

3 Graphen und Netzwerke

In der Praxis häufigst angewandte Methoden des OR:

- lineare Optimierung
- Methoden auf Graphen und Netzen
 - anschauliches Bild komplexer Zusammenhänge, natürliches Bild bei Transport, Versorgung, Beschaffung, Belieferung, Durchfluß, Entsorgung, Terminplanung, ...
 - effiziente (polynomiale) Behandlung auch großer Probleme

Themen diese Kapitels (andere Klassifizierung)

auch: **Kombinatorische Optimierung**

Dynamische Optimierung

...

3.1 Grundlegendes (Erinnerung; s. Datenstrukturen)

Graph (ungerichteter Graph):

$G := [V, E]$

Menge V von **Knoten** V , V endlich

Menge E von **Kanten** $E \subseteq V \times V =$

welche je 2 Knoten verbinden:

Kante $e = \{i, j\}$ $e \in E; i, j \in V$ (hier: $e = [i, j]$)

ist zweielementige Untermenge von V

(auch: Inzidenzabbildung ordnet

jedem $e \in E$ genau 2 Elemente $i, j \in V$ zu)

für $e = [i, j]$ heißt e **inzident** mit i und j
heißten i und j **adjazent**

G heißt **vollständig**,

wenn mit i, j auch $[i, j]$ enthalten

für $e = [i, j]$ heißen i und j **Nachbarn**,

$N(i)$ bezeichne Menge von Nachbarn eines Knoten i ,

Anzahl von Nachbarn heißt **Grad** $d(i)$

Knoten ohne Nachbarn heißen **isoliert**

$G' := [V', E']$ heißt

- **Teilgraph** von $G = [V, E]$,
wenn $V' \subseteq V$ und $E' \subseteq E$

- **echter Teilgraph** von $G = [V, E]$,
wenn G' Teilgraph von G
sowie $V' \subset V$ oder $E' \subset E$

- durch V' **induzierter Teilgraph** von $G = [V, E]$,
wenn $V' \subseteq V$ und $E' = \{ [i, j] \mid i, j \in V'; [i, j] \in E \}$

Haben 2 oder mehr Kanten (bzw. Pfeile)
 beide Anfangsknoten (Anfangs- und Endknoten)
 gemeinsam, heißen sie **parallel**

Sind für eine Kante
 beide Anfangsknoten (Anfangs- und Endknoten)
 identisch, sprechen wir von einer **Schlinge**

Annahme (ab jetzt):

Graphen (Digraphen) besitzen

- keine parallelen Kanten (Pfeile) (**antiparallel** erlaubt)
- besitzen keine Schlingen

Beschreibungen von Graphen

Digraphen

sei $V = \{v_1, v_2, \dots, v_n\}$ $E = \{e_1, e_2, \dots, e_m\}$

- Inzidenzmatrix

$H_{n,m}(G)$

$h_{ij} = 1$, e_i inzident mit v_j

$h_{ij} = 0$, sonst

$H_{n,m}(G)$

$h_{ij} = +1$, e_i pos. inz. mit v_j

$h_{ij} = -1$, e_i neg. inz. mit v_j

$h_{ij} = 0$, sonst

- Adjazenzmatrix

$U_{n,n}(G)$

$u_{ij} = 1$, $[v_i, v_j] \in E$

$u_{ij} = 0$, sonst

$U_{n,n}(G)$

$u_{ij} = 1$, $\langle v_i, v_j \rangle \in E$

$u_{ij} = 0$, sonst

- **Adjazenzliste**: n-array für Knoten ("**Forward Star**")
 je Eintrag: Verweis auf Liste
 aller Nachbarn aller Nachfolger

- + "**andere**" (s. Bäume)

Kantenfolge:

Folge von Knoten und Kanten in Graph

$$i_0, [i_0, i_1], i_1, \dots, i_{r-1}, [i_{r-1}, i_r], i_r =: [i_0, i_1, \dots, i_r]$$

i_0, i_r : Endknoten

$i_0 \ i_r$: offene Kantenfolge

mit verschiedenen Knoten:

Kette

$i_0 = i_r$: geschlossene Kantenfolge

m. versch. Zwischen-Knoten:

Kreis

Graph ohne enthaltenen Kreis: **kreisfrei**

Knoten i, j heißen **verbunden**,

wenn es Kantenfolge (Kette) mit Endknoten i, j gibt

Knoten per def mit sich selbst verbunden

Graph G heißt **zusammenhängend**,

wenn alle Knotenpaare verbunden

zusammenhängender Teilgraph G' von G

heißt **Zusammenhangskomponente**,

wenn G' "maximal",

dh G' ist nicht echter Teilgraph

eines zusammenhängenden Teilgraphen von G

Pfeilfolge:

Folge von Knoten und Pfeilen in Digraph

$$i_0, \langle i_0, i_1 \rangle, i_1, \dots, i_{r-1}, \langle i_{r-1}, i_r \rangle, i_r =: \langle i_0, i_1, \dots, i_r \rangle$$

i_0, i_r : Endknoten

$i_0 \ i_r$: offene Pfeilfolge

mit verschiedenen Knoten:

Weg

$i_0 = i_r$: geschlossene Pfeilfolge

m. versch. Zwischen-Knoten:

Zyklus

Digraph ohne enthaltenen Zyklus: **zyklenfrei**

Folge von Knoten und Pfeilen mit (potentiell)

nichtidentischer Orientierung:

Semipfeilfolge

analog:

Semiweg, Semizyklus

Knoten i, j heißen **verbunden**,

wenn es Semipfeilfolge mit Endknoten i, j gibt

Knoten j ist von Knoten i **erreichbar**,

wenn es Pfeilfolge (Weg) von i nach j gibt

$R^+(i)$ bezeichne Menge Knoten, die von i erreichbar

$R^-(i)$ bezeichne Menge Knoten, von denen i erreichbar

Digraph G heißt **schwach zusammenhängend**,

wenn alle Knotenpaare verbunden

Digraph G heißt **stark zusammenhängend**,

wenn jeder Knoten von jedem Knoten erreichbar

Schwache (starke) Zusammenhangskomponente von G

ist maximaler schwach (stark) zusammenhängender

Teilgraph von G

Baum : zusammenhängender, kreisfreier Graph,
 $|V| - 1$ Kanten, Zufügen Kante erzeugt Kreis

Wald: kreisfreier Graph
 mit k Bäumen mit k Zusammenhangskomponenten

gerichteter Baum / Wurzelbaum (mit Wurzel r)
 schwach zusammenhängender Digraph,
 mit Quelle r und $\text{in}(i)=1, i \neq r$
 (genau 1 Vorgänger)

..., **Blatt, Ast, Vater/Mutter/Elter, Sohn/Tochter/Kind,**
(Knoten-)Tiefe, ...

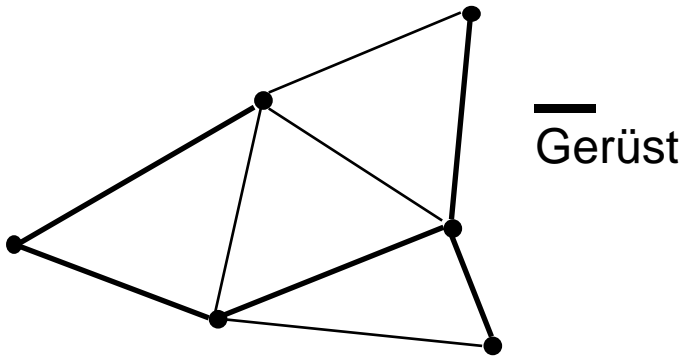
Binärbaum: Baum mit 2 Kindern je Knoten

balancierter Binärbaum der Tiefe t :
 Binärbaum mit genau 2^{t-1} Knoten der Tiefe $t-1$,
 Knoten der Tiefe $t-1$
 - mit 2 Kindern: links
 - mit 0 Kindern: rechts
 - mit 1 Kind: dazwischen (maximal 1 Knoten)

Heap: balancierter Binärbaum,
 wo "Wert" jedes Knotens
 "Wert" seiner Töchter

Gerüst eines Graphen G :

- zusammenhängender Teilgraph von G
- enthält alle Knoten von G
- besitzt minimale Kantenzahl



Gerüst ist Baum (spanning tree, Spannbaum)

Graph G besitzt (mindestens 1) Gerüst,
wenn er zusammenhängend ist

gerichtetes Gerüst eines Digraphen G :

- Teilgraph von G
- enthält alle Knoten von G
- stellt gerichteten Baum dar

in diesem Zusammenhang gelte

R

$$a < \infty \quad a \in \mathbb{R}$$

$$a + \infty = \infty + a = \infty \quad a \in \mathbb{R}$$

$$a = \begin{cases} 0 & a = 0 \\ \infty & \text{sonst} \end{cases} \quad a \in \mathbb{R}_+ \setminus \{0\}$$

offene Kantenfolge

mit Endknoten i, j

mit kleinster / größter Länge unter allen Folgen i nach j

heißt **kürzeste / längste**

Kantenfolge

offene Pfeilfolge

mit Anf.kn. i + Endkn. j

mit kleinster / größter Länge unter allen Folgen i nach j

Pfeilfolge

analog: kürzeste / längste

Ketten

Wege

Länge kürzester

Kantenfolge (Kette)

von i nach j heißt **Entfernung**

$d[i, j]$

Pfeilfolge (Weg)

$d\langle i, j \rangle$

Enthält

G keine Kreise

G keine Zyklen

negativer Länge, dann existiert Entfernung für

verbundene i, j

von i erreichbares j

3.2 Minimalgerüste / minimal spanning trees

Problemstellung:

Versorgungs-, Verbreitungsnetz
zu planen

Strom, Gas, ...,
Daten (WAN), Telefon

Versorgungspunkte
Knoten eines Graphen

potentielle Verbindungsstrecken
Kanten des Graphen,
bewertet mit Kosten

Installation, Betrieb, ...

gesucht ist kostengünstigstes Netz

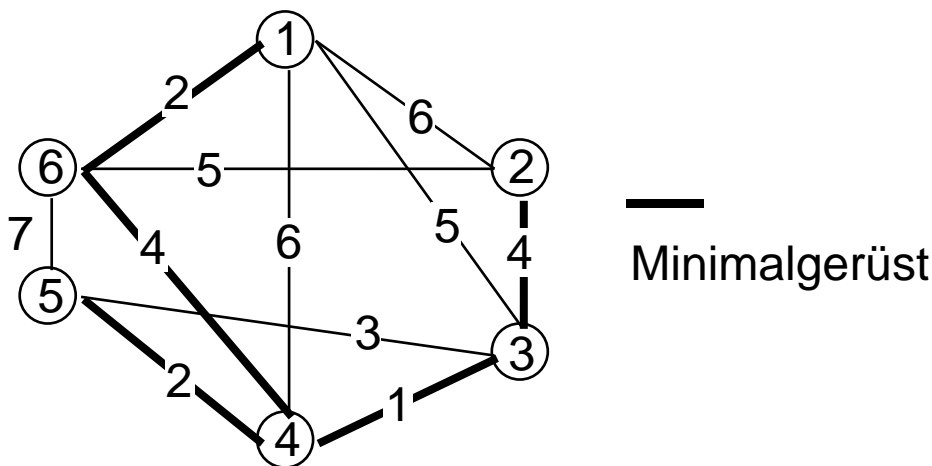
Minimalgerüst des Graphen:

