Operationeel onderzoek

Geheeltallige programmering (IP)

LP-gebaseerde B&B

Algoritme voor max. (min.)

1. LP-relaxatie: GUB (GLB) → geheeltallig: STOP
2. Branching op fractionele variabele
3. Los LP in vertakking op: LUB (LLB) → geheeltallig: GLB (GUB) = kandidaat optimum
4. Verder vertakken → doelfunctiewaarde ≤ GLB (≥ GUB): GEDOMINEERD

Knapzak

Geheeltalligheidseis ➔ rekentijd ↑↑

Set cover/pakking/partitie

↓ ↓ ↓

$A\mathbf{x} \geq \mathbf{1}$ $A\mathbf{x} \leq \mathbf{1}$ $A\mathbf{x} = \mathbf{1}$ waarbij A = incidentiematrix met a_{ij} = 0 of 1

rijen = # objecten

kolommen = # deelverz.

- Alternatieve optima
  - Voordeel: meerdere (conflicterende) objectieven

Vaste kosten/fixed charge

- Doelfunctie: max. … - vaste kost * Y_i …
  $X_i > 0 \Rightarrow Y_i = 1$ dus $X_i \leq MY_i$ "Big M contraint"
- Kwaliteit van formuleringen
  - Gap (min.) = $\frac{Z_* - Z_{LP}}{Z_{LP}}$ x 100
  - Gap (max.) = $\frac{Z_{LP} - Z_*}{Z_*}$ x 100
  - ! M mag niet té groot zijn !

Logische restricties

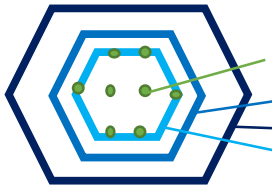
- Als $x_1 > 0$ of $x_2 > 0$ of allebei (dus $x_1 + x_2 > 0$), dan $x_3 + x_4 \geq m$
  - Indicator variabele $\delta \in \{0,1\}$: $x_j \leq M\delta$ (M groot)
  - $x_3 + x_4 \geq \delta m$
- Als $x_1 > 0$ én $x_2 > 0$, dan …
  - $z_1, z_2 \in \{0,1\}$: $x_j \leq Mz_j$ (M groot)
  - $\delta \in \{0,1\}$: $z_1 + z_2 \leq 1 + \delta$ en …
- Stel $a_1x \leq b_1$ of $a_2x \leq b_2$ of allebei moeten voldaan zijn
  - $\delta_1, \delta_2 \in \{0,1\}$: $\delta_1 + \delta_2 \geq 1$ en $a_1x \leq b_1 + M(1-\delta_1)$ en $a_2x \leq b_2 + M(1-\delta_2)$
- Minstens k beperkingen moeten voldaan zijn
  - $\delta_i \in \{0,1\}$: $\sum_i \delta_i \geq k$ en …

Magazijnvestiging

Geaggregeerd model	Gedesaggregeerd model
$\min \quad \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} x_{ij}$	
Odb	
$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,m$	
$x_{ij}, y_j \in \{0,1\} \qquad i = 1,\ldots,m; j = 1,\ldots,n$	
$\sum_{i=1}^{m} x_{ij} \leq m y_j \qquad j = 1,\ldots,n$	$x_{ij} \leq y_j \qquad i = 1,\ldots,m; j = 1,\ldots,n$
↓	↓
m + n beperkingen	m + mn beperkingen

De geheeltallige oplossingsruimte is hetzelfde voor beide, maar LP-relaxaties zijn anders!
→ **Rekentijd?**

Formulering A is sterker dan B indien
0) Zelfde IP oplossingsruimte
1) $P_A \subset P_B$
2) $P_A \neq P_B$ of $P_B \setminus P_A \neq \phi$



➔ Meer beperkingen is dus niet per se slechter, want oplossingsruimte kan dan nauwer aansluiten bij geheeltallige oplossingsruimte!

Cutting-plane algoritme

= alternatieve oplossingsmethode voor IP (naast B&B)
Een snede
- Bewaart alle geheeltallige oplossingen
- Snijdt een deel van de oplossingsruimte van LP-relaxatie weg

→ Geeft dus een **sterkere** formulering!

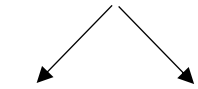
Algoritme:
1. LP-relaxatie: x* = optimum → geheeltallig: STOP
2. Voeg snede toe waaraan alle oplossingen van IP voldoen, maar x* niet (= separatie)
   Vb. enkel geldig voor **zuiver geheeltallige** programma's in **standaardvorm**:
   - Gomory-snede
     - O.b.v. equatie uit optimaal tableau met fractioneel rechterlid
   - Niet-nul snede
     - Omdat x* fractioneel is, weten we dat er geen enkele IP oplossing bestaat met alle NBV=0 → min. 1 NBV > 0 ≥ 1 dus snede: $\sum_{i \in NBV} x_i \geq 1$

   Vb. probleem-specifieke snede voor 0/1 knapzak:
   - Cover inequalities
     - Zoek geschonden voorwaarden o.b.v. $\sum_i w_i >$ toegelaten gewicht
     - Evt. **sterker** maken door alle x_i met $w_i \geq$ grootste gewicht dat al mee is, ook toe te voegen

- Combineren van B&B en cutting-planes?
  B&B: groeit exponentieel in aantal variabelen
  Cutting-planes: kan soms ook exponentieel oplopen
  - Sequentieel combineren

    LP-relaxatie

    → Snedes toevoegen tot IP-optimum OF "tailing off" (convergentie)

    Formulering is nu sterker dus kleinere zoekboom!

