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∩V =
welche je 2 Knoten verbinden:
Kante e = {i,j} e E; i,j V(hier: e = [i,j])
ist zweielementige Untermenge von V
(auch: Inzidenzabbildung ordnet
jedem e E genau 2 Elemente i,j V zu)

für e = [i,j] heißt e **inzident** mit i und j
heißen 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** δ(i)

Knoten ohne Nachbarn heißen **isoliert**

G' := [V',E'] heißt
- **Teilgraph** von G = [V,E],
  wenn V' V und E' E
- **echter Teilgraph** von G = [V,E],
  wenn G' Teilgraph von G
  sowie V' V oder E' E
- durch V' **induzierter Teilgraph** von G = [V,E],
  wenn V' V und E'={ [i,j] | i,j V'; [i,j] E }

Digraph (gerichteter Graph):

$G^{\rightarrow}$:= <V,E>
V Knotenmenge, E Menge gerichteter Kanten (**Pfeile**)

für e = <i,j> heißt i **Anfangs**-, j **Endknoten** von e
heißt e positiv **inzident** mit i,
negativ **inzident** mit j
heißen i und j **adjazent**

$G^{\rightarrow}$ heißt **vollständig**,
wenn mit i,j auch <i,j> und <j,i> enthalten

$G^{\rightarrow}$ heißt **symmetrisch**,
wenn mit <i,j> auch <j,i> enthalten

für e = <i,j> heißi i **Vorgänger** von j
und j **Nachfolger** von i

P(i) bezeichne Menge von Vorgängern eines Knoten i,
Anzahl von Vorgängern heißt **Eingangsgrad** $\delta^-(i)$

S(i) bezeichne Menge von Nachfolgern eines Knoten i,
Anzahl von Nachfolgern heißt **Ausgangsgrad** $\delta^+(i)$

Knoten ohne Vorgänger heißt **Quelle**
Knoten ohne Nachfolger heißt **Senke**

Knoten ohne Vorgänger und Nachfolger heißen **isoliert**

(echte / induzierte) **Teil(di)graphen** von $G^{\rightarrow}$ analog zu G

$G^{\rightarrow}$ heißt **topologisch sortiert**, wenn (uU Umnumerier'g)
j P(i) j<i bzw. i S(j) j<i

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

- **Inzidenzmatrix**

$\mathbf{H}_{n,m}(G)$	$\mathbf{H}_{n,m}(G^{\rightarrow})$
h_{ij} =1, e_i inzident mit v_j	h_{ij} =+1, e_i pos. inz. mit v_j
h_{ij} =0, sonst	h_{ij} = -1, e_i neg. inz. mit v_j
	h_{ij} =0, sonst

- **Adjazenzmatrix**

$\mathbf{U}_{n,n}(G)$	$\mathbf{U}_{n,n}(G^{\rightarrow})$
u_{ij} =1, $[v_i,v_j]$ E	u_{ij} =1, $<v_i,v_j>$ E
u_{ij} =0, sonst	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
$i_0,[i_0,i_1],i_1,\ldots,i_{r-1},[i_{r-1},i_r],i_r =: [i_0,i_1,\ldots,i_r]$

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

i_0,i_r:	Endknoten	
$i_0 \neq i_r$:	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 $G^{\rightarrow}$ heißt **schwach zusammenhängend**,
wenn alle Knotenpaare verbunden

Digraph $G^{\rightarrow}$ heißt **stark zusammenhängend**,
wenn jeder Knoten von jedem Knoten erreichbar

Schwache (starke) Zusammenhangskomponente von $G^{\rightarrow}$
ist maximaler schwach (stark) zusammenhängender
Teilgraph von $G^{\rightarrow}$

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 $\delta^-(i)=1$, $i \neq r$
(genau 1 Vorgänger)

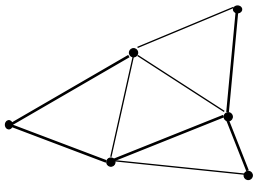
…, **Blatt**, **Ast**, **Vater/Mutter/Elter**, **Sohn/Tochter/Kind**,
(Knoten-)Tiefe, …

Binärbaum: Baum mit ≤ 2 Kindern je Knoten

balancierter Binärbaum der Tiefe t:
Binärbaum mit genau 2^{t-1} Knoten der Tiele 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