- Gerüst
- mit minimaler Länge

alle erreicht

Summe Streckenkosten

Beispiel



Bestimmung Minimalgerüst in bewertetem Graphen

- $G = [V, E; c]$
- zusammenhängend
 - $n := |V|$ Knoten, $n \geq 2$; $m := |E|$ Kanten
 - reellwertige Gewichtsfunktion c

Algorithmen von Prim, von Kruskal
 ("hier mit die ältesten", 50er)

Prim (Skizze): "Folge von Bäumen"

- wähle Kante mit geringstem Gewicht (oder eine davon)
 + Endknoten "Initialbaum"
- wiederhole $(n-2)$ -mal:
 erweitere Baum zu neuem Baum,
 durch Zufügen einer (ungenutzten) Kante,
 unter Wahl des kleinstmöglichen Kantengewichts
- Ergebnis ist Baum mit $n-1$ Kanten Gerüst,
 aufgrund des Auswahlmechanismus Minimalgerüst

häufiger verwendet Variante nach
Kruskal (Skizze): "Folge von Wäldern"

- wähle Kante mit geringstem Gewicht (oder eine davon)
 + Endknoten "kreisfreier Initialwald"
- wiederhole $(n-2)$ -mal:
 - erweitere kreisfreien Wald zu kreisfreiem Wald,
 durch Zufügen einer (ungenutzten) Kante,
 unter Wahl des kleinstmöglichen Kantengewichts
- Ergebnis ist krfr. Wald mit $n-1$ Kanten Baum, Gerüst,
 aufgrund des Auswahlmechanismus Minimalgerüst

Prim- und Kruskal-Algorithmen sind Greedy-Algorithmen:
in jedem Schritt "lokal" optimale Entscheidung

dennoch "globale" Optimalität gegeben:
glücklicher (eher seltener) Fall

Zur Korrektheit der "MST"-Algorithmen:

Satz 3.2.01 : Walderweiterung und MST-Eigenschaft

Seien

$G := [V, E]$:	Graph
$[U_i, T_i]; i=1, \dots, k$:	Bäume in G,
$U_i \cap U_j = \emptyset$:	disjunkt
$\{ [U_i, T_i] \mid i=1, \dots, k \}$:	Wald in G,
$\bigcup (U_i) = V$:	V aufspannend

sowie

$[v, u]$ kürzeste aller Kanten
mit (lediglich) 1 Endpunkt u in U_1

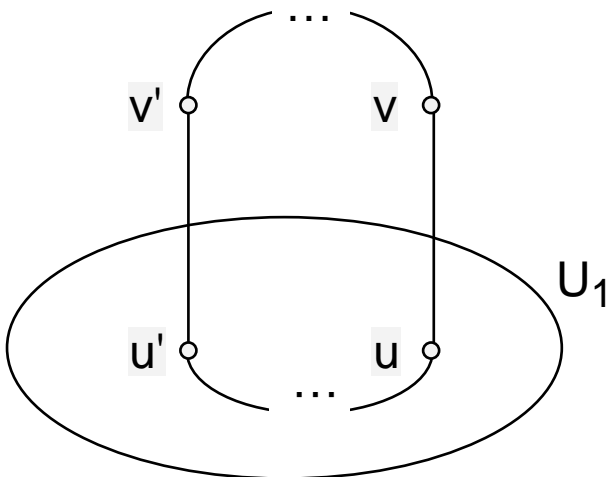
Dann existiert unter allen Gerüsten, welche alle Kanten
 $T = \bigcup (T_i)$ des Waldes
enthalten, ein minimales Gerüst, das Kante
 $[v, u]$ enthält

Beweis per Widerspruch:

Sei $[V, F]$ Gerüst mit $F \subset T$
und $[v, u] \notin F$,
kürzer als alle Gerüste, die T und $[v, u]$ enthalten

Füge $[v, u]$ zu F hinzu:

Kreis resultiert (Baumeigenschaften)
enthält nicht nur Knoten aus U_1 , da $v \in U_1$
Kante $[v', u'] \in [v, u]$ auf Kreis, $u' \in U_1, v' \in V - U_1$



Entfernung von $[v', u']$

Spannbaum $[V, F']$, $F' = F - \{[v, u]\} + \{[v', u']\}$
 mit Länge (Kosten) Länge (V, F)

Widerspruch zu $[V, F]$ kürzer als jeder Spannbaum,
 der T und $[v, u]$ enthält

Prim-Algorithmus:

Initialwald: kürzeste Strecke + (restliche) Einzelknoten
 Aufwand: $O(n^2)$

Kruskal-Algorithmus:

Initialwald: Spannbäume aus kürzesten Kanten
 Aufwand: $O(m \log m)$, bei heap-Implementierung G
 worst case (G vollständig): $O(n^2 \log n)$

Vergleich: fallabhängig,
 dünn besetzt pro Kruskal

3.3 Kürzeste Wege

Sei $N = \langle V, E; c \rangle$ bewerteter Digraph

mit $V = \{1, \dots, n\}$
 $|E| = m$

$c: E \rightarrow \mathbf{R} \quad c_{ij} := c\langle i, j \rangle$

und N ohne Zyklen negativer Länge

Gesucht seien

(alternative, aber überlappende Fragestellungen)

(a) für festen **Startknoten** r und festen **Zielknoten** s
 Entfernung $d\langle r, s \rangle$ (kürzester Weg r nach s)

(b) für festen **Startknoten** r und alle Knoten $j \in V$
 Entfernungen $d\langle r, j \rangle$ (kürzeste Wege r nach j)

(c) für alle Knoten $i, j \in V$
 Entfernungen $d\langle i, j \rangle$ (kürzeste Wege i nach j)

Lösungen (in Folge) decken zusätzlich ab

- längste Wege:
 entweder Vorzeichenänderung aller Bewertungen
 oder "Minimierung" "Maximierung" (in Folge)
- kürzeste / längste Ketten in bewerteten Graphen
 mit nichtnegativem c :
 kürzeste / längste Wege in "zugeordneten"
 symmetrischen bewerteten Digraphen
 (Kante $[i, j]$ Pfeilpaar $\langle i, j \rangle$ und $\langle j, i \rangle$,
 $c\langle i, j \rangle = c\langle j, i \rangle = c[i, j]$)

Fragestellung (b) ("r zu allen")

definiere

$$d_j := \begin{cases} 0 & j = r \\ d_{\langle r, j \rangle} & j \in R(r), \quad r \neq j \quad \forall \\ & j \in R(r) \end{cases}$$

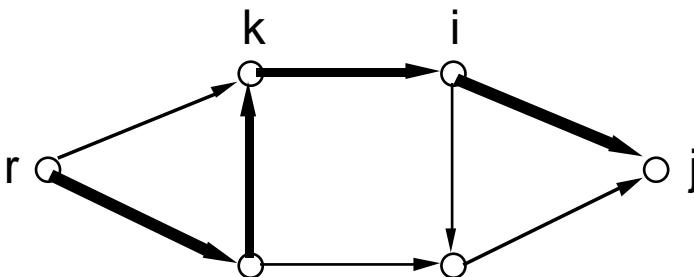
$W_j :=$ "entsprechender" (kürzester) Weg

Gibt es Beziehungen zwischen den d_j ?

Ja: "Bellmann'sches Optimalitätsprinzip"
(hier Spezialfall, grundl. Bedeutung für
"Dynamische Optimierung")

Teilwege in kürzesten Wegen sind (selbst) kürzeste Wege

im Beispiel:



wäre (fetter) Weg r nach i nicht = W_i ,
dann könnte (fetter) Weg r nach j nicht = W_j sein

demnach gilt: $d_j = d_i + c_{ij}$
und allgemein die "Bellmann'sche Gleichung"

$$(3.3.01) \quad d_j = \min_{i \in P(j)} (d_i + c_{ij}) \quad j \in R(r), \quad r$$

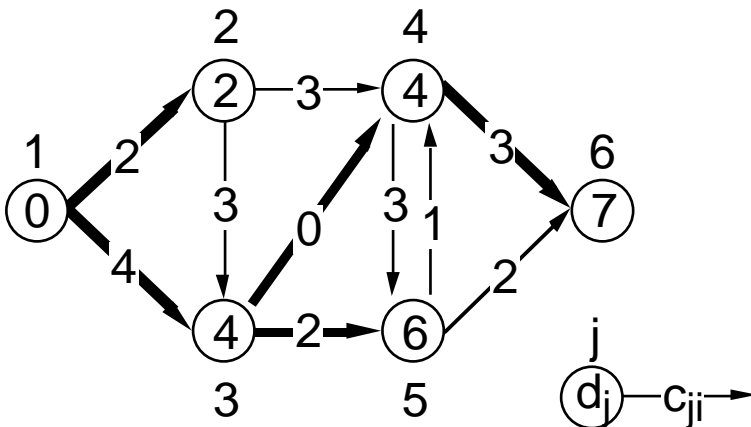
In Anwendung:

Algorithmen kürzeste Wege, **Baumalgorithmen**

- konstruieren für alle Knoten $j \in R(r) \setminus \{r\}$
 kürzesten Weg W_j ,
 wo alle Teilwege von W_j kürzeste Wege sind

konstruieren Teildigraph $=:T$ von N ,
 mit Knotenmenge $R(r)$,
 je $j \in R(r) \setminus \{r\}$ genau 1 Vorgänger p_j in T
 konstruieren gerichteten Baum mit Wurzel r ,
 wo alle Wege von r aus kürzeste Wege sind,
 genannt **Wegebaum**

im Beispiel:



tabellarisch:

j	1	2	3	4	5	6
d_j	0	2	4	4	6	7
p_j	1	1	1	3	3	4

Prinzip Baumalgorithmen

Baumalgorithmen arbeiten iterativ:

Im Verlauf / zu Ende der Verfahren

- ist T der bisher gefundene Wegebaum
/ der optimale Wegebaum
- tragen Knoten
"Marke (label)" d_j
= kürzeste bisher gefundene
/ kürzeste Entfernung von r aus
- wird Menge Q "neu markierter" Knoten unterhalten
/ ist Q leer

Notierung in "nahezu" Umgangssprache-Sprache

{Netz als Adjazenzliste / Nachfolgerliste}

initialisiere netz, r ;

{Initialisierung Algorithmus:}

$d_r := 0$; $p_r := r$; $Q := \{r\}$;

$d_i := \infty$; $p_i := 0$; $i = 1, \dots, n$; $i \neq r$

{Iteration:}

while Q

 extract_any $i \in Q$; {Entnahme 1 mark. Knoten}

for all $j \in \text{succ}(i) \mid d_j > d_i + c_{ij}$

$d_j := d_i + c_{ij}$; $p_j := i$; $Q := Q \cup \{j\}$

Korrektheit: "offensichtlich"

