Flußprobleme und Dualität

Inhalt

1 Grundlagen

Definition. Sei ein einfacher gerichteter Graph $D = (V, E)$ mit *Kantenkapazitäten* $c\colon E \to \mathbb{R}_0^+$ und ausgezeichneten Knoten $s, t \in V$, s *Quelle (source)* und t *Senke (target)* gegeben. Man bezeichnet das Tupel $(D; s, t; c)$ dann als *Netzwerk*. Eine Abbildung $f\colon E \to \mathbb{R}_0^+$ heißt *Fluß*, wenn sie die folgenden beiden Eigenschaften hat:

(i) Für alle $(i, j) \in E$ ist die *Kapazitätsbedingung*

$$0 \le f(i,j) \le c(i,j) \tag{1}$$

erfüllt.

(ii) Für alle $i \in V \setminus \{s, t\}$ ist die *Flußerhaltungsbedingung*

$$\sum_{\{j|(i,j)\in E\}} f(i,j) - \sum_{\{j|(j,i)\in E\}} f(j,i) = 0 \tag{2}$$

erfüllt.

Lemma 1. *Für einen Fluß f in einem Netzwerk $(D; s, t; c)$ gilt*

$$\sum_{(s,i)\in E} f(s,i) - \sum_{(i,s)\in E} f(i,s) = \sum_{(i,t)\in E} f(i,t) - \sum_{(t,i)\in E} f(t,i) \ .$$

Beweis. Es gilt

$$\begin{aligned}\sum_{(i,j)\in E} f(i,j) &= \sum_{(i,s)\in E} f(i,s) + \sum_{(i,t)\in E} f(i,t) + \sum_{j\in V\setminus\{s,t\}} \sum_{(i,j)\in E} f(i,j) \\ &= \sum_{(s,i)\in E} f(s,i) + \sum_{(t,i)\in E} f(t,i) + \sum_{j\in V\setminus\{s,t\}} \sum_{(j,i)\in E} f(j,i) \ .\end{aligned}$$

Wegen der Flußerhaltungsbedingung (2) folgt die Behauptung. □

Definition. Der Ausdruck

$$w(f) := \sum_{(s,i)\in E} f(s,i) - \sum_{(i,s)\in E} f(i,s)$$

heißt *Wert* des Flusses f.

Ein Fluß f, für den $w(f)$ maximal ist, d.h. $w(f') \le w(f)$ für alle Flüsse f' in Netzwerk $(D; s, t; c)$, heißt *Maximalfluß* in $(D; s, t; c)$.

Problem. In einem Netzwerk $(D; s, t; e)$ soll ein Maximalfluß gefunden werden.

Definition. Eine Menge $S \subset V$ induziert eine Partition $(S, V\setminus S)$ der Knotenmenge V, die wir *Schnitt* im Graphen $D = (V, E)$ nennen. In einem Netzwerk $(D; s, t; c)$ heißt $(S, V \setminus S)$ ein s-t-Schnitt, wenn $s \in S$ und $t \in V \setminus S$. Die *Kapazität* eines Schnittes $(S, V \setminus S)$ ist definiert als

$$c(S, V \setminus S) := \sum_{\substack{(i,j)\in E \\ i\in S \\ j\in V\setminus S}} c(i,j) \ .$$

Ein Schnitt $(S, V \setminus S)$ heißt *minimal*, wenn $c(S, V \setminus S)$ minimalen Wert unter allen Schnitten $(S', V \setminus S')$ in D hat, d.h. $c(S', V \setminus S') \ge c(S, V \setminus S)$ für alle $S' \subset V$ mit $\emptyset \ne S' \ne V$.

Schnitt-Lemma 2. *Sei $(S, V \setminus S)$ ein s-t-Schnitt im Netzwerk $(D; s, t; c)$. Für jeden Fluß f gilt, daß*

$$w(f) = \sum_{\substack{(i,j)\in E \\ i\in S \\ j\in V\setminus S}} f(i,j) - \sum_{\substack{(i,j)\in E \\ j\in S \\ i\in V\setminus S}} f(i,j) \ .$$

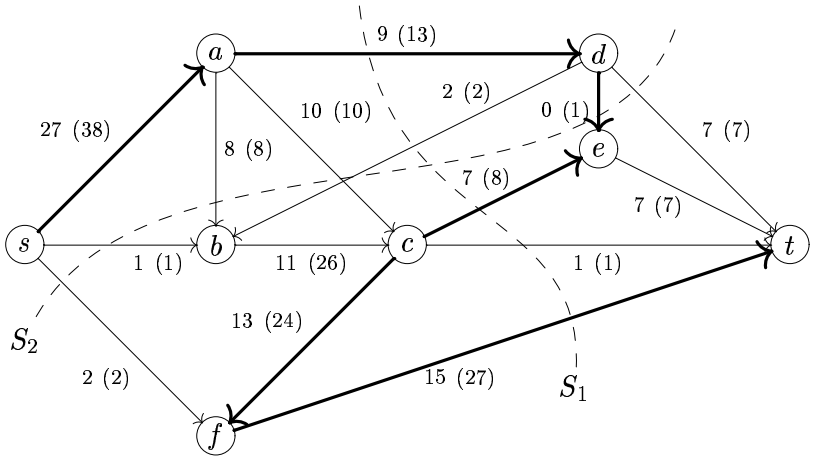
Insbesondere ist $w(f) \le c(S, V \setminus S)$.

Beweis. Es gilt

$$\begin{aligned} w(f) &= \sum_{i \in S} \left(\sum_{(i,j) \in E} f(i,j) - \sum_{(j,i) \in E} f(j,i) \right) \\ &= \underbrace{\sum_{\substack{(i,j) \in E \\ i,j \in S}} f(i,j) - \sum_{\substack{(j,i) \in E \\ i,j \in S}} f(j,i)}_{=0} + \sum_{\substack{(i,j) \in E \\ i \in S \\ j \in V \setminus S}} f(i,j) - \underbrace{\sum_{\substack{(j,i) \in E \\ i \in S \\ j \in V \setminus S}} f(j,i)}_{\geq 0} \\ &\leq \sum_{\substack{(i,j) \in E \\ i \in S \\ j \in V \setminus S}} c(i,j) = c(S, V \setminus S) \end{aligned}$$

□

Beispiel. Der Graph



zeigt ein Flußnetzwerk mit einem eingezeichneten Fluß mit Wert 30. Die Kapazitäten der Kanten sind in Klammern angegeben. Außerdem sind zwei s-t-Schnitte S_1 und S_2 mit den Kapazitäten 49 und 31 eingezeichnet, wobei letzterer minimal ist. Der Schlüssel zum Finden eines Flusses mit größerem Wert scheint ein „erhöhender Weg“ von s nach t zu sein, bestehend aus „Vorwärtskanten“ (i,j), auf denen $f(i,j) < c(i,j)$ und „Rückwärtskanten“ (i,j), auf denen $f(i,j) > 0$. Die hervorgehobenen Kanten zeigen einen solchen erhöhenden Weg.

Definition. Zu einem Fluß f im Netzwerk $(D; s, t; c)$ betrachten wir einen (ungerichteten) Weg von s nach t. Alle Kanten auf diesem Weg, die von s in Richtung t gerichtet sind, heißen *Vorwärtskanten*, alle anderen *Rückwärtskanten*. Ein solcher Weg heißt *erhöhender Weg* (bezüglich f), wenn für jede Vorwärtskante (i,j) des Weges $f(i,j) < c(i,j)$ gilt und wenn für jede Rückwärtskante $f(i,j) > 0$.

Satz vom erhöhenden Weg 3. *Ein Fluß f in einem Netzwerk $(D; s, t; c)$ ist genau dann ein Maximalfluß, wenn es bezüglich f keinen erhöhenden Weg gibt.*

Beweis. $\Longrightarrow$: Sei f ein Maximalfluß. Angenommen, es existiert bezüglich f ein erhöhender Weg. Sei für Kanten (i, j) dieses Weges

$$\Delta(i,j) := \begin{cases} c(i,j) - f(i,j) & \text{falls } (i,j) \text{ Vorwärtskante} \\ f(i,j) & \text{falls } (i,j) \text{ Rückwärtskante} \end{cases}$$

und

$$\Delta := \min\{\Delta(i,j) \mid (i,j) \text{ auf erhöhendem Weg } W\} \ .$$

Dann ist $\Delta > 0$. Sei nun $f' \colon E \to \mathbb{R}_0^+$ definiert als

$$f' := \begin{cases} f(i,j) + \Delta & \text{falls } (i,j) \text{ Vorwärtskante auf } W \\ f(i,j) - \Delta & \text{falls } (i,j) \text{ Rückwärtskante auf } W \\ f(i,j) & \text{sonst } . \end{cases}$$

Dann ist f' wieder ein Fluß, und $w(f') > w(f)$ im Widerspruch zu der Annahme, daß f ein Maximalfluß ist.