Gerüst ist Baum (spanning tree, Spannbaum)

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

gerichtetes Gerüst eines Digraphen $G^{\rightarrow}$:
- Teilgraph von $G^{\rightarrow}$
- enthält alle Knoten von $G^{\rightarrow}$
- stellt gerichteten Baum dar

Bewertete (kanten-attributierte) **Graphen**

Graph	bzw.	Digraph
G = [V,E]		$G^{\rightarrow}$ = <V,E>
+ "Bewertung" aller Kanten		aller Pfeile
e E		e E
mit Elementen aus Menge M		
c: E → M		

heißt

bewerteter Graph	bewerteter Digraph
G := [V,E;c]	$G^{\rightarrow}$:= <V,E;c>

c heißt Bewertungs- / Gewichtungs-Funktion, wo

Bewertung einer Kante	Bewertung eines Pfeils
c(e) = c[i,j]	c(e) = c<i,j>

Netzwerk
bewerteter Digraph ohne isolierte Knoten

für M = **R** {∞} kann der

Kantenfolge	Pfeilfolge
$F = [i_0,i_1,\ldots,i_r]$	$F = <i_0,i_1,\ldots,i_r>$

eine **Länge** zugeordnet werden gemäß

$$c(F) := \sum_{k=1}^{r} c_{i_{k-1}i_k}$$

in diesem Zusammenhang gelte

∞ **R**

$a < \infty$ a **R**

$a + \infty = \infty + a = \infty$ a **R**

$a \cdot \infty = \infty \cdot a = \begin{cases} 0 & a = 0 \\ \infty & \text{sonst} \end{cases}$ a $\mathbf{R}_+ \cup \{\infty\}$

offene Kantenfolge	offene Pfeilfolge
mit Endknoten i,j	mit Anf.kn. i + Endkn. j

mit kleinster / größter Länge unter allen Folgen i nach j heißt **kürzeste** / **längste**

Kantenfolge	Pfeilfolge

analog: kürzeste / längste

Ketten	Wege

Länge kürzester

Kantenfolge (Kette)	Pfeilfolge (Weg)

von i nach j heißt **Entfernung**

d[i,j]	d<i,j>

Enthält

G keine Kreise	$G^{\rightarrow}$ 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

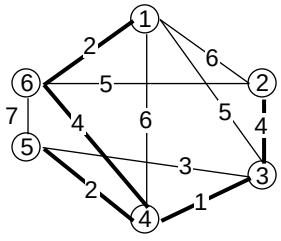
Versorgungspunkte
→ Knoten eines Graphen

potentielle Verbindungsstrecken → Kanten des Graphen, bewertet mit Kosten	Installation, Betrieb, …

gesucht ist kostengünstigstes Netz
→ **Minimalgerüst** des Graphen:

- Gerüst	alle erreicht
- mit minimaler Länge	Summe Streckenkosten

Beispiel



Minimalgerüst

Bestimmung Minimalgerüst in bewertetem Graphen

G = [V,E;c]
- zusammenhängend
- n:=|V| Knoten, n≥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

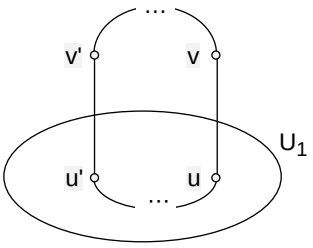
Seien
G := [V,E] : Graph
$[U_i,T_i]$; i=1,…,k : Bäume in G,
$U_i \cap U_j =$ disjunkt
$\{ [U_i,T_i] \mid i=1,\ldots,k \}$: Wald in G,
$\cap (U_i) = V$ V aufspannend
sowie
[v,u] kürzeste aller Kanten
mit (lediglich) 1 Endpunkt u in U_1

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

Beweis per Widerspruch:

Sei [V,F] Gerüst mit F T
und [v,u] 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 U_1
Kante [v',u'] ≠ [v,u] auf Kreis, u' U_1, v' $V-U_1$



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 = <V,E;c> bewerteter Digraph

mit V = {1,…,n}
|E| = m
c: E → **R** c_{ij} := c<i,j>

und N ohne Zyklen negativer Länge

Gesucht seien
(alternative, aber überlappende Fragestellungen)