Algorithmus 3.3.01 : Kürzeste Wege - Baumalgorithmus

Notierung in Pseudo-Sprache

```

{Netz als erweiterte Adjazenzliste
                                     / Nachfolgerliste:}
succ_info record (succ int, c real);
knoten record (s list of succ_info, d real, p int);
netz array[1:n] of knoten;

{zusätzliche Daten des Algorithmus:}
r int;           {Startknoten}
Q set of int;   {Menge markierter Knoten}

initialisiere netz, r;

{Initalisierung Algorithmus:}
netz[r].d:=0; netz[r].p:=r; Q.insert(r);
for i=1 step 1 until n | i r do
begin netz[i].d:=" "; netz[i].p:="0" end;

{Iteration:}
while not Q.empty do
begin
  i:=Q.extract_any;      {Entnahme 1 mark. Knoten}
  for all (j,c) netz[i].s
    | netz[j].d > netz[i].d + c do
    begin
      netz[j].d := netz[i].d + c;
      netz[j].p := i;
      Q.insert(j)      {Einfügung 1 mark. Knoten}
    end
end

{: Wegebäum als Nachfolgerliste}

```

verschiedene Baumalgorithmen entwickelt,
Unterschiede in

- Entnahme Knoten aus Menge Q markierter Knoten,
Datenstruktur für Q

einmalige Aufnahme in Q: **label-setting** Verfahren (LS)

wiederholte Aufnahme in Q: **label-correcting** Verf. (LC)

Im folgenden behandelt / erwähnt + eingeordnet

- LC-Algorithmus A (nach Ford, modifiziert):
Q als **queue** implementiert
Einfügen "hinten",
Entnahme "vorne"
- LC-Algorithmus B
Q als Paar von **queues** Q1, Q2 implementiert
Einfügen erstmalig in Q2, "hinten",
sonst in Q1, "hinten",
Entnahme falls Q1 aus Q1, "vorne",
sonst aus Q2, "vorne"

Variante: Q1 als **stack**

- LS-Algorithmus A nach Dijkstra (Bewert'gen nichtnegativ)
Q als **heap** implementiert
- LS-Algorithmen B / C nach Bellmann (zyklenfreie Netze)
N **topologisch sortiert**
/ Q als **queue** + Nutzung **Eingangsgrad**-Information $\bar{(.)}$

Algorithmus 3.3.02: Kürzeste Wege - LC à la Ford

Entnahmen aus Q in der Reihenfolge der Einfügungen

```
{Netz als erweiterte Adjazenzliste:}
succ_info record (succ int, c real);
knoten record (s list of succ_info, d real, p int);
netz array[1:n] of knoten;
```

```
{zusätzliche Daten des Algorithmus:}
r int;           {Startknoten}
Q queue of int; {Menge markierter Knoten}
```

```
initialisiere netz, r;
```

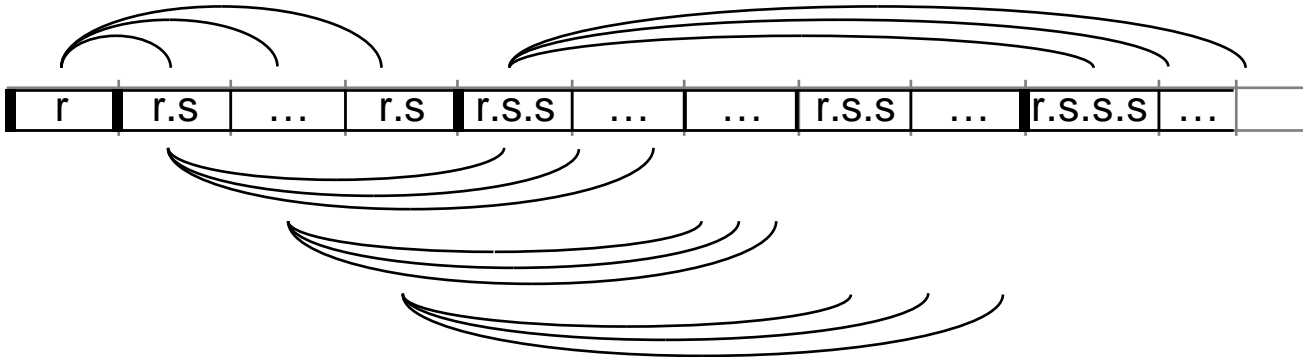
```
{Initialisierung Algorithmus:}
netz[r].d:=0; netz[r].p:=r; Q.enqueue(r);
for i=1 step 1 until n | i r do
begin netz[i].d:=" "; netz[i].p:="0" end;
```

```
{Iteration:}
while not Q.empty do
begin
  i:=Q.dequeue; {Entnahme mark. Knoten vorne}
  for all (j,c) netz[i].s
    | netz[j].d > netz[i].d + c do
    begin
      netz[j].d := netz[i].d + c;
      netz[j].p := i;
      if not Q.element(j) then Q.enqueue(j) {hinten}
    end
  end
end
```

```
{: Wegebaum als Nachfolgerliste}
```

Korrektheit

Ablauf in Phasen vorstellbar:



nach **sind in Q**

Phase 0: Knoten r

Phase 1: Nachfolger von r
 kürzeste Wege bestimmt aus Pfeilfolgen der Länge 1

Phase 2: Nachfolger von Nachfolger von r
 kürzeste Wege bestimmt aus Pfeilfolgen der Länge 2

...
 Phase n-1: Nachfolger (n-1)-ter Stufe von r
 kürz. Wege bestimmt aus Pfeilfolgen der Länge n-1
 alle kürzesten Wege bestimmt

Phase n: keine Knoten (falls keine Zyklen neg. Länge)

Aufwand

- je Phase höchstens jeder Pfeil (aus insgesamt m)
 höchstens 1-mal inspiziert
- n Phasen

Aufwand ist $O(m \cdot n)$ (Erinnerung: $m < n^2$)

LC-Algorithmus B (hier nicht ausgeführt) hat

- schlechtere worst case Komplexität $O(m n^2)$
 - bessere mittlere Komplexität
bei steigenden Pfeilzahlen von kürzesten Wegen
-

LS-Algorithmus nach Dijkstra
(nichtnegative Bewertungen)

- verfolgt Idee, nur Knoten j aus Q zu entnehmen,
deren label d_j endgültig feststeht
(für die kürzester Weg festliegt)
- realisiert Idee durch Q-Entnahme Knoten k
mit minimalem label d_k in Q

Algorithmus

Algorithmus 3.3.03: Kürzeste Wege - LS à la Dijkstra

Entnahmen jeweils des Elementes aus Q mit minimalem d

```

{Netz als erweiterte Adjazenzliste:}
succ_info record (succ int, c real);
knoten record (s list of succ_info, d real, p int);
netz array[1:n] of knoten;

{zusätzliche Daten des Algorithmus:}
r int;                {Startknoten}
Q_info record (i int, d real)
Q heap of Q_info;     {Menge markierter Knoten}

initialisiere netz, r;

{Initialisierung Algorithmus:}
netz[r].d:=0; netz[r].p:=r; Q.insert(r,0);
for i=1 step 1 until n | i r do
begin netz[i].d:=" "; netz[i].p:="0" end;

{Iteration:}
while not Q.empty do
begin
  (i,d):=Q.extract_min; {Entnahme min. mark. Knoten}
  for all (j,c) netz[i].s
    | netz[j].d > netz[i].d + c do
    begin
      netz[j].d := netz[i].d + c;
      netz[j].p := i;
      if not Q.element(j) then Q.insert(j,netz[j].d)
      else Q.update(j,netz[j].d)
    end
  end
end

{: Wegebäum als Nachfolgerliste}

```


Korrektheit

aufsetzend auf "offensichtlicher" Korrektheit
allgemeinen Baumalgorithmus

Behauptung in Q ist Knoten k mit kleinstem label d_k
endgültig bewertet

Begründung Algorithmus untersucht Nachfolger j von k ,
nimmt $u \cup j$ in Q auf mit $d_j = d_k + c_{ki}$
wegen $c_{ki} \geq 0$
labels $< d_k$ können nicht nach Q gelangen,
insbesondere nicht für Knoten k
Behauptung

Aufwand

- je Q -Entnahme Operationen extract / insert / update
bzgl. heap (n Einträge): jeweils $O(\log n)$
(s. Datenstrukturen)
- jeder Knoten maximal 1-mal in Q aufgenommen,
mit insgesamt maximal m Pfeilüberprüfungen

Aufwand ist $O(m \log n)$

Trotz besserer theoretischer Zeitkomplexität
können LC-Algorithmen überlegen sein

- bei größeren Netzen
- bei "dünn besetzten" Netzen (wenig Pfeile)

LS-Algorithmen B, C (zyklenfreie Netze)
(hier nicht diskutiert)
haben Zeitkomplexität $O(m)$

Bis jetzt Konzentration auf

(b) "von einem r zu allen j"

Zu (a) "von einem r zu einem s"

LS-Verfahren können abgebrochen werden,
wenn s (erreicht) entfernt
LC-Verfahren dagegen nicht (Nachteil)

Zu (c) "von allen i zu allen j"

(nach wie vor : keine Zyklen negativer Länge)

definiere

$$d_{ij} := \begin{cases} 0 & j = i \\ d_{\langle i,j \rangle} & j \in R(i), \quad i, j \in V \\ \text{sonst} & \end{cases}$$

$W_{ij} :=$ "entsprechender" (kürzester) Weg von i nach j

$$p_{ij} := \begin{cases} i & j = i \\ \text{Vorgänger j auf } W_{ij} & j \in R(i), \quad i, j \in V \\ 0 & \text{sonst} \end{cases}$$

$D := (d_{ij} \mid i, j = 1, \dots, n)$ "Entfernungsmatrix"

$P := (p_{ij} \mid i, j = 1, \dots, n)$ "Wegematrix"

Idee "**Tripel-Algorithmus**" nach Floyd / Warshall:

für $i, j \in V = \{1, \dots, n\}$, $j \neq i$,

$k \in V \setminus \{i, j\}$

sei $d_{ij}^{(k-1)}$ Länge kürzesten Weges i nach j , $k \geq 2$,

der (außer i, j) allenfalls Knoten $k-1$ enthält

$d_{ij}^{(k-1)} :=$ falls kein solcher Weg existiert

für $i=j$ $d_{ij}^{(k-1)} := 0$

offensichtlich gilt

$$d_{ij} = d_{ij}^{(n)} \quad i, j \in V$$

Enthält W_{ij} (außer i und j) allenfalls Knoten k ,

- und ist k nicht enthalten,

$$\text{dann } d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- und ist k enthalten,

$$\text{dann } d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

Es gilt also:

$$d_{ij}^{(0)} = c_{ij} \quad i, j = 1, \dots, n$$

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \quad i, j = 1, \dots, n; k \geq 1$$

"Tripeloperation"

(wo, wie üblich, $c_{ii}=0$

$$c_{ij} = \begin{cases} c_{ij} & \text{falls } \langle i, j \rangle \in E \\ \infty & \text{sonst} \end{cases}$$

Folgend skizzierter Tripel-Algorithmus

verwendet Entfernungsmatrix, Wegematrix "direkt"

zweidimensionale arrays

Algorithmus 3.3.04: Kürzeste Wege - Tripelalgorithmus

{Netz (der Einfachheit halber) als Gewichtsmatrix:}
N array [1:n,1:n] **of real**;

{zusätzliche Daten des Algorithmus:}
D array [1:n,1:n] **of real**;
P array [1:n,1:n] **of int**;

initialisiere N;

{Initialisierung Algorithmus:}

```

for i=1 step 1 until n do
  for j=1 step 1 until n do
    begin
      D[i,j]:=C[i,j];
      if D[i,j]<" " then P[i,j]:=i
        else P[i,j]:="0" {"keiner"}
    end;

```

{Iteration:}

```

for k=1 step 1 until n do
  for i=1 step 1 until n | D[i,k]<" " do
    for j=1 step 1 until n
      | D[i,j]>D[i,k]+D[k,j] do
      begin
        D[i,j]:=D[i,k]+D[k,j];
        P[i,j]:=P[k,j]
      end

```

{: kürzeste Wege als Entfernungs- + Wegematrix}

Korrektheit

siehe Vorüberlegungen zum Algorithmus

Aufwand

- k, i, j - Schleifen je $O(n)$
Zeitkomplexität ist $O(n^3)$
- Speicherung als arrays (n,n)
Raumkomplexität ist $O(n^2)$: hoch!