    … …
  - Alternerend combineren: "branch and cut"

    LP-relaxatie

    → Snedes toevoegen tot "tailing off" (convergentie)

    LP(1) → + snedes

    LP(2) → + snedes

    …

Toewijzingsprobleem < transportprobleem < transitoprobleem

Toewijzingsprobleem

Bewerkingstijd van taak j (aantal = m) op machine i (aantal = m): c_{ij}

x_{ij} = 1 indien machine i aan taak j wordt toegewezen

= 0 anders

Min. $\sum_i \sum_j c_{ij}x_{ij}$

Odb $\sum_j x_{ij} = 1$ voor i = 1, …, m

$\sum_i x_{ij} = 1$ voor j = 1, …, m

$x_{ij} \geq 0$ voor i, j = 1, …, m

→ Coëfficientenmatrix A = totaal unimodulair (TU) i.e. elke vierkante submatrix heeft determinant 0, 1 of -1. Daarnaast: alle rechterleden b zijn geheeltallig

➔ **Elke tbo is geheeltallig**!!

Ook $x_{ij} \leq 1$ dus opl. zal automatisch $x_{ij} \in \{0,1\}$ hebben

(Mogelijke oplossingsmethode: Hongaarse methode)

Transitoprobleem

p transitoknooppunten

Transportprobleem

Transport i→j: x_{ij}

Capaciteit bron i: s_i (aantal = m)

Vraag knooppunt j: d_j (aantal = n)

Kostenmatrix: $[c_{ij}]$

Min. $\sum_i \sum_j c_{ij}x_{ij}$

Odb $\sum_j x_{ij} \leq s_i$ voor i = 1, …, m

$\sum_i x_{ij} \geq d_j$ voor j = 1, …, n

$x_{ij} \geq 0$ voor i = 1, …, m en j = 1, …, n

in = uit voor alle transitoknooppunten

→ **TU**!

Handelsreizigersprobleem (TSP)

Symmetrisch TSP	Asymmetrisch TSP
Steden: N (aantal = n) Lengte tussen i en j: c_{ij} $x_{\{i,j\}} \in \{0,1\}$	= toewijzingsprobleem!! Steden: N (aantal = n) Lengte i→ j: c_{ij} , mogelijks verschillend van lengte j→i: c_{ji} $x_{ij} \in \{0,1\}$
Doelfunctie	
Min. $\sum_{\{i,j\}} c_{ij}x_{\{i,j\}}$ Odb	Min. $\sum_i \sum_j c_{ij}x_{ij}$ Odb
Toewijzing	
$\sum_{i\neq j} x_{\{i,j\}} = 2$ voor i = 1, …, n	$\sum_{j,j\neq i} x_{ij} = 1$ voor i = 1, …, n $\sum_{i,j\neq i} x_{ij} = 1$ voor j = 1, …, n → **TU** dus "gratis" IP oplossingen!
Subtour eliminatie	
$\sum_{\{i,j\}\subset S} x_{\{i,j\}} \leq \lvert S\rvert - 1$ voor S ⊂ N, $2 \leq \lvert S\rvert < \lvert N\rvert$ → DFJ formulering OF **even sterke** formulering: $\sum_{i\in S, j\notin S} x_{\{i,j\}} \geq 2$ voor S ⊂ N, $2 \leq \lvert S\rvert < \lvert N\rvert$) → Géén compacter alternatief zoals MTZ	DFJ formulering of variant mogelijk Alternatief: MTZ formulering Definieer u_i voor ordening van steden $u_1 = 1$ $2 \leq u_i \leq n$ voor i = 2, …, n $u_j \geq u_i + 1 - M(1-x_{ij})$ voor i,j ≠ 1 en i≠j (kies vb. M = n)
↓	↓
DFJ = $O(2^n)$! → Los LP-relaxatie op *zonder subtour eliminaties* en dan **separeren** door stap voor stap subtour eliminatie beperkingen toe te voegen en geschonden beperkingen te zoeken. Vervolgens: B&B of snedes zoeken en in elke stap weer separeren!	MTZ = O(n)! MAAR we zijn nu wel TU eigenschap kwijt

Merk op: DFJ-formulering is sterker dan MTZ formulering!
↔ MTZ-formulering is echter wel veel compacter…
→ Combineer:

1. Los **IP** op zonder subtour eliminatie beperkingen (k=1)
2. Zolang k ≤ max. aantal rondes:
    a. Los huidig IP op. Veronderstel dat er r subtours S_1, …, S_r zijn.
    b. Als r = 1 → STOP; anders voegen we ≤ 1000 DFJ beperkingen toe en k = k+1
3. Voeg MTZ formuleringen toe en los IP op tot optimum.

Combinatorische optimalisatie

B&B voor 1-machineplanning

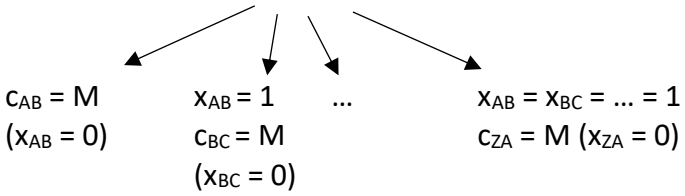
Algoritme voor min. overschrijdingstijd:

1. Beginknooppunt: vb. via heuristiek “Earliest Due Date First”: GUB
2. Branching: stel **van achter naar voor** een planning op
3. Bereken huidige overschrijdingstijd in elk knooppunt: LLB
4. Verder vertakken tot volledige planning is opgesteld: GUB
5. LLB ≥ GUB: GEDOMINEERD

B&B voor asymmetrische TSP

Algoritme min. afstand:

1. Relaxatie: laat subtour eliminatie beperkingen weg (→ TU matrix!): GLB
2. Branching: elimineer minimum-cardinaliteit subtour x_{AB}, x_{BC}, …, x_{ZA}



3. Bereken per knooppunt LLB
4. Verder vertakken tot tbo: GUB
5. LLB ≥ GUB: GEDOMINEERD

Heuristieken

- Constructieheuristieken
  - Nearest neighbour
  - Cheapest insertion
- Verbeteringsheuristieken:
  Start met constructieheuristiek en dan …
  - Swap move
  - Insert move
  - 2-opt move

  → MAAR kan gevangen zitten in **lokaal optimum**
- Meta-heuristieken:
  - Kan uit lokaal optimum ontsnappen
  - Vb. Simulated annealing, tabu search, genetische algoritmes, ant colony optimization, …

Projectplanning

- Project is uitvoerbaar ⇔ volgordenetwerk is acyclisch ⇔ ∃ topologische ordening
- AoA vs. AoN
- Kritieke pad methode (CPM): lengte langste pad = kortst mogelijke duurtijd!

