5 Netzwerkflussprobleme

Ein Unternehmer möchte täglich möglichst viele seiner Waren von einem seiner Standorte zu einem anderen Standort transportieren. Er nutzt dazu LKW‘s anderer Unternehmen, die zwischen verschiedenen Städten hin und her fahren. Jeder hat eine maximale Kapazität, die er von einer zu einer andern Stadt transportieren kann. Jeder dieser LKW‘s hat allerdings eine maximale Kapazität an Waren, die er transportieren kann. Da es in den einzelnen Städten keine Lagerhäuser gibt, können die Waren nicht zwischengelagert werden, sondern müssen sofort weiter transportiert werden.
Der Ausgangsstandort in diesem Szenario stellt den Startknoten s und der zweite Standort den Endknoten t dar. Die LKW ‘s fahren entlang der Kanten und ihre Kapazität entspricht der Kapazität der jeweiligen Kante.

Ziele

i) MaxFlow-Problem
Maximiere den Transport von einem Knoten s (source, Quelle) zu einem Knoten t (target, Senke)
Beispiel: Transport von Tonnen Stahl pro Tag über das Eisenbahnnetz.

ii) MinCostFlow
Konkretes Transportproblem (z.B. zwischen Produzenten und Konsumenten)
Ziel: Minimiere die Transportkosten.
Dazu wird zusätzlich du der Kapazität jeder Kante auch Kosten zugeordnet.

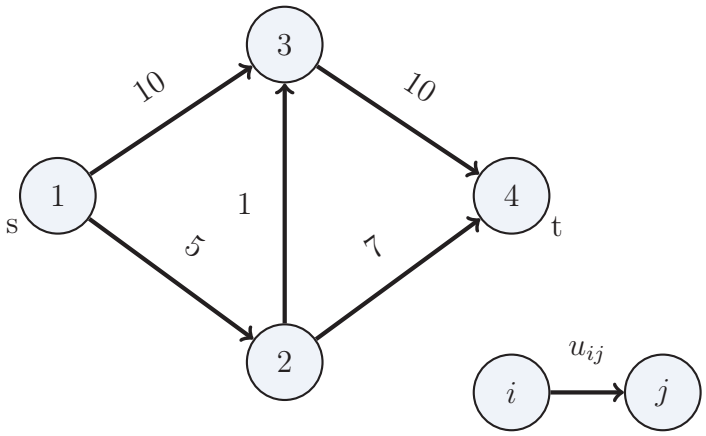
5.1 MaxFlow-Problem

5.1.1 Formale Definition

Gegeben sei

- ein gerichteter Graph $G = (V, E)$
- eine Kapazitätsfunktion $u : E \to \mathbb{R}_0^+$ und
- zwei Knoten $s, t \in V$ mit $s \neq t$.

$u((v,w))$ heißt Kapazität von (v,w). s heißt Quelle (source) und t Senke (target).



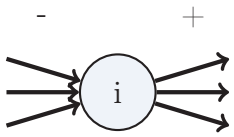
Gesucht ist eine Flussfunktion $x : E \to \mathbb{R}_0^+$ mit folgenden Eigenschaften:
(x_{ij} ist der Fluss über die Kante (i,j)) mit:

i) Kapazitätsbedingung:
$\forall\, e \in E : 0 \leq x(e) \leq u(e)$

ii) Massenbalance-Bedingung:
Sei $v \in V$ (beliebig aber fest!)

$$\underbrace{\sum_{(v,u)\in E}}_{\substack{\text{gehe über alle}\\ \textbf{ausgehenden}\\ \text{Kanten von v}}} x((v,u)) - \underbrace{\sum_{(w,v)\in E}}_{\substack{\text{gehe über alle}\\ \textbf{eingehenden}\\ \text{Kanten von v}}} x((w,v)) = \begin{cases} F, & v = s \\ 0, & \forall\, v \notin \{s,t\} \\ -F, & v = t \end{cases}$$

$F \geq 0$



$$\delta(i) = \sum x((i,j)) - \sum x((k,i))$$

ii) Optimalitätsbedingung:
F soll maximal sein.