(a) für festen **Startknoten** r und festen **Zielknoten** s
Entfernung d<r,s> (kürzester Weg r nach s)

(b) für festen **Startknoten** r und alle Knoten j V
Entfernungen d<r,j> (kürzeste Wege r nach j)

(c) für alle Knoten i,j V
Entfernungen d<i,j> (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 "zugeordeten"
symmetrischen bewerteten Digraphen
(Kante [i,j] → Pfeilpaar <i,j> und <j,i>,
c<i,j> = c<j,i> = c[i,j])

Fragestellung (b) ("r zu allen")

definiere

$$d_j := \begin{cases} 0 & j = r \\ d<r,j> & j \quad R(r), \neq r \quad j \quad V \\ \infty & j \quad R(r) \end{cases}$$

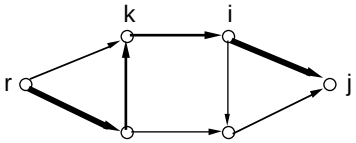
W_j := "entsprechender" (kürzester) Weg

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"

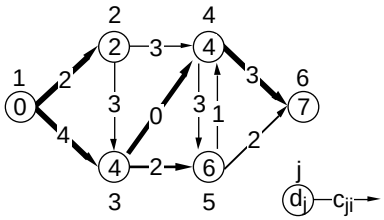
(3.3.01) $d_j = \min_{P(j)} \left(d_i + c_{ij} \right)$ j R(r), ≠ r

In Anwendung:
Algorithmen kürzeste Wege, **Baumalgorithmen**

- konstruieren für alle Knoten j R(r)\{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 R(r)\{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

Baumalgorihtmen 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;

{Initalisierung Algorithmus:}
dr:=0; pr:=r; Q:={r};
di:=∞; pi:=r; i=1,…,n; i≠r

{Iteration:}
while Q≠
  extract_any i  Q;     {Entnahme 1 mark. Knoten}
  for all j  succ(i) | dj > di + cij
    dj:=di+cij; pj:=i; Q:=Q {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

{: Wegebaum 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 **queue**s 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 $\bar{\delta}(.)$

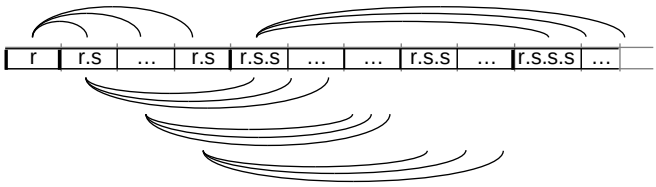
Algorithmus 3.3.02 : Kürzeste Wege - LC à la Ford

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;

{Initalisierung 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

{: 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 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;

{Initalisierung 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

{: Wegebaum 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 uU j in Q auf mit $d_j = d_k + c_{kj} \geq d_k$
wegen $c_{..} \geq 0$
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

$$d_{ij} := \begin{cases} 0 & j = i \\ d<i,j> & j \in R(i), \neq i \quad i,j \in V \\ \infty & \text{sonst} \end{cases}$$

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

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

$D := (d_{ij} \mid i,j=1,\ldots,n)$ "Entfernungsmatrix"
$P := (p_{ij} \mid i,j=1,\ldots,n)$ "Wegematrix"

Idee "**Tripel-Algorithmus**" nach Floyd / Warshall:
für i,j $\in$ V={1,…,n}, j≠i,
k $\in$ V \ {i,j}
sei $d_{ij}^{(k-1)}$ Länge kürzesten Weges i nach j, k≥2,
der (außer i,j) allenfalls Knoten ≤ k-1 enthält
$d_{ij}^{(k-1)} := \infty$ falls kein solcher Weg existiert
für i=j $d_{ij}^{(k-1)} := 0$

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

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

- und ist k nicht enthalten,
  dann $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
- und ist k enthalten,
  dann $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Es gilt also:

$d_{ij}^{(0)} = c_{ij}$ i,j = 1,…,n

$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ i,j = 1,…,n; k ≥ 1
"Tripeloperation"

(wo, wie üblich, $c_{ii}=0$
$c_{ij}=\infty$, falls <i,j> $\notin$ E)

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;

{Initalisierung 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,…,n zu allen

Dazu:

Lemma 3.3.05 : Kürzeste Wege (Hilfssatz)

