Introduktion til

# Modellering inden for Prescriptive Analytics

E2022

Introduktion til

# Modellering inden for Prescriptive Analytics

E2022

Sune Lauth Gadegaard
Institut for Økonomi, Aarhus Universitet

*Sidst opdateret 11. november 2022*

Sune Lauth Gadegaard
Institut for Økonomi
Aarhus BSS
Aarhus Universitet
Fuglesangs Allé 4,
8210 Aarhus V
pure.au.dk/sgadegaard@econ.au.dk



Dette dokument er sat vha. LaTeX af Sune Lauth Gadegaard
Skrifttype: iwona, light, condensed. 12pt
Produceret vha. *memoir*–klassen
Alle figurer er produceret vha. pakken Ti*k*Z

En særlig tak går til mine studerende, som gennem årene har været så venlige at bidrage til materialet ved at finde trykfejl, manglende kommaer og ved at foreslå nye og anderledes måder at fremstille emner på.

Følgende studerende har ydet en ekstraordinær indsats i forhold til at forbedre denne bogs fremstilling og indhold:

| Navn | Hovedbidrag | Årgang |
|---|---|---|
| Rasmus Bertelsen Madsen | Korrektur | 2021 |
| Elisa le Fevre Leineweber | Korrektur (komma-ninja) | 2022 |
| Alexander Strarup | Korrektur | 2022 |

*Til Selma og Ida*

*Og til Kim Allan Andersen, som forlod os alt for tidligt*

# Indholdsfortegnelse


**1 Introduktion** 1
- 1.1 Hvad er "prescriptive analytics" . . . 1

**2 Introduktion til Python og en let indføring i brugen af Pyomo** 5
- 2.1 Installation af Python og Pyomo . . . 6
- 2.2 Introduktion til programmering i Python . . . 9
- 2.3 Opbygning af Pythonkode og det første program . . . 11
- 2.4 Introduktion til Pyomo . . . 17
- 2.5 Pakker som kan have interesse for "*the prescriptive analyst*" . . . 31

**3 Clustering** 33
- 3.1 Hvad er clustering . . . 33
- 3.2 Tilgange til Clustering . . . 35

**4 Lokationsplanlægning og netværksdesign** 53
- 4.1 Hvad er lokaliseringsplanlægning og netværksdesign? . . . 53
- 4.2 $p$-median og $p$-center problemet . . . 55
- 4.3 Lokaliseringsplanlægning med faste åbningsomkostninger . . . 59
- 4.4 Covering-problemer . . . 61
- 4.5 Netværksdesign problemer . . . 67

**5 Produktionsplanlægning** 77
- 5.1 Modelelementer . . . 77
- 5.2 En første simpel model - Uncapacitated lot-sizing model . . . 78
- 5.3 Master Production Scheduling udvidelse . . . 85
- 5.4 Material Requirement Planning udvidelse . . . 93

**6 Skemaplanlægning** 99
- 6.1 Hvad er skemaplanlægning? . . . 99
- 6.2 $n$ jobs på én maskine . . . 101

6.3 *n* jobs på *m* identiske maskiner . . . . . 107

**7 Ruteplanlægning** 117
7.1 Hvad er ruteplanlægning? . . . . . 117
7.2 Traveling salesperson problemet . . . . . 118
7.3 Multiple-TSP: mere end et køretøj . . . . . 126
7.4 Kapacitetsbegrænsninger: CVRP . . . . . 131
7.5 Yderligere udvidelser af ruteplanlægningsproblemer . . . . . 136
7.6 En "unified" tilgang til ruteplanlægning . . . . . 143

**8 Verifikation, validering og repræsentation af modeller og løsninger** 149
8.1 Hvorfor verificere, validere, og repræsentere modeller og løsninger 149
8.2 Modellen har ingen brugbar løsning . . . . . 151
8.3 Modellen er ubegrænset . . . . . 154
8.4 Modellen kan løses . . . . . 156

**9 Avancerede emner** 163
9.1 Stokastisk optimering . . . . . 163
9.2 Multikriterie optimering . . . . . 172

**Appendices** 191

**A Forudsætninger og matematisk notation** 193
A.1 Matematiske symboler og deres definitioner . . . . . 193
A.2 Summer i kompakt form . . . . . 196
A.3 Lineær programmering og blandet heltalsprogrammering . . . . . 201

**Bibliografi** 211

# 1 Introduktion

Denne bog er tænkt som undervisningsmateriale til kurset "Modellering inden for Prescriptive Analytics". Kurset har som hovedmål at sætte den studerende i stand til at modellere en lang række problemstillinger inden for erhvervsøkonomi, logistik og operationsanalyse. Hovedvægten vil ligge på lineære blandede heltalsprogrammeringsmodeller (Mixed Integer Linear Programming (MILP) models) men vi kommer også til at se hvorledes nogle problemstillinger kan formuleres ved hjælp af constraint programming og også ved hjælp af software-specifik syntaks.