Erklärungen:

i) Für alle Kanten muss gelten, dass ihr Flusswert zwischen Null und ihrer maximalen Kapazität liegt.

ii) Für alle Knoten die nicht s oder t sind, muss gelten, dass alles was in sie rein fließt auch wieder abfließt. Es kann also nichts in diesen Knoten gelagert werden. F entspricht dem Fluss der fließt. Aus dem Knoten s fließt der komplette Fluss F heraus, aber nichts herein. Beim Knoten t ist es genau umgekehrt.

iii) Ohne diese Bedingung wäre das Problem trivial, da der Null-Fluss ($x = 0$) die anderen beiden Bedingungen immer erfüllt.

Notation:
Knoten: i, j
Die Kapazität wird anstatt mit $u((i, j))$ jetzt nur noch mit u_{ij} und die Flussfunktion anstatt mit $x((i, j))$ mit x_{ij} bezeichnet.

Beobachtung

$\delta(t) = -\delta(s)$

1. Idee für einen Algorithmus: Erhöhe Pfade

i) Starte mit $x = 0(\forall\ (i, j) \in E\ x_{ij} \leftarrow 0)$

ii) Erhöhe x entlang von Pfaden von s nach t

5.1.2 Das Restnetzwerk

(residual network)
Ein Restnetzwerk beschreibt mögliche Flussrichtungen.

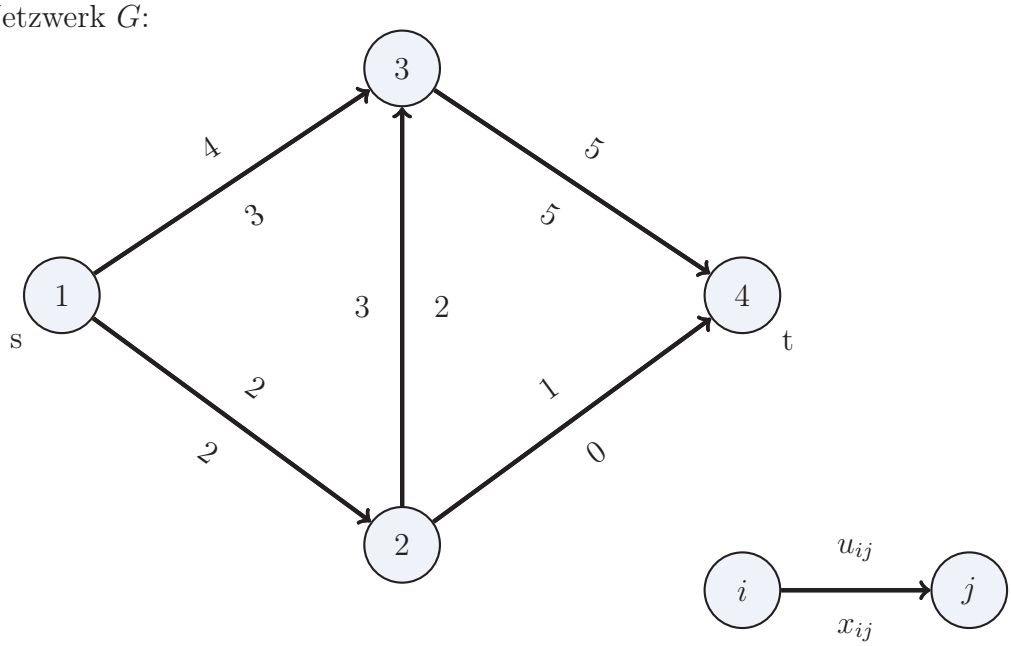
Definition

Sei x eine aktuelle Flussfunktion, dh. sie erfüllt die Kapazitätsbedingung (z.B. der Nullfluss ($\forall\ i, j\ x_{ij} = 0$)).
Das Restnetzwerk $G(x)$ besteht aus allen Kanten von G und deren Gegenkante. Es existieren also für jede Kante (i, j) aus G zwei Kanten in $G(x)$, nämlich die Kante (i, j) und ihre Gegenkante (j, i). Sie besitzen die Restkapazitäten $r_{ij} = u_{ij} - x_{ij}$ und $r_{ji} = x_{ij}$.