Werden in einem Netzwerk oZnL N = <V,E;c>
die Kantenbewertungen geändert gemäß
$c_{ij}' := c_{ij} + b_i - b_j$ b_i, b_j beliebig, reell
dann sind die kürzesten Wege des Netzwerks
N' = <V,E;c'>
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 $b_i := d_{ri}$ $i \in V$
wegen $d_{rj} = d_{ri} + c_{ij}$ ist $c_{ij}'=0$ für <i,j> 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 <i,j> 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

zur Formulierung Anordnungsbeziehung(en)
mehrere Möglichkeiten:
- Ende-Start: B beginnbar, wenn A abgeschlossen
- Start-Start: B beginnbar, sobald A begonnen
- Start-Ende: B abschließbar, sobald A begonnen
- Ende-Ende: B abschließbar, wenn A abgeschlossen

konkrete Netzplantechniken nutzen
jeweils genau 1 der Möglichkeiten

Beispiel: Projekt "Brückenbau" →
mit Aufstellung **Vorgangsliste**
unter Einsatz der Ende-Start Option

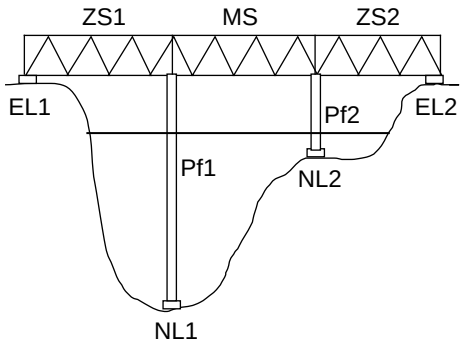
---

daraus: Netzplan als

- **Vorgangspfeilnetz** → CPM
  Critical Path Method
  Vorgang → Pfeil, Dauer → Bewertung Pfeil,
  unmittelbare Folge → Pfeile
  aneinander geheftet,
  Ereignis → Knoten

- **Vorgangsknotennetz** → MPM
  Metra Potential Method
  Vorgang → Knoten, unmittelbare Folge → Pfeil
  min, max Abstand → Bewertung Pfeil,
  "Ereignis" nicht benutzt

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

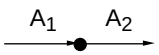
3.4.1 CPM-Netzpläne

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

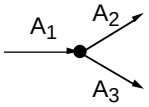
Startvorgang A: kein anderer Vorgang vor A beendet

Zielvorgang A: kein anderer Vorgang nach A begonnen

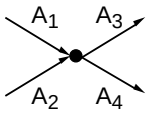
Direkte Abbildungen aus Vorgangsliste



(Start) A_2 nach (Ende) A_1

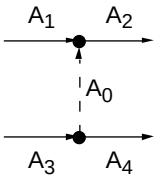


A_2,A_3 nach A_1

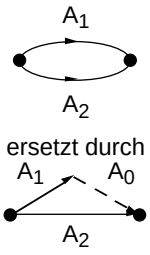


A_3,A_4 nach A_1,A_2

Abbildungen mit Scheinvorgängen (Dauer 0)



A_4 nach A_3,
A_2 nach A_1,A_3

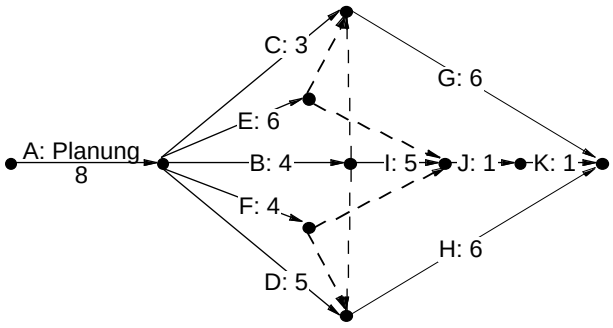


+ weitere Vorschriften:

- 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,…,n}
Pfeile <i, j> **mit Zeitdauern bewertet** D_{ij}

Netzplan Beispiel:



Eigenschaften CPM-Netzpläne

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

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

→

- **kürzeste Projektdauer**: Länge längster Weg
- **kritische Vorgänge**: Pfeile <i,j> 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 <i,j>
FEZ_{ij} frühest mögliches Ende Vorgang <i,j>
SAZ_{ij} spätest möglicher Start Vorgang <i,j>
SEZ_{ij} spätest mögliches Ende Vorgang <i,j>