$\Longleftarrow$: Das Netzwerk $(D; s, t; c)$ habe keinen bezüglich f erhöhenden Weg. Sei S die Menge aller Knoten in V, zu denen ein erhöhender Weg von s aus bezüglich f existiert. Es gilt $S \neq \emptyset$, weil $s \in S$, und $S \neq V$, weil $t \notin S$. Dann induziert s einen s-t-Schnitt, und es muß gelten, daß $f(i,j) = c(i,j)$ für alle (i,j) mit $i \in S$, $j \in V \setminus S$ und daß $f(i,j) = 0$ für alle (i,j) mit $i \in V \setminus S$, $j \in S$ (d.h. alle Kanten (i,j) mit $i \in S$, $j \in V \setminus S$ sind „saturiert“, und alle Kanten (i,j) mit $i \in V \setminus S$, $j \in S$ sind „leer“). Nach Schnittlemma 2 ergibt sich $w(f) = c(S, V \setminus S)$. Es muß also $w(f)$ maximal sein. □

Max-Flow Min-Cut Theorem 4 (Ford und Fulkerson 1956[1]). *In einem Netzwerk $(D; s, t; c)$ ist der Wert eines Maximalflusses gleich der minimalen Kapazität eines s-t-Schnittes.*

Beweis. Die Behauptung folgt direkt aus dem Satz vom erhöhenden Weg 3. Denn ist f ein Maximalfluß, dann existiert ein Schnitt $(S, V \setminus S)$ mit $s \in S$ und $t \in V \setminus S$ (wobei S die Menge aller auf einem erhöhenden Weg von s erreichbaren Knoten ist). Für $(S, V \setminus S)$ gilt, daß

$$w(f) = c(S, V \setminus S) \quad \text{und} \quad c(S, V \setminus S) = \min_{\substack{\emptyset \neq S' \neq V \\ s \in S \\ t \in V \setminus S}} c(S', V \setminus S) \ . \qquad \square$$

[1] ebenso **Elias, Feinstein und Shanon 1956**

Folgerung. *Für einen Fluß f in einem Netzwerk $(D; s, t; c)$ ist äquivalent:*

(i) Der Wert $w(f)$ ist maximal.

(ii) Es gibt keinen bezüglich f erhöhenden Weg.

(iii) Die Kapazität eines minimalen s-t-Schnittes $(S, V \setminus S)$ ist $w(f)$.

Ganzzahligkeitssatz 5. *Sei $(D; s, t; c)$ ein Netzwerk mit $c\colon E \to \mathbb{N}_0$. Dann gibt es einen Maximalfluß f mit $f(i,j) \in \mathbb{N}_0$ für alle $(i,j) \in E$ und damit $w(f) \in \mathbb{N}_0$.*

Beweis. Definieren einen ganzzahligen „Anfangsfluß“ $f_0\colon E \to \mathbb{N}_0$ (zum Beispiel $f(i,j) = 0$ für alle $(i,j) \in E$). Ist f_0 nicht maximal, so existiert ein erhöhender Weg bezüglich f_0, und für diesen ist $\Delta_0 > 0$ (definiert wie Δ im Satz vom erhöhenden Weg 3) ganzzahlig. Entsprechend kann f_1 mit $w(f_0) = w(f_0) + \Delta_0$ konstruiert werden, und f_1 ist wiederum ganzzahlig. Das Verfahren kann so lange iteriert werden, bis ein ganzzahliger Fluß f_i erreicht ist, bezüglich dessen es keinen erhöhenden Weg mehr gibt. □

2 Bestimmung maximaler Flüsse („Max Flow“)

Entsprechend dem Beweis des Satzes vom erhöhenden Weg 3 können wir folgendes Verfahren zur Bestimmung eines Maximalflusses (und eines minimalen Schnittes) angeben:

1. Setze $f(i,j) := 0$ für alle Kanten $(i,j) \in E$.
2. Solange es einen erhöhenden Weg bezüglich f gibt, führe aus:
3. Sei $\langle e_1, e_2, \ldots, e_k \rangle$ mit $e_1, \ldots, e_k \in E$ erhöhender Weg.
4. Setze $\Delta := \min(\{c(e_i) - f(e_i) \mid e_i \text{ VwK}\} \cup \{f(e_i) \mid e_i \text{ RwK}\})$ (wobei VwK für Vorwärts-, RwK für Rückwärtskante steht).
5. Setze $f(e_i) := f(e_i) + \Delta$, falls e_i eine Vorwärtskante ist, und setze $f(e_i) := f(e_i) - \Delta$, falls e_i eine Rückwärtskante ist.

Einen erhöhenden Weg kann man systematisch mittels einer Breitensuche finden.

2.1 Ford-Fulkerson-Algorithmus

Eingabe. Ein Netzwerk $(D; s, t; c)$ mit dem Graph $D = (V, E)$, wobei $V := \{1, \ldots, n\}$, und Kapazitätsfunktion $c\colon E \to \mathbb{R}_0^+$.

Ausgabe. Ein Maximalfluß f und ein minimaler s-t-Schnitt $(S, V \setminus S)$.

Ford-Fulkerson

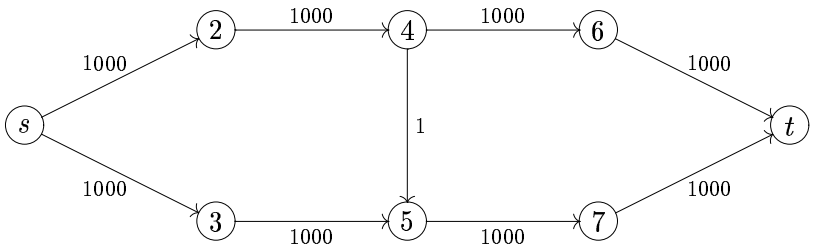
1. Für $(i,j) \in E$ setze $f(i,j) := 0$.
2. Lege Datenstrukturen an:
   - S (Menge der markierten Knoten)
   - *Vor* (Array der Länge $n-1$, in dem für alle Knoten aus $V \setminus \{s\}$ der Vorgänger auf einem erhöhenden Weg von s nach t gespeichert wird)
   - Δ (Array der Länge $n-1$ zur sukzessiven Bestimmung der „Erhöhungswerte“ Δ)
   - u (Hilfsarray der Länge n zur Durchführung der Breitensuche)
3. Setze $S := \{s\}$.
4. Setze für alle $v \in V$ sowohl $u(v) := \mathit{False}$ als auch $\Delta(v) := \infty$.
5. Solange es ein $v \in S$ mit $u(v) = \mathit{False}$ gibt, führe aus:
6. Wähle $v \in S$ mit $u(v) = \mathit{False}$.
7. Für alle $(v,w) \in E$ mit $w \notin S$ führe aus:
8. Falls $f(v,w) < c(v,w)$, dann:
9. Setze $\mathit{Vor}(w) := +v$.
10. Setze $\Delta(w) := \min\{c(v,w) - f(v,w), \Delta(v)\}$.
11. Setze $S := S \cup \{w\}$.
12. Für alle $(w,v) \in E$ mit $w \notin S$ führe aus:
13. Falls $f(w,v) > 0$, dann:
14. Setze $\mathit{Vor}(w) := -v$.
15. Setze $\Delta(w) := \min\{f(w,v), \Delta(v)\}$.
16. Setze $S := S \cup \{w\}$.
17. Setze $u(v) := \mathit{True}$.
18. Falls $t \in S$, dann führe aus:
19. Setze $w := t$.
20. Solange $w \neq s$, führe aus:
21. Falls $\mathit{Vor}(w) > 0$, dann:
22. Setze $f(\mathit{Vor}(w), w) := f(\mathit{Vor}(w), w) + \Delta(t)$.
23. Sonst:
24. Setze $f(w, -\mathit{Vor}(w)) := f(w, -\mathit{Vor}(w)) - \Delta(t)$.
25. Setze $w := \mathit{Vor}(w)$.
26. Setze $S := \{s\}$.
27. Setze für alle $v \in V$ sowohl $u(v) := \mathit{False}$ als auch $\Delta(v) := \infty$.
28. Gib f und $(S, V \setminus S)$ aus.

In den Schritten 6 bis 17 wird ein erhöhender s-t-Weg gesucht. Dazu wird die Menge $S \subset V$ der Knoten bestimmt, die auf erhöhenden Wegen von s aus erreichbar sind.

Laufzeit. Die Laufzeit des Algorithmus hängt davon ab, wie geschickt v ausgewählt wird (in Schritt 6), und davon, wie oft erhöht wird. Die Anzahl der Erhöhungen hängt hier auch ab von $C := \max\{c(i,j) \mid (i,j) \in E\}$.

Bei nichtrationalen Werten $c(i,j)$ kann es passieren, daß das Verfahren nicht terminiert. Bei rationalen Werten geht C im allgemeinen in die Laufzeit ein.

Beispiel.



