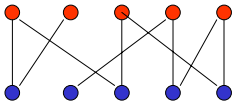
INF 3/4130
29. september 2005

- Dagens tema: Kapittel 14:
  - Matchinger i (urettede) grafer (matching = pardannelse)
  - Flyt i nettverk (nettverk = rettede grafer med kapasiteter etc.)
  - Dagens tema er kraftig forbundet med konveksitet, polyedre med heltallige hjørner etc., og dette ser man nærmere på i Geir Dahls kurs. Vi går ikke inn på det her.
- Obligatorisk oppgave 1 er lagt ut. Frist: Mandag 17. oktober
  - Det kommer en presisert utgave senest mandag morgen (3/10).
  - Men det er bare å begynne å arbeidet ut fra den som nå ligger ute
- Forelesning neste uke:
  - Antakeligvis avsluttende om kapittel 14 ?
  - Noe om matchinger i generelle grafer (eget notat)
  - Fra kapittel 10 og 23 (søk etter løsninger på kombinatoriske problemer)

Matchinger i urettede bipartite grafer, kap. 14.1

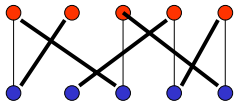
Bipartit graf = tofargbar graf = graf som ikke har (slike) "odde løkker":



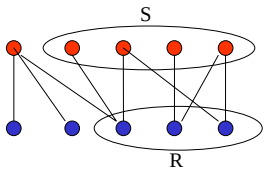
Nodemengden X, f.eks. håndverkere

Kantene: F.eks. hvem har kompetanse til hvilken jobb?

Nodemengden Y, f.eks. dagens jobber



Vi klarte å finne en "perfekt matching", som altså gjør at vi kan få utført alle jobbene.



Boka: R = Gamma(S)

Her: En undermengde S av X er forbundet (bare) med nodemengden R i Y, og R har færre noder enn S. Da finnes opplagt ingen perfekt matching. Men dette gjelder også andre veien:

Halls Teorem: Det finnes en perfekt matching **hvis og bare hvis** det ikke finnes noe utplukk S av X slik at R har færre noder enn S.

Bevis den "vanskelige" veien: Den "ungarske" algoritme vil enten gi en perfekt matching, eller den vil komme opp med en slik S.

Den naive algoritme virker ikke

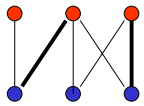
Problem: Gitt bipartite urettet graf. Finn, om mulig, en "perfekt" matching

Grådighets-tilnærmelse (lønner seg av og til, men ikke her):

"Velg kanter tilfeldig, og ta dem inn i matchingen om de ikke har en felles node med en som allerede er i matchingen"

Eksempel på at grådighets-tilnærmelsen ikke virker:

Det finnes opplagt en matching med tre kanter, men denne kan ikke utvides til en slik!



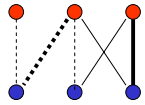
I parentes bemerket:

Et sted der grådighet faktisk virker er når en vil finne det letteste spenntreet i en sammenhengende urettet graf med vektede kanter:

"Se på kantene i rekkefølge av stigende vekt, og ta med de som ikke danner en løkke med de nodene som allerede er plukket" (Kruskals algoritme).

Den ungarske algoritme for å finne perfekt matching

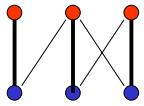
- Det viser seg imidlertid at om vi, i stedet for å lete etter helt "ledige" kanter, leter etter "forbedringsveier", så vil vi helt sikkert finne en slik en, dersom en større matching i det hele tatt eksisterer.
- En slik forbedringsvei er stiplet i denne figuren:



- Forbedringsvei:
  - En "alternerende vei" (annenhver kant er med og ikke med i matchingen)
  - Begge endenodene er "umatchet" (er ikke endenode i en kant i matchingen)

- Vi "bruker" en forbedringsvei ved å bytte kanter i matchingen langs forbedringsveien:
-

  - Dette må opplagt føre til en ny matching, som er én større.
  - I vårt tilfelle får vi denne:



Hvordan finne forbedringsveier?