Kortste/langste pad via mathematische programmering

- **Transito probleem**: min. of max. $\sum_i \sum_j c_{ij}x_{ij}$
  - X_{ij} = 1 indien (i,j) op geselecteerde pad ligt; 0 anders
  - Start = bron met capaciteit 1
  - Einde = vraag met vraag 1
  - Alle tussenliggende punten = transitopunten

  → X_{ij} zal "gratis" 0 of 1 zijn!

  → Optimale doelfunctiewaarde is eindig als er geen negatieve/positieve lussen zijn
- **Duale model**: max. of min. $t_{eind} - t_{start}$
  - t_i = duale variabele voor elk knooppunt
  - Odb $t_j - t_i \leqq$ of $\geqq c_{ij}$

    → Interpretatie:
    - Kortste pad: stel $t_j - t_i$ voor als een touwtje met lengte c_{ij} en trek t_{start} en t_{einde} zo ver mogelijk uit elkaar (~max.). Strakke lijn = kortste pad.
    - Langste pad: t_i = realisatie-tijdstip van event i, dus t_j kan pas ten vroegste c_{ij} tijdseenheden na t_i starten.
  - Alle t_i zijn vrij in teken
  - Toepassing: lineaire crash-kost per activiteit
    - r_{ij} = reductie in tijd/lengte voor pad (i,j)
    - c_{ij} = kost per eenheid tijdsreductie
    - Min. $\sum_i \sum_j c_{ij}r_{ij}$
    - Odb $t_j - t_i \geq c_{ij} - r_{ij}$

Kortste/langste pad via dynamische programmering

Algemeen: kortste/langste pad van A(0,0) naar B(n,m)

- Volledige enumeratie

  → **Rekentijd?** Voor m = n: totaal aantal operaties = **$O(2^{2n})$**
- DP-recursie
  1. Initialisatie: stel TO op; d(1) = 0
  2. Zoek kortste/langste pad tot elk knooppunt **volgens TO**
  3. Stop bij eindknooppunt

  → **Rekentijd?**
  - TO opstellen = O(m)
  - Aantal knooppunten = (n+1) x (m+1) – 1
  - 2 optellingen en 1 vergelijking per knooppunt

  } O(nxm) of $O(n^2)$ of O(m)

  → Totaal = O(2m) = **O(m)**
- Alternatieve DP-recursie voor langste pad
  1. Initialisatie: stel TO op; **late event time** LT(n) = due date
  2. Zoek kleinste LT voor elk knooppunt volgens omgekeerde TO
  3. Stop bij beginknooppunt
  - Totale speling TF(i,j) = LT(j) – ET(i) – c_{ij}

    = maximale verlengingsduur zonder impact op due date
  - Indien due date = ET(n), dan noemen we alle activiteiten met TF(i,j) = 0 kritiek
  - Vrije speling FF(i,j) = ET(j) – ET(i) – c_{ij}

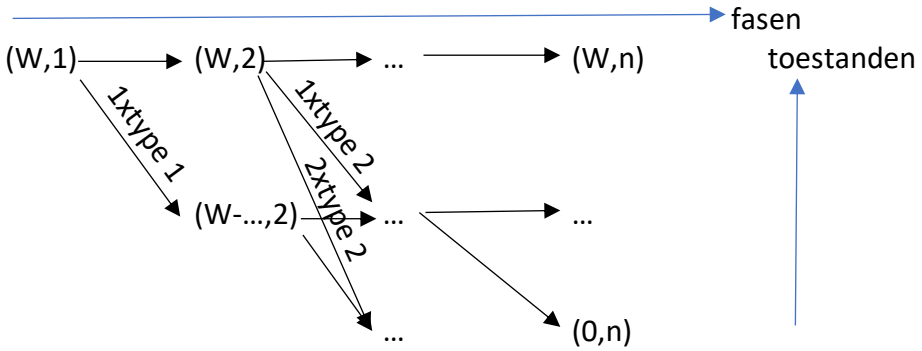
    = max. verlengingsduur zonder impact op ET van volgers

DP-recursie voor (geheeltallige) knapzak

f(d,t) = waardefunctie met toestand d (capaciteit) en fase t (type product)
Knapzak met capaciteit W en n types producten – achterwaartse DP-recursie:

1. Initialisatie: bereken f(d,n) voor d = 1, …, W
2. Bereken f(d,n-1) voor d = 1, …, W via **recursie**
3. …
4. Tenslotte: f(d,1)

➔ Analoog aan zoeken van een **langste pad in een netwerk**:



→ **Rekentijd?**

- DP-recursie langste pad = O(m) → polynomiaal
- DP-recursie knapzak = O(nK) → polynomiaal in n én in grootste getal K
  = pseudo-polynomiaal (= exponentieel)

  Verkiezen we voor knapzak dan DP-recursie of B&B?
  - Input = **O(n log K)**
  - B&B = $O(n^2 2^n)$ → nuttig indien grote getallen (K) en klein # getallen (n)
  - DP-recursie = O(nK) → nuttig indien kleine getallen (K) en groot # getallen (n)

Alternatieve recursie (NIET voor 0/1 knapzak):
g(d) = maximaal nut voor capaciteit d

1. Bereken g(1)
2. Bereken g(2) via recursie
3. …
4. Tenslotte: g(W)

→ **Rekentijd?** O(nW)

DP-recursie voor TSP

Algemeen TSP met n steden:

- Toestand = huidige locatie **+ reeds bezochte steden**
  → Wegens optimaliteitsprincipe: optimale beslissing voor resterende fasen mag enkel afhangen van huidige toestand (geen geheugen!)
- Fase = positie in de route
- **Rekentijd?** $O(n^2 2^n)$ → **exponentieel**
  Dus gebruik beter B&B voor TSP!