Reduktion Raumaufwand durch sequentielle Wegealgorithmen

- nicht kürzeste Wege zwischen allen Knoten auf einmal
- sondern jeweils nacheinander von $i=1, \dots, n$ zu allen

Dazu:

Lemma 3.3.05 : Kürzeste Wege (Hilfssatz)

Werden in einem Netzwerk $N = \langle V, E; c \rangle$
die Kantenbewertungen geändert gemäß

$$c'_{ij} := c_{ij} + b_i - b_j \quad b_i, b_j \text{ beliebig, reell}$$

dann sind die kürzesten Wege des Netzwerks

$$N' = \langle V, E; c' \rangle$$

identisch zu jenen aus N . Die kürzesten Wegelängen N' sind

$$c'(W_{kl}) = c(W_{kl}) + b_k - b_l$$

Beweis: einfach ausrechnen

Seien kürzeste Wege N von Knoten r bestimmt
 (Wegebaum T_r)

Setze $b_i := d_{ri} \quad i \in V$

wegen $d_{rj} = d_{ri} + c_{ij} \quad \text{ist} \quad c_{ij}' = 0 \quad \text{für } \langle i, j \rangle \text{ aus } T_r$
 N' nichtnegativ bewertet,
 für N' Dijkstra einsetzbar

Somit (Skizze) sequentielle Wegealgorithmen

*bestimme kürzeste Wege von Knoten 1 aus
 unter Verwendung anwendbaren Algorithmus' ;*

```

for k=2 step 1 until n do
  begin
    for all  $\langle i, j \rangle \in E$  do
       $C[i, j] := C[i, j] + D[k-1, i] - D[k-1, j];$ 
      bestimme kürzeste Wege von Knoten k aus  

      unter Verwendung Dijkstra-Algorithmus' ;
      for j=1 step 1 until n do
         $D[k, j] := D[k, j] + D[k-1, j] - D[k-1, k];$ 
    end;
  
```

Zwischenergebnisse können "vergessen" werden,
 Raumkomplexität für D ist $O(n)$
 (auch Zeitkomplexität besser als Tripel-Algorithmus)

Hier "Abbruch", aber

- viele weitere Algorithmus-Varianten
- zusätzliche Probleme + Verfahren
 (k-kürzeste Wege, ...)

3.4 Netzplantechnik

Einsatz zur Planung + Überwachung von Projekten

Projekt:

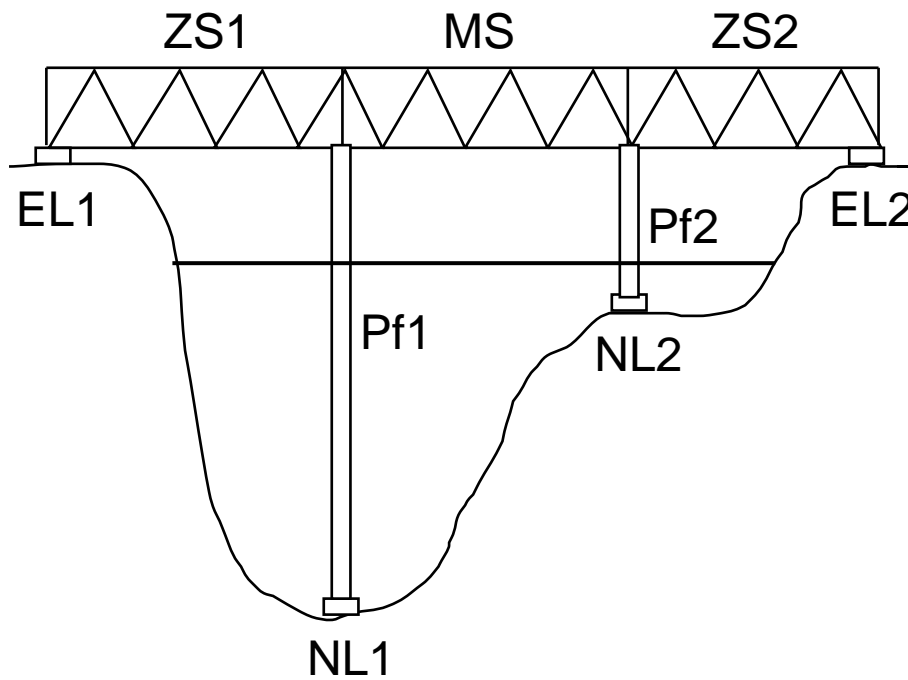
Vorhaben

- aus **Vorgängen** zusammengesetzt
(zeitbeanspruchenden Teilarbeiten)
- durch **Anordnungsbeziehungen** verknüpft
(Abfolge von Vorgängen)
- wo **Ereignisse** Abschluß von (uU mehreren) Vorgängen
Möglichkeit z. Beginn von Vorgängen
markieren
- mit Problemen der **Zeit- und Terminplanung**
 - kürzeste Projektdauer
 - kritische Vorgänge (Verlängerung gleiche
Verlängerung Projektdauer)
 - Anfangs- + Endtermine aller Vorgänge
 - Pufferzeiten aller Vorgänge
(max. Zeitspanne Verschiebung
ohne Beeinfluss'g Projektdauer)

Aufstellung **Netzplan**,

Bestimmung längster Wege in zugeordnetem Netzwerk

Brückenbau



Vorgangsliste (Start - Ende - Verknüpfung)

Vorgang	Dauer / Monate	unmittelbare Vorgänger
A: vorbereitende Planung	8	
B: Fertigung aller Einzelteile	4	A
C: Fertigung + Montage EL1	3	A
D: Fertigung + Montage EL2	5	A
E: Fertig'g + Montage NL1 + Pf1	6	A
F: Fertig'g + Montage NL2 + Pf2	4	A
G: Fertig'g + Montage ZS1	6	B,C,E
H: Fertigung + Montage ZS2	6	B,D,F
I: Fertigung MS	5	B
J: Einschwimmen + Montage MS	1	E,F,I
K: Wartezeit nach Montage MS	1	J

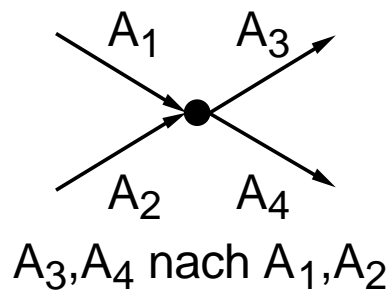
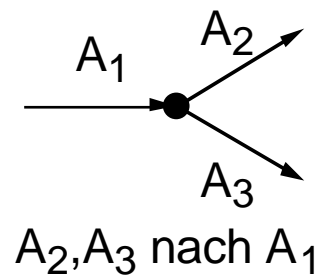
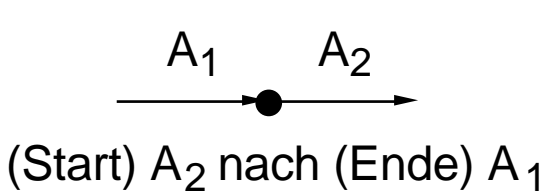
3.4.1 CPM-Netzpläne

ZUR ABBILDUNG VORGANGSLISTE (Ende-Start Option)
VORGANGSPFEILNETZ (CPM)

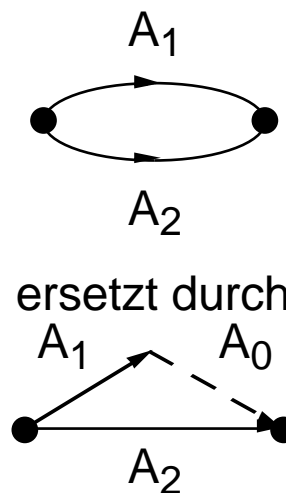
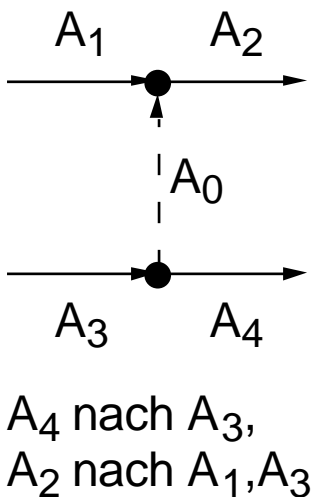
Startvorgang A: kein anderer Vorgang vor A beendet

Zielvorgang A: kein anderer Vorgang nach A begonnen

Direkte Abbildungen aus Vorgangsliste



Abbildungen mit Scheinvorgängen (Dauer 0)

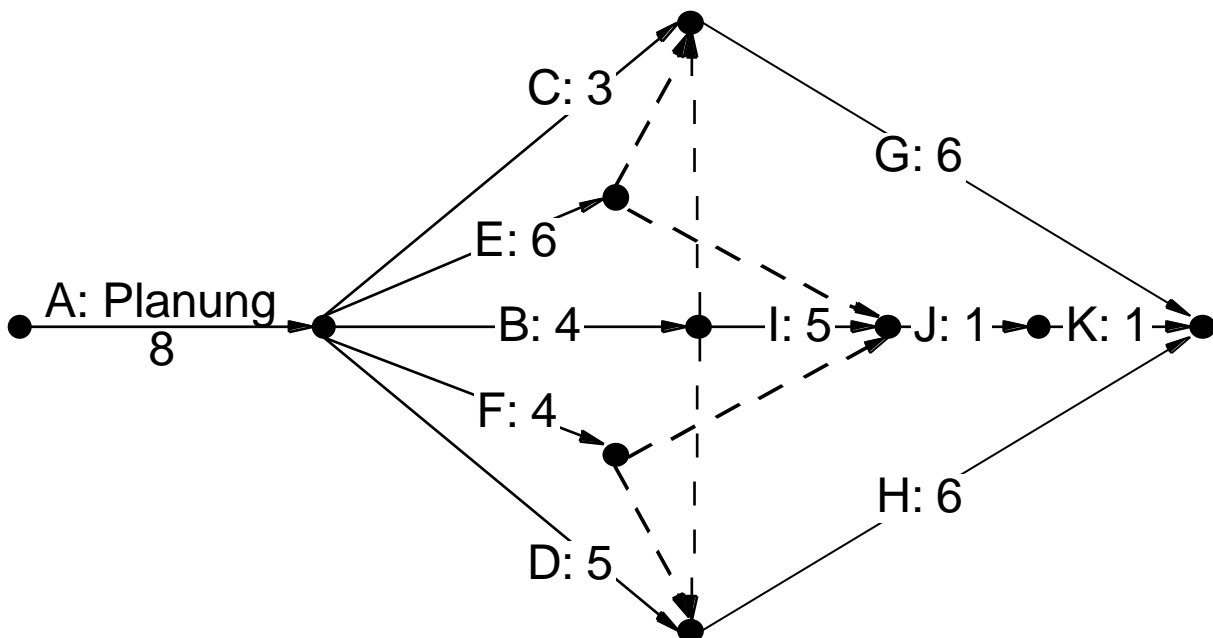


+ weitere Vorschriften:

- Start A_2 nach Teil von A_1 :
 A_1 unterstrukturieren
- alle Startvorgänge von einem Knoten aus: "Quelle"
- alle Zielvorgänge zu einem Knoten: "Senke"
- Start A_2 frühestens T nach Ende A_1 :
entsprechenden Hilfsvorgang einfügen
- Start A_2 spätestens T nach Ende A_1 :
in CPM nicht berücksichtigt

+ **Knoten (Ereignisse) numeriert** $V=\{1,\dots,n\}$
Pfeile $\langle i, j \rangle$ mit Zeitdauern bewertet D_{ij}

Netzplan Beispiel:



Eigenschaften CPM-Netzpläne

- genau 1 Quelle, 1 Senke
- schwach zusammenhängend, zyklensfrei
- alle Knoten (incl. Senke) von Quelle erreichbar
mindestens 1 Weg Quelle → Senke,

zyklensfrei → längster Weg W^* Quelle → Senke
(u.a.) effizienter Bellmann-Alg. einsetzbar

→

- **kürzeste Projektdauer:** Länge längster Weg
- **kritische Vorgänge:** Pfeile $\langle i,j \rangle$ auf W^*
(i,j Knoten, "Ereignisse")
- **kritischer Weg:** W^*

Zeitplanung umfaßt zusätzlich

- **Vorgangstermine** (Anfangs- + Endtermine Vorgänge)
- **Pufferzeiten** (aller Vorgänge)

Bezeichnungen (für Zeitpunkte):

FAZ_{ij} frühest möglicher Start Vorgang $\langle i,j \rangle$
FEZ_{ij} frühest mögliches Ende Vorgang $\langle i,j \rangle$
SAZ_{ij} spätest möglicher Start Vorgang $\langle i,j \rangle$
SEZ_{ij} spätest mögliches Ende Vorgang $\langle i,j \rangle$

FZ_i frühest möglicher Zeitpunkt Ereignis i
SZ_i spätest möglicher Zeitpunkt Ereignis i

ohne Beeinträchtigung "gesetzter" Projektdauer := T

Knotenmenge Netzplan $V = \{1, \dots, n\}$,
 topologisch sortiert Knoten 1: Quelle, n: Senke

- Projekt beginne zu Zeitpunkt 0
 $FZ_1 = 0$
- andere FZ_i : FZ_i ist Mindestabstand
 von Projektbeginn bis Zeitpunkt Ereignis i
 = Länge längster Weg 1 i

so daß, mit Bellmann (3.3.01), $\min \quad \max$

$$FZ_i = \max_k \left(FZ_k + D_{ki} \right) \quad i = 2, \dots, n$$

"vorwärts"-Anwendung; wg topol. Sortier'g $P(i) \{1, \dots, i-1\}$

- kürzeste Projektdauer FZ_n
 "sinnvolle" Setzung $T \leq FZ_n$
 + Setzung $SZ_n = T$, falls vorgegeben
 $= FZ_n$, sonst

- andere SZ_i : $SZ_n - SZ_i$ ist Mindestabstand
 von Zeitpunkt Ereignis i bis Projektende
 = Länge längster Weg i n

so daß, mit Bellmann (3.3.01), $\min \quad \max$

$$SZ_n - SZ_i = \max_j \left(SZ_n - (SZ_j - D_{ij}) \right)$$

$$SZ_i = \min_j \left(SZ_j - D_{ij} \right) \quad i = n-1, \dots, 1$$

"rückwärts"-Anwendung; wg topol. Sortier'g $S(i) \{i+1, \dots, n\}$

- Vorgangstermine
(nur für reale, nicht-Schein- Vorgänge interessant)

$$\begin{aligned} \text{FAZ}_{ij} &= \text{FZ}_i \\ \text{FEZ}_{ij} &= \text{FZ}_i + D_{ij} \\ \text{SAZ}_{ij} &= \text{SZ}_j - D_{ij} \\ \text{SEZ}_{ij} &= \text{SZ}_j \end{aligned}$$

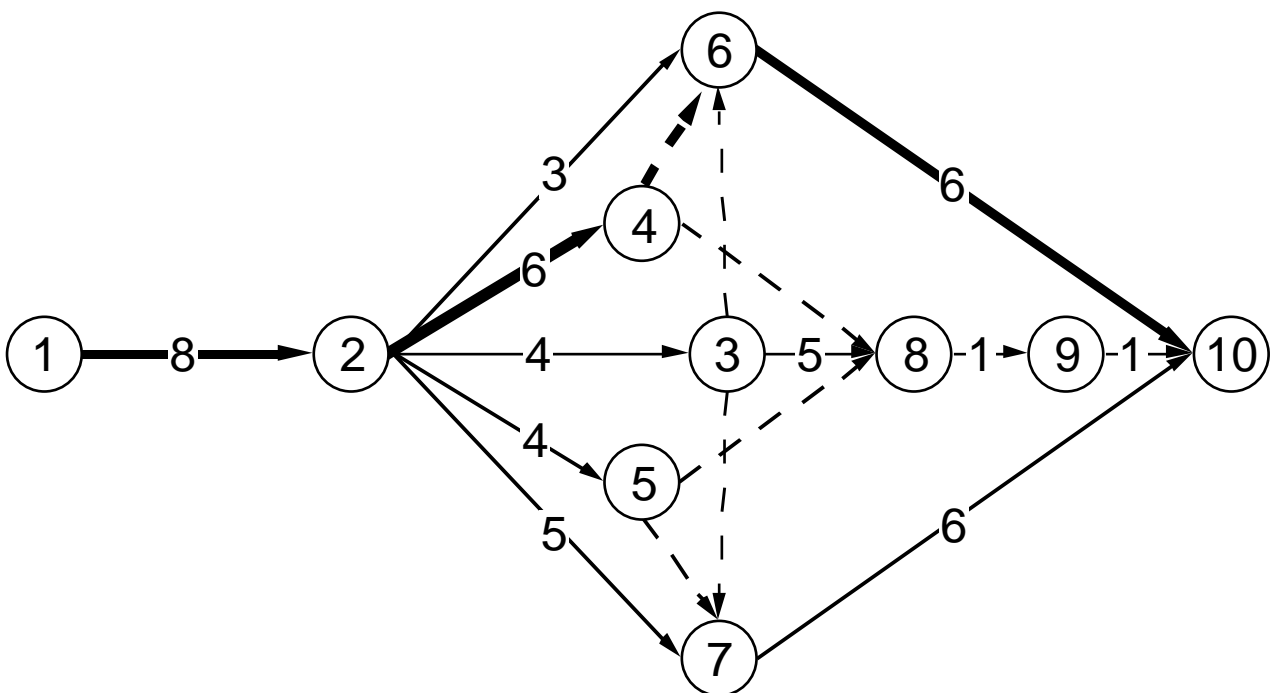
- Pufferzeiten ... (Vorsicht bzgl. Def. FZ_i , SZ_j + Schein-Vs)

Ergebnisse Beispiel:

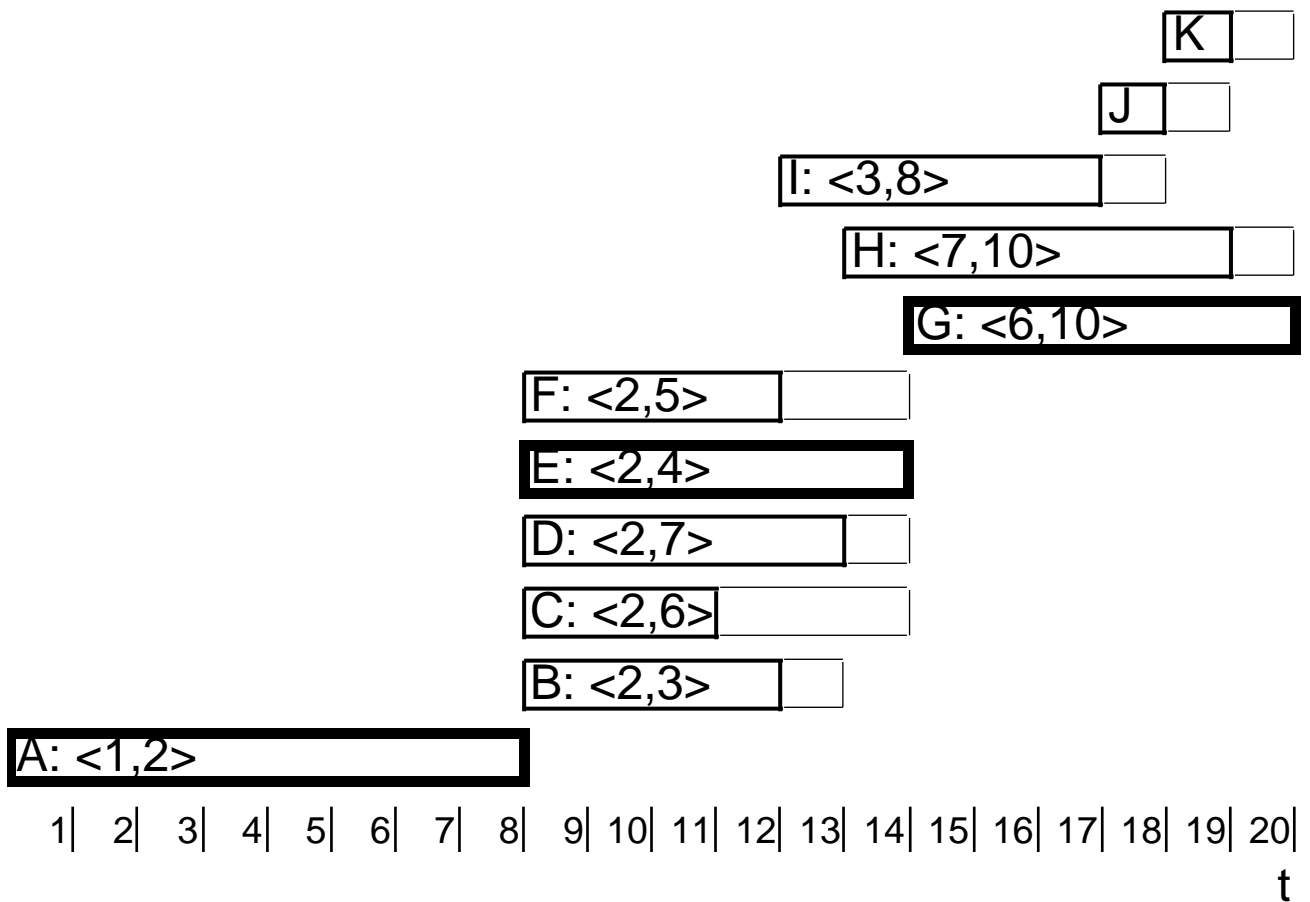
tabellarisch, ohne Vorgabe T:

i	1	2	3	4	5	6	7	8	9	10
FZ_i	0	8	12	14	12	14	13	17	18	20
SZ_i	0	8	13	14	14	14	14	18	19	20
T	20									

