

**Verbesserungen:**

1. Versuche Pfade mit möglichst hoher Restkapazität zu finden. (\rightarrow Capacity Scaling)
2. Suche nach kürzesten erhöhenden Pfaden ($\#$ Kanten) (\rightarrow BFS)

Laufzeiten von Algorithmen:

Man kann Algorithmen in Bezug auf ihre Laufzeit verschiedenen Klassen zuordnen.

- Nicht polynomielle Algorithmen
Die Laufzeit ist nicht polynomiell in n und m .
Z.B. der Labeling-Algorithmus (n mal \mathcal{U} Erhöhungen)
- Polynomielle Algorithmen
Die Laufzeit ist polynomiell in n und m und $\log(\mathcal{U})$
Z.B. Wortlänge des Rechners (z.B. 64 Bit)
- Streng polynomielle Algorithmen
Die Laufzeit ist nur in n und m polynomiell

5.4.3 Algorithmus Capacity-Scaling

Capacity-Scaling ist ein polynomieller Algorithmus für MaxFlow.

Idee:

Versuche erhöhende Pfade mit großer Restkapazität (δ) zu finden.

Beginne mit \mathcal{U} , dann $\mathcal{U}/2, \mathcal{U}/4, \dots, 1$
 $\underbrace{\hspace{10em}}_{\log(\mathcal{U}) \text{ Phasen} \rightarrow \Delta\text{-Phasen}}$

Modifikation des Labeling-Algorithmus:

Führe einen neuen Parameter Δ ein.

\rightarrow Ignoriere alle Kanten (i, j) in $G(x)$ deren Restkapazität $r_{ij} < \Delta$

Dadurch ändert sich ausschließlich das Ausfiltern der Kanten (5.3).

```
1 forall_out_edges(e,v){
2     if(cap[e] - flow[e] < delta)
3         continue;
4     //Rumpf
5 }
6 forall_in_edges(e,v){
7     if(flow[e] < delta)
8         continue;
9     //Rumpf
10 }
```

Listing 5.4: Ausfiltern der Kanten

Der neue Parameter muss dem Algorithmus MF_Labeling noch zusätzlich übergeben werden.

→ MF_Labeling($G, s, t, cap, flow, \Delta$)

Beobachtung:

Für $\Delta = 1$ entspricht der Algorithmus dem normalen Labeling-Algorithmus (ganzzahlig!).

```
1 Capacity_Scaling(const graph& G, node s, node t, const
2     edge_array<int>& cap, int U){
3     edge_array<int>& flow(G,0);
4     int delta = 2^log(U);
5     while(delta < 0){
6         MF_Labeling(G, s, t, cap, flow, delta)
7         delta = delta/2
8     }
9 }
```

Listing 5.5: Capacity_Scaling

Andere Betrachtungsweise:

Δ -Restnetzwerk $G(x, \Delta)$ ist $G(x)$ ohne alle Kanten mit $r_{ij} < \Delta$.

Jede Δ -Phase sucht einen erhöhenden Pfad in $G(x, \Delta)$.



Korrektheit

Die letzte Δ -Phase ($\Delta = 1$) entspricht dem normalen Labeling-Algorithmus.

Laufzeit

Die Anzahl der Phasen (Hauptschleife) ist kleiner gleich $\log(\mathcal{U})$.

Lemma 5

Jede Δ -Phase führt maximal $2 \cdot m$ Erhöhungen aus.

Beweis

Betrachte das Ende einer Δ -Phase.

Sei S die Menge der gelabelten Knoten und $\bar{S} = V \setminus S$.

Dann ist $[S, \bar{S}]$ ein (s,t) -Schnitt ($s \in S, t \in \bar{S}$).

Die Restkapazität $r[S, \bar{S}] \leq m \cdot \Delta$, weil jede Kante zwischen S und \bar{S} höchstens (eigentlich $<$) Restkapazität Δ hat.

Betrachte die (nächste) $\Delta/2$ -Phase.