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

ohne Beeinträchtigung "gesetzter" Projektdauer := T

Knotenmenge Netzplan V ={1,…,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 → max

$$FZ_i = \max_{k\ P(i)} \left(FZ_k + D_{ki}\right) \qquad i = 2,\ldots,n$$

  "vorwärts"-Anwendung; wg. top. Sortier'g P(i) {1….,i-1}

- kürzeste Projektdauer FZ_n
  "sinnvolle" Setzung $T \geq FZ_n$
  + 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 → max

$$SZ_n - SZ_i = \max_{j\ S(i)} \left(SZ_n - \left(SZ_j - D_{ij}\right)\right)$$

$$SZ_i = \min_{j\ S(i)} \left(SZ_j - D_{ij}\right) \qquad i = n-1,\ldots,1$$

  "rückwärts"-Anwendung; wg. top. Sortier'g S(i) {i+1….,n}

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

$FAZ_{ij} = FZ_i$
$FEZ_{ij} = FZ_i + D_{ij}$
$SAZ_{ij} = SZ_j - D_{ij}$
$SEZ_{ij} = SZ_j$

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

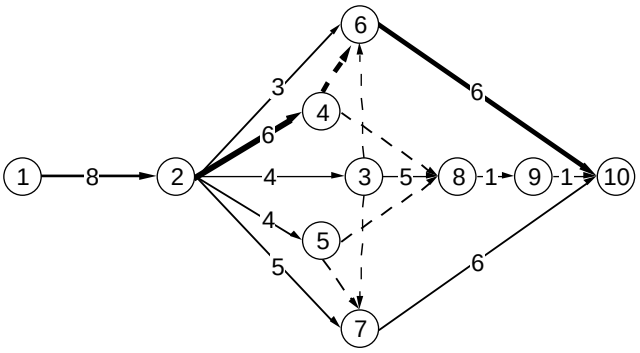
Ergebnisse Beispiel:

tabellarisch, ohne Vorgabe T:

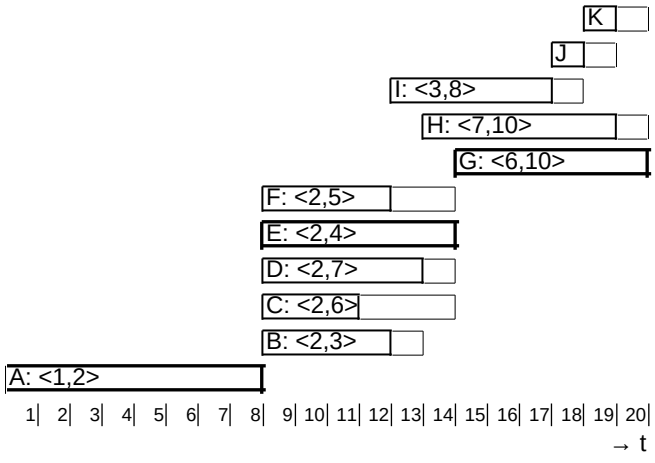
i	1	2	3	4	5	6	7	8	9	10
FZ_i	0	8	12	14	12	14	13	17	18	20
SZ_i	0	8	13	14	14	14	14	18	19	20

$T \geq 20$

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 jedem Pfeil <i,j>
eines Digraphen <V,E> ohne isolierte Knoten
eine **Minimalkapazität** u_{ij} $\mathbf{R_+}$
und eine **Maximalkapazität** o_{ij} $\mathbf{R_+}$ {∞} zugeordnet.
Der doppelt bewertete Graph
N = <V,E;u,o>
heißt **Netzwerk mit Kapazitäten**

u_{ij} bzw o_{ij} bezeichnen den zulässigen
Mindest- bzw Höchstwert des Flusses auf Pfeil <i,j>,
u_{ij}=0 ist ein häufiger Sonderfall,
o_{ij}=∞ steht für fehlende obere Beschränkung

Definition 3.5.02 : Flüsse in Netzwerken

Seien
r V, s R(r) Knoten des Netzes

Eine Abbildung
f: E → $\mathbf{R_+}$ {∞}
heißt **Fluß** in N **von** r **nach** s mit **Flußstärke** F(f) ≥ 0,
wenn die **Flußbedingung**

