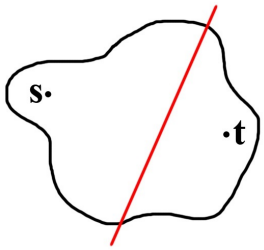
5.2 Schnitte

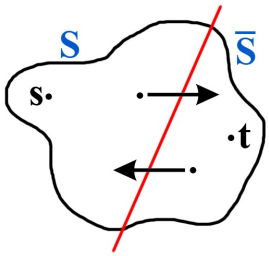


Betrachte einen Schnitt der s und t trennt. Alles was von s nach t fließt, fließt auch über den Schnitt.
$\Rightarrow$ Kapazität des Schnittes ist obere Schranke für den maximalen Fluss.
Das gilt für alle Schnitte und damit auch für den minimalen Schnitt ($MinCut$).

5.2.1 Definition des (s,t)-Schnittes

Ein (s,t)-Schnitt $[S,\bar{S}]$ ist eine Partitionierung der Knoten V in zwei disjunkte Teilmengen S und $\bar{S}$ $(= V \setminus S)$ mit $s \in S$ und $t \in \bar{S}$.

Notation:



$(i,j) \in (S,\bar{S})$ oder $(i,j) \in (\bar{S},S)$.

Der Fluss von s nach t muss über die Kanten $\underbrace{(S,\bar{S})}_{Vorwärtskanten}$ $(= E \cap S \times \bar{S})$

$(\bar{S},S)$ sind die Rückwärtskanten.

Kapazität des (s,t)-Schnittes

Die Kapazität des (s,t)-Schnittes ist die Summe der Kapazitäten der Kanten, deren source-Knoten in S und deren target-Knoten in $\bar{S}$ liegen.

$$u[S,\bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$$

Restkapazität des (s,t)-Schnittes

Die Restkapazität des (s,t)-Schnittes ist die Summe der Restkapazitäten der Kanten, deren source-Knoten in S und deren target-Knoten in $\bar{S}$ liegen.

$$r[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} r_{ij}$$

Fluss über den Schnitt

Der Wert des Flusses über den Schnitt ist die Differenz zwischen der Summe der Flusswerte der Kanten, die von S nach $\bar{S}$ laufen und der, die von $\bar{S}$ nach S laufen. Da der Fluss von $\bar{S}$ nach S zurück fließender Fluss ist, muss er abgezogen werden.

$$F = \underbrace{\sum_{(i,j)\in(S,\bar{S})} x_{ij}}_{maximal} - \underbrace{\sum_{(i,j)\in(\bar{S},S)} x_{ij}}_{\geq 0}$$

$$\Rightarrow F \leq u[S, \bar{S}]$$

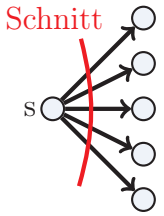
Dies gilt für alle (s,t)-Schnitte. Insbesondere für den mit minimaler Kapazität.

$u[S, \bar{S}] \Rightarrow$ Der Wert eines maximalen Flusses ist nicht größer als die Kapazität eines minimalen Schnittes.

$F_{max} \leq MinCut$ (Schwache Variante des MaxFlow/MinCut-Theorem)

Obere Schranke der Kapazität des (s,t)-Schnittes

Sei $\mathcal{U}$ = maximale Kapazität aller Kanten $(max\{u_{ij}|(i,j) \in E\})$
Kapazität eines Schnittes $\leq (n-1) \cdot \mathcal{U}$
$\Rightarrow F_{max} \leq n \cdot \mathcal{U}$



5.3 Algorithmen auf G(x)

Es gibt zwei Möglichkeiten auf $G(x)$ zu arbeiten. Explizit darauf zu arbeiten, bedeutet, dass intern ein zweiter Graph aufgebaut wird. Dieser Graph besitzt sowohl die Originalkanten, als auch die Gegenkanten, solange es keine Null-Kanten sind.

Bei diesen Graphen kann z.B. DFS oder BFS zur Pfadsuche eingesetzt werden. Asymptotisch schneller ist es, wenn man implizit arbeitet, also auf dem Originalgraphen. Dabei müssen bei einer Pfaderhöhung Kanten eingefügt bzw. gelöscht werden. Im Gegensatz zum expliziten Restnetzwerk müssen hier nicht nur die ausgehenden, sondern auch die eingehenden Kanten betrachtet werden.

cap: Kapazitäten (u_{ij})
$flow$: Aktueller Flusswert (x_{ij})

```
1  forall_out_edges(e,v){
2       r = cap[e] - flow[e];
3       if(r == 0)
4            continue; // Ausfiltern aller Kanten, die nicht in
                G(x) sind
5       //Rumpf
6  }
7  forall_in_edges(e,v){
8       r = flow[e];
9       if(r == 0)
10           continue;
11      //Rumpf
12 }
```

Listing 5.1: Ausfiltern der Kanten

5.4 Erhöhende-Pfad-Algorithmen

Der Labeling-Algorithmus ist ein Erhöhender-Pfad-Algorithmus (augmenting path).