Diese führt dann maximal $\frac{m \cdot \Delta}{\Delta/2}$ Erhöhungen aus.

$$\leq 2 \cdot m \text{ Erhöhungen.}$$

\Rightarrow Laufzeit einer einzelnen Δ -Phase ist $\mathcal{O}\left(\underbrace{2m}_{\text{Anzahl}} \cdot \underbrace{m}_{\substack{1x \\ \text{Labeling}}}\right)$

□

Lemma 6

Capacity_Scaling löst das *MaxFlow-Problem* in Zeit $\mathcal{O}(m^2 \cdot \log(\mathcal{U}))$.

Beweis

Eine Δ -Phase kostet $\mathcal{O}(m^2)$ und die Anzahl der Phasen beträgt $\log(\mathcal{U})$. □

Bemerkung:

Eine andere Möglichkeit den Labeling-Algorithmus zu verbessern ist, dass man kürzeste ($\#$ Kanten) Pfade verwendet (\rightarrow Ford/Fulkerson) und nur Kanten zwischen benachbarten Levels nutzt.



5.5 Preflow-Push

Idee:

Sende von s aus möglichst viel Fluss ins Netzwerk k und versuche ihn schrittweise bis zum Knoten t zu leiten. Lasse dabei Überschuss wieder zurück nach s fließen. Die Richtung findet man mit Hilfe von Distanz-Labels.

5.5.1 Definition

- i) Ein Preflow x ist eine Funktion $x : E \rightarrow \mathbb{N}_0$ mit
- $0 \leq x_{ij} \leq u_{ij}$ Kapazitätsbedingung
 - $\sum_{\substack{\text{für ein-} \\ \text{gehende} \\ \text{Kanten}}} x_{ji} - \sum_{\substack{\text{für aus-} \\ \text{gehende} \\ \text{Kanten}}} x_{ik} \geq 0 \quad \forall i \in V \setminus \{s, t\}$
(dh. eventuell mehr einfließender als abfließender Fluss)
- ii) $e(i) := \sum x_{ji} - \sum x_{ik}$ heißt Überschuss (oder Excess) von i .
Ein Preflow mit $e(i) = 0 \quad \forall i \in V \setminus \{s, t\}$ ist ein Flow.
- iii) Ein Knoten $i \in V \setminus \{s, t\}$ mit $e(i) > 0$ heißt aktiv.

Idee für Algorithmus:

- Schiebe (push) Fluss von s nach t
(von Knoten zu Knoten \leftrightarrow erhöhender Pfad)
- Während des Ablaufs entsteht dadurch ein Preflow (mit Excess-Knoten)
- Am Ende soll der Preflow ein echter Flow sein.
(dh. es existieren keine aktiven Knoten mehr)

Für die Richtung ($s \rightarrow t$) verwenden wir eine Distanzfunktion.

5.5.2 Distanz-Funktion

Die Distanz ist die Länge (= Anzahl) von Pfaden nach t .



Definition

Für einen Preflow x definieren wir in $G(x)$ eine Distanzfunktion $d = V \rightarrow \mathcal{N}_0$ mit

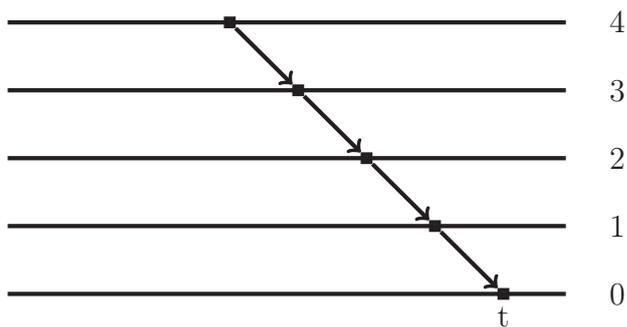
- i) $d(t) = 0$
- ii) $d(i) \leq d(j) + 1 \forall (i, j) \in G(x)$

Beobachtung:

- i) d ist untere Schranke für exakte Distanzwerte.
- ii) $d = 0$ (Nullfunktion) ist gültige Distanzfunktion.