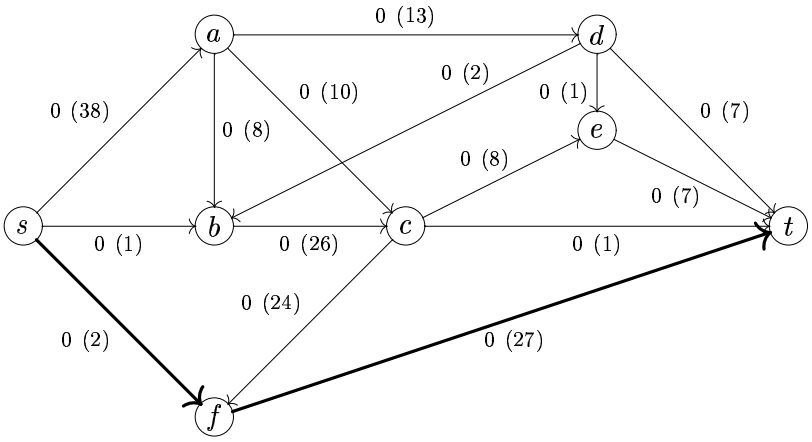
Der Wert eines maximalen Flusses ist 2000. Es kann passieren, daß abwechselnd entlang der Wege $s \to 2 \to 4 \to 5 \to 7 \to t$ und $s \to 3 \to 5 \leftarrow 4 \to 6 \to t$ um jeweils eine Flußeinheit erhöht wird.

2.2 Der Algorithmus von Edmonds und Karp (1972)

Geschickter geht der Algorithmus von Edmonds und Karp vor. Dort wird der Schritt 6 des Algorithmus „wähle $v \in S$ mit $u(v) = \mathit{False}$“ ersetzt durch “wähle unter allen $v \in S$ mit $u(v) = \mathit{False}$ das v aus, welches schon am längsten in S ist.“

Dazu wird S als QUEUE implementiert. Dieser Algorithmus kann dann in $O(|V||E|^2)$ implementiert werden. Der Fluß wird maximal $O(|V||E|)$ oft erhöht, und die Erhöhung kostet jeweils höchstens $O(|E|)$.

Beispiel. Wir führen den Algorithmus von Edmonds und Karp an dem folgenden Beispielgraph vor, der den Anfangsfluß (überall 0) und die Kapazitäten in Klammern zeigt. Hervorgehoben ist außerdem der erste erhöhende Weg, der gefunden werden könnte.



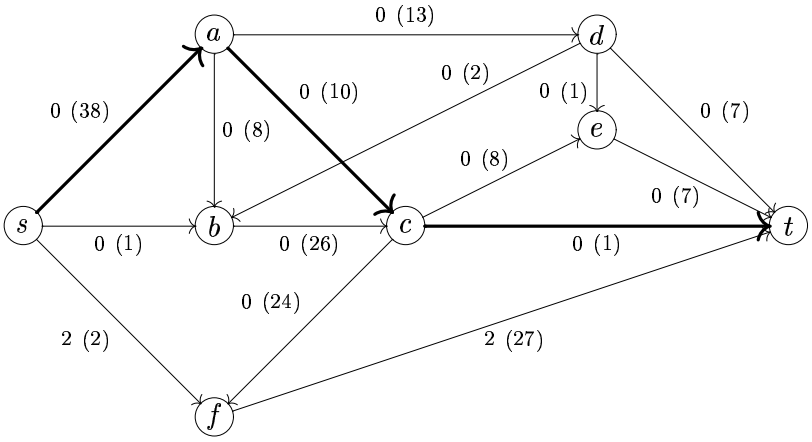
Der eingezeichnete erhöhende Weg wurde wie folgt gefunden: Zunächst ist $S = \{s\}$ und $u(v) = \mathit{False}$ für alle $v \in V$. Von s aus werden die Knoten a, b und f erreicht, das heißt $S := \{s, a, b, f\}$ und

$$\begin{aligned} \mathit{Vor}(a) &:= s & \Delta(a) &:= 38 \\ \mathit{Vor}(b) &:= s & \Delta(b) &:= 1 \\ \mathit{Vor}(f) &:= s & \Delta(f) &:= 2 \ . \end{aligned}$$

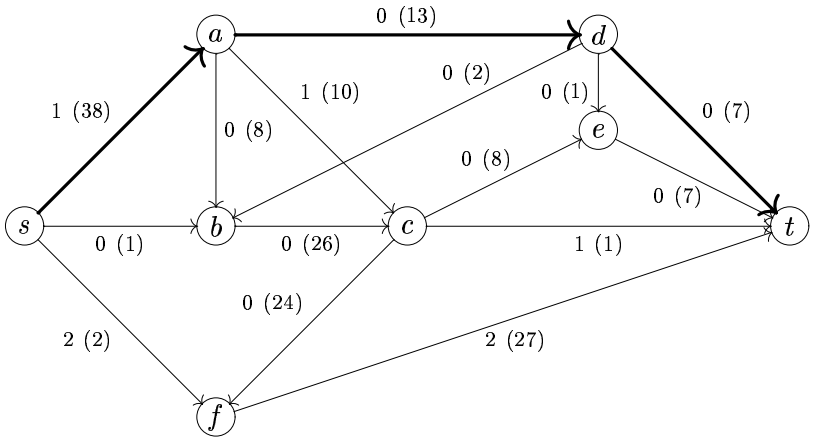
Nun werden von a aus die Knoten c und d gefunden und von f aus der Knoten t, womit $t \in S$ ist und

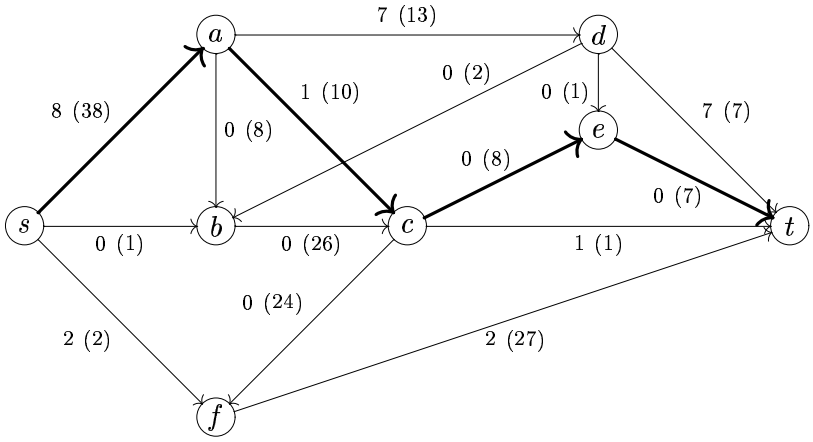
$$\begin{aligned} \mathit{Vor}(c) &:= a & \Delta(c) &:= 10 \\ \mathit{Vor}(d) &:= a & \Delta(a) &:= 13 \\ \mathit{Vor}(t) &:= f & \Delta(t) &:= 2 \ . \end{aligned}$$

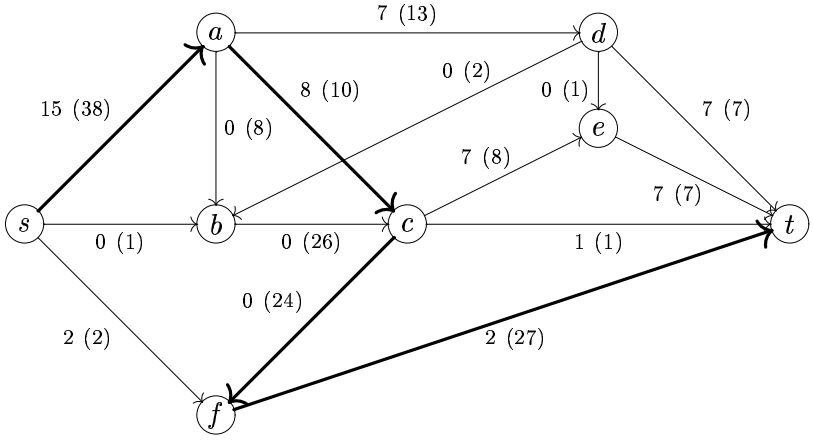
Der nächste Graph zeigt den Fluß nach Auswertung des erhöhenden Weges sowie den neuen gefundenen erhöhenden Weg.

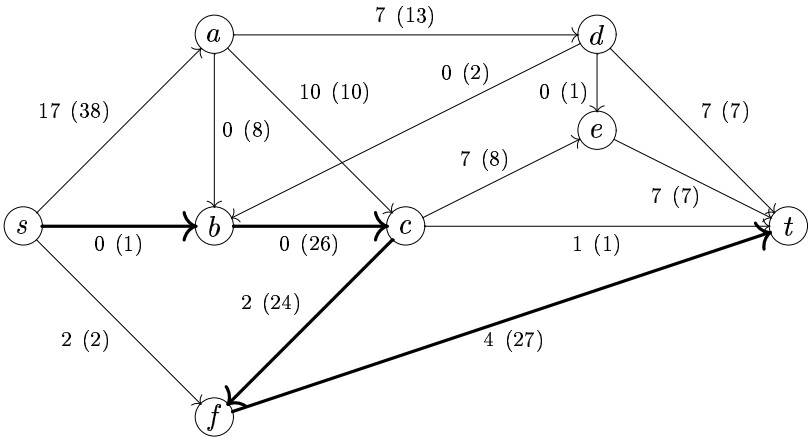


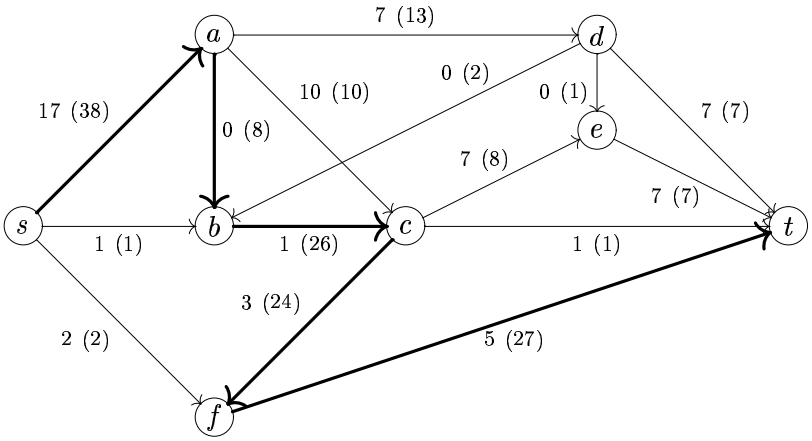
Es folgen die weiteren Zwischenstände des Algorithmus bis zum Ende.