Kortste pad van één knooppunt naar alle andere

G(N,A) met n knooppunten en m bogen

- DP-recursie
  - **Als G acyclisch** (→ TO)
  - Lengtes kunnen > 0 zijn
  - Kortste én langste pad
  - **Rekentijd** O(m)
- Dijkstra
  - Wel cycli mogelijk (→ geen TO)
  - Als lengtes ≥ 0
  - Enkel kortste pad
  - **Rekentijd** O(nxn + m) = $O(n^2)$
  - Algoritme:
    1. Initialisatie:

It.	Knooppunt 1	Knooppunt 2	…	Knooppunt n
0	0	+∞	+∞	+∞

    2. Afstand huidig knooppunt (1) naar andere knooppunten invullen
    3. Kleinste label permanent maken → wordt huidig knooppunt
    4. Afstand huidig knooppunt naar andere invullen INDIEN kleiner
    5. Herhaal tot geen knooppunten overblijven
- Bellman-Ford
  - Wel cycli mogelijk
  - Lengtes kunnen > 0 zijn
  - Enkel kortste pad
  - **Rekentijd** O(nm)
  - Algoritme:
    1. Initialisatie:

It.	Knooppunt 1	Knooppunt 2	…	Knooppunt n
0	0	+∞	+∞	+∞

    2. Afstand huidig knooppunt (1) naar andere knooppunten invullen
    3. Kleinste labels invullen
    4. Afstand vanuit alle knooppunten met gewijzigd label invullen INDIEN kleiner
    5. Herhaal tot convergentie OF # iteraties = # knooppunten (geen convergentie → negatieve lus)

Vervangingsproblemen

C_{ij} = aankoopkost in jaar i + onderhoudskost tijdens jaren i, …, j-1 – inruilwaarde jaar j
→ Los op via achterwaartse DP-recursie met

- Ofwel f(t) = min. kost vanaf jaar t indien je in t nieuwe auto koopt
- Ofwel f(t,i) = min. kost vanaf t indien auto i perioden oud is (keuze: houd of vervang)

Niet-additieve recursie

↳Doelfunctie

Voorbeelden:

- Systeembetrouwbaarheid van serieschakeling
  - Niet-additieve variant van knapzak
  - Max $B_1(X_1)$ x $B_2(X_2)$ x … x $B_n(X_n)$
- Discounting
- Evenredige vertegenwoordiging
  - N districten met populatie P_i en R vertegenwoordigers aan te stellen
  - Ideaal: X_i uit district i met $X_i = R(P_i/P)$
  - Doelfunctie:
    - Min.$_{(\sum Xi = R)}$ $\sum_{i=1}^{N} |X_i - RP_i/P|$
    - Min.$_{(\sum Xi = R)}$ {max.$_i$ { $|X_i - RP_i/P|$ }}

Stochastische DP

Ofwel onzekerheid vanwege stochastische opbrengst

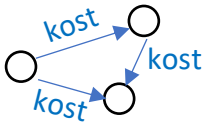
→ Gebruik verwachte waarden

Ofwel onzekerheid vanwege stochastische transformatie

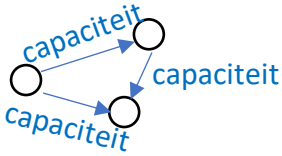
→ Optimale politiek is conditioneel!

Netwerk(stroom)modellen

Netwerkmodellen:
Vb. transitoprobleem > transport > toewijzing



Max. stroommodellen:



Maximum stroommodellen via mathematische programmering

Netwerk G(N,A)
x_{ij} = stroom over boog (i,j)
u_{ij} = capaciteit boog (i,j)
v = stroom over artificiële terugkeerboog van einde t naar start s
Max. v

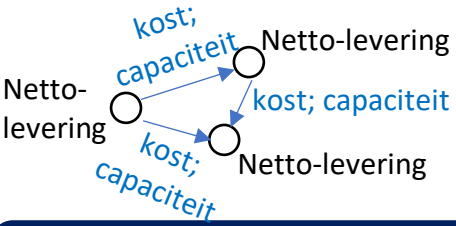
Odb $\sum_j x_{ij} - \sum_i x_{ij} = \begin{cases} v \text{ voor } i = s \\ 0 \ \forall i \in N \setminus \{s,t\} \\ -v \text{ voor } i = t \end{cases}$

$0 \leq x_{ij} \leq u_{ij}$ voor alle $(i,j) \in A$
$0 \leq v < +\infty$

→ Knooppunt-boog incidentiematrix → **TU**!
Zelfs incl. capaciteitsbeperkingen krijgen we "gratis" $x_{ij} \in \{0,1,2,\dots\}$

MCNFP via mathematische programmering

= moederprobleem van alle netwerk- en stroomproblemen

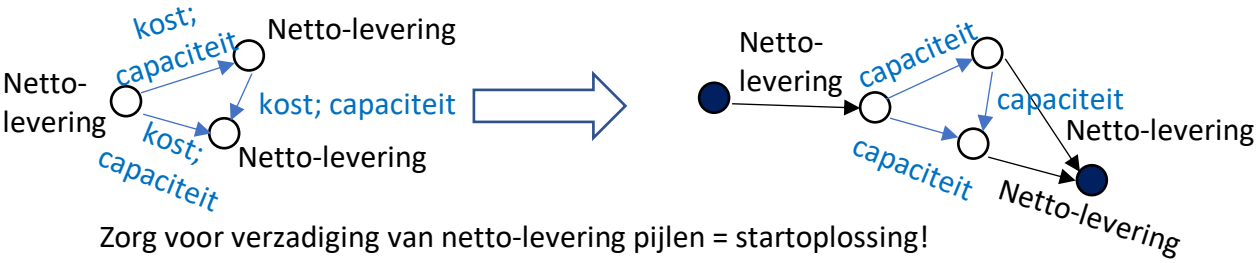


Netwerk G(N,A)
x_{ij} = stroom over boog (i,j)
u_{ij} = max. capaciteit boog (i,j)
l_{ij} = min. capaciteit boog (i,j)
c_{ij} = kost voor vervoer over boog (i,j)
b_i = netto levering knooppunt i
Min. $\sum_{(i,j)} c_{ij}x_{ij}$
Odb $\sum_j x_{ij} - \sum_i x_{ij} = b_i$ voor alle $i \in N$
$l_{ij} \leq x_{ij} \leq u_{ij}$ voor alle $(i,j) \in A$