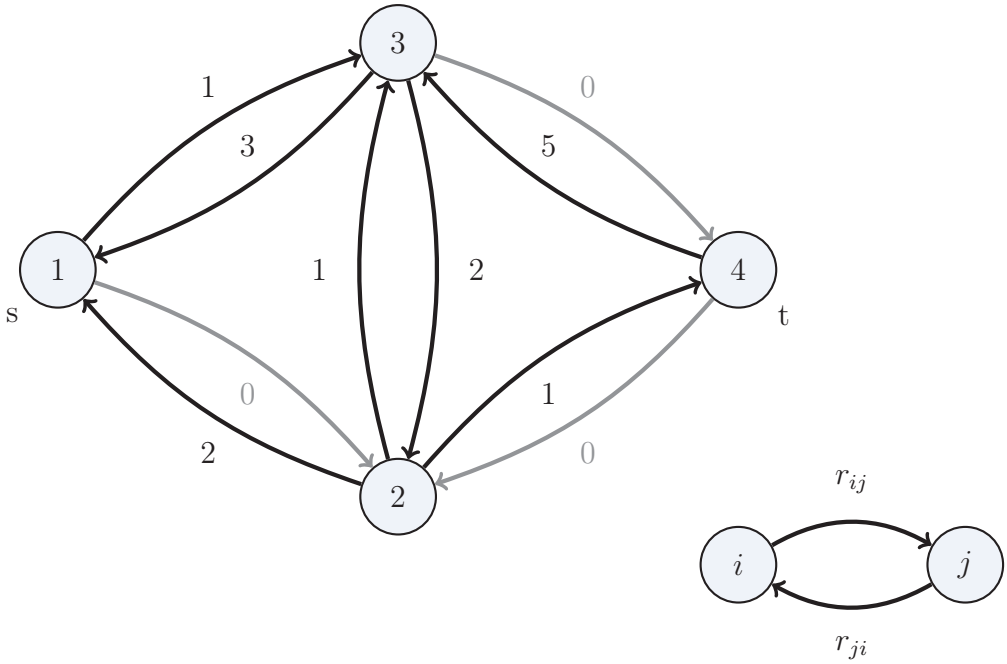
Die Restkapazitäten beschreiben mögliche Änderungen an der Flussfunktion. Eine Erhöhung geschieht in Richtung der Kante (i, j) (max $r_{ij} = u_{ij} - x_{ij}$) und eine Verminderung in die Gegenrichtung (max $r_{ji} = x_{ij}$).

Beispiel 5.1

Netzwerk G:



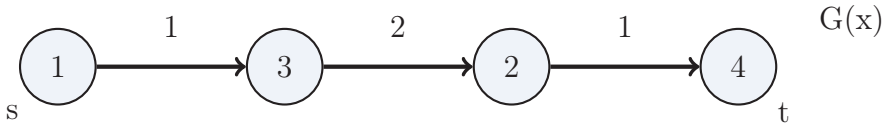
Aktueller Flusswert: $F = 5$
Restnetzwerk $G(x)$:



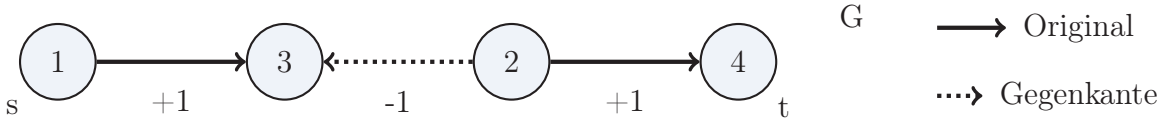
Da über Kanten mit einer Restkapazität von Null, nichts mehr fließen kann, werden sie weggelassen. Dadurch ist die folgende Beobachtung möglich.