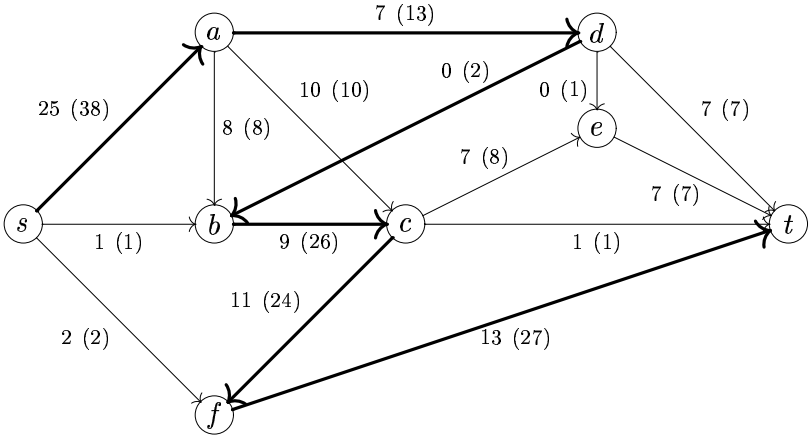

a
d
e
s
b
c
t
f
0 (13)
1 (38)
0 (2)
1 (10)
0 (1)
0 (7)
0 (8)
0 (8)
0 (7)
0 (1)
0 (26)
1 (1)
0 (24)
2 (2)
2 (27)

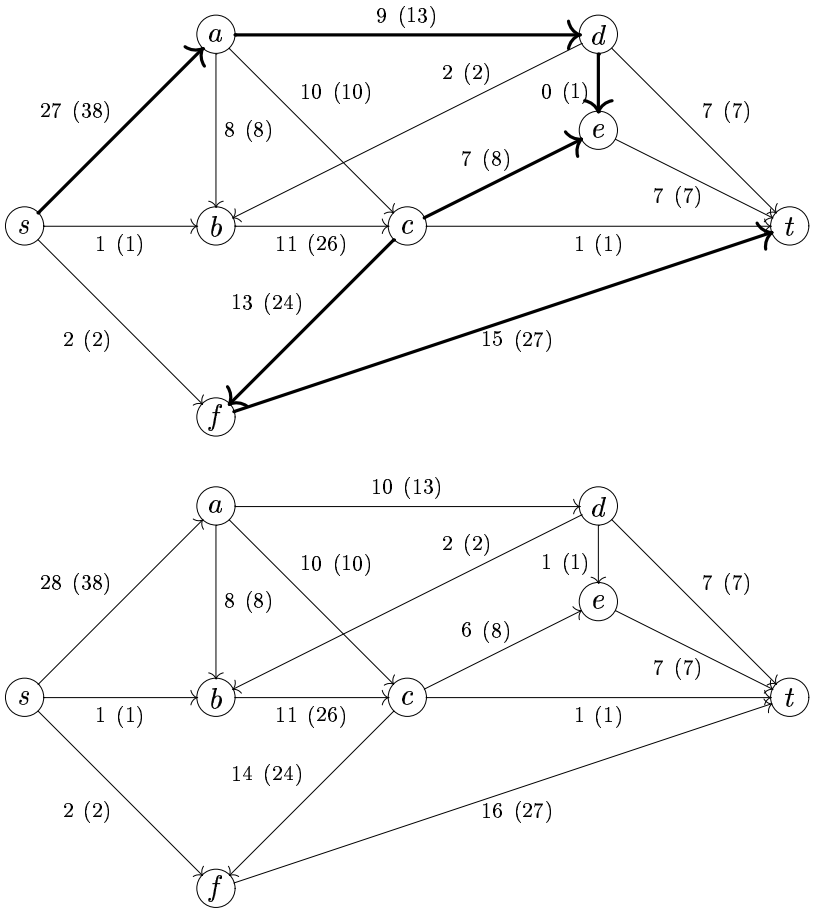

a
d
e
s
b
c
t
f
7 (13)
8 (38)
0 (2)
1 (10)
0 (1)
7 (7)
0 (8)
0 (8)
0 (7)
0 (1)
0 (26)
1 (1)
0 (24)
2 (2)
2 (27)


a
d
e
s
b
c
t
f
7 (13)
15 (38)
0 (2)
8 (10)
0 (1)
7 (7)
0 (8)
7 (8)
7 (7)
0 (1)
0 (26)
1 (1)
0 (24)
2 (2)
2 (27)


a
d
e
s
b
c
t
f
7 (13)
17 (38)
0 (8)
10 (10)
0 (2)
0 (1)
7 (7)
7 (8)
7 (7)
0 (1)
0 (26)
1 (1)
2 (24)
2 (2)
4 (27)


a
d
e
s
b
c
t
f
7 (13)
17 (38)
0 (8)
10 (10)
0 (2)
0 (1)
7 (7)
7 (8)
7 (7)
1 (1)
1 (26)
1 (1)
3 (24)
2 (2)
5 (27)


a
d
e
s
b
c
t
f
7 (13)
25 (38)
8 (8)
10 (10)
0 (2)
0 (1)
7 (7)
7 (8)
7 (7)
1 (1)
9 (26)
1 (1)
11 (24)
2 (2)
13 (27)



2.3 Der Algorithmus von Goldberg und Tarjan (1988)

Der Algorithmus von Goldberg und Tarjan ist der *effizienteste* bekannte Algorithmus zur Konstruktion eines Maximalflusses. Er kann unter Benutzung geeigneter Datenstrukturen so implementiert werden, daß er eine Laufzeit von $O(|V|^2|E|^{\frac{1}{2}})$ hat.

Der Algorithmus beruht nicht auf „erhöhenden Wegen“, sondern auf der Verwendung von „Präflüssen“, bei denen für Knoten die Flußerhaltungsbedingung verletzt sein darf: In einem Präfluß darf in einen Knoten mehr hineinfließen als hinausfließt. Der Algorithmus erhält diese „Präfluß-Eigenschaft“. Erst am Ende des Algorithmus wird der Präfluß zu einem Fluß gemacht, der dann maximal ist.

Grundidee. Aus Knoten mit „Flußüberschuß“ wird dieser Überschuß in Richtung t geschoben (PUSH). Es werden dazu nicht kürzeste Wege nach t, sondern Wege, die momentan „ungefähr“ kürzeste Wege sind, verwendet (RELABEL). Wenn es nicht mehr möglich ist, einen Flußüberschuß in Richtung t zu schieben, so wird er zurück zur Quelle s geschoben.

Zur Vereinfachung der Darstellung erweitern wir das Netzwerk $D = (V, E)$ zu $D' = (V, V \times V)$. Sei $E' := E \cup \{(v, w) \mid (w, v) \in E \wedge (v, w) \notin E\}$. Außerdem wird $c\colon E \to \mathbb{R}_0^+$ fortgesetzt zu $c'\colon V \times V \to \mathbb{R}_0^+$ durch $c'(v, w) = 0$ für $(v, w) \notin E$.

Ein *Fluß* f ist dann eine Abbildung $f\colon V \times V \to \mathbb{R}$ mit *Kapazitätsbedingung*

$$\forall (v, w) \in V \times V \quad f(v, w) \leq c'(v, w) \ , \tag{3}$$

Antisymmetrie-Forderung

$$\forall (v, w) \in V \times V \quad f(v, w) = -f(w, v) \tag{4}$$

und *Flußerhaltungsbedingung*

$$\forall v \in V \setminus \{s, t\} \quad \sum_{u \in V} f(u, v) = 0 \ . \tag{5}$$

Der *Wert* eines Flusses f ist dann

$$w(f) = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t) \ .$$

Die Antisymmetriebedingung (4) bewirkt, daß nicht beide Kanten (v, w) und (w, v) „echten" Fluß tragen. Dadurch wird die Flußerhaltungsbedingung und die Berechnung des Flußwertes vereinfacht.

Definition. Ein *Präfluß* ist eine Abbildung $f\colon V \times V \to \mathbb{R}$, welche die Bedingungen (3) und (4) erfüllt sowie

$$\forall v \in V \setminus \{s\} \quad \sum_{u \in V} f(u, v) \geq 0 \ . \tag{5'}$$

Die Bedingung (5') besagt, daß für alle Knoten $v \in V \setminus \{s\}$ mindestens soviel Fluß hineinfließt wie auch hinausfließt.