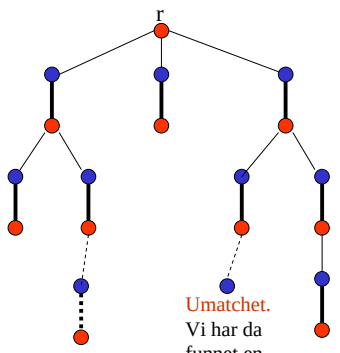
Den ungarske algoritme går på å starte med en tom matching, og så lete etter en forbedringsvei, bruke denne, finne ny forbedringsvei, bruke denne osv. til vi:

- enten har en perfekt matching
- eller vi ikke finner noen ny forbedringsvei
  - og da vil situasjonen forhåpentligvis vise oss en undermengde S i X som er forbundet med mengden R= Gamma(S) i Y, og at R er mindre enn S (slik at vi vet at det ikke finnes noen perfekt matching)

- Det generelle steget i den ungarske algoritme er altså ut på å ha en foreløpig (ikke-perfekt) matching M, og å prøve å finne forbedringsvei som vi kan bruke til å lage en matchingen som er én kant større. Søket etter en slik forbedringsvei kan greiest gjøres slik:
  - Velg en umatchet node r i X. Den skal bli roten av et tre T vi skal bygge, der alle veier ut fra roten skal være alternerende veier.
  - Vi har da funnet en forøkningsvei dersom en forgrening av treet kan få kontakt med en umatchet node i Y (se figur neste foil).

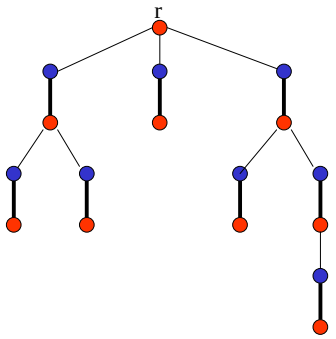
Steget i den ungarske algoritme - 1

- Vi antar altså at vi står med en ikke-perfekt matching M, og vil finne en forbedringsvei.
- I starten av trebyggingen består treet T bare av rotnoden r
- Anta at vi generelt har bygget et tre T ut fra r, og at vi vil utvide det.
- Vi ser så etter kanter ut fra noder i T som er i X (røde) og ikke går til en node som allerede er med i T.
- Om vi finner en slik kant vil den andre enden være i Y (blå). Det er to tilfeller:
  - Kanten går til en umatchet node i Y, da har vi *funnet en forbedringsvei*, og vi kan da bruke denne (øker altså M), og starte helt på nytt, med neste steg.
  - Kanten går til en matchet node i Y, og da tar vi også med i T den tilhørende kanten i M. Treet blir altså utvidet.



Steget i den ungarske algoritme - 2

- Om vi ikke finner en kant fra en rød node i T til en kant utenfor treet, ønsker vi å finne et bevis på at ingen perfekt matching finnes:
  - Ønsker: En delmengde S av nodene i X (røde) som er slik at de nodene R den er forbundet med i Y er færre enn i S.
  - Som S velger vi da rett og slett de røde nodene i T. Av dem er det én mer enn antall kanter fra M i treet.
  - Vi lar så R være de blå nodene i T. R har opplagt én node færre enn S (like mange som kanter fra M i treet)
  - Vi påstår nå at nodene i S ikke har kanter til noen andre blå noder enn de i R. Begrunnelse:
    - Algoritmen har stoppet nettopp fordi det ikke finnes kanter fra røde noder i treet til blå noder utenfor treet.



Dermed er også Halls Teorem bevist: Denne algoritmen kan kjøres på enhver bipartit graf (med like store X og Y), og den vil gi enten en perfekt matching eller en S slik at Gamma(S) er mindre enn S.

Varianter over problemstillingen

- Sett på til nå:
  - Finn en perfekt matching i en bipartit graf (eller vis at slik ikke finnes)
  - Et programskisse av denne algoritmen er gitt på side 422/423
- Andre spørsmål:
  - Finn en matching med flest mulig kanter (og da behøver ikke nodemengdene X og Y være like store)
    - Skal dere se på i gruppene neste uke
  - Gitt vekter på kantene: Finn en matching med størst mulig vekt
    - Skal vi se på på forelesningen neste uke

- Generaliseringer av det bipartite matching-problemet
  - Gå over til generelle grafer, og still de tilsvarende spørsmål angående matchinger
    - Det uveiede tilfellet av dette skal vi se på i neste uke (eget notat)
  - Flyt i "nettverk" (der matching-problemet for bipartite grafer fremkommer som et spesialtilfelle)
    - Dette skal vi se på nå.