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

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) $u_{ij} \le f_{ij} \le o_{ij}$ <i,j > E

dann heißt f **zulässiger Fluß**

f | (f_{ij}=0; <i,j> E) heißt **Nullfluß**
Nullfluß ist zulässig für (u_{ij}=0; <i,j> 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

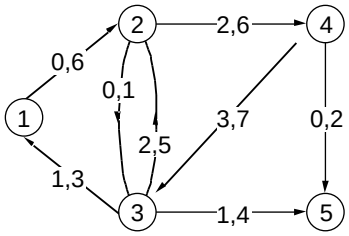
max F
udN (3.5.02),(3.5.03)

lineares Optimierungsproblem mit
(Entscheidungs-) Variablen {f_{ij} ;<i,j> E}

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

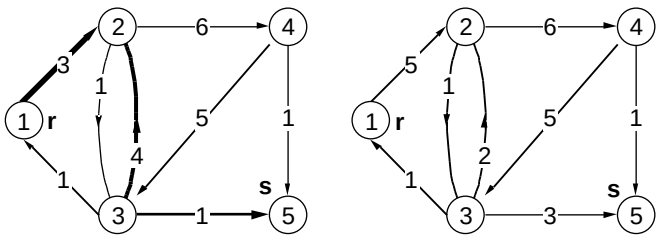
Frage: Gibt es "elegantere" / effizientere
Lösungsverfahren als
"allgemeine" lineare Optimierung ?

Beispiel



Pfeilbeschriftung:
u_{ij},o_{ij}

2 zulässige Flüsse
mit $F(f^{links}) < F(f^{rechts})$



Pfeilbeschriftung: f_{ij}

Bemerkung

wegen der summarischen Behandlung von
Flüssen auf Pfeilen <i,j> bzw <j,i>
kompensiert sich deren Auswirkung in den Knoten i,j

sei S ein Semiweg
von (Flußquelle) r
nach Knoten k ≠ r
mit Orientierung r→k: **(r,k)-Semiweg**

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

(im Beispiel enthält ausgezeichneter Semiweg
Vorwärtspfeile <1,2>, <3,5>,
Rückwärtspfeil <3,2>)

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

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

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 <i,j> von S

mit $s := \min_{<i,j>\ S} s_{ij}$

heißt S **flußvergrößernder Semiweg** für f
und Fluß r→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 $G^{\rightarrow} = <V,E>$ ein Digraph
$A \subset V, B \subset V$ eine Zerlegung der Kantenmenge
$V=A \cup B, A \cap B= \emptyset, A,B \neq \emptyset$

Menge der aus A nach B führenden Pfeile
C<A,B>
heißt (A von B) **trennender Schnitt** in $G^{\rightarrow}$

Menge der aus B nach A führenden Pfeile
C<B,A>
heißt **konträrer Schnitt** zu C<A,B>

C<A,B> C<B,A>
heißt **(A,B)-Schnitt**
bzw, für $i \in A, j \in B$
auch **(i,j)-Schnitt**

Schnitt C<A,B> eines Netzwerks N mit Kapazitäten lassen sich zuordnen
Minimal- und Maximalkapazität U und O
sowie
Kapazität K
gemäß:

$$U(C<A,B>) := \sum_{<i,j> \; C<A,B>} u_{ij}$$
$$O(C<A,B>) := \sum_{<i,j> \; C<A,B>} o_{ij}$$
$$K(C<A,B>) := O(C<A,B>) - U(C<B,A>)$$

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

$$\sum_{i \in A} \left(\sum_{j \in S(i)} f_{ij} - \sum_{k \in P(i)} f_{ki} \right)$$
$$= \sum_{i \in A} \left(\sum_{j \in S(i) \cap A} f_{ij} - \sum_{k \in P(i) \cap A} f_{ki} \right) + \sum_{i \in A} \left(\sum_{j \in S(i) \cap B} f_{ij} - \sum_{k \in P(i) \cap B} f_{ki} \right)$$
$$= 0 + F(C<A,B>)$$