Definition. Sei f ein Präfluß. Für $v \in V$ heißt der Wert

$$e(v) := \sum_{u \in V} f(u, v)$$

Flußüberschuß, und die Abbildung $r_f\colon E' \to \mathbb{R}$ mit

$$\forall (u, v) \in E' \quad r_f(u, v) := c'(u, v) - f(u, v)$$

heißt *Restkapazität*.

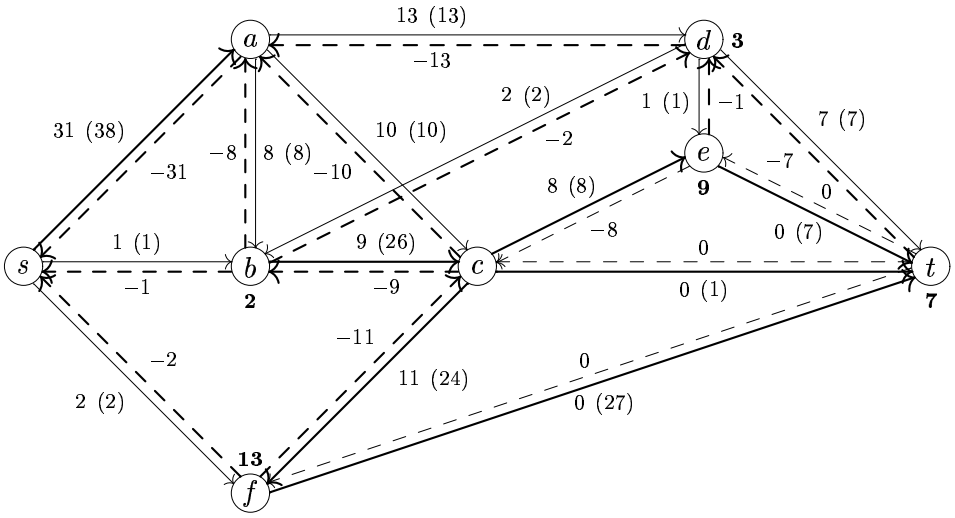


Abbildung 1: Flußnetzwerk mit Präfluß

Bemerkung. Wenn für eine Kante $(u, v) \in E'$ gilt, daß $r_f(u, v) > 0$, so kann der Fluß auf dieser Kante erhöht werden. Falls für $(u, v) \in E'$ gilt, daß $0 \leq f(u, v) < c'(u, v)$, so heißt (u, v) *nicht saturiert*. Ist $0 < f(u, v) \leq c'(u, v)$, so ist (u, v) *nicht leer*, also $f(v, u) = -f(u, v) < 0 \leq c'(v, u)$.

Definition. Eine Kante $(v, w) \in E'$ heißt *Residualkante* bezüglich Präfluß f, falls $r_f(v, w) > 0$. Der *Residualgraph* zu f ist gegeben durch $D_f(V, E_f)$ mit $E_f := \{(v, w) \in E' \mid r_f(v, w) > 0\}$.

Beispiel. Die Abbildung 1 zeigt in unserem altbekannten Flußnetzwerk einen Präfluß. Die Kapazitäten der Kanten sind in Klammern angegeben, soweit sie ungleich 0 sind. Kanten, die nur in E' sind, sind gestrichelt gezeichnet. Ist in einem Knoten ein Überschuß vorhanden, so ist dieser ebenfalls (in fetter Schrift) angegeben. Residualkanten sind hervorgehoben.

Definition. Eine Abbildung $dist\colon V \to \mathbb{N}_0 \cup \{\infty\}$ heißt *zulässige Markierung* bezüglich eines Präflusses f, falls $dist(s) = |V|$, $dist(t) = 0$, und für alle $v \in V \setminus \{s, t\}$, falls $(v, w) \in E_f$, $dist(v) \leq dist(w) + 1$ gilt. Ein Knoten $v \in V$ heißt *aktiv* im Laufe des Algorithmus, wenn $v \in V \setminus \{s, t\}$, $e(v) > 0$ und $dist(v) < \infty$.

Bemerkung. Zu Beginn des Algorithmus von Goldberg und Tarjan wird $dist(s) := |V|$ und $dist(v) := 0$ für $v \in V \setminus \{s\}$ gesetzt. Im Laufe des Algorithmus werden die Werte $dist(v)$ verändert, allerdings immer so, daß $dist$ zulässig ist. Die Markierung $dist$ erfüllt stets:

- Falls $dist(v) < |V|$ für $v \in V$, so ist $dist(v)$ eine untere Schranke für den Abstand von v zu t in D_f.
- Falls $dist(v) > |V|$, so ist t von v in D_f nicht erreichbar, und der Ausdruck $dist(v) - |V|$ ist eine untere Schranke für den Abstand von v zu s in D_f.

Der Algorithmus beginnt mit einem Präfluß f mit $f(s,v) = -f(v,s) = c'(s,v)$ für $(s,v),(v,s) \in E'$ und $f(v,w) = 0$ sonst. Der Präfluß f wird im Laufe des Algorithmus verändert, bis er am Ende ein Fluß ist.

Formale Beschreibung

Eingabe. Netzwerk $(D; s, t; c)$ mit $D = (V, E)$ und $c\colon E \to \mathbb{R}_0^+$.

Goldberg-Tarjan

1. Für $(v,w) \in V \times V$ mit $(v,w) \notin E$ setze $c(v,w) := 0$.
2. Für $(v,w) \in V \times V$ setze $f(v,w) := 0$ und $r_f(v,w) := c(v,w)$.
3. Setze $dist(s) := |V|$.
4. Für $v \in V \setminus \{s\}$ setze:
5. $\quad f(s,v) := c(s,v)$, $r_f(s,v) := 0$.
6. $\quad f(v,s) := -c(s,v)$, $r_f(v,s) := c(v,s) - f(v,s)$.
7. $\quad dist(v) := 0$.
8. $\quad e(v) := c(s,v)$.
9. Solange es einen aktiven Knoten gibt:
10. $\quad$ Wähle einen aktiven Knoten v aus.
11. $\quad$ Führe für v eine zulässige Operation PUSH oder RELABEL aus.
12. Gib f aus.

Die Operation PUSH ist zulässig, falls der betreffende Knoten v aktiv ist und falls es ein $w \in V$ mit $r_f(v,w) > 0$ und $dist(v) = dist(w) + 1$ gibt.

Prozedur PUSH(D, f, v, w)

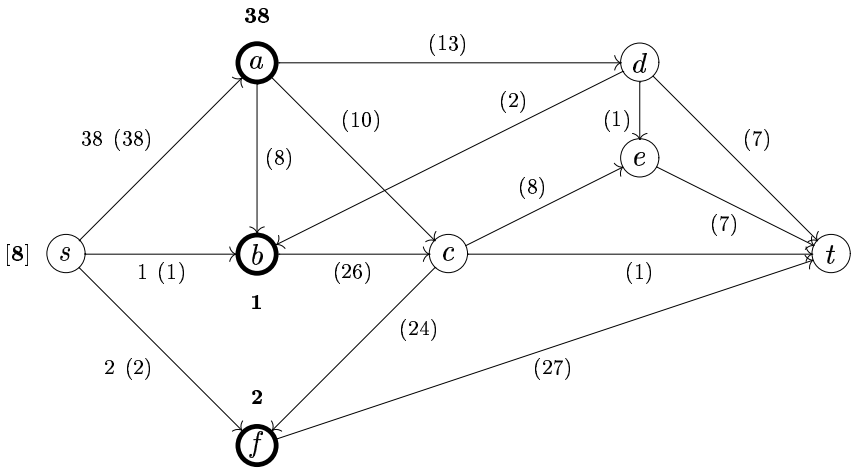
1. $\Delta := \min(e(v), r_f(v,w))$.
2. $f(v,w) := f(v,w) + \Delta$, $f(w,v) := f(w,v) - \Delta$.
3. $r_f(v,w) := r_f(v,w) - \Delta$, $r_f(w,v) := r_f(w,v) + \Delta$.
4. $e(v) := e(v) - \Delta$, $e(w) := e(w) + \Delta$.

Die Operation RELABEL ist zulässig, falls v aktiv ist und falls für alle w mit $r_f(v,w) > 0$ gilt, daß $dist(v) \leq dist(w)$.

Prozedur RELABEL$(D, f, v, dist)$

1. $$dist(v) := \begin{cases} \infty & \text{, falls } \{w \mid r_f(v,w) > 0\} = \emptyset \\ \min\{dist(w) + 1 \mid r_f(v,w) > 0\} & \\ & \text{sonst} \end{cases},$$

Beispiel. Die Gegenkanten bleiben implizit. Die Kapazitäten werden in Klammern dargestellt. Der Präfluß ist ebenfalls notiert, soweit von 0 verschieden. Für jeden Knoten wird in fetter Schrift die Distanz in eckigen Klammern sowie der Überschuß (jeweils soweit von 0 verschieden) dargestellt. Aktive Knoten werden außerdem besonders hervorgehoben. Die erste Darstellung zeigt das Netzwerk zu Beginn des Algorithmus.



Wir führen jetzt die folgenden Operationen durch:

- RELABEL(a).

  Dann ist $dist(a) = 1$.

- PUSH(a, b) mit $\Delta = 8$ und PUSH(a, c) mit $\Delta = 10$ und PUSH(a, d) mit $\Delta = 13$.