→ Knooppunt-boog incidentiematrix → **TU**! Zelfs incl. capaciteitsbeperkingen.
Merk op: voor elke tbo geldt $\sum_i b_i = 0$

Negatieve-lus algoritme voor MCNFP

1. Initialisatie: vind een toelaatbare stroom **x**
   → Maak van MCNFP een max. stroom-instantie:



   Zorg voor verzadiging van netto-levering pijlen = startoplossing!
2. Zolang het residueel netwerk G(N,R(**x**)) een negatieve lus C bevat:
   a. Stuur max. toelaatbare stroom over de negatieve lus
      (verhoog/verminder stroom over voorwaartse/achterwaartse bogen in C)
   b. Update G(N,R(**x**))

   → Vinden van negatieve lus in G(N,R(x)): **Bellman Ford algoritme**!
   ↳ (Initialisatie: eindig getal i.p.v. ∞)
3. Toelaatbare stroom **x*** is optimaal ⇔ residuele netwerk G(N,R(**x***)) bevat geen negatieve lus meer!

→ **Rekentijd?**
Algemeen: netwerk G(N,A) met |N| = n, |A| = m, C = grootst voorkomende kost,
U = grootst voorkomende netto-leveringen/capaciteit

- Bellman-Ford = O(mn)
- Max. # iteraties = max. doelfunctiewaarde = mCU
  → Tijdscomplexiteit $O(nm^2CU)$ → **exponentieel**!
  MAAR indien zorgvuldig gekozen negatieve lus: **polynomiaal** algoritme!

Ford-Fulkerson algoritme voor max. stroomproblemen

= toepassing van negatieve-lus algoritme voor max. stroomproblemen
(met kost van elk pad = 0, behalve c_{ts} = -1)

1. Initialisatie: vind een toelaatbare stroom **x**
   → Vb. **0-stroom**
2. Zolang het residueel netwerk G(N,R(**x**)) een negatieve lus C bevat:
   a. Stuur max. toelaatbare stroom over de negatieve lus
      (verhoog/verminder stroom over voorwaartse/achterwaartse bogen in C)
   b. Update G(N,R(**x**))

   → Vinden van negatieve lus in G(N,R(x)): **labelling-algoritme**!
3. Toelaatbare stroom **x*** is optimaal ⇔ residuele netwerk G(N,R(**x***)) bevat geen negatieve lus meer!

➔ Naast **maximum stroom** vinden we hierdoor ook **minimum snedes**:

- S = {alle bereikbare knooppunten (dus niet-verzadigde pijlen) vanuit s}; $\bar{S}$ = N\S
- Snede = [S, $\bar{S}$] = {noodzakelijke pijlen voor stroom s→t}
- C[S, $\bar{S}$] = capaciteit van de snede = UB voor max. stroom
- Einde Ford-Fulkerson algoritme: C[S, $\bar{S}$] = optimale (max.) s-t stroom

→ **Rekentijd?**

Algemeen: netwerk G(N,A) met |N| = n, |A| = m, C = grootst voorkomende kost,
U = grootst voorkomende netto-leveringen/capaciteit

- Labelen = O(m)
- Max. # iteraties = max. doelfunctiewaarde = nU
  → Tijdscomplexiteit O(nmU) → **pseudopolynomiaal** dus grootte ~ log U!
  MAAR indien zorgvuldig gekozen verbeterende paden: **polynomiaal** algoritme!

Complexiteitstheorie

= “In welke mate stijgt de rekentijd?”
Polynomiaal begrensde rekentijd = efficiënt/gemakkelijk
>< **exponentieel** = niet efficiënt/moeilijk

- Voor n groot: exponentiële functie > polynomiale functie
- Algoritme met polynomiale tijdscomplexiteit: multiplicatieve verbetering van max. op te lossen instantie
  >< algoritme met exponentiële tijdscomplexiteit: additieve verbetering

➔ Probleem is <u>makkelijk</u> *indien* polynomiaal algoritme bestaat
Vb.

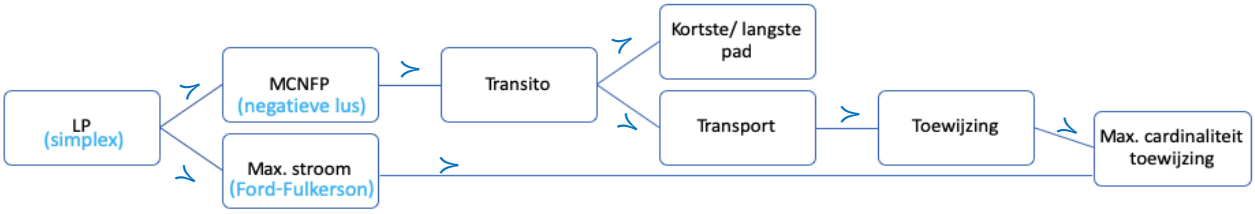
- **LP**: ellipsoïde methode en inwendige puntmethode
  !! Simplex algoritme is exponentieel in rekentijd !! Hoewel empirisch wel goed…
- **Toewijzing, transport, MCNFP**: speciaal geval van LP
- **Kritiek pad in acyclisch netwerk zoeken**: DP-recursie
- **TO vinden in georiënteerde graaf**: rekentijd neemt toe in aantal pijlen

➔ Probleem is <u>moeilijk</u> indien bewijs dat poly-tijd algoritme niet bestaat…
Vb. **IP**?