5.4.1 Der allgemeine Labeling-Algorithmus

Algorithmus 3 : Allgemeiner Labeling-Algorithmus

```
1 x = 0 // Nullfunktion (alle x_ij = 0)
2 while G(x) enthält einen Pfad von s nach t do
3 |   Sei P ein solcher (erhöhender) Pfad
4 |   δ = min{r_ij|(i, j) ∈ P}
5 |   Erhöhe den Fluss x entlang von P um δ Einheiten
6 |   Berechne G(x) neu
7 end
```

5.4.2 Der Labeling-Algorithmus

(Konkrete Implementation des allgemeinen Algorithmus)

Idee:
(Ähnlich wie bei DFS, BFS, explore from ...)
Finde die Menge S aller Knoten, die von s aus in $G(x)$ erreichbar sind. Diese werden markiert. Wenn t markiert wurde, dann gilt $t \in S$, dh. es existiert ein erhöhender Pfad $\rightarrow$ Pfaderhöhung

Darstellung der Pfade:
Pred-Array (siehe kürzeste Wege)
$pred[v]$ ist die Kante über die v zum ersten Mal erreicht wurde.

```
void MF_Labeling(const graph& G, node s, node t, const
   edge_array<int>& cap, edge_array<int>& flow){
     list<node> L; \\ Menge S
     node_array<bool> labeled(G, false);
     node_array<edge> PRED(G, NULL);
     while(true){
          labeled[s] = true;
          L.append(s);
          while(!L.empty()){
               node v = L.pop();
               edge e;
               forall_out_edges(e,v){
                    if(flow[e] == cap[e])
                         continue; // e nicht in G(x)
                    node w = G.target(e);
                    if(labeled[w])
                         continue;
                    labeled[w] = true;
                    PRED[w] = e;
                    L.append(w);
               }
               forall_in_edges(e,v){
                    if(flow[e] == 0) continue;
                    node w = G.source(e);
                    if(labeled[w]) continue;
                    labeled[w] = true;
                    PRED[w] = e;
                    L.append(w);
               }
               if(labeled[t])
                    L.clear();
          } // Ende while-Schleife
          if(labeled[t])
               AUGMENT(G,s,t,PRED,cap,flow);
          else
               break; // ex. kein erhöhender Pfad
     }
}
```

Listing 5.2: MF_Labeling

```
1  void AUGMENT(const graph& G, node s, node t, const
       node_array<edge>& PRED, const edge_array<int>& cap,
       edge_array<int>& flow){
2      int delta = MAXINT; // Restkapazität von P
3      node v = t;
4      while(v != s){
5            int r;
6            edge e = PRED[v];
7            if(v == G.source(e)){ //Rückwärtskante
8                  r = flow[e];
9                  v = G.target(e);
10           }
11           else{ // Vorwärtskante
12                 r = cap[e] - flow[e];
13                 v = G.source(e);
14           }
15           if(r < delta)
16                 delta = r; // min
17     }
18     // Eigentliche Flusserhöhung
19     v = t;
20     while(v != s){
21           edge e = PRED[v];
22           if(v == G.source(e)){ //Rückwärtskante
23                 flow[e] = flow[e] - delta;
24                 v = G.target(e);
25           }
26           else{ // Vorwärtskante
27                 flow[e] = flow[e] + delta;
28                 v = G.source(e);
29           }
30     }
31 }
```

Listing 5.3: Augment

Erklärung:
Die äußere *while*-Schleife läuft so lange, bis explizit aus ihr herausgesprungen wird. Dies geschieht, wenn nach der inneren Schleife t nicht gelabelt ist, es also keinen erhöhenden Pfad mehr gibt.
Da s der Startknoten ist und jeder von s aus erreichte Knoten gelabelt und zur Men-