Ud over denne introduktion til modellering er håbet også, at den succesfulde studerende efter endt kursus kan identificere svagheder ved sine modeller og at den studerende kan validere modeller og løsninger ved hjælpe af andre teknikker fra prescriptive analytics.

## 1.1 Hvad er "prescriptive analytics"

Et første, åbenlyst og meget fornuftigt spørgsmål at stille sig selv er; hvad er prescriptive analytics? Det kan jo ved første øjekast opfattes en smule besynderligt, at denne del af kursets titel ikke er på dansk, og det er den af gode grunde ikke. Der er på nuværende tidspunkt (efterår 2022) ikke nogen god dansk oversættelse af begrebet *prescriptive analytics*. Men det vi kan gøre er, at bryde begrebet op i sine to bestanddele "prescriptive" og "analytics" for på den måde at få en idé om, hvad det dækker over. For at introducere begrebet bedst, sætter vi det her ind i den kontekst, der hedder "business analytics"

### 1.1.1 Hvad er business analytics

Business Analytics som begreb blev, hvis ikke introduceret i, så i hvert fald gjort populært i Davenport og Harris (2007). Begrebet dækker over brugen af analytics til at fremme en forretnings konkurrencemæssige fordele. Det er blevet diskuteret bredt, også internationalt, hvorledes man mere præcist kan definere *analytics* og

den definition, som denne bog tager sit udgangspunkt i, er den, som kommer fra det amerikanske "Institute for Operations Research and the Management Sciences" (INFORMS):

> Analytics is the scientific process of transforming data into insights *for the purpose of making better decisions*. Analytics is always an action-driven approach. There is always a decision to be made when we look at doing analytics.

Der er altså en åbenlys forskel mellem analytics og så statistik som også, unægteligt, er en videnskabelig process, som bruges til at give indsigt ud fra data, nemlig det *handlingsorienterede*. Analytics bruges til at skabe indsigt ud fra data og denne opnåede indsigt skal så bruges til at drive beslutningstagning.

### 1.1.2 Hierarki inden for business analytics

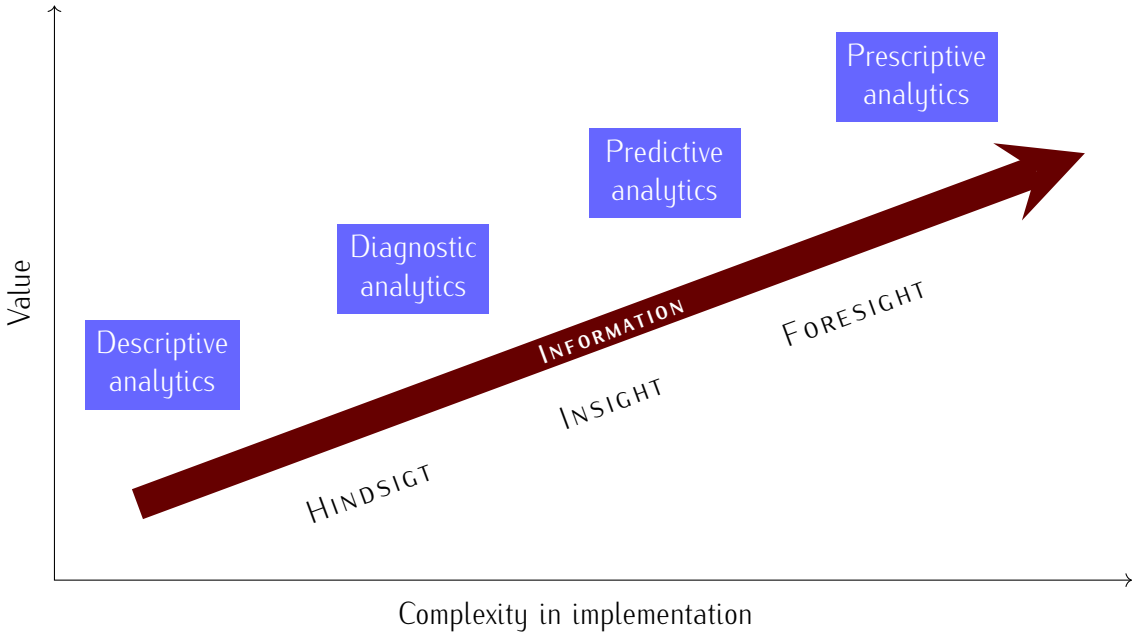
Business analytics kan groft sagt opdeles i (mindst) fire underkategorier af analytics; *descriptive analytics*, *diagnostic analytics*, *predictive analytics* og *prescriptive analytics*. Nedenfor vil der kort blive redegjort for hver underkategori.