Netzplan, mit kritischem Weg



Gantt-Diagramm, mit Puffern



3.4.2 Weitere Netzplantypen

- MPM Vorgangsknotennetze
- PERT Ereignisknotennetze
stochastische Anordnungsbeziehungen,
stochastische Vorgangsdauern
- GERT + Rückkopplungen, bedingte Ausführungen
- ...

hier nicht betrachtet

3.5 Flüsse in Netzwerken

Netzwerke "natürliche" Vorstellung für Probleme, bei denen

- Versorgungs- / Verteilungsflüsse (Dimension: Menge/Zeiteinheit)
- in (für betrachtete Zeitintervalle) konstanter "Stärke"
- über Verbindungen Wege von Quellen zu Senken gelangen
- wo zusätzlich auf Verbindungen Fluß-Beschränkungen nach unten und / oder oben ("Kapazitäten") existieren

Problembereiche sind von der Art

- größtmöglicher / maximaler Fluß
 - kostengünstigster / kostenminimaler Fluß
- speziell gelagerte lineare Optimierungsprobleme

Anwendungen "offensichtlich":

Verkehr, Gas, Öl, Wasser, ..., Daten

3.5.1 Maximale Flüsse

Definition 3.5.01 : Netzwerk mit Kapazitäten

Sei f jedem Pfeil $\langle i, j \rangle$
 eines Digraphen $\langle V, E \rangle$ ohne isolierte Knoten
 eine **Minimalkapazität** $u_{ij} \in \mathbb{R}_+$
 und eine **Maximalkapazität** $o_{ij} \in \mathbb{R}_+ \cup \{ \infty \}$ zugeordnet.
 Der doppelt bewertete Graph
 $N = \langle V, E; u, o \rangle$
 heißt **Netzwerk mit Kapazitäten**

u_{ij} bzw o_{ij} bezeichnen den zulässigen
 Mindest- bzw Höchstwert des Flusses auf Pfeil $\langle i, j \rangle$,
 $u_{ij}=0$ ist ein häufiger Sonderfall,
 $o_{ij}=\infty$ steht für fehlende obere Beschränkung

Definition 3.5.02 : Flüsse in Netzwerken

Seien
 $r \in V, s \in V \setminus \{r\}$ Knoten des Netzes

Eine Abbildung

$$f: E \rightarrow \mathbb{R}_+ \cup \{ \infty \}$$

heißt **Fluß** in N **von** r **nach** s mit **Flußstärke** $F(f) \geq 0$,
 wenn die **Flußbedingung**

$$(3.5.02) \quad \sum_{j \in S(i)} f_{ij} - \sum_{k \in P(i)} f_{ki} = \begin{cases} F & i = r \\ -F & i = s \\ 0 & i \in V \setminus \{r, s\} \end{cases}$$

erfüllt ist. r bzw s heißen **Flußquelle** bzw **Flußsenke**

r / s müssen nicht Quelle / Senke des Netzwerks sein

Erfüllt f die Bedingungen

$$(3.5.03) \quad u_{ij} \geq f_{ij} \geq o_{ij} \quad \langle i,j \rangle \in E$$

dann heißt f **zulässiger Fluß**

$f \mid (f_{ij}=0; \langle i,j \rangle \in E)$ heißt **Nullfluß**

Nullfluß ist zulässig für $(u_{ij}=0; \langle i,j \rangle \in E)$

f^1 heißt **größer** als f^2 , wenn $F(f^1) > F(f^2)$;
ein zulässiger Fluß maximaler Stärke
in N von r nach s heißt **maximaler Fluß**

Maximale Flüsse sind optimale Lösungen des
Maximalflußproblems

$$\max_{u \in N} F \quad (3.5.02), (3.5.03)$$

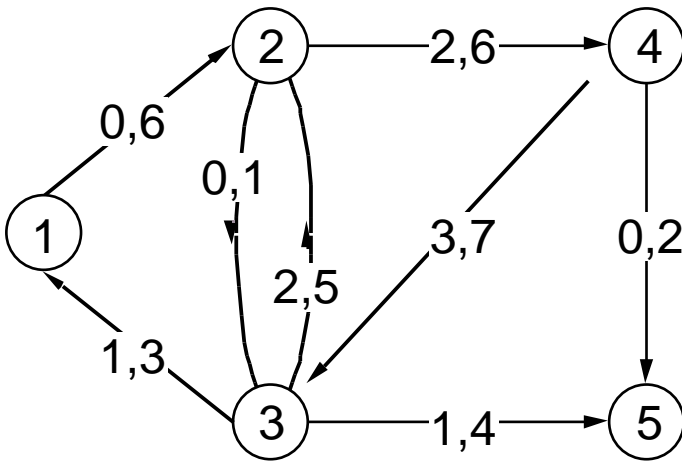
lineares Optimierungsproblem mit
(Entscheidungs-) Variablen $\{f_{ij}; \langle i,j \rangle \in E\}$

Offensichtlich existiert maximaler Fluß

- wenn alle o_{ij} endlich
- und ein zulässiger Fluß existiert

Frage: Gibt es "elegantere" / effizientere
Lösungsverfahren als
"allgemeine" lineare Optimierung ?

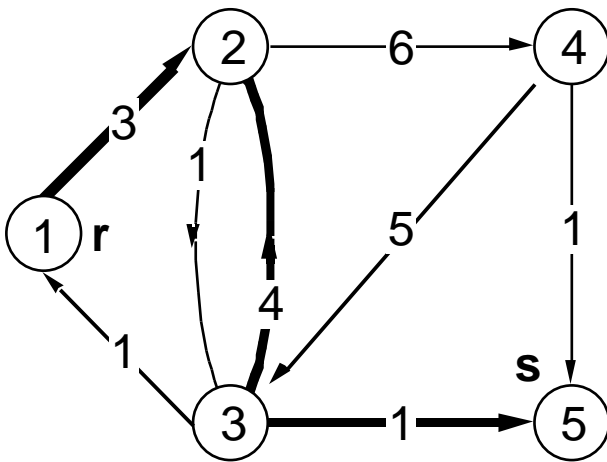
Beispiel



Pfeilbeschriftung:
 u_{ij}, o_{ij}

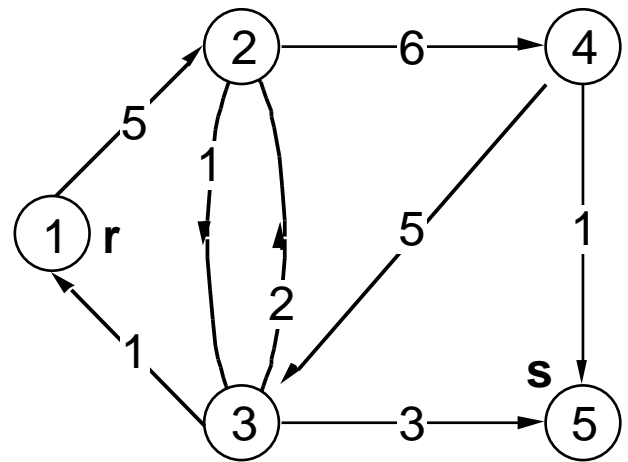
2 zulässige Flüsse

mit $F(\text{links}) < F(\text{rechts})$



links

Pfeilbeschriftung: f_{ij}



rechts

Bemerkung

wegen der summarischen Behandlung von
 Flüssen auf Pfeilen $\langle i,j \rangle$ bzw $\langle j,i \rangle$
 kompensiert sich deren Auswirkung in den Knoten i,j

sei S ein Semiweg
 von (Flußquelle) r
 nach Knoten k r
 mit Orientierung $r \rightarrow k$: **(r,k)-Semiweg**

Pfeil auf S heißt
Vorwärtspfeil, wenn er S -Orientierung hat
Rückwärtspfeil, sonst

(im Beispiel enthält ausgezeichneter Semiweg
 Vorwärtspfeile $\langle 1,2 \rangle$, $\langle 3,5 \rangle$,
 Rückwärtspfeil $\langle 3,2 \rangle$)

für zulässigen Fluß f (in N , von r nach s)
 und (r,k) -Semiweg ist

$$s_{ij} := \begin{cases} u_{ij} - f_{ij} & \langle i,j \rangle \text{ Vorwärtspfeil } S \\ f_{ij} - u_{ij} & \langle i,j \rangle \text{ Rückwärtspfeil } S \end{cases}$$

maximaler Betrag, um den
 Fluß auf Vorwärtspfeil vergrößert
 (Rück-)Fluß auf Rückwärtspfeil verkleinert
 werden kann,
 ohne Zulässigkeitsbedingungen (3.5.03) zu verletzen

(im Beispiel ist $s_{12}=3$, $s_{32}=2$, $s_{35}=3$)

Ist $s_{ij} > 0$ für alle Pfeile $\langle i,j \rangle$ von S

mit $s := \min_{\langle i,j \rangle \in S} s_{ij}$

heißt S **flußvergrößernder Semiweg** für f
 und Fluß $r \rightarrow k$ kann um s vergrößert werden
 ohne Flußbedingungen (3.5.02) zu verletzen

(im Beispiel ist $s=2$, und Fluß gemäß f^{rechts} erzielbar)

Satz 3.5.04 : Maximaler Fluß

Ein zulässiger Fluß f von r nach s in einem Netzwerk mit Kapazitäten ist genau dann maximal, wenn für f kein flußvergrößernder (r,s) -Semiweg existiert.