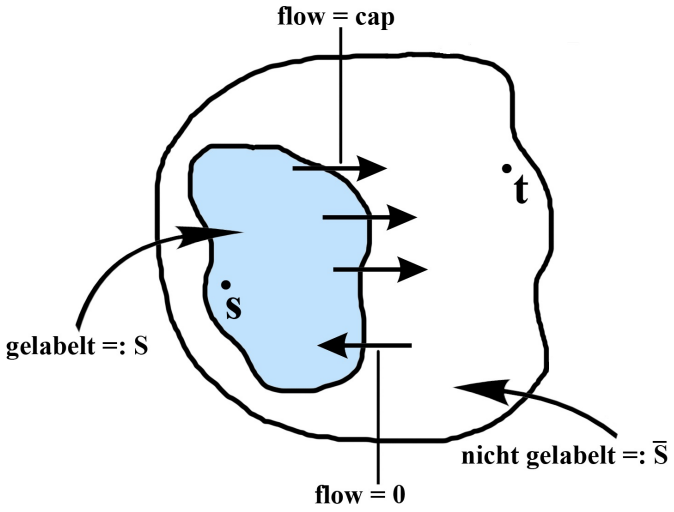
ge L hinzugefügt wird, wird s zu Beginn immer gelabelt und in L eingefügt. In der inneren Schleife wird zuerst ein beliebiger Knoten u aus der Menge L entnommen. Von diesem Knoten u aus werden nun sowohl die ausgehenden, als auch die eingehenden Kanten betrachtet. Dabei werden diejenigen Kanten, die keine Restkapazität mehr besitzen, sofort aussortiert und nicht weiter betrachtet. Bei den restlichen wird überprüft, ob der durch die Kante erreichbare Knoten schon gelabelt wurde. Ist dies nicht der Fall wird er gelabelt, sein $PRED$-Verweis auf die Kante gesetzt, über die er erreicht wurde und der Knoten in die Menge L aufgenommen.
Die innere Schleife bricht ab, sobald L leer ist, was der Fall ist, wenn t gelabelt wurde, da dann ein erhöhender Pfad existiert und entlang diesem, im Algorithmus AUGMENT, Fluss geschickt wird.
Hierbei wird zuerst über den gefundenen Pfad von t nach s gelaufen, und dabei die maximale Flusserhöhung ausfindig gemacht. Ist diese gefunden, also s erreicht, wird erneut über den gefundenen Pfad von t nach s gelaufen und dabei die Flussänderung um den herausgefundenen Wert durchgeführt.
Es wird immer von t nach s gelaufen, da man nur den Vorgänger, nicht aber den Nachfolger eines Knotens kennt.

Korrektheit

In jedem Schritt (Hauptschleife) gibt es zwei Möglichkeiten:

1. Der Algorithmus findet einen erhöhenden Pfad → AUGMENT
2. Der Algorithmus findet keinen erhöhenden Pfad. t wird also nicht gelabelt → STOP

Zu zeigen: Im zweiten Fall ist der berechnete Fluss maximal.



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

Beobachtung:
$\forall\ (i,j) \in [S, \bar{S}]$ gilt: $u_i j = 0$ (deshalb werden diese Kanten vom Algorithmus nicht mehr benutzt, um weitere Knoten zu labeln).
Bei uns bedeutet das:
$\forall$ Kanten $e \in (S, \bar{S}) :\ flow[e] = cap[e]$ und
$\forall\ e \in (\bar{S}, S) :\ flow[e] = 0$

Beobachtung über Flüsse und Schnitte
Flusswert: $F = \sum\limits_{(i,j)\in(S,\bar{S})} x_{ij} - \sum\limits_{(i,j)\in(\bar{S},S)} x_{ij}$
hier gilt:
$$F = \underbrace{\sum_{(i,j)\in(S,\bar{S})} \overset{cap[e]}{u_{ij}}}_{u[S,\bar{S}]} - \sum_{(i,j)\in(\bar{S},S)} \overset{flow[e]}{0}$$
$\Rightarrow F = u[S, \bar{S}]$

Aus schwacher Version des MaxFlow/MinCut-Theorems ($F_{max} \leq \mathcal{U}_{min}$) folgt:
$\underbrace{F\text{ist maximal}}_{\Rightarrow \text{Korrektheit}}$ und $u[S, \bar{S}]$ ist minimal.

Allgemein:
Der Algorithmus liefert zusätzlich zur optimalen Lösung des primalen Problems (MaxFlow) eine optimale Lösung des sogenannten dualen Problems (MinCut). Dies bietet eine Möglichkeit das Ergebnis zu überprüfen (hier: MaxFlow = MinCut).

Theorem 1 (MaxFlow/MinCut-Theorem)
Der Wert eines maximalen Flusses ist gleich der Kapazität eines minimalen (s,t)-Schnittes.

Beweis
Die Korrektheit des Theorems folgt aus der Korrektheit des Labeling-Algorithmus. □