_{u \in V} f(u, v) - \sum_{w \in V} f(v, w)$. Ein Knoten $v \neq t$ heißt *aktiv*, falls $e_f(v) > 0$.

Der Algorithmus weist jedem Knoten v eine *Höhe* $h(v)$ zu. Die Höhenfunktion is *legal*, falls für alle Kanten (v, w) im residualen Netzwerk G_f gilt $h(v) \leq h(w) + 1$. Eine Kante $(v, w) \in E_f$ heißt *zulässig*, falls $h(v) > h(w)$. Zusammen mit der vorigen Bedingung folgt daraus, dass $h(v) = h(w) + 1$.

Lemma 1.5 *Die zulässigen Kanten formen einen gerichteten azyklischen Graphen (oder DAG).*

Beweis. Angenommen, es gebe einen Kreis $(v_1, v_2, \ldots, v_\ell, v_1)$ über zulässige Kanten. Dann muss gelten, dass $h(v_1) > h(v_2) > \ldots > h(v_\ell) > h(v_1)$ ist, ein Widerspruch. $\square$

Wir stellen jetzt die grundlegenden Operationen des Präfluss-Algorithmus vor.

1. **Push(u, v):**
   $\delta := \min\{e_f(u), c_f(u,v)\}$
   $f(u,v) := f(u,v) + \delta$
   $e_f(u) := e_f(u) - \delta$
   $e_f(v) := e_f(v) + \delta$

2. **Lift(u):**
   $h(u) := \min\{h(v) + 1 \mid (u,v) \in E_f\}$

Der generische Präflussalgorithmus arbeitet wie folgt:

```
for each u ∈ V \ {s} do h(u) := 0; e_f(u) := 0
for each (u,v) ∈ E do f(u,v) := 0; f(v,u) := 0
h(s) := |V|
for each (s,u) ∈ E do
  f(s,u) = c(s,u); e_f(u) := c(s,u)
while (es gibt aktiven Knoten u) do
  if (es gibt zulässige Kante (u,v))
    then Push(u,v)
    else Lift(u)
```

Zunächst beweisen wir die Korrektheit des Algorithmus, und dann berechnen wir die worst case Laufzeit. Wir nehmen im folgenden an, dass $n = |V|$ und $m = |E|$ ist.

Lemma 1.6 *Zu jedem Zeitpunkt des Algorithmus gilt $e_f(s) \leq 0$ und für alle Knoten $v \in V \setminus \{s\}$, dass $e_f(v) \geq 0$.*

Beweis. Wir zeigen das Lemma durch vollständige Induktion über die Anzahl der ausgeführten Push- und Lift-Operationen. Am Anfang ist das Lemma offensichtlich wahr. Wir nehmen also an, es ist wahr. Dann erhält jede Flussveränderung in der Push-Operation aufgrund der Wahl von δ die Eigenschaft, dass $e_f(v) \geq 0$ für alle $v \in V \setminus \{s\}$. Es gilt also

$$\sum_{v \in V \setminus \{s\}} \left(\sum_{u \in V} f(u,v) - \sum_{w \in V} f(v,w) \right) \geq 0$$

Desweitern wissen wir aus Kapitel 1.1, dass für jede Funktion f gilt

$$\sum_{v \in V} \left(\sum_{u \in V} f(u,v) - \sum_{w \in V} f(v,w) \right) = 0$$

Daraus folgt, dass $e_f(s) = \sum_{u \in V} f(u,s) - \sum_{w \in V} f(s,w) \leq 0$ sein muss. $\square$

Lemma 1.7 *Jeder Lift(u) Aufruf erhält die Legalität der Höhenfunktion und erhöht $h(u)$ um mindestens 1.*

Beweis. Eine Lift-Operation wird nur dann für einen Knoten u durchgeführt, wenn es keine zulässige Kante (u, v) gibt und damit $h(u) \leq h(v)$ für alle $(u, v) \in E_f$ ist. Da die neue Höhe $h'(u) = \min\{h(v) + 1 \mid (u, v) \in E_f\}$ ist, folgt, dass $h'(u) > h(u)$ und $h'(u) \leq h(v) + 1$ für alle $(u, v) \in E_f$ ist. Wir erhalten also das Lemma. □

Lemma 1.8 (Superoptimalität) *Für einen legalen Präfluss f und eine legale Höhenfunktion h gibt es keinen augmentierenden Pfad in G_f.*