  Nach Durchführung dieser drei Operationen ist $e(a) = 7$, $e(b) = 8+1 = 9$, $e(c) = 10$ und $e(d) = 13$. Auch der Präfluß hat sich entsprechend verändert.

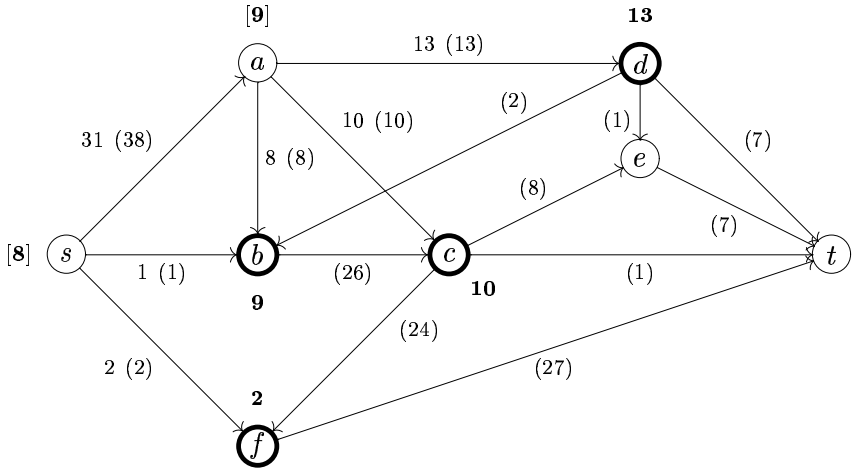
- RELABEL(a).

  Da $r_f(a, s) > 0$ und $dist(a) \leq dist(s)$ ist, folgt $dist(a) := 9$.

- PUSH(a, s) mit $\Delta = 7$.

  Danach ist der verringerte Präfluß $f(s, a) = 31$ und $e(a) = 0$.

Nach Durchführung dieser Operationen haben wir den folgenden Zwischenstand im Netzwerk.



Nun folgen diese Operationen:

- RELABEL(b).

  Dann ist $dist(b) = 1$.

- PUSH(b, c) mit $\Delta = 9$.

  Dann ist $e(b) = 0$ und $e(c) = 19$.

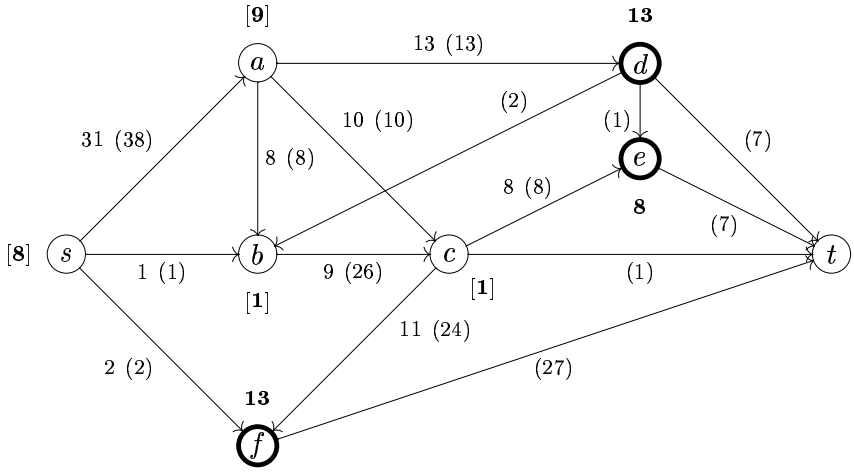
- RELABEL(c).

  Dann ist $dist(c) = 1$.

- PUSH(c, e) mit $\Delta = 8$ sowie PUSH(c, f) mit $\Delta = 11$.

  Dies hat zur Folge, daß $e(c) = 0$ ist. Dafür haben wir $e(e) = 8$ und $e(f) = 11$.

Der neue Zwischenstand sieht dann so aus:



Weiter:

- RELABEL(d).

  Dann ist $dist(d) = 1$.

- PUSH(d, e) mit $\Delta = 1$ sowie PUSH(d, t) mit $\Delta = 7$.

  Nach Durchführung dieser zwei Operationen hat sich $e(d)$ auf 5 verringert, dafür ist jetzt $e(e) = 9$ und $e(t) = 7$.

- RELABEL(d).

  Wegen $r_f(d, b) = 2$ und $dist(b) = 1$ folgt $dist(d) = 2$.

- PUSH(d, b) mit $\Delta = 2$.

  Nun ist $e(d) = 3$ und $e(b) = 2$.

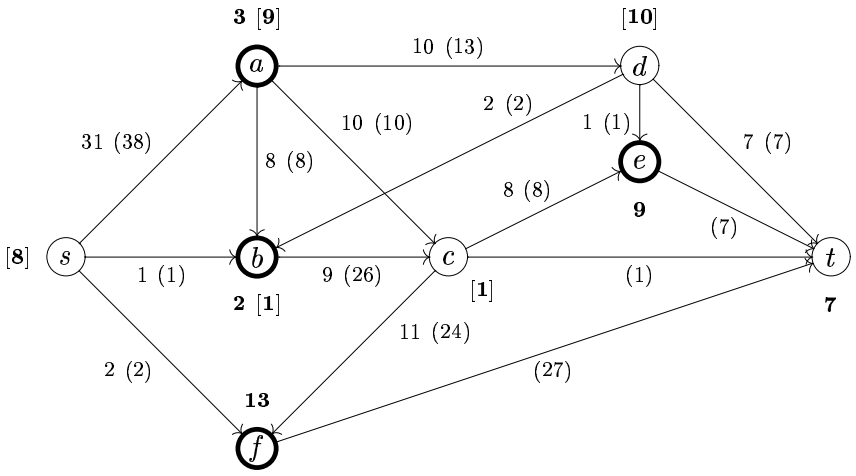
- RELABEL(d).

  Da $r_f(d, a) > 0$, folgt $dist(d) = 10$.

- PUSH(d, a) mit $\Delta = 3$.

  Der Präfluß von a nach d verringert sich auf 10, gleichzeitig ist $e(d) = 0$, dafür jedoch $e(a) = 3$.

Wir sehen uns erneut den Zwischenstand an:



Nun die folgenden Operationen:

- PUSH(a, s) mit $\Delta = 3$.

  Der Präfluß $f(s, a)$ verringert sich auf 28, und damit ist $e(a)$ wieder 0.

- RELABEL(b).

  Wegen $r_f(b, c) = 17$ und $dist(c) = 1$ folgt $dist(b) = 2$.

- PUSH(b, c) mit $\Delta = 2$.

  Hiermit ist auch $e(b)$ wieder 0, dafür ist $e(c) = 2$.

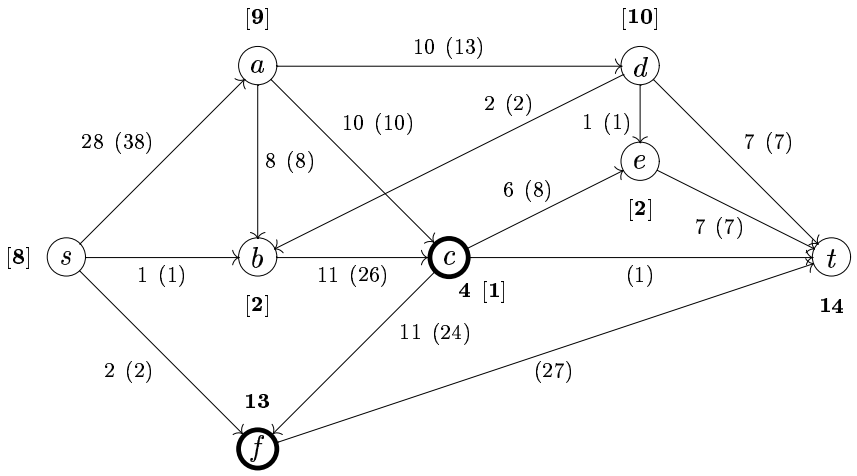
- RELABEL(e), dann PUSH(e, t) mit $\Delta = 7$.

  Zunächst ist damit $dist(e) = 1$. Anschließend verringert sich $e(e)$ auf 2, dafür haben wir dann $e(t) = 14$.

- RELABEL(e), dann PUSH(e, c) mit $\Delta = 2$.

  Dann ist $dist(e) = 2$. Der Präfluß von c nach e verringert sich auf 6, außerdem ist $e(e) = 0$ und $e(c) = 4$.

Das Netzwerk sieht nun so aus:



Zum Abschluß noch:

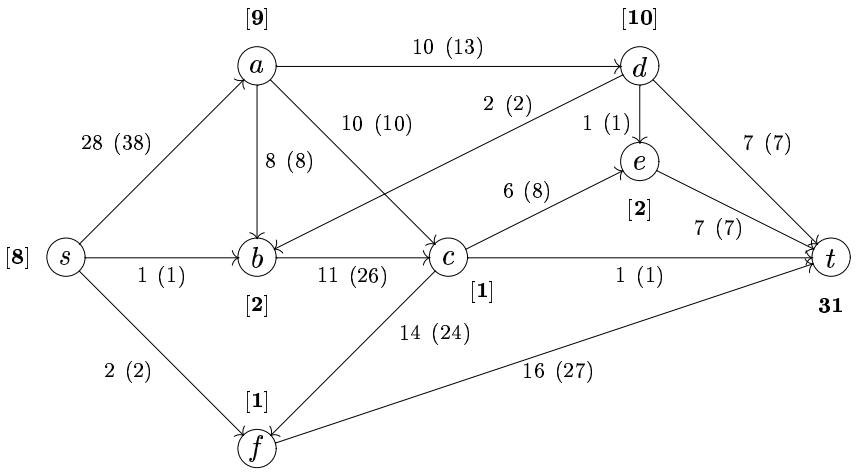
- PUSH(c, t) mit $\Delta = 1$ und PUSH(c, f) mit $\Delta = 3$.