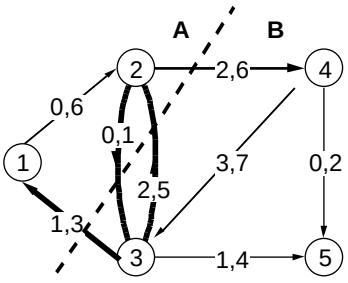
wo F(C<A,B>) Netto-Flußmenge $A \rightarrow B$, bei vorliegendem f

mit Def von K(C<A,B>) ist

$$F(f) = F(C<A,B>) \leq K(C<A,B>) \qquad A: r \in A, s \in A$$
$$\leq \min_{C<A,B>: \; r \in A, \; s \in A} K(C<A,B>)$$

Satz

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



Pfeilbeschriftung:
u_{ij},o_{ij}

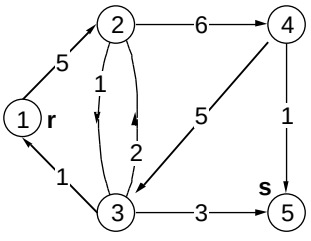
ist (A,B)-Schnitt
C<A,B> = {<2,3>,<2,4>}

hierzu konträrer Schnitt
C<B,A>={<3,1>,<3,2>}

Kapazität
K(C<A,B>) = 7 - 3 = 4

minimaler (1,5)-Schnitt
stark ausgezeichnet

und Fluß $r \rightarrow s$ 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 $N = <V,E;u,o>$
zulässiger Fluß $f: E \rightarrow \mathbf{R_+} \quad \{\infty\}$

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≠r,
  falls flußvergrößernder (r,k)-Semiweg existent
  (zusätzlicher positiver Fluß r → k möglich)

  falls nicht, maximaler Fluß F_{max} gefunden

- Flußvergrößerungsphase
  falls flußvergrößernder (r,s)-Semiweg existent
  entsprechende Flußvergrößerung
  (zul. Fluß f → 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≠r initial unmarkiert,
  unmarkierter Knoten wird markiert
  bei "Überprüfung" markierten Nachbars,
  Markierung gibt Anlaß zur "Überprüfung"
  Markierung unmarkierter Nachbarn

- Marken für i≠r haben Form $(+l,\varepsilon_i)$
  oder $(-l,\varepsilon_i)$

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

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

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

- Markierungen tabellarisch

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

- 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 $(+/-q,\varepsilon_s)$, $\varepsilon := \varepsilon_s$
- bei Markierung $(+q,\varepsilon_s)$ $f_{qs} := f_{qs} + \varepsilon$
- bei Markierung $(-q,\varepsilon_s)$ $f_{sq} := f_{sq} - \varepsilon$

in Knoten k (auf Rückweg) k=pred s,…,r
- bei Markierung $(+q,\varepsilon_k)$ $f_{qk} := f_{qk} + \varepsilon$
- bei Markierung $(-q,\varepsilon_k)$ $f_{kq} := f_{kq} - \varepsilon$

Neuer Fluß F(f') = F(f) + ε

Algorithmus 3.5.08 startet mit zulässigem Fluß,
- für **u=0** (häufiger Fall)
  ist Nullfluß offensichtlich zulässiger Initialfluß
- für **u≠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
$F_{max} \le \min($ maximaler Nettoausfluß r,
maximaler Nettoeinfluß s)

Je Schritt
- Markierungsphase: maximal m Pfeile zu prüfen
- Flußverg'gsphase: 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
$N = <V,E;u,o>$
jedem Pfeil $<i,j>$ E zusätzlich **Kosten** c_{ij} $\mathbf{R_+}$ zugeordnet.
Der dreifach bewertete Graph
$N = <V,E;u,o,c>$
heißt **Netzwerk mit Kapazitäten und Kosten**

c_{ij} bezeichnet dabei die
Kosten einer Flußeinheit auf Pfeil $<i,j>$

Bei zulässigem Fluß f der Stärke F* := F(f)
heißt Fluß dieser Stärke, der Kosten $\Sigma\, c_{ij}\, f_{ij}$
minimiert, **kostenminimaler Fluß der Stärke** F*
(beachte: F* ist fest, f ist gesucht)

f ist Lösung des linearen Optimierungsmodells Typ "KFS"

$$\min \sum_{<i,j>\ E} c_{ij} f_{ij}$$

(3.5.10)

(KFS) udN

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

$$u_{ij} \le f_{ij} \le o_{ij} \qquad <i,j> \quad E$$

Bestimmung kürzester Wege
+ Maximalflußproblem
sind Spezialfälle von KFS
KFS ist allgemeinstes der bisher betrachteten Optimierungsmodelle in Netzen