Beweis. Angenommen, es gäbe einen augmentierenden Pfad $(s = v_1, v_2, \ldots, v_\ell = t)$ in G_f. Da die Höhen der Knoten mit der Zeit nur anwachsen können (siehe Lemma 1.7), gilt $h(s) \geq n$, und da t niemals aktiv sein kann, gilt $h(t) = 0$. Daraus folgt, dass

$$n \leq h(s) \leq h(v_2) + 1 \leq h(v_3) + 2 \leq \ldots \leq h(t) + \ell - 1 = \ell - 1 \leq n - 1$$

da der augmentierende Pfad einfach ist und somit höchstens n Knoten enthalten kann. Wir haben also einen Widerspruch. □

Lemma 1.9 (Optimalität) *Wenn es keinen aktiven Knoten in G_f gibt, dann ist der Präfluss ein maximaler Fluss.*

Beweis. Wenn es keinen Knoten $u \in V \setminus \{s, t\}$ gibt mit $e_f(u) > 0$, dann gilt für alle Knoten $u \in V \setminus \{s, t\}$ nach Lemma 1.6, dass $e_f(u) = 0$. Der Präfluss ist also ein Fluss. Die Maximalität des Flusses folgt aus Lemma 1.8 und dem Maxflow-Mincut-Theorem. □

D.h. wenn der Algorithmus terminiert (es gibt keinen aktiven Knoten mehr), dann liefert er einen maximalen Fluss. Wir müssen noch zeigen, dass der Algorithmus terminiert.

Lemma 1.10 *Für jeden aktiven Knoten u gibt es einen Pfad in G_f von u nach s.*

Beweis. Sei U die Menge der Knoten, die in G_f von u erreichbar sind. Falls $s \notin U$ ist, dann haben alle Knoten in U wegen Lemma 1.6 einen nichtnegativen Überschuss. Der Fluss in U hinein muss 0 sein, denn gäbe es eine Kante $(v, w) \in E$ mit $v \notin U$ und $w \in U$ und $f(v, w) > 0$, dann wäre auch $c_f(w, v) = f(v, w) > 0$. Also gilt

$$\begin{aligned}
0 &= \sum_{v \in V \setminus U} \sum_{w \in U} f(v, w) \\
&\geq \sum_{v \in V \setminus U} \sum_{w \in U} f(v, w) - \sum_{v \in U} \sum_{w \in V \setminus U} f(v, w) \\
&= \sum_{v \in V \setminus U} \sum_{w \in U} f(v, w) - \sum_{v \in U} \sum_{w \in V \setminus U} f(v, w) + \sum_{v \in U} \sum_{w \in U} f(v, w) - \sum_{v \in U} \sum_{w \in U} f(v, w) \\
&= \sum_{v \in U} \left(\sum_{w \in V} f(w, v) - \sum_{w \in V} f(v, w) \right) \\
&= \sum_{v \in U \setminus \{u\}} \left(\sum_{w \in V} f(w, v) - \sum_{w \in V} f(v, w) \right) + e_f(u) \\
&> 0
\end{aligned}$$

Wir haben also einen Widerspruch, und somit muss s in U sein. □

Lemma 1.11 *Für jeden aktiven Knoten $v \in V$ gilt $h(v) \leq 2n - 1$.*

Beweis. Da s aufgrund Lemma 1.6 und t per Definition nie aktiv sein können, gilt zu jeder Zeit $h(s) = n$ und $h(t) = 0$. Betrachte also einen beliebigen aktiven Knoten $u \in V \setminus \{s, t\}$. Nach Lemma 1.10 ist s von u erreichbar in G_f. Sei $p = (u = v_1, v_2, \ldots, v_\ell = s)$ ein einfacher Pfad von u nach s. Wir wissen, dass $h(s) = n$. Desweiteren gilt $h(v_i) \leq h(v_{i+1}) + 1$ für alle $i \in \{1, \ldots, \ell - 1\}$. Daraus folgt, dass $h(u) \leq n + \ell \leq 2n - 1$, da der Weg t nicht enthalten kann und somit $\ell \leq n - 1$ ist. □

Lemma 1.12 (Komplexität von Lift) *Die Gesamtzahl aller Lift-Operationen ist $O(n^2)$ und deren Zeitkomplexität ist $O(n \cdot m)$.*

Beweis. Aufgrund von Lemma 1.7 und Lemma 1.11 können höchstens $2n - 1$ Lift-Operationen auf jeden Knoten angewandt werden. Über alle Knoten gesehen wird die List-Operation daher höchstens $O(n^2)$-mal angewandt.