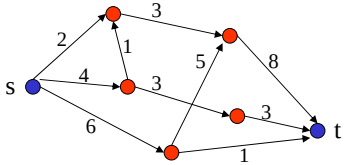
Flyt i nettverk, kap. 14.2

- Dette stoffet er også noe dekket i Weiss-boka, så man kan også lese der.
- At man her bruker ordet ”nettverk” (og ikke noe med ”grafer”) er bare ren tradisjon. Grovt sett er nettverk rettede grafer med forskjellige vekter, kapasiteter etc. på kantene.
- Svært mange praktiske problemer faller inn under nettverksproblemer, og spesielt flyt i nettverk:
  - Datanett
  - Forskjellige typer rør-nettverk
  - Vei-nettverk
  - …
- De nettverk vi skal studere her har en ”kilde-node” s og en ”sluk-node” t, og oppgaven er generelt å presse så mye ”flyt” fra s til t som mulig.
  - En flyt *f* er sammensatt av en flyt *f(e)* på hver kant *e*, som er slik at:
    - I hver node, bortsett fra i s og t, er summen av flyt inn i noden lik sum av flyt ut av noden (definert i forhold til kantenes retning). (”flytkonserverings-prinsippet”)
    - *f(e)* kan i utgangspunktet også være negativ, og dette brukes i spesielle tilfelle.
  - Men oftest: Hver kant *e* har en viss kapasitet *c(e)*, og flyten *f(e)* gjennom kanten *e* må da ligge mellom 0 og *c(e)*.

Flyt i nettverk uten kapasiteter

Forutsetter i denne fremstilling: Det går ikke kanter inn i s eller ut av t.

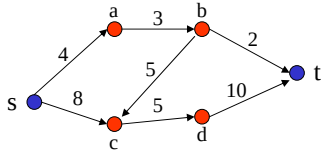
- *val(f)* er summen av flyten som går ut av s.
- Lemma: Summen av flyten som går inn i t er også *val(f)*
  - Vises grovt sett ved summering av flyten inn/ut over alle noder

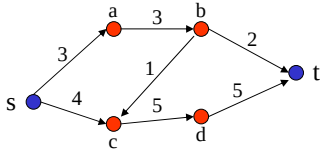


- Begreper etc. i boka, men som vi ikke nevner direkte i de følgende foilene:
  - En semi-vei gjennom grafen = en vei i den underliggende urettede grafen
  - Enhets-flyt: Definert av en semi-vei fra s til t der det er flyt +1 på de kantene som følger veiens retning, og –1 på de kantene som går mot veien.
  - Lemma: To flyter som summeres kant for kant gir en ny lovlig flyt.
  - Lemma: Om hver kant-flyt multipliseres med en fast konstant får vi en ny lovlig flyt.

Flyt i nettverk, med kapasiteter

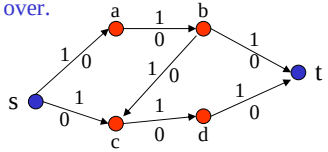
- Hver kant *e* har en viss kapasitet *c(e)*, og flyten *f(e)* gjennom kanten *e* må ligge mellom 0 og *c(e)*.
- Ønske:
  - Gitt et nettverk med kapasiteter.
  - Vi ønsker å finne kantflyter *f(e)* som
    - holder seg innenfor kapasitetene
    - utgjør en maksimal flyt, altså en som gir en så stor *val(f)* som mulig
- Eksempelet under til venstre, er et nettverk med gitte kapasiteter.
  - Vi ser intuitivt: Maksimal flyt er her 7, og en slik flyt er gitt til høyre.