Reducties

Als P1 $\prec$ P2, dan P2 makkelijk $\Rightarrow$ P1 makkelijk
P1 moeilijk $\Rightarrow$ P2 moeilijk

Vb. <u>netwerkproblemen</u>



Vb. <u>beslissingsproblemen</u>

- Partitie $\prec$ 0/1 knapzak
  - Partitie: Bestaat er een deelverzameling A’ van A zodat som van de getallen in A’ = som van getallen in A\ A’ ?
  - 0/1 knapzak: nut en gewicht object i = element i uit A
    → max. nut zonder overschrijding capaciteit: als optimum = capaciteit: “ja”
- Hamiltoniaans circuit $\prec$ TSP
  - HC: Bestaat er een tour die elk knooppunt precies 1 keer bezoekt?
  - TSP: elke boog in gegeven netwerk: lengte 1, alle andere bogen: lengte > 1
    → los TSP op: als optimum = som van bogen met lengte 1: “ja”
- Euler circuit $\prec$ Hamiltoniaans circuit
  - EC: Bestaat er een tour die elke boog precies 1 keer bezoekt?
    → Een graaf bevat een EC $\Leftrightarrow$ de graaf verbonden is en alle knooppunten hebben een even aantal aanliggende bogen (oplosbaar in lineaire tijd).

!! Pseudo-polynomiale algoritmes leiden tot **niet-polynomiale reducties** !!
Vb. knapzak (grootte instantie O(n log M)) $\prec$ langste pad (grootte instantie O(nxM)

nxM is exponentieel in O(n log M)

P versus NP

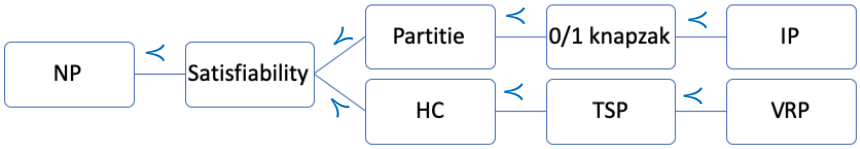
↳ = alle beslissingsproblemen

↳ = alle beslissingsproblemen die oplosbaar zijn in polynomiale tijd

Dus P $\subseteq$ NP, maar is P = NP?

→ Probleem X is **NP-hard** $\Leftrightarrow \forall Y \in NP: Y \prec X$

NP-hard probleem makkelijk $\Rightarrow$ gans NP makkelijk: Onmogelijk? Neen. Onwaarschijnlijk? Ja!



Wat is dan nog relevant?

- Afmetingen
- Empirisch vs. worst case
- Speciale structuur
- Heuristieken
- Relaxeer

Complexiteitsstatus

NP-hard Polynomiaal oplosbaar

Kan erg verschillen voor gelijkaardige problemen!!

- IP (NP-hard) vs. LP (makkelijk)
- HC (NP-hard) vs. EC (makkelijk)
- TSP (NP-hard) vs. MST (makkelijk)

  ↳ = Minimum Spanning Tree

  ↳ Oplossen via **algoritme van Prim**

  1. Willekeurig beginknooppunt
  2. Voeg boog en knooppunt toe met kleinste gewicht
  3. Herhaal tot # bogen = # knooppunten – 1

  → **Rekentijd?**

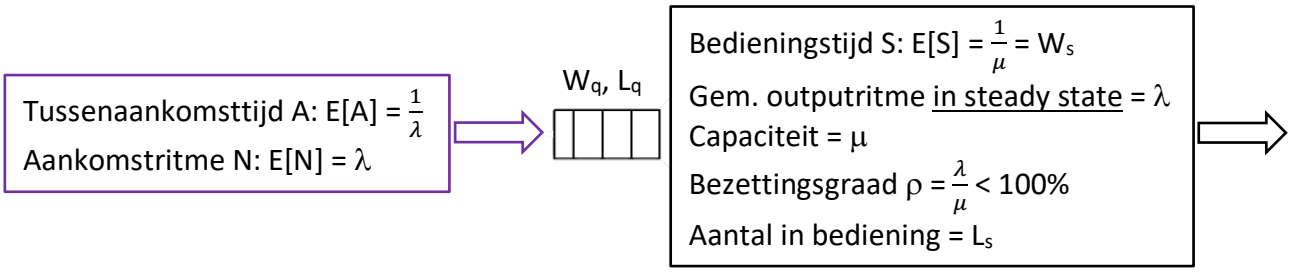
  O(n-1) iteraties met O(m) operaties per iteratie = O(nxm)

Wachtlijnen

Wachttijd paradox

Gem. tijd tot volgende aankomst $\left[= \frac{E[A]}{2} \cdot (1 + \frac{var[A]}{(E[A])^2})\right]$ is onafhankelijk van vorige aankomst!

Algemeen wachtlijnsysteem



- Voor elk systeem geldt Wet van Little: $L = \lambda_{eff} W$
- Doorlooptijd van een klant i W_i = stochastisch proces in **continue ruimte**, **discrete tijd**
- Aantal klanten in systeem $\mathcal{L}(t)$ = stochastisch proces in **discrete ruimte**, **continue tijd**

Poisson-proces

= geboorteproces

Voorwaarden

1) Geheugenloosheid: wachttijd paradox!
2) Kans op één aankomst binnen zeer klein tijdsinterval $\Delta t \rightarrow 0 = \lambda$
3) Kans op meer dan één aankomst binnen tijdsinterval $\Delta t \rightarrow 0 = 0$

Palm-Khinchine theorema

Σ veel aparte stochastische processen die elk heel zelden een event genereren ~ Poisson

Eigenschappen

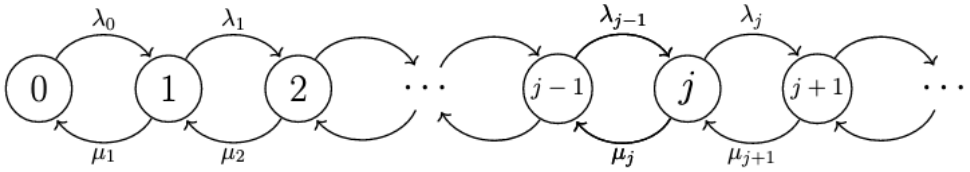
Aankomstritme N ~ Poisson(λt) ⇔ tussenaankomsttijd A ~ exponentieel($\frac{1}{\lambda}$)