Die Kosten einer $Lift(v)$-Operation sind gleich dem ausgehenden Grad von v in G_f, da wir für jedes v alle seine adjazenten Knoten überprüfen müssen. Die Gesamtkosten der List-Operation sind also nach oben hin beschränkt durch

$$\sum_{v \in V} (2n - 1) \cdot \deg(v) = O(n \cdot m)$$

wobei $\deg(v)$ den Grad von Knoten v angibt. □

Eine Push-Operation heisst *sättigend*, falls $\delta = c_f(u, v)$ ist, und sonst nichtsättigend.

Lemma 1.13 (Komplexität von Push) *Die Gesamtzahl aller sättigenden Push-Operationen ist $O(n \cdot m)$.*

Beweis. Nach einem sättigenden Push auf (u, v) können wir nicht nochmal Fluss von u nach v schieben, sofern v nicht vorher eine Push-Operation auf (v, u) durchgeführt hat. Da $h(u) = h(v) + 1$ bei Push(u, v) und $h(v) = h(u) + 1$ bei Push(v, u) gelten muss und die Höhen der Knoten monoton steigen, muss sich also die Höhe von u für einen nochmaligen sättigenden Push entlang (u, v) um mindestens 2 erhöht haben. Über (u, v) kann es somit höchstens $(2n - 1)/2$ sättigende Push-Operationen geben, woraus eine Gesamtzahl von $O(n \cdot m)$ sättigenden Push-Operationen folgt. □

Lemma 1.14 *Die Gesamtzahl aller nichtsättigenden Push-Operationen ist $O(n^2 m)$.*

Beweis. Wir verwenden die Potentialfunktion $\Phi = \sum_{aktive\ v \in V} h(v)$. Am Anfang ist $\Phi = 0$, da alle Höhen aktiver Knoten gleich 0 sind. Es gibt nun drei Typen von Operationen, die Φ verändern.

1. Nichtsättigender Push: Findet ein nichtsättigender Push entlang der Kante (v, w) statt, dann wird Knoten v deaktiviert und somit Φ um $h(v)$ vermindert. Auf der anderen Seite kann Knoten w nun aktiv werden, was Φ um $h(w)$ erhöhen kann. Da aber $h(v) = h(w) + 1$ gilt, wird Φ immer um mindestens 1 erniedrigt.

2. Sättigender Push: Ein sättigender Push entlang (v, w) kann Φ um höchstens $2n - 1$ erhöhen, da im schlimmsten Fall v aktiv bleibt und w aktiv wird und die Höhen der Knoten durch $2n - 1$ begrenzt sind. Da die Gesamtzahl sättigender Push-Operationen duch Lemma 1.13 auf $O(n \cdot m)$ begrenzt ist, können sättigende Push-Operationen Φ insgesamt um höchstens $O(n^2 m)$ erhöhen.

3. Lift-Operation: Jede List-Operation erhöht Φ um 1. Da es nach Lemma 1.12 nur $O(n^2)$ viele Lift-Operationen geben kann, können Lift-Operationen Φ insgesamt um höchstens $O(n^2)$ erhöhen.

Insgesamt kann also Φ duch sättigende Push-Operationen und Lift-Operationen um höchstens $O(n^2 m)$ erhöht werden, und jeder nicht sättigende Push erniedrigt Φ um mindestens 1. Da Φ immer größer gleich 0 sein muss, ist somit die Anzahl der nichtsättigenden Push-Operationen auf $O(n^2 m)$ beschränkt.

$\square$

Da eine Push-Operation nur konstante Zeit benötigt, ergibt sich aus den Lemmas oben eine Gesamtlaufzeit von $O(n^2 \cdot m)$. Durch eine verbesserte Auswahl der Push- und Lift-Operationen kann man diese Laufzeit noch verbessern:

1. FIFO: Ein Knoten, der aktiv wird, wird an den Beginn der Liste aller aktiven Knoten gesetzt. Aktive Knoten werden vom Anfang der Liste genommen. Laufzeit: $O(n^3)$.