  Danach ist $e(c) = 0$, $e(t) = 15$ und $e(f) = 16$.

- RELABEL(f), dann PUSH(f, t) mit $\Delta = 16$.

  Zunächst ist $dist(f) = 1$. Durch die PUSH-Operation wird $e(f) = 0$ und $e(t) = 31$.

Nun ist kein Knoten mehr aktiv, und ein maximaler Fluß ist gefunden:



Korrektheit

Wir beweisen die Korrektheit des Algorithmus von Goldberg und Tarjan, indem wir zunächst zeigen, daß er korrekt ist, falls er terminiert. Dann zeigen wir, daß die maximale Anzahl zulässiger Operationen endlich ist. Dies ist für alle Wahlen aktiver Knoten v und alle Wahlen zulässiger Operationen für v gültig.

Lemma 6. *Sei f ein Präfluß auf D, die Funktion dist eine bezüglich f zulässige Markierung auf V und $v \in V$ ein aktiver Knoten. Dann ist entweder eine* PUSH*-Operation von v oder eine* RELABEL*-Operation von v zulässig.*

Beweis. Wenn v aktiv ist, so ist $e(v) > 0$ und $dist(v) < \infty$. Da *dist* zulässig ist, gilt für alle w mit $r_f(v,w) > 0$, daß $dist(v) \leq dist(w) + 1$. Falls nun PUSH(v,w) für kein solches w zulässig ist, muß $dist(v) \leq dist(w)$ sein für alle w mit $r_f(v,w) > 0$. Dann ist aber RELABEL zulässig für v. □

Lemma 7. *Während des Ablaufs des Algorithmus von Goldberg und Tarjan ist f stets ein Präfluß und dist stets eine bezüglich f zulässige Markierung.*

Beweis. Wir führen eine Induktion über die Anzahl k der durchgeführten zulässigen Operationen durch.

Für $k = 0$ ist die Behauptung auf Grund der Initialisierung erfüllt. Wir nehmen also an, die Behauptung gelte nach der k-ten Operation, und betrachten die $(k+1)$-te Operation.

Fall 1: Die $(k+1)$-te Operation ist eine Operation PUSH(v,w).

Die Präfluß-Eigenschaft von f bleibt erhalten. Die Markierung *dist* ändert sich nicht, jedoch der Präfluß f. Die Operation PUSH(v,w) erhöht $f(v,w)$ um $\Delta > 0$. Falls dadurch $r_f(v,w) = 0$ wird, bleibt die Markierung *dist* trivialerweise zulässig. Wird $r_f(w,v) > 0$, so bleibt *dist*

ebenfalls zulässig, da für die Zulässigkeit von PUSH(v, w) die Bedingung $dist(w) = dist(v) - 1 \leq dist(v) + 1$ gelten muß.

Fall 2: Die $(k+1)$-te Operation ist eine Operation RELABEL(v).

Dann gilt vor der Operation, daß $dist(v) \leq dist(w)$ für alle w mit $r_f(v, w) > 0$. Die Operation RELABEL setzt $dist(v)$ auf das Minimum aller $dist(w) + 1$ mit $r_f(v, w) > 0$. Danach ist also *dist* wieder zulässig. Der Präfluß f wird nicht geändert. □

Lemma 8. *Sei f ein Präfluß und dist bezüglich f zulässig. Dann ist t im Residualgraph D_f von s aus nicht erreichbar (es gibt also keinen gerichteten s-t-Weg in D_f).*

Beweis. Angenommen, es existiert ein einfacher gerichteter s-t-Weg in D_f, etwa

$$s = v_0 \to v_1 \to v_2 \to \ldots \to v_l = t \ .$$

Da *dist* zulässig ist, gilt $dist(v_i) \leq dist(v_{i+1}) + 1$ für $0 \leq i \leq l - 1$. Es ist dann also $dist(s) \leq dist(t) + l < |V|$, da $dist(t) = 0$ und $l \leq |V| - 1$. Dies ist ein Widerspruch zu $dist(s) = |V|$ wegen der Zulässigkeit von *dist*. □

Satz 9. *Falls der Algorithmus von Goldberg und Tarjan terminiert und am Ende alle Markierungen endlich sind, dann ist der konstruierte Präfluß ein Maximalfluß im Netzwerk $(D; s, t; c)$.*

Beweis. Wegen Lemma 6 kann der Algorithmus nur dann abbrechen, wenn kein aktiver Knoten existiert. Da nach Voraussetzung alle Markierungen endlich sind, muß $e(v) = 0$ gelten für alle $v \in V \setminus \{s, t\}$. Damit ist f ein Fluß. Nach Lemma 8 gibt es in D_f keinen Weg von s nach t. Dann gibt es aber in D keinen bezüglich f erhöhenden s-t-Weg. □

Es bleibt zu zeigen, daß der Algorithmus terminiert und daß die Markierungen endlich bleiben.

Lemma 10. *Sei f ein Präfluß auf D. Wenn für v gilt, daß $e(v) > 0$, so ist s in D_f von v aus erreichbar.*

Beweis. Sei S_v die Menge aller von v in D_f erreichbaren Knoten. Für alle $u \in V \setminus S_v$ und alle $w \in S_v$ ist $f(u, w) \leq 0$, da

$$0 = r_f(w, u) = c(w, u) - f(w, u) \geq 0 + f(u, w) \ .$$

Wegen der Antisymmetriebedingung (4) gilt

$$\begin{aligned}\sum_{w\in S_v} e(v) &= \sum_{\substack{u\in V\\ w\in S_v}} f(u,w)\\ &= \sum_{\substack{u\in V\setminus S_v\\ w\in S_v}} f(u,w) + \underbrace{\sum_{u,w\in S_v} f(u,w)}_{=0}\\ &\le 0\ .\end{aligned}$$

Da f ein Präfluß ist, ist $e(v)\ge 0$ für alle $w\in V\setminus\{s\}$, also $\sum_{w\in S_v\setminus\{s\}} e(v)\ge 0$. Da aber $e(v)>0$ und $v\in S_v$, ist auch $s\in S_v$. □

Lemma 11. *Während des gesamten Algorithmus gilt*

$$\forall v\in V\quad dist(v)\le 2|V|-1\ .$$

Beweis. Zu Beginn des Algorithmus gilt die Behauptung. Betrachten also einen beliebigen Zeitpunkt, zu dem für einen Knoten v die Markierung $dist(v)$ geändert wird. Dann muß v aktiv sein, also $e(v)>0$ gelten. Wegen Lemma 10 ist s von v aus in D_f erreichbar, es existiert also ein einfacher Weg $v=v_0\to v_1\to v_2\to\ldots\to v_l=s$ mit $dist(v_i)\le dist(v_{i+1})$ für $0\le i\le l-1$. Da $l\le|V|-1$, folgt

$$dist(v)\le dist(s)+l\le 2|V|-1\ .$$ □

Lemma 12. *Während des Algorithmus werden höchstens* $2|V|-1$ *Operationen* RELABEL *pro Knoten ausgeführt. Die Gesamtzahl der* RELABEL*-Operationen ist also höchstens* $2|V|^2$.

Beweis. Jede Operation RELABEL an v erhöht $dist(v)$. Da während des gesamten Algorithmus die Ungleichung $dist(v)\le 2|V|-1$ gilt, folgt die Behauptung. □

Um die Gesamtzahl der PUSH-Operationen abzuschätzen, unterscheiden wir zwei Arten von PUSH-Operationen:

Definition. Eine Operation PUSH(v,w) heißt *saturierend*, wenn hinterher $r_f(v,w)=0$ gilt. Ansonsten heißt PUSH(v,w) *nicht saturierend.*

Lemma 13. *Während des Algorithmus werden höchstens* $2|V||E|$ *saturierende* PUSH *ausgeführt.*

Beweis. Ein PUSH(v,w) wird nur für eine Kante (v,w) aus D_f ausgeführt, und $dist(v)=dist(w)+1$. Falls PUSH(v,w) saturierend ist, so kann nur dann zu einem späteren Zeitpunkt des Algorithmus noch einmal PUSH(v,w) ausgeführt werden, wenn in der Zwischenzeit ein PUSH(w,v) ausgeführt wurde. Dazu muß dann aber $dist(w)=dist(v)+1$ sein, wozu $dist(w)$ zwischen

dem ersten PUSH(v,w) und PUSH(w,v) um mindestens zwei gewachsen sein muß. Ebenso muß vor Ausführung des zweiten PUSH(v,w) auch $dist(v)$ um mindestens zwei gewachsen sein.