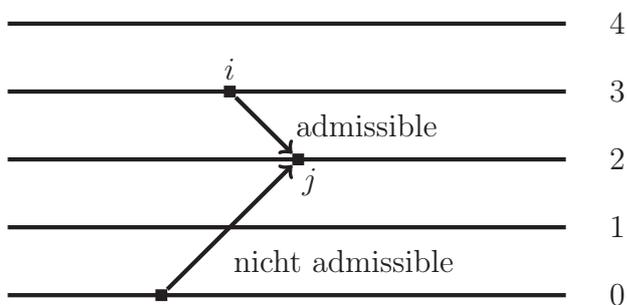
Intention:

$d(i)$ entspricht der Höhe oder dem Level des Knoten i .



Definition admissible

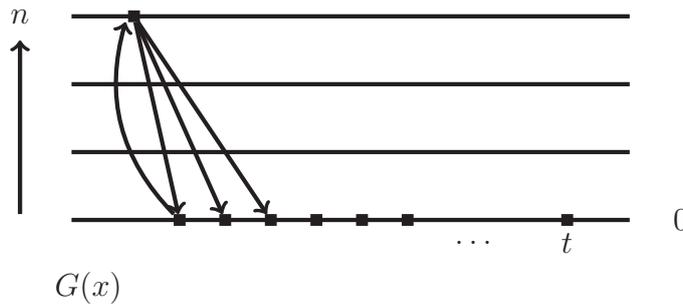
Eine Kante (i, j) in $G(x)$ heißt admissible, wenn $d(i) = d(j) + 1$ dh. der Höhenunterschied genau 1 beträgt.



5.5.3 Preflow-Push-Algorithmus

1. Satturiere alle von s ausgehenden Kanten.

2. Setze $d(i) = \begin{cases} 0, & \text{für } i \neq s \\ n, & \text{für } i = s \end{cases}$



Überschussknoten = alle Nachbarn von s

3. Betrachte einen aktiven Knoten i und schiebe Fluss über eine zulässige Kante
→ PUSH

4. Falls ein aktiver Knoten i keine zulässige, ausgehende Kante hat

→ RELABEL

$$d(i) \leftarrow \min\{d(j) \mid j \text{ Nachbar in } G(x)\} + 1$$

Der generische Preflow-Push-Algorithmus

Der generische Preflow-Push-Algorithmus nutzt keine besondere Strategie (wie z.B. highest-label, FIFO, ...) zur Auswahl aktiver Knoten.



Algorithmus 4 : Generischer Preflow-Push-Algorithmus

```
1 foreach  $v \in V$  do
2   |  $d(v) \rightarrow 0$ 
3   |  $e_v \rightarrow 0$ 
4 end
5 foreach  $(u, v) \in E$  do
6   |  $x_{uv} \rightarrow 0$ 
7 end
8 foreach  $j \in V$  mit  $(s, j) \in E$  do
9   |  $x_{sj} \leftarrow u_{sj}$  // volle Kapazität
10  |  $e_j \leftarrow e_j + x_{sj}$ 
11 end
12  $d(s) \leftarrow n$ 
13 while  $\exists$  aktive Knoten (dh.  $e(i) > 0$ ) do
14   | Wähle einen aktiven Knoten  $i$ 
15   | PUSH/RELABEL( $i$ )
16 end
```

Algorithmus 5 : Push/Relabel(i)

```
1 if  $\exists$  admissible Edge  $(i, j)$  in  $G(x)$  then
2   | Wähle eine solche Kante  $(i, j)$ 
3   |  $\delta \leftarrow \min\{e(i), r_{ij}\}$  Überschuss, Restkapazität
4   |  $x_{ij} \leftarrow x_{ij} + \delta$ 
5   |  $e(i) \leftarrow e(i) - \delta$ 
6   |  $e(j) \leftarrow e(j) + \delta$ 
7 end
8 else
9   |  $d(i) \leftarrow \min\{d(j) | (i, j) \text{ in } G(x)\} + 1$ 
10 end
```