2. Highest-label-first: Nimm immer den aktiven Knoten mit der höchsten Höhe. Laufzeit: $O(\sqrt{m} \cdot n^2)$.

1.8 Goldbergs Algorithmus mit Highest-Label-First

Wir zeigen hier lediglich eine Laufzeit von $O(n^3)$. Dazu genügt es zu zeigen, dass die Anzahl nichtsättigender Push-Operationen höchstens $O(n^3)$ ist. Da wir $O(n^2)$ Ausführungen von Lift-Operationen haben, kann man die Anzahl nichtsättigender Push-Operationen dadurch einschränken, dass man zeigt, dass zwischen zwei aufeinanderfolgenden Lift-Operationen höchstens eine nichtsättigende Push-Operation für jeden Knoten erfolgen kann. Betrachten wir also den Zeitpunkt nach einer beliebigen Lift-Operation. Wir wissen aus Lemma 1.5, dass die zulässigen Kanten einen DAG formen. Überschuss kann nur entlang dieser Kanten abfließen. In der Übung ist dann zu zeigen, dass dadurch mit der Highest-Label-First jeder Knoten höchstens einen nichtsättigenden Push durchziehen kann, bevor eine weiter Lift-Operation notwendig ist.

1.9 Flüsse mit minimalen Kosten

Bisher haben wir angenommen, dass die Kanten keine Kosten haben. Hier nehmen wir an, die Kosten der Kanten werden bestimmt durch eine Funktion $w : E \to \mathbb{R}^+$. Dann würden wir gerne einen maximalen Fluss f mit minimalen Kosten finden, wobei die Kosten von f definiert sind als

$$w(f) = \sum_{e \in E} w(e) \cdot f(e)$$

Wie aber findet man so einen Fluss? Zunächst definieren wir die Kosten eines augmentierenden Pfades p als

$$w(p) = \sum_{(u,v) \in p : (u,v) \in E} w(u, v) - \sum_{(u,v) \in p : (v,u) \in E} w(v, u)$$

Ein *augmentierender Kreis* p bezüglich f ist ein augmentierender Pfad, dessen erster und letzer Knoten identisch (und nicht notwendigerweise s oder t) ist, d.h. für $p = (v_1, v_2, \ldots, v_\ell)$ gilt

- p ist ein einfacher Pfad bis auf $v_1 = v_\ell$
- $(v_i, v_{i+1}) \in E$ oder $(v_{i+1}, v_i) \in E$ für alle $i \in \{1, \ldots, \ell - 1\}$
- $(v_i, v_{i+1}) \in E_f$ für alle $i \in \{1, \ldots, \ell - 1\}$

Residuale Kapazität und Kosten übertragen sich sinngemäss auf augmentierende Kreise. Zu beachten ist, dass die Augmentierung entlang eines Kreises keine Flussverletzung oder Flusswertveränderung herbeiführt, sondern nur die Kosten des Flusses verändert.

Lemma 1.15 *Zu jedem augmentierenden Kreis p bezüglich eines Flusses f gibt es einen Fluss f' mit $|f'| = |f|$ und $w(f') = w(f) + w(p) \cdot c_f(p)$.*

Beweis. Wir definieren f' analog zu Satz 1.1. Dann folgt aus den Argumenten in Satz 1.1, dass f' ein legaler Fluss ist. Ferner gilt für das Gewicht von f':

$$\begin{aligned} w(f') &= w(f) + \sum_{(u,v) \in p : (u,v) \in E} c_f(p) \cdot w(u,v) - \sum_{(u,v) \in p : (v,u) \in E} c_f(p) \cdot w(v,u) \\ &= w(f) + w(p) \cdot c_f(p) \end{aligned}$$

□

Für den folgenden Satz nehmen wir ohne Beschränkung der Allgemeinheit an, dass das gegebene Flussnetzwerk G ein einfacher gerichteter Graph ist.

Satz 1.16 *Ein Fluss f hat minimale Kosten unter allen Flüssen mit Flusswert $|f|$ genau dann, wenn es keinen augmentierenden Kreis in G_f gibt mit negativen Kosten.*

Beweis. $\Rightarrow$: Folgt aus Lemma 1.15 mittels Kontraposition.