N is discreet	A is continu
$E[N(t)] = \lambda t$	$E[A] = \frac{1}{\lambda}$
$Var[N(t)] = \lambda t$	$Var[A] = \frac{1}{\lambda^2}$
$Pr[N(t) = k] = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$	$f_A(t) = \lambda e^{-\lambda t}$
	$Pr[A \leq t] = 1 - Pr[N(t) = 0] = 1 - e^{-\lambda t}$ ⟲ integraal
	Geheugenloosheid: $Pr[A > T+t \mid A > T] = Pr[A > t]$

Geboorte-/sterfteproces $\mathcal{L}(t)$

= veralgemening Poisson-proces:

- Geboorte én sterfte
- Toestandsafhankelijke aankomstritmes (λ_j) en vertrekritmes (μ_j)



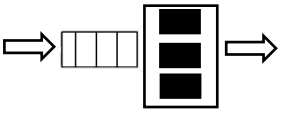
Voorwaarden

1) Tijd tot volgende geboorte in toestand j ~ Exp.(λ_j)
2) Tijd tot volgende sterfte in toestand j ~ Exp.(μ_j)
3) Tijd tot volgende gebeurtenis in toestand j = minimum van 1 en 2

Stappenplan

1. Toestandsafhankelijke λ_j en μ_j
   a. Controleer voorwaarden
      - Concurrerende exponentiëlen:
        Indien meerdere parallelle stations met onafhankelijke S_i ~ Exp.(μ)
        → tijd tot volgende sterfte exponentieel in elke toestand!
      - Desaggregatie:
        Poisson-proces (λ) → verschillende types i met kans p_i volgen elk apart ook een Poisson proces (λp_i)
   b. Bepaal λ_j en μ_j
   c. Stel ritme-diagram op
2. Evenwichtskansen π_j
   Stel steady state: $\lim_{t\to\infty} P_{ij}(t) = \pi_j$ → onafhankelijk van begintoestand i en tijd t
3. Prestatiemaatstaven L, L_s, L_q
4. Wet van Little: W, W_s, W_q
   ! Gebruik λ_{eff} !

Kendall-Lee notatie: a/b/c/d/e/f

Geldt enkel voor  of 

a: kansdichtheid tussenaankomsttijden
→ M/D/E_k/GI
b: kansdichtheid bedieningstijd
→ M/D/E_k/G

M/M/… is geboorte-/sterfteproces!

c: aantal parallelle stations
d: wachtlijndiscipline
→ FCFS/LCFS/SIRO/GD
waarbij $E[W_{FCFS}] = E[W_{LCFS}] = E[W_{SIRO}]$ maar $var[W_{FCFS}] \leq var[W_{SIRO}] \leq var[W_{LCFS}]$
e: max. klanten in systeem (wachten + bediening), vb. **balking**
f: grootte klantenpopulatie

Eén server 

M/M/1/GD/∞/∞ (geboorte-/sterfteproces)

1. Toestandsafhankelijke λ_j en μ_j
   $$\begin{cases} \lambda_j = \lambda \\ \mu_0 = 0 \\ \mu_j = \mu \end{cases}$$
2. Evenwichtskansen π_j
   Bezettingsgraad = $\frac{L_s}{s}$ = L_s = ρ = $\frac{\lambda}{\mu}$ < 100%!
   $\lambda_{eff} = \mu\,\rho = \mu\,(1-\pi_0)$
3. Prestatiemaatstaven L, L_s, L_q
   $L = \sum_{j=0}^{\infty} j\pi_j$ = …
   $L_s = 1 - \pi_0$
   $L_q = \sum_{j=1}^{\infty} (j-1)\pi_j$ = …
4. Wet van Little: W, W_s, W_q

Speciaal geval: M/M/1/**FCFS**/∞/∞

- $Pr[W > t] = e^{-\mu(1-\rho)t}$ → W ~ Exp.($\mu(1-\rho)$) = Exp.($\mu(1-\frac{\lambda}{\mu})$) = Exp.($\mu - \lambda$) = Exp.($\frac{1}{W}$)
- $Pr[W_s > t] = e^{-\mu t}$ → W_s ~ Exp.(μ)
- $Pr[W_q > t] = \rho\, e^{-\mu(1-\rho)t}$

M/M/1/GD/c/∞ (geboorte-sterfteproces)

1. Toestandsafhankelijke λ_j en μ_j
   $$\begin{cases} \lambda_j = \lambda \\ \lambda_c = 0 \\ \mu_0 = 0 \\ \mu_j = \mu \end{cases}$$
2. Evenwichtskansen π_j
   Ook steady state als $\rho = \frac{\lambda}{\mu} \geq 100\%$!
   Bezettingsgraad = $\frac{L_s}{s}$ = $1 - \pi_0 \neq \rho$
3. Prestatiemaatstaven L, L_s, L_q
   $L = \sum_{j=0}^{\infty} j\pi_j$ = …
   $L_s = 1 - \pi_0$
   $L_q = L - L_s$
4. Wet van Little: W, W_s, W_q
   $\lambda_{eff} = \lambda\,(1-\pi_c)$

M/G/1/GD/∞/∞

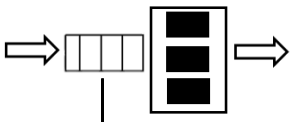
W_q = wachttijd voorgangers in de rij + klant in bediening

= L_q x $\frac{1}{\mu}$ + $(1-\pi_0)$ x $\frac{E[S]}{2} \cdot \left(1 + \frac{var[S]}{(E[S])^2}\right)$

$W = W_s + W_q = \frac{1}{\mu} + W_q$ (→ Wet van Little: L, L_s, L_q)

Bezettingsgraad = $\frac{L_s}{s} = \frac{\lambda}{\mu} = \rho$

Meerdere servers in parallel



M/M/s/GD/∞/∞ (geboorte-/sterfteproces)

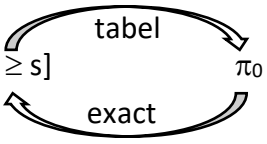
↳ Gepoolde wachtrij → efficiëntiewinst!