Bei Ausführung der ersten Operation PUSH(v,w) oder PUSH(w,v) überhaupt muß außerdem $dist(v) + dist(w) \geq 1$ sein. Am Ende gilt

$$dist(v) \leq 2|V| - 1 \quad \text{und} \quad dist(w) \leq 2|V| - 1 \ ,$$

das heißt, daß bei der Ausführung des letzten PUSH(v,w) oder PUSH(w,v) die Ungleichung $dist(v) + dist(w) \leq 4|V| - 3$ gilt. Für eine Kante (v,w) kann es also höchstens $2|V| - 1$ saturierende PUSH-Operationen geben. Damit ist insgesamt $2|V||E|$ eine obere Schranke für die Gesamtzahl der saturierenden PUSH. □

Lemma 14. *Während des Algorithmus werden höchstens* $4|V|^2|E|$ *nicht saturierende* PUSH *ausgeführt.*

Beweis. Wir betrachten die Veränderung des Wertes

$$D := \sum_{\substack{v \in V \setminus \{s,t\} \\ v \text{ aktiv}}} dist(v)$$

im Laufe des Algorithmus. Zu Beginn ist $D = 0$, und es gilt immer $D \geq 0$.

Jedes nicht saturierende PUSH(v,w) setzt D um mindestens 1 herab, da danach $e(v) = 0$, also v nicht aktiv, und eventuell w danach aktiv, aber $dist(w) = dist(v) - 1$. Jedes saturierende PUSH(v,w) erhöht D um höchstens $2|V| - 1$, da der eventuell aktivierte Knoten w nach Lemma 11 erfüllt, daß $dist(w) \leq 2|V| - 1$. Die saturierenden PUSH können also insgesamt (Lemma 13) D um höchstens $(2|V| - 1)(2|V||E|)$ erhöhen. Nach Lemma 12 kann durch RELABEL der Wert D um höchstens $(2|V| - 1)|V|$ erhöht werden.

Da die Gesamtzahl der Erhöhungen von D gleich der Gesamtzahl der Erniedrigungen von D ist, haben wir also eine obere Schranke für die Gesamtzahl der Erniedrigungen, und damit ist die Gesamtzahl der nicht saturierenden PUSH höchstens

$$(2|V| - 1)(2|V||E| + |V|) \leq 4|V|^2|E| \ . \qquad \square$$

Satz 15. *Der Algorithmus von Goldberg und Tarjan terminiert nach spätestens* $O(|V|^2|E|)$ *Ausführungen zulässiger* PUSH- *oder* RELABEL-*Operationen.*

Spezielle Implementationen des Algorithmus von Goldberg und Tarjan

Der Algorithmus von Goldberg und Tarjan ist sehr flexibel. Je nach Wahl der Operationen PUSH oder RELABEL kann man zu unterschiedlichen Worstcase-Laufzeiten kommen. Die Laufzeit ist im wesentlichen abhängig von der

Anzahl der nichtsaturierenden PUSH. Dies ist wiederum abhängig von der Wahl des aktiven Knotens. Besonders günstig ist es, aktive Knoten so zu wählen, daß sie nicht „unnötig“ wechseln.

FIFO-Implementation. Die aktiven Knoten werden entsprechend der Reihenfolge „first-in-first-out“ gewählt. Dies führt zu $O(|V|^3)$ (Goldberg 1985, Shiloach und Vishkin 1982).

Mit „Dynamischen Bäumen“ kommt man sogar mit $O(|V||E| \log \frac{|V|^2}{|E|})$ (Goldberg und Tarjan 1988).

Highest-Label-Implementation. Für PUSH wird unter den aktiven Knoten derjenige mit höchstem Wert von *dist* gewählt. Dies führt zu $O(|V|^2|E|^{\frac{1}{2}})$ (Cheriyan und Motvekwani 1989).

Excess-Scaling-Implementation. Für PUSH(v, w) wird die Kante (v, w) so gewählt, daß v aktiv, $e(v)$ „geeignet groß“ und $e(w)$ „geeignet klein“ ist. Dies führt zu $O(|E|+|V|^2 \log C)$, wobei $C := \max_{(u,v)} c(u, v)$ ist (Ahuja und Orlin 1989).

2.4 Anwendungsbeispiel: Mehrmaschinen-Scheduling

Gegeben: Eine Menge von *Aufträgen* (*Jobs*) $j \in J$, $|J| < \infty$, und für jede Aufgabe j eine Bearbeitungszeit $p_j \in \mathbb{R}_0^+$, eine früheste Anfangszeit $r_j \in \mathbb{R}_0^+$ und eine Deadline $d_j \geq r_j + p_j$, außerdem M *Maschinen*. Jede Maschine kann zu einem Zeitpunkt nur einen Job bearbeiten, und jeder Job kann zu einem Zeitpunkt nur von einer Maschine bearbeitet werden. Jobs können allerdings unterbrochen werden und später auf derselben oder einer anderen Maschine weiterbearbeitet werden.

Gesucht: Eine Bearbeitungsreihenfolge der Jobs auf den Maschinen, die alle Bedingungen erfüllt, sofern eine solche existiert.

Rückführung des Problems auf ein Flußproblem

Die Zeiten r_j und d_j werden für alle $j \in J$ in nichtabsteigender Reihenfolge betrachtet. Es werden die höchstens $2|J| - 1$ paarweise disjunkten Zeitintervalle zwischen diesen Zeitpunkten betrachtet. Diese werden mit T_{kl} für $[k, l)$ bezeichnet.

Das Flußnetzwerk wird dann folgendermaßen gebildet:

Knoten: Es gibt $|J|$ Knoten für die Jobs, außerdem für jedes T_{kl} einen Knoten, und schließlich eine Quelle s und eine Senke t.

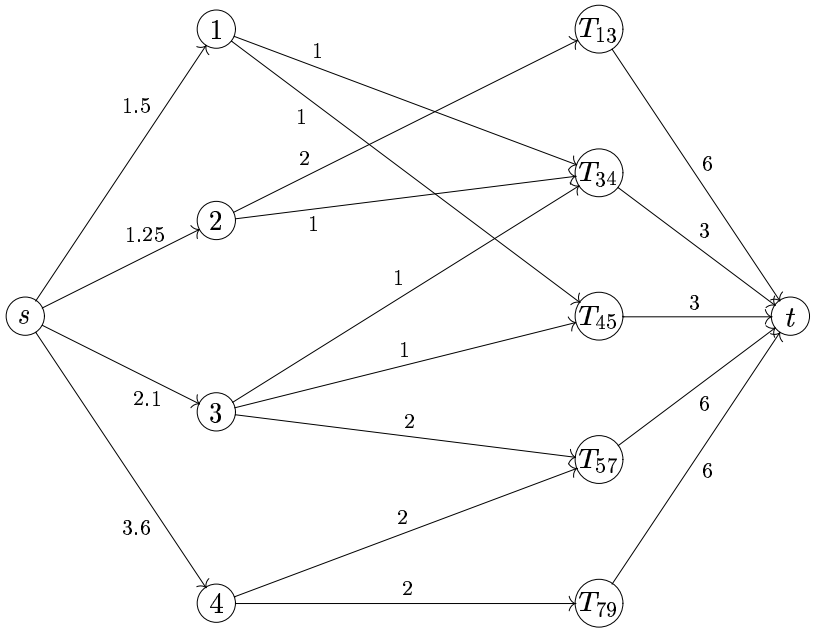


Abbildung 2: Flußnetzwerk für Mehrmaschinen-Scheduling

Kanten: Für alle $j \in J$ werden Kanten (s, j) mit $c(s, j) := p_j$ eingeführt, um die Bearbeitungszeiten eines jeden Knotens zu repräsentieren. Außerdem (T_{kl}, t) mit $c(T_{kl}, t) := (l - k)M$, um die Bearbeitungszeit zu repräsentieren, die im Zeitintervall $[k, l)$ zur Verfügung steht, und dann noch (j, T_{kl}), falls $r_j \leq k$ und $d_j \geq l$ mit $c(j, T_{kl}) := l - k$, um die maximale Zeit zu repräsentieren, die für Job j im Intervall $[k, l)$ zur Verfügung steht.

Es gibt eine Bearbeitungsreihenfolge, die die Bedingungen erfüllt (*zulässiger Schedule*) genau dann, wenn für einen Maximalfluß f in D gilt, daß $w(f) = \sum_{j \in J} p_j$.

Beispiel. Wir wählen $M := 3$, $J := \{1, 2, 3, 4\}$, und

$$\begin{array}{lll} p_1 := 1.5 \ , & r_1 := 3 \ , & d_1 := 5 \ , \\ p_2 := 1.25 \ , & r_2 := 1 \ , & d_2 := 4 \ , \\ p_3 := 2.1 \ , & r_3 := 3 \ , & d_3 := 7 \ , \\ p_4 := 3.6 \ , & r_4 := 5 \ , & d_4 := 9 \ . \end{array}$$

Daraus ergibt sich, daß wir die Zeitpunkte $1, 3, 4, 5, 7, 9$ betrachten und die Intervalle $T_{13}, T_{34}, T_{45}, T_{57}, T_{79}$. Das zugehörige Netzwerk zeigt dann Abbildung 2.