$\Leftarrow$: (Kontraposition) Sei f ein Fluss der keine minimalen Kosten besitzt. Sei g ein Fluss mit minimalen Kosten und $|g| = |f|$. Betrachte das residuale Netzwerk $G_{g-f} = (V, E_{g-f})$ mit $E_{g-f} = E^+_{g-f} \cup E^-_{g-f}$, wobei

$$E^+_{g-f} = \{(u,v) \mid g(u,v) > f(u,v)\} \quad \text{und} \quad E^-_{g-f} = \{(v,u) \mid g(u,v) < f(u,v)\}$$

Seien die Kantenkapazitäten definiert als $c_{g-f}(e) = |g(e) - f(e)|$. Dann gilt für alle Knoten $v \in V$ aufgrund der Tatsache, dass g und f legale Flüsse sind und $|f| = |g|$:

$$\begin{aligned} 0 &= \sum_{u \in V} (g(u,v) - f(u,v)) - \sum_{w \in V} (g(v,w) - f(u,w)) \\ &= \sum_{u \in V : g(u,v) > f(u,v)} (g(u,v) - f(u,v)) + \sum_{u \in V : g(u,v) < f(u,v)} (g(u,v) - f(u,v)) - \\ &\quad \sum_{w \in V : g(v,w) > f(v,w)} (g(v,w) - f(v,w)) - \sum_{w \in V : g(v,w) < f(v,w)} (g(v,w) - f(v,w)) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{u\in V:g(u,v)>f(u,v)} |g(u,v)-f(u,v)| - \sum_{u\in V:g(u,v)<f(u,v)} |g(u,v)-f(u,v)| - \\
&\quad \sum_{w\in V:g(v,w)>f(v,w)} |g(v,w)-f(v,w)| + \sum_{w\in V:g(v,w)<f(v,w)} |g(v,w)-f(v,w)| \\
&= \sum_{u\in V:g(u,v)>f(u,v)} c_{g-f}(u,v) - \sum_{u\in V:g(u,v)<f(u,v)} c_{g-f}(v,u) - \\
&\quad \sum_{w\in V:g(v,w)>f(v,w)} c_{g-f}(v,w) + \sum_{w\in V:g(v,w)<f(v,w)} c_{g-f}(w,v) \\
&= \sum_{u\in V} c_{g-f}(u,v) - \sum_{w\in V} c(v,w)
\end{aligned}
$$

Für alle Knoten ist also die Summer der eingehenden Kapazitäten gleich der Summer der ausgehenden Kapazitäten. Sei nun $c_{\min} = \min\{c_{g-f}(u,v) \mid (u,v) \in E_{g-f}\}$. Wenn wir bei einem beliebigen Knoten $v \in V$ starten, dann gibt es immer eine Kante (v,w) mit $c_{g-f}(v,w) \geq c_{\min}$, und jeder noch nicht vorher besuchte Knoten muss aufgrund der Kapazitätserhaltung auch wieder verlassen werden können. Da die Anzahl der Knoten durch $|V|$ beschränkt, ist daher ein augmentierender Kreis in G_{g-f} konstruierbar. Entfernt man diesen, ist die Kapazitätserhaltung nach wie vor erfüllt. G_{g-f} kann also in eine Menge augmentierender Kreise zerlegt werden. Wendet man diese Kreise auf f an, so erhält man g. Da $w(g) < w(f)$ ist, muss also mindestens einer dieser Kreise negative Kosten haben, was den Beweis beendet. $\square$

Falls die Kantenkapazitäten und Kantenkosten ganzzahlig und durch eine Konstante C beschränkt sind, können wir mit den Erkenntnissen oben einen Polynomialzeitalgorithmus zur Berechung des maximalen Flusses mit minimalen Kosten durchführen:

- Verwende Ford-Fulkerson zur Berechnun eines maximalen Flusses. Laufzeit: $O(C \cdot n \cdot m)$.
- Suche nach augmentierenden Kreisen mit negativen Kosten in G_f bis keine solche Kreise mehr zu finden sind. Ein negativer augmentierender Kreis kann mittels Bellman-Ford in $O(n \cdot m)$ Zeit gefunden werden. Jeder solche augmentierende Kreis hat eine ganzzahlige Kapazität größer 0 und ganzzahlige Kosten kleiner 0. D.h. die Kosten des maximalen Flusses verringern sich um mindestens 1 für jeden augmentierenden Kreis mit negativen Kosten. Die Gesamtlaufzeit für diesen Teil ist also $O(C \cdot n^2 \cdot m)$.

Fortgeschrittenere Techniken können einen maximalen Fluss mit minimalen Kosten auch in Polynomialzeit für beliebige Kapazitäten und Kosten berechnen. Wir werden später nochmal darauf zurückkommen, wenn wir lineare Programme behandeln.