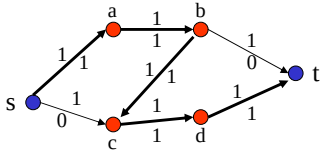




Ren grådighet virker ikke

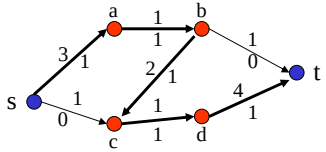
- Den naive grådighets-algoritmen virker ikke:
  - Steget: Finn ”enkel flytøkningsvei”:
    - Finn en rettet vei fra s til t som er slik at alle flyter *f(e)* er lavere enn *c(e)* langs veien
    - Øk flyten langs denne så mye som mulig (gitt av den kanten som har minst *c(e) – f(e)* langs veien)
  - Gjenta dette til ingen slike veier finnes.
- På figuren under er kapasitetene skrevet *over* kantene (alle 1) og flyten *under* (initielt er den 0 overalt).
  - Vi finner så en tilfeldig slik *enkel flytøkningsvei*, f.eks. s-a-b-c-d-t. Langs denne kan vi øke flyten med 1, og vi får neste situasjon under.
  - De fete kantene har nå brukt hele sin kapasitet (er mettet = ”saturated”)
  - *val(f)* er nå 1, men det er opplagt at vi kan oppnå *val(f)*=2
  - MEN, det finnes ingen *enkel flytøkningsvei* av den enkle typen som er definert over.



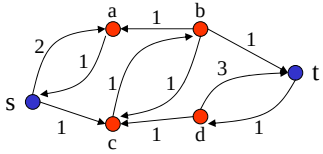


Det *f*-avledede nettverket *N(f)*

- Det vi tydeligvis ikke har tatt hensyn til i den enkle betraktningen med flytøkende veier, er at vi også kan *minke* flyten i noen kanter når vi vil gjøre en forandring – og ved å ta hensyn til det får vi faktisk en fullgod algoritme
- For å få oversikt over forandrings-mulighetene på den enkelte kant kan vi, ut fra et gitt nettverk med kapsiteter *c(e)* og en gitt lovlig flyt *f(e)*, tegne det *f-avledede nettverket* (Merk her: nye kapasiteter ift. forrige foil):



Nettverk med kapasiteter (over) og flyt (under)

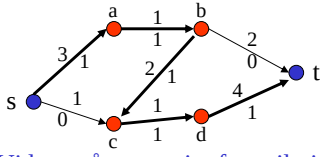


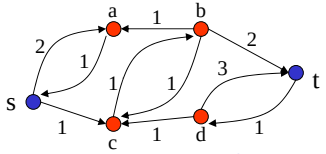
Det *f*-avledede nettverket (angir mulige flytforandringer)

- Det f-avledede nettverket betegnes her *N(f)*
- Se også figur 14.8 i boka (side 435)

f-forbedringsveier

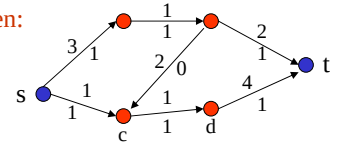
Samme som på forrige foil, det opprinnelige nettverket N til venstre.





- Vi leter så etter veier fra s til t i det *f*-avledede nettverket *N(f)*
  - Slike veier kalles "*f*-forbedringsveier"
  - Vi kan for eksempel velge s-c-b-t. Den maksimale flytforandringen langs denne er her 1 (generelt h).
- Vi gjør så den tilsvarende flytforandringen, ved å
  - øke flyten med h i de kantene i N der *f*-forbedringsveien går samme vei som i N
  - minke flyten med h der kanten i *f*-forbedringsveien går motsatt vei av i N

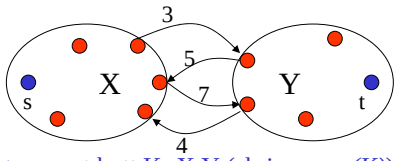
Dette gir den nye flyten:



- Ut fra denne må vi så lage et nytt *f*-avledet nettverk *N(f)*, osv

Kutt i nettverk

- Et "kutt" (Cut) i et nettverk er rett og slett en todeling av nodemengden i mengdene X og Y. Her skal vi bare se på kutt der s er i X og t er i Y.