"offensichtlich"

Definition 3.5.05 : Schnitte in Digraphen

Sei $G = \langle V, E \rangle$ ein Digraph
 $A \subseteq V, B \subseteq V$ eine Zerlegung der Kantenmenge
 $V = A \cup B, A \cap B = \emptyset, A, B$

Menge der aus A nach B führenden Pfeile
 $C_{\langle A, B \rangle}$

heißt $(A \text{ von } B)$ **trennender Schnitt** in G

Menge der aus B nach A führenden Pfeile
 $C_{\langle B, A \rangle}$

heißt **konträrer Schnitt** zu $C_{\langle A, B \rangle}$

$C_{\langle A, B \rangle} \cup C_{\langle B, A \rangle}$

heißt **(A,B)-Schnitt**

bzw, für $i \in A, j \in B$

auch **(i,j)-Schnitt**

Schnitt $C\langle A, B \rangle$ eines Netzwerks N mit Kapazitäten lassen sich zuordnen

Minimal- und Maximalkapazität U und O
sowie

Kapazität K
gemäß:

$$U(C\langle A, B \rangle) := \sum_{\langle i, j \rangle \in C\langle A, B \rangle} u_{ij}$$

$$O(C\langle A, B \rangle) := \sum_{\langle i, j \rangle \in C\langle A, B \rangle} o_{ij}$$

$$K(C\langle A, B \rangle) := O(C\langle A, B \rangle) - U(C\langle B, A \rangle)$$

K ist Differenz größter Flußmenge $A \rightarrow B$
und kleinster Rückflußmenge $B \rightarrow A$
ist maximale Netto-Flußmenge $A \rightarrow B$

Definition 3.5.06 : Minimalschnitte

Ein Schnitt, der unter allen (r, s) -Schnitten eines Netzwerks mit Kapazitäten die minimale Kapazität aufweist heißt **minimaler (r, s) -Schnitt**

Satz 3.5.07 : Maximalfluß-Minimalschnitt-Theorem

Die Stärke eines maximalen Flusses
von r nach s in einem Netzwerk N mit Kapazitäten
ist gleich der Kapazität eines minimalen
 (r, s) -Schnittes in N

Üblicherweise gezeigt über Nachweis,
daß hier duale lineare Optimierungsprobleme vorliegen

Verständlicher:

Jeder (r,s) Schnitt beruht
auf (A,B) -Schnitt mit $r \in A$ und $s \in B$

gemäß Flußbedingungen (3.5.02) gilt

- für jeden zulässigen Fluß
- für jedes derartige A

$$\sum_{i \in A} \left(\sum_{j \in S(i)} f_{ij} - \sum_{k \in P(i)} f_{ki} \right) = F(f)$$

sowie

$$\begin{aligned} & \sum_{i \in A} \left(\sum_{j \in S(i)} f_{ij} - \sum_{k \in P(i)} f_{ki} \right) \\ &= \sum_{i \in A} \left(\sum_{j \in S(i) \cap A} f_{ij} - \sum_{k \in P(i) \cap A} f_{ki} \right) \\ & \quad + \sum_{i \in A} \left(\sum_{j \in S(i) \cap B} f_{ij} - \sum_{k \in P(i) \cap B} f_{ki} \right) \\ &= 0 + F(C \langle A, B \rangle) \end{aligned}$$

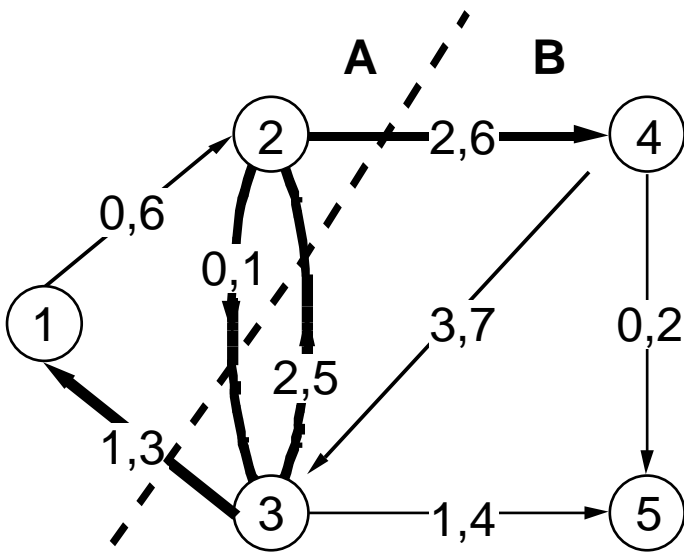
wo $F(C \langle A, B \rangle)$ Netto-Flußmenge $A \rightarrow B$, bei vorliegendem f
mit Def von $K(C \langle A, B \rangle)$ ist

$$F(f) = F(C \langle A, B \rangle) + K(C \langle A, B \rangle) \quad A: r \in A, s \in A$$

$$\min_{C \langle A, B \rangle: r \in A, s \in A} K(C \langle A, B \rangle)$$

Satz

Im Beispiel
mit $A=\{1,2\}$ $B=\{3,4,5\}$



Pfeilbeschriftung:
 u_{ij}, o_{ij}

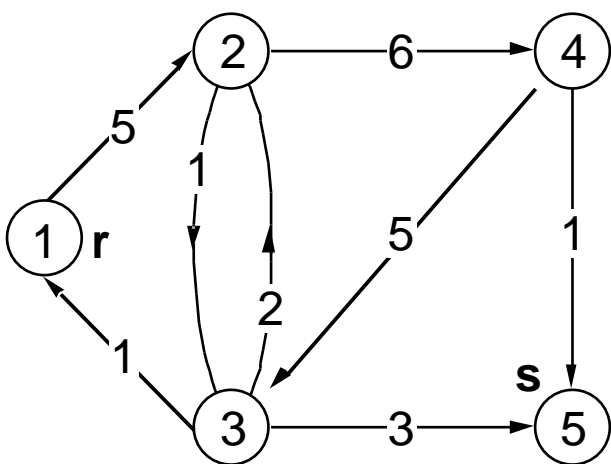
ist (A,B)-Schnitt
 $C\langle A,B \rangle = \{ \langle 2,3 \rangle, \langle 2,4 \rangle \}$

hierzu konträrer Schnitt
 $C\langle B,A \rangle = \{ \langle 3,1 \rangle, \langle 3,2 \rangle \}$

Kapazität
 $K(C\langle A,B \rangle) = 7 - 3 = 4$

minimaler (1,5)-Schnitt
stark ausgezeichnet

und Fluß r s Beispiel "rechts" maximal:



rechts

Ableitung maximaler Flüsse in Netzwerken mit Kapazitäten

- lange Reihe von Algorithmen (bis in jüngste Zeit)
- mit Basisidee ("Urahn") Ford/Fulkerson (1962)

Algorithmus 3.5.08 : Maximale Flüsse - Ford / Fulkerson Skizze

Ausgangspunkt:

Netzwerk mit Kapazitäten
zulässiger Fluß

$N = \langle V, E; u, o \rangle$
 $f: E \rightarrow \mathbb{R}_+ \cup \{0\}$

Idee:

Konstruktion Folge	zulässiger Flüsse wachsender Stärke
mittels Konstruktion	flußvergrößernder (r,s)-Semiwege

Ablauf:

iterativ, mit 2 Phasen je Iteration

- Markierungsphase:
Markierung aller Knoten $k \neq r$,
falls flußvergrößernder (r,k)-Semiweg existent
(zusätzlicher positiver Fluß $r \rightarrow k$ möglich)

falls nicht, maximaler Fluß F_{\max} gefunden
- Flußvergrößerungsphase
falls flußvergrößernder (r,s)-Semiweg existent
entsprechende Flußvergrößerung
(zul. Fluß $f \rightarrow$ zul. Fluß f' , mit $F(f') > F(f)$)

Zu Marken und Markierungsphase

- Flußquelle initial markiert mit $(+, \infty)$
Bedeutung: aus r kann beliebig Fluß bezogen werden
- Knoten $i \in r$ initial unmarkiert,
unmarkierter Knoten wird markiert
bei "Überprüfung" markierten Nachbars,
Markierung gibt Anlaß zur "Überprüfung"
Markierung unmarkierter Nachbarn
- Marken für $i \in r$ haben Form $(+l, f_{ij})$
oder $(-l, f_{ij})$

Bedeutung "+": i wurde bei Überprüfung von $l \in P(i)$
"vorwärts" markiert
mit $l_{ij} := \min(l, o_{ij} - f_{ij})$
: bisheriger "Vorwärts"-Fluß f_{ij}
kann um l_{ij} vergrößert werden,
ohne Flußbedingungen zu verletzen

Bedeutung "-": i wurde bei Überprüfung von $l \in S(i)$
"rückwärts" markiert
mit $l_{ij} := \min(l, f_{ij} - u_{ij})$
: bisheriger "Rückwärts"-Fluß f_{ij}
kann um l_{ij} verkleinert werden,
ohne Flußbedingungen zu verletzen

-Werte = 0 führen **nicht** zur Markierung

- Markierungen tabellarisch

Knoten	Markierungsbedingung	Marke	-Wert
j	j S(i) $f_{ij} < o_{ij}$	(+i, j)	$j = \min(i, o_{ij} - f_{ij})$
k	k P(i) $f_{ki} > u_{ij}$	(-i, k)	$k = \min(i, f_{ki} - u_{ki})$

- Überprüfungs-/Markierungsreihenfolge (z.B.) BFS
- Markierungsprozeß bricht ab, wenn
entweder Flußsenke markiert
Flußvergrößerungsphase
oder Flußsenke nicht markiert
+ kein weiterer Knoten markierbar
maximaler Fluß entdeckt
(+ simultan minimaler Schnitt,
hier nicht diskutiert)

Zur Flußvergrößerungsphase

Senke s markiert,
genau 1 Vorgänger + (Vorwärts-)Fluß-Erhöhung
bzw. (Rückwärts-)Fluß-Erniedrigung
dieser genau 1 Vorgänger +
...

Konstruktion (r,s)-Semiweg rückwärts (von s aus)
mit Angleichung der Pfeilflüsse

in Knoten s Markierung (+/-q, s), $q := s$

- bei Markierung (+q, s) $f_{qs} := f_{qs} +$
 - bei Markierung (-q, s) $f_{sq} := f_{sq} -$
- in Knoten k (auf Rückweg) $k = \text{pred } s, \dots, r$
- bei Markierung (+q, k) $f_{qk} := f_{qk} +$
 - bei Markierung (-q, k) $f_{kq} := f_{kq} -$

Neuer Fluß $F(f') = F(f) +$

Algorithmus 3.5.08 startet mit zulässigem Fluß,

- für $u=0$ (häufiger Fall)
ist Nullfluß offensichtlich zulässiger Initialfluß
- für $u \neq 0$ Vorphase erforderlich (hier nicht diskutiert)

Falls u, o, f^{init} ganzzahlig, beschränkt

zulässiges f maximales f
welches Algorithmus findet (ganzzahlige Erhöhungen)

(bei rationalen Größen "auch",
bei irrationalen Größen schwierigeres Problem,
hier nicht diskutiert)

Aufwand

Ist F_{max} maximaler Fluß,

dann (von $F=0$) aus

maximal F_{max} Schritte im Algorithmus

wo F_{max} abschätzbar

$F_{max} = \min(\text{maximaler Nettoausfluß } r, \text{ maximaler Nettoeinfluß } s)$

Je Schritt

- Markierungsphase: maximal m Pfeile zu prüfen
- Flußverg'sphase: maximal n Knoten berührt

Algorithmus hat Zeitkomplexität

$O(F_{max}(m+n))$

Bessere bekannt (hier nicht diskutiert)

3.5.2 Kostenminimale Flüsse

Definition 3.5.09 : Netzwerk mit Kapazitäten und Kosten

Seien in einem Netzwerk mit Kapazitäten

$$N = \langle V, E; u, o \rangle$$

jedem Pfeil $\langle i, j \rangle \in E$ zusätzlich **Kosten** $c_{ij} \in \mathbb{R}_+$ zugeordnet.