Beobachtung: (ohne 0-Kanten)
Wenn alle Null-Kanten im Graphen weggelassen werden, entspricht die Frage ob ein Pfad existiert der Frage ob ein erhöhender Pfad existiert.
Jeder Pfad in $G(x)$ von s nach t beschreibt eine mögliche Erhöhung des Gesamtflusses F. ($\rightarrow$ Pfaderhöhung)

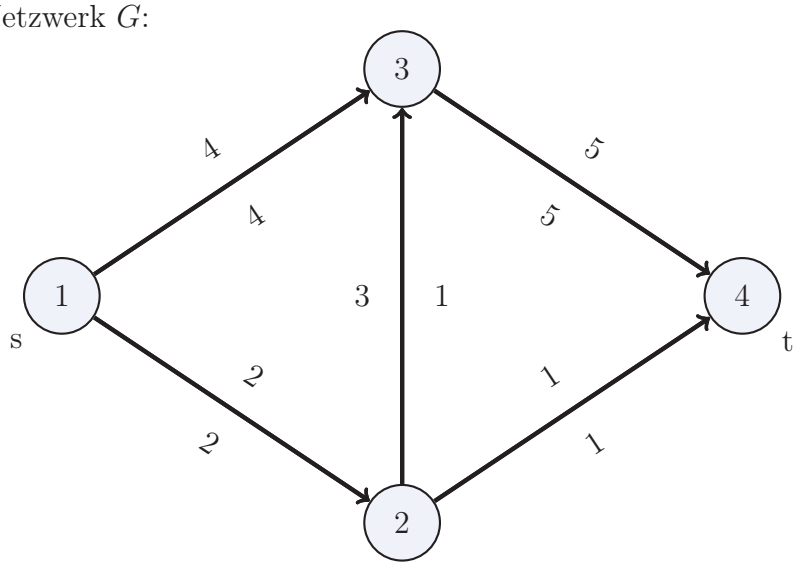
Im Beispiel 5.1 existiert nur ein solcher Pfad:



Eine Erhöhung um $\delta = min\{r_{ij}|(i,j) \in P\} = 1$ ist möglich.

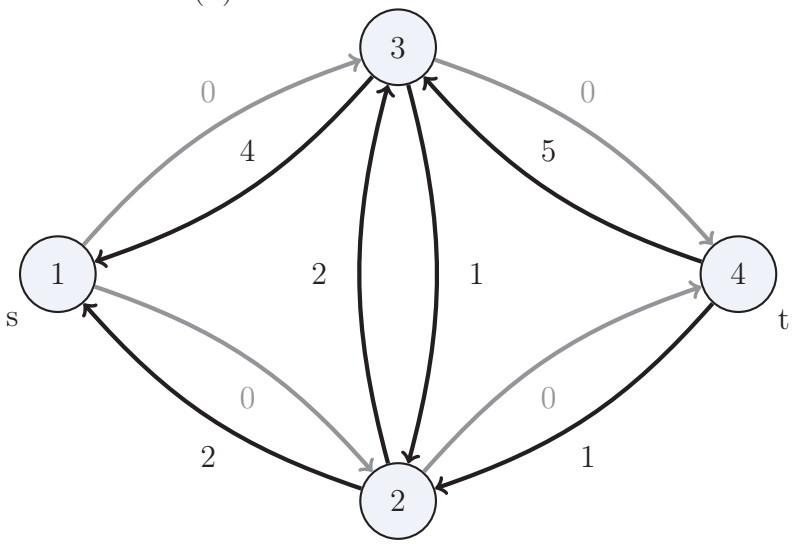


Netzwerk G:



Flusswert $F = 6$

Restnetzwerk $G(x)$:



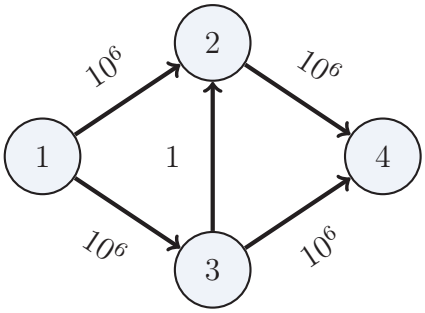
Hier existiert in $G(x)$ kein Pfad mehr von s nach t. Daher ist keine weitere Erhöhung mehr möglich.