Kürzeste Wege

$N = <V,E;c>$ bewerteter Digraph ohne Zyklen negativer Länge
$r \ V, s \ R(r)\backslash r$ Start- und Zielknoten

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

$$f_{ij} := \begin{cases} 1 & < i,j > \text{ auf } W \\ 0 & \text{sonst} \end{cases} \qquad < i,j > \quad E$$

Mit $u_{ij} := 0$, $o_{ij} := \infty$ $<i,j>$ E
ist jeder kürzeste Weg Lösung von KFS mit F*=1

Maximalfluß

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

Setze $c_{ij} := 0$ $<i,j>$ E
+
füge Pfeil $<s,r>$ zu
mit $u_{sr} := 0$, $o_{sr} := \infty$, $c_{sr} := -1$

KFS Problem für
Netz N' mit Kapazitäten + Kosten (E'=E {<r,s>})
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)

$N = <V,E;u,o>$ Netzwerk mit Kapazitäten, antisymmetrisch (keine ↔ Pfeilpaare)
$f: E \to \mathbf{R_+}$ zulässiger Fluß

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

mit inkrementellen Kapazitäten

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

Weg in N' von r nach s
entspricht flußvergrößerndem (r,s)-Semiweg in N
ist in N' s 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
$X \quad E$ **matching**, wenn kein Kantenpaar in X mit selbem Knoten inzident
$i \ V$ **überdeckt** durch matching X, wenn X mit i inzidente Kante enthält
X **vollständig** (auch: **perfekt**), wenn X alle i 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

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

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

$$\min c(X) \quad \text{udN} \quad X \quad X_p(G)$$

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 R,j S]

Zuordnungsproblem

Probleme der "besten" Zuordnung von
- Bewerbern zu Stellen (vgl. Abschn. 1)
- Jobs zu Maschinen
- Fahrzeugen zu Transportaufträgen
- …

heißen Zuordnungsprobleme, erfaßt durch

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

bipartites bewertetes Netz

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

Jede vollständige Zuordnung "ist" ein matching X

Mit (binären, zweiwertigen) Entscheidungsvariablen

$$x_{ij} := \begin{cases} 1 & <i,j> \quad X \\ 0 & <i,j> \quad X \end{cases} \qquad <i,j> \quad E$$

ist "beste" Zuordnung definiert als

$$\min \quad \sum_{<i,j>} \quad _E c_{ij}x_{ij}$$
$$\text{udN} \quad \sum_{j} \quad _{S(i)} x_{ij} = 1 \quad i \quad R$$
$$\sum_{i} \quad _{P(j)} x_{ij} = 1 \quad j \quad S$$
$$x_{ij} \quad \{0,1\} \quad <i,j> \quad E$$

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,…,n} =: V
- alle direkten Transportmöglichkeiten i → j
  als Pfeile {<i,j>} =: 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
$0 \le x_{ij} \le o_{ij}$
befriedigen

sowie bzgl. abtransportierter Nettomengen =: a_i (i V)
mit $a_i > 0$ für Angebotsknoten
$a_i = 0$ für Umladeknoten
$a_i < 0$ für Nachfrageknoten
wo $\Sigma\, a_i = 0$ "nichts fällt vom Himmel"
die Flußbedingungen

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

Das Optimierungsproblem

$$\min \quad \sum_{<i,j>} \quad _E c_{ij}x_{ij}$$
$$\text{udN} \quad \sum_{j} \quad _{S(i)} x_{ij} - \sum_{k} \quad _{P(i)} x_{ki} = a_i \quad i \quad V$$
$$0 \le x_{ij} \le o_{ij} \qquad <i,j> \quad E$$

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)

$$\min \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij}$$
$$\text{udN} \quad \sum_{j=1}^{n} x_{ij} = a_i \quad i \quad \text{"Produzenten"}$$
$$\sum_{i=1}^{m} x_{ij} = b_j \quad j \quad \text{"Verbraucher"}$$
$$x_{ij} \ge 0 \qquad i,j \quad \text{"Produzenten"} \quad \text{"Verbraucher"}$$

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

Lösungsverfahren arbeiten mit / auf
(vergleichweise kleinem)
m x n -"Tableau" der x_{ij}

Abbruch der Skizze

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