1. Toestandsafhankelijke λ_j en μ_j

$$\begin{cases} \lambda_j = \lambda \\ \mu_j = j\mu \ (voor\ j = 0,1,\dots,s-1) \\ \mu_j = s\mu \ (voor\ j = s, s+1,\dots) \end{cases}$$

2. Evenwichtskansen π_j

   Bezettingsgraad = $\frac{L_s}{s} = \frac{\lambda}{s\mu} = \rho$ < 100%

3. Prestatiemaatstaven L, L_s, L_q

   $L = L_q + L_s$

   $L_s = \frac{\lambda}{\mu}$

   $L_q = \sum_{j=s}^{\infty} (j - s)\pi_j$

4. Wet van Little: W, W_s, W_q

M/G/∞/GD/∞/∞ ("self service")

- M/**M**/∞/GD/∞/∞ (geboorte-/sterfteproces)
  1. Toestandsafhankelijke λ_j en μ_j

$$\begin{cases} \lambda_j = \lambda \\ \mu_k = k\mu \ (voor\ elke\ k \in \mathbb{N}) \end{cases}$$

  2. Evenwichtskansen π_j

     Bezettingsgraad = $\frac{L_s}{s} \neq \rho$

     $\pi_j = \frac{\rho^k}{k!} \cdot \pi_0$ ~ Poisson(ρ)

     $\pi_0 = e^{-\rho}$

  3. Prestatiemaatstaven L, L_s, L_q

     $L_q = 0$

     $L = L_s = \rho$

  4. Wet van Little: W, W_s, W_q

     $W_q = 0$

     $W = W_s = \frac{1}{\mu}$

- M/**D**/∞/GD/∞/∞

  ↳ Elke klant blijft exact d tijdseenheden in het systeem

  π_j ~ Poisson($\lambda d = \frac{\lambda}{\mu} = \rho$)

- M/**G**/∞/GD/∞/∞

  Via desaggregatie en aggregatie: π_j ~ Poisson($\sum_{k=1}^{\infty} \lambda p_k d_k = \frac{\lambda}{\mu} = \rho$)

→ Als aankomstritme ~ Poisson, dan $\pi_j = \frac{\rho^k}{k!} \cdot e^{-\rho}$ ~ Poisson(ρ)

onafhankelijk van bedieningstijd-verdeling!! Enkel gemiddelde $\frac{1}{\mu}$ hebben we nodig!

M/M/s/GD/s/∞ (geboorte-/sterfteproces)

= Blocked Customers Cleared = Erlang verliesmodel

1. Toestandsafhankelijke λ_j en μ_j

$$\begin{cases} \lambda_j = \lambda \\ \lambda_s = 0 \\ \mu_j = j\mu \ (voor\ j = 0,1, \dots, s-1) \\ \mu_j = s\mu \ (voor\ j = s, s+1, \dots) \end{cases}$$

2. Evenwichtskansen π_j

   π_s = fractie van klanten die niet bediend kunnen worden

3. Prestatiemaatstaven L, L_s, L_q

   $L_q = 0$

   $L = L_s = \frac{\lambda_{eff}}{\mu} = \frac{\lambda(1-\pi_s)}{\mu}$

4. Wet van Little: W, W_s, W_q

   $W_q = 0$

   $W = W_s = \frac{1}{\mu}$

M/M/R/GD/K/K (geboorte-/sterfteproces)

= machine repair = finite source

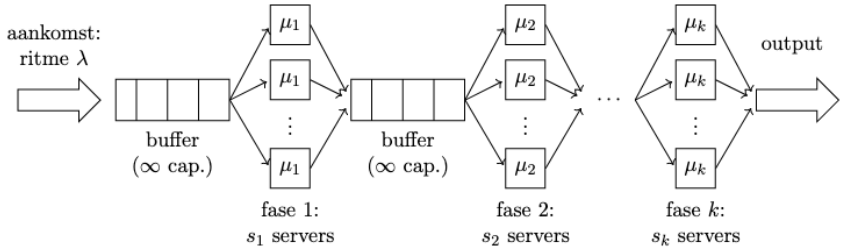
1. Toestandsafhankelijke λ_j en μ_j

$$\begin{cases} \lambda_j = (K-j)\lambda \\ \mu_j = j\mu \ (voor\ j = 0,1, \dots, R) \\ \mu_j = R\mu \ (voor\ j = R, \dots, K) \end{cases}$$

2. Evenwichtskansen π_j
3. Prestatiemaatstaven L, L_s, L_q
4. Wet van Little: W, W_s, W_q

Netwerken

Seriële netwerken



Stelling van Jackson: indien

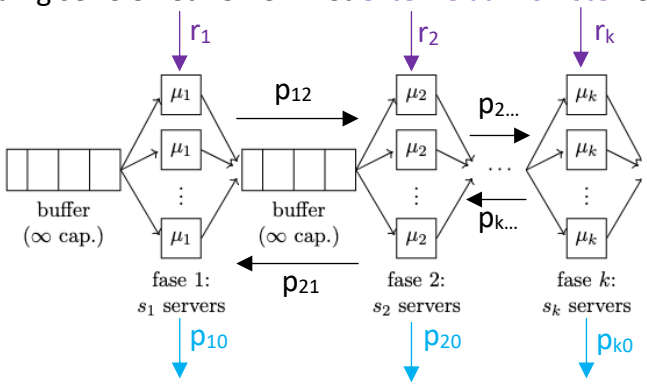
1) Tussenaankomsttijden in eerste fase ~ Exp.(λ)
2) Bedieningstijd in elke fase j (identieke servers) ~ Exp.(μ_j)
3) Elke fase j voldoende capaciteit: $\lambda < s_j \mu_j$
4) Buffers met ∞ capaciteit en FCFS

dan tussenaankomsttijden in iedere fase j ~ Exp.(λ)

➔ Elke fase j: M/M/s_j/FCFS/∞/∞

Open netwerken

= uitbreiding seriële netwerken met externe aankomsten en vertrek uit het systeem



Gelijkaardig aan stelling van Jackson: toestand van elk station onafhankelijk van andere!
Voldoende capaciteit?

$$\begin{cases} \lambda_1 = r_1 + p_{21}\lambda_2 \\ \lambda_2 = r_2 + p_{12}\lambda_1 + \cdots \\ \quad \cdots \end{cases}$$