Anmerkungen:

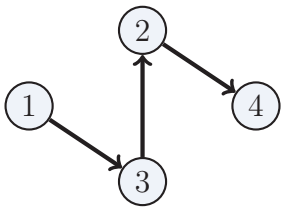
- Für den Nullfluss ($x = 0$) gilt $G(x) = G$.
- Wird eine Kante (i, j) in G saturiert (dh. $x_{ij} = u_{ij}$), so wird die Kante (i, j) aus $G(x)$ entfernt.
- Setzt man $x_{ij} = 0$, so wird die Gegenkante (j, i) aus $G(x)$ entfernt.

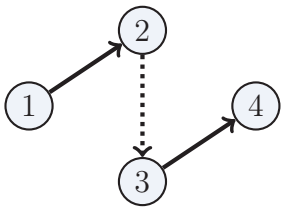
$\Rightarrow$ Änderungen an der Flussfunktion führen zu Änderungen in $G(x)$.

Bemerkung
Der einfache Pfaderhöhungsalgorithmus kann sich sehr schlecht verhalten.



Im schlechtesten Fall





2×10^6 Pfaderhöhungen

Korrektheit

i) x ist immer ein Fluss (Kapazitätsbedingung + Massenbalance-Bedingung)

ii) terminiert mit Flus x, sodass in $G(x)$ kein Pfad von s nach t existiert.

Beweis
Beweis mit Hilfe vom MaxFlow/MinCut-Theorem.

Implementierung

1. Keine expliziete Darstellung von G(x).
   Stattdessen: Navigiere im Originalgraphen.
   Verwende ausgehende Kanten, wenn $u_{ij} > x_{ij}$ und
   eingehende Kanten, wenn $x_{ij} > 0$.
2. Berechnung eines erhöhenden Pfades
   Verwende DFS oder BFS vom Knoten s aus und berechne die Kanten wie in 1. beschrieben. Stoppe, wenn t erreicht wurde.
3. Darstellung des Pfades durch pred-Verweise (Vaterverweise im DFS- oder BFS-Baum).

5.1.3 Pfaderhöhung: (Maxflow) s, t

Idee: Versuche im Restnetzwerk einen Pfad mit Restkapazität zu finden, indem z.B mit `explorefrom(s)`, `dfs(s)` oder `bfs(s)` alle von s aus erreichbaren Knoten gesucht werden, bis t gefunden wird.

Labeling-Algorithmus

```
void MF_Labeling(const graph& G, node s, node t,const
   edge_array<int>& cap, edge_array<int>& flow){
   node_array<bool> labeled; // besuchte Knoten
   node_array<edge> PRED; // Pfade

   // Initialisierung
   edge e;
   forall_edges(e,G){
      flow[e] = 0
   }
   node v;
   forall_nodes(v,G){
      labeled[v] = false
      pred[v] = NULL
   }

   while(true){
      queue<node> L; // Breitensuche
      labeled[s] = true;
      L.append(s);
      while(!L.empty()){
         node v = L.pop();
         // Iteriere über alle im Restnetzwerk adjazenten Kanten
         edge e;
         forall_out_edges(e,v){
            if(flow[e] == cap[e]) // Restkapazität ist null
               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 5.1: MF_Labeling

```
void AUGMENT(const graph& G, node s, node t, const
   node_array<edge>& PRED, const edge_array<int>& cap,
   edge_array<int>& flow){
   int delta = MAXINT; // Restkapazität von P
   node v = t;
   while(v != s){
      int r;
      edge e = PRED[v];
      if(v == G.source(e)){ //Rückwärtskante
         r = flow[e];
         v = G.target(e);
      }
      else{ // Vorwärtskante
         r = cap[e] - flow[e];
         v = G.source(e);
      }
      if(r < delta)
         delta = r; // min
   }
```

```
    // Eigentliche Flusserhöhung
    v = t;
    while(v != s){
        edge e = PRED[v];
        if(v == G.source(e)){ //Rückwärtskante
            flow[e] = flow[e] - delta;
            v = G.target(e);
        }
        else{ // Vorwärtskante
            flow[e] = flow[e] + delta;
            v = G.source(e);
        }
    }
}
```

Listing 5.2: 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 Menge 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.