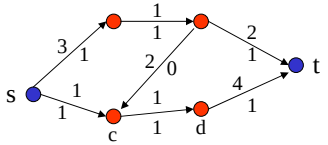


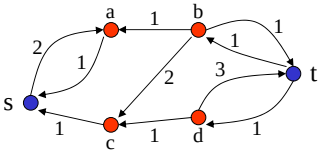
- *Kapasiteten* av et kutt K=X,Y (skrives *cap(K)*) er summen av kapasitetene på de kantene som går fra X til Y (altså ikke de som går motsatt) . Over blir den 3+7=10
- Lemma: Gitt en lovlig flyt *f* og et kutt K=X,Y. Da er *val(f)* ikke større enn *cap(K)*. Vises grovt sett slik:
  - Ved summering av flyten inn/ut over alle noder i X finner vi at "flyten over kuttet K" (med opplagt definisjon) må være lik *val(f)*.
  - Ut fra hvordan kapasiteten på et kutt er definert, ser vi cap(K) er minst like stor som *val(f)*.
- Dette gir oss en mulighet til å vise at vi har en maksimal flyt: Om vi har en flyt *f* og et kutt K slik at *val(f) = cap(K)* så er flyten maksimal!

FordFulkerson-algoritmen

Denne algoritmen går slik:

- Start med null flyt
- Steget (der vi nå har en eller annen lovlig flyt *f*):
  - Lag det *f*-avledede nettverket *N(f)* (som angir alle forandringsmuligheter)
  - Finn en *f*-forbedringsvei gjennom dette nettverket, og finn maksimal økning for denne (bestemt av den kanten langs veien med minst forandringsmulighet)
  - Gjør den forandringen i flyten som dette angir
- Gjenta steget til vi ikke lenger kan finne en *f*-forbedingsvei fra s til t i *N(f)*
- Når algoritmen slutter er det altså ikke forbindelse fra s til t i *N(f)*.



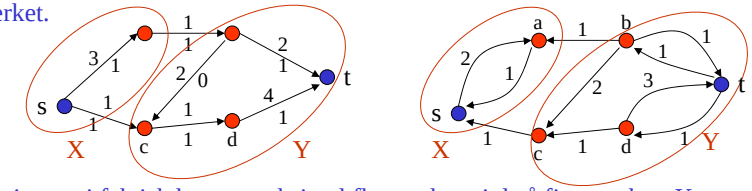


Se også programmet på side 438

Dere skal følge en variant av algoritmen ut fra figur 14.9 på gruppene neste uke

Avslutning av FordFulkerson-algoritmen

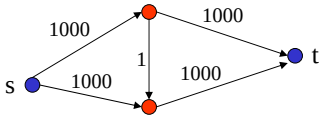
Den slutter altså med at det ikke er noen forbindelse fra s til t i det *f*-avledede nettverket.



- For å vise at vi faktisk har en maksimal flyt ønsker vi da å finne et kutt K som har nøyaktig samme kapasitet som flyten *f*: *cap(K)=val(f)*.
- Det viser at et slikt er lett å finne: La X være de nodene som kan nåes i det *f*-avledede nettverket fra X, og la Y være resten av nodene. Se figuren over.
- Siden ingen kanter i *N(f)* går fra X til Y er det lett å se at
  - Alle kanter i N som går fra X til Y bruker hele sin kapasitet (er mettet)
  - Alle kanter i N som går fra Y til X har flyt *f = 0*.
- Demed vet vi at flyten er maksimal, og vi har vi bevist følgende teorem:
- Teorem (Max-flyt min-kutt): I et nettverk med kapasiteter kan vi finne en flyt *f* og et kutt K slik at *val(f)=cap(K)*. Da vet vi at flyten er maksimal, og at intet kutt K har mindre kapasitet.

Varianter av FordFulkerson-algoritmen

- FordFulkerson-algoritmen sier bare at man skal velge en eller annen forbedringsvei i forhold til kapasitetene og den nåværende flyten, deretter finne den maksimale flytøkningen vi kan gjøre langs denne, og så legge til denne flyten.
- Når vi ikke lenger kan finne noen slik forbedringsvei har vi en maksimal flyt (bevist ved at algoritmen gir oss et kutt med kapasitet = flyten)
- Om vi ikke legger på ytterligere styring av veivalg, gjelder:
- Om vektene er heltall, så kan antall steg bli den maksimale flyten for nettverket. Eksempel:



- Om kapasitetene er reelle tall kan algoritmen teoretisk sett ikke teminere.
- Man kan hele tiden velge den forbedringsveien som gir størst forbedring (kan lett finnes med en algoritme analog til en korteste-vei-algoritme)
  - Dette gir worst-case-tid: O(m log(n) log(val(f)))
- (Edmodns og Karp) Man kan også hele tiden velge den veien som er kortest i antall kanter
  - Dette gir worst-case-tid O(n m^2)