Der dreifach bewertete Graph

$$N = \langle V, E; u, o, c \rangle$$

heißt **Netzwerk mit Kapazitäten und Kosten**

c_{ij} bezeichnet dabei die

Kosten einer Flußeinheit auf Pfeil $\langle i, j \rangle$

Bei zulässigem Fluß f der Stärke $F^* := F(f)$

heißt Fluß dieser Stärke, der Kosten $\sum_{\langle i, j \rangle \in E} c_{ij} f_{ij}$

minimiert, **kostenminimaler Fluß der Stärke F^***

(beachte: F^* ist fest, f ist gesucht)

f ist Lösung des linearen Optimierungsmodells Typ "KFS"

$$\begin{array}{ll}
 \text{(3.5.10)} & \min \sum_{\langle i, j \rangle \in E} c_{ij} f_{ij} \\
 \text{(KFS)} & \text{udN} \quad \sum_{j \in S(i)} f_{ij} - \sum_{k \in P(i)} f_{ki} = \begin{cases} F^* & i = r \\ -F^* & i = s \\ 0 & i \in V \setminus \{r, s\} \end{cases} \\
 & \sum_{\langle i, j \rangle \in E} u_{ij} f_{ij} \leq \sum_{\langle i, j \rangle \in E} o_{ij}
 \end{array}$$

Bestimmung kürzester Wege

+ Maximalflußproblem

sind Spezialfälle von KFS

KFS ist allgemeinstes der bisher betrachteten

Optimierungsmodelle in Netzen

Kürzeste Wege

$N = \langle V, E; c \rangle$ bewerteter Digraph
 ohne Zyklen negativer Länge
 $r \in V, s \in R(r) \setminus r$ Start- und Zielknoten

Ordne jedem Weg W in N von r nach s Fluß zu gemäß

$$f_{ij} := \begin{cases} 1 & \langle i, j \rangle \text{ auf } W \\ 0 & \text{sonst} \end{cases} \quad \langle i, j \rangle \in E$$

Mit $u_{ij} := 0, o_{ij} := \langle i, j \rangle \in E$
 ist jeder kürzeste Weg Lösung von KFS mit $F^* = 1$

Maximalfluß

$N = \langle V, E; u, o \rangle$ Netzwerk mit Kapazitäten
 ohne Pfeil $\langle s, r \rangle$
 (falls doch: Zwischenknoten
 einführen)

Setze $c_{ij} := 0 \quad \langle i, j \rangle \in E$
 +
 füge Pfeil $\langle s, r \rangle$ zu
 mit $u_{sr} := 0, o_{sr} := \langle s, r \rangle, c_{sr} := -1$

KFS Problem für
 Netz N' mit Kapazitäten + Kosten $(E' = E \setminus \{\langle s, r \rangle\})$
 und Flußstärke $F^* := 0$

Lösung für N' maximiert "Rückfluß" f_{sr} (s. Zielfunktion)
 = Gesamtflußstärke F^* in N

Spezielle Lösungsverfahren KFS existieren

zB Busacker/Gowen

incl. (neben kostenminimalem Fluß)

Bestimmung kostenminimalen maximalen Flusses

- unter Zuhilfenahme primal / dualer Prinzipien
- auf Basis "Inkrementnetzwerk"
- inkrementell (ähnlich Ford/Fulkerson)

Idee Inkrementnetzwerk

(auch für Ford/Fulkerson, statt "Markierung", einsetzbar)

$N = \langle V, E; u, o \rangle$ Netzwerk mit Kapazitäten,
antisymmetrisch (keine Pfeilpaare)
 $f: E \rightarrow \mathbf{R}_+$ zulässiger Fluß

$N'(f) := \langle V, E'(f); o'(f) \rangle$ Inkrementnetzwerk,
ohne Minimalkapazitäten
 $E'(f) := E^+(f) \cup E^-(f) \cup E$ enthält Kandidaten für
- Vorwärtspfeile ($f_{ij} < o_{ij}$)
- Rückwärtspfeile ($f_{ij} > u_{ij}$)
mit inkrementellen Kapazitäten

$$E^+(f) := \{ \langle i, j \rangle \in E \mid f_{ij} < o_{ij} \} \quad \text{und} \quad o'_{ij} := o_{ij} - f_{ij}$$

$$E^-(f) := \{ \langle i, j \rangle \in E \mid f_{ij} > u_{ij} \} \quad \text{und} \quad o'_{ij} := f_{ij} - u_{ij}$$

Weg in N' von r nach s

entspricht flußvergrößerndem (r, s) -Semiweg in N

ist in N' $s \in R(r)$, dann ist f maximal

3.6 Weitere spezielle Probleme (Skizze)

Auf Graphen / Netzen beträchtliche Menge weiterer

- praktisch relevanter Familien von (hier: linearen) Optimierungsproblemen
- für deren speziellen Aufgabentyp jeweils spezielle (effizientere) Lösungstechniken entwickelt wurden

Hier kurz Nennung / Charakterisierung von "Typen":

Matching-Probleme

$G = [V, E; c]$ (bewerteter) Graph

$X \subseteq E$ **matching**, wenn kein Kantenpaar in X mit selbem Knoten inzident

$i \in V$ **überdeckt** durch matching X , wenn X mit i inzidente Kante enthält

X **vollständig** (auch: **perfekt**), wenn X alle $i \in V$ überdeckt

X **maximal**, wenn kein matching X' mit $|X'| > |X|$ existiert

$X_p(G)$ Menge der perfekten matchings in G

Für bewerteten Graphen mit reellwertigem c heißt

$$c(X) := \sum_{[i,j] \in X} c_{ij}$$

Länge (auch: **Gewicht**) des matchings x in G und

$$\min_{X \in X_p(G)} c(X)$$

Summen-Matching-Problem,
und dessen Lösung **minimales Summen-Matching** in G

Lösungsverfahren i.allg.Fall kompliziert (aber: $O(n^3)$)
einfacher in bipartiten Graphen,

wo Graph $G = [V,E]$ **bipartit**, wenn
 V derart (in R,S) zerlegbar,
daß alle Kanten Form $[i \in R, j \in S]$

Zuordnungsproblem

Probleme der "besten" Zuordnung von

- Bewerbern zu Stellen (vgl. Abschn. 1)
- Jobs zu Maschinen
- Fahrzeugen zu Transportaufträgen
- ...

heißen Zuordnungsprobleme, erfaßt durch

$R := \{A_1, \dots, A_n\}$ "Zuzuordnende"
 $S := \{T_1, \dots, T_n\}$ "Zuordnungsstellen"
 $|R| = |S|$ ist zwar Spezialfall, aber oBdA
 $c_{ij} :=$ Kosten der Zuordnung A_i zu T_j (gerichtet: "Pfeil")

bipartites bewertetes Netz

$N = [V,E;c]$ mit $V = R \cup S$
 $E =$ Pfeile (potentieller) Zuordnungen
 $c =$ Kosten (potentieller) Zuordnungen

Jede vollständige Zuordnung "ist" ein matching X

Mit (binären, zweiwertigen) Entscheidungsvariablen

$$x_{ij} := \begin{cases} 1 & \langle i,j \rangle \in X \\ 0 & \langle i,j \rangle \notin X \end{cases} \quad \langle i,j \rangle \in E$$

ist "beste" Zuordnung definiert als

$$\begin{array}{ll}
 \min & \sum_{\langle i,j \rangle \in E} c_{ij} x_{ij} \\
 \text{udN} & \sum_{j \in S(i)} x_{ij} = 1 \quad i \in R \\
 & \sum_{i \in P(j)} x_{ij} = 1 \quad j \in S \\
 & x_{ij} \in \{0,1\} \quad \langle i,j \rangle \in E
 \end{array}$$

Summen-Matching-Problem in bipartiten Graphen

aber mit speziellen (angepaßten) Lösungsverfahren, zB
 "Ungarische Methode" "Glover-Klingsman-Algorithmus"

Umladeproblem

Probleme des "besten" Transports
 eines Gutes (Wasser, Energie, Daten, ...)

- von Angebotsorten
- über Umladeorte
- zu Nachfrageorten

heißen Umladeprobleme, erfaßbar durch

Netzwerk N mit Kapazitäten und Kosten

- alle Angebots-, Umlade-, Nachfrageorte
 als Knoten $\{1, \dots, n\} =: V$
- alle direkten Transportmöglichkeiten $i \rightarrow j$
 als Pfeile $\{\langle i,j \rangle\} =: E$

(vorausgesetzt wird: N schwach zusammenhängend)

- mit Maximalkapazitäten o_{ij} in "Transport-Einheiten"
- mit Transportkosten c_{ij} je TE

und ("günstigst") zu ermitteln

- Transportmengen x_{ij} in TEs

Lösung soll sicher

$$0 \leq x_{ij} \leq o_{ij}$$

befriedigen

sowie bzgl. abtransportierter Nettomengen $=: a_i \quad (i \in V)$

mit $a_i > 0$ für Angebotsknoten

$a_i = 0$ für Umladeknoten

$a_i < 0$ für Nachfrageknoten

wo $a_i = 0$ "nichts fällt vom Himmel"

die Flußbedingungen

$$\sum_{j \in S(i)} x_{ij} - \sum_{k \in P(i)} x_{ki} = a_i \quad i \in V$$

Das Optimierungsproblem

$$\min \sum_{\langle i,j \rangle \in E} c_{ij} x_{ij}$$

$$\text{udN} \quad \sum_{j \in S(i)} x_{ij} - \sum_{k \in P(i)} x_{ki} = a_i \quad i \in V$$

$$0 \leq x_{ij} \leq o_{ij} \quad \langle i,j \rangle \in E$$

heißt **Umladeproblem**

KFS-Problem (kostenminimaler Fluß) hat im Vergleich nur 1 Quelle, 1 Senke

ist spezielles Umladeproblem

Kürzeste Wege, Maximalfluß sind spezielle KFS

sind spezielle Umladeprobleme

Zuordnungsproblem ist Spezialfall des Transportproblems (= Umladeproblem ohne Umladeknoten, vgl. später)

ist spezielles Umladeproblem

Umladeproblem eines der allgemeinsten (linearen) Optimierungsprobleme in Netzen

aus diesem Grunde hierfür speziell adaptierte Version
des Simplex-Verfahrens entwickelt:
"Netzwerk-Simplexmethode"

Wie schon erwähnt,
Umladeproblem ohne Umladeknoten
ist "traditionelles" **Transportproblem**
(m Produzenten, n Verbraucher)

$$\begin{array}{ll}
 \min & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\
 \text{udN} & \sum_{j=1}^n x_{ij} = a_i \quad i \text{ "Produzenten"} \\
 & \sum_{i=1}^m x_{ij} = b_j \quad j \text{ "Verbraucher"} \\
 & x_{ij} \geq 0 \quad i,j \text{ "Produzenten" "Verbraucher"}
 \end{array}$$

($m=n$, $a_i=b_j=1$ Zuordnungsproblem)

Lösungsverfahren arbeiten mit / auf
(vergleichsweise kleinem)
 $m \times n$ -"Tableau" der x_{ij}

Abbruch der Skizze

bei (späteren, beruflichen) "Nöten":
es lohnt sich, die Spezialliteratur heranzuziehen
(vor "Neuerfindung" !)