Descriptive analytics, eller beskrivende analytics, er analytics, som bruger for eksempel data-minining og statistik til at give os indsigt i fortiden. Det vil altså sige analytics, som kan hjælpe med at besvare spørgsmål som "Hvad er sket?".

Diagnostic analytics, eller diagnosticerende analytics, er analytics, som bruger for eksempel machine learning teknikker som mønstergenkendelse (pattern recognition) og data-mining eller korrelationsanalyse til at få indsigt i, hvorfor en hændelse indtraf. Man kan altså besvare spørgsmål som "Hvorfor skete det?".

Predictive analytics, eller forudsigende/prædiktiv analytics, er analytics, som bruger for eksempel statistik, forecasting og machine learning/regression til at give en forståelse af, hvordan fremtiden kan komme til at se ud. Altså besvarer predictive analytics spørgsmål som "Hvad kan komme til at ske?".

Prescriptive analytics, eller ordinerende/præskriptiv analytics, er analytics, som bruger optimering og/eller simulation til at foreslå en eller flere mulige handlingsmuligheder. Det vil sige, at prescriptive analytics besvarer spørgsmål som "Hvad skal vi gøre?".




**Figur 1.1: Dette er en tilpasning baseret på Davenport og Harris 2007. Figurens tekst er holdt på engelsk for at bibeholde muligheden for at fremsøge yderligere information på baggrund af begreberne.**

Det er særdeles vigtigt at gøre sig klart, at hver gang man tager et niveau ned af denne liste af analytics underkategorier, så er man afhængig af den forrige. For eksempel kan vi sjældent lave gode diagnostikker, hvis vi ikke kan beskrive problemet og vi kan heller ikke ordinere løsninger, hvis vi ikke ved hvordan fremtiden kan se ud.

Man kan tænke på business analytics som kombinationen af de ovenforstående analytics undergrupper, hvilket leder til en illustration som i Figur 1.1. I denne figur ser vi, at som kompleksiteten i forhold til implementering i en virksomhed stiger, så stiger den potentielle værdi også. Figuren skal ikke forstås således, at prescriptive analytics er sværere i sig selv end descriptive analytics. Derimod skal man notere sig, at for at man kan implementere prescriptive analytics succesfuldt, kræver det succesfuld implementering af de underliggende analytics discipliner, hvorved kompleksiteten følgeligt stiger, efterhånden som lagene lægges på.

Vi har altså nu sat begrebet prescriptive analytics ind i en "business analytics" kontekst og samtidigt fået defineret præcist hvad, der skal forstås ved *analytics*. Som bogens titel antyder, vil den hovedsageligt beskæftige sig med prescriptive analytics, og særligt matematisk modellering anvendt til prescriptive analytics. Bogen er opdelt i kapitler, som beskæftiger sig med problemstillinger med forskellige planlægningshorisonter og er blot et lille udpluk af den myriade af problemstil-

linger, hvor matematisk modellering kan benyttes. Formålet med denne opdeling er at skabe en eksempel-baseret tilgang til modellering, da det er forfatterens opfattelse, at det at modellere noget konkret i høj grad øger motivationen og dermed indlæringen. Bogen kan ikke nødvendigvis stå alene og for kurset på Aarhus BSS, Institut for Økonomi E2022vil der i tillæg blive brugt cases, yderligere noter samt videnskabelige artikler.

I denne version af bogen, er der tilføjet et nyt kapitel omkring Pythonprogrammering og en kort guide til modelleringssproget Pyomo (se kapitel 2). Dette kapitel er tilføjet i forbindelse med efterårssemmestret 2022 og derfor er det stadig et "ungt" kaiptel i den forstand, at der kan være ting, der skal ændres fremadrettet. Forfatteren håber på læserens forståelse. Tillige er et appendiks med fokus på matematisk notation og en smule teori omkring lineær programmering og blandet heltalsprogrammering tilføjet. Dette er ligeledes i en tidlig modningsfase og skal blot opfattes som en hjælp til læseren.

Efter endt studeren af denne bog er det overordnede mål, at den studerende

1. er i stand til at bruge matematisk modellering og programmering til at skabe indsigter baseret på data, som kan foreslå bedste eller i det mindste gode løsninger på konkrete problemstillinger .
2. er klar over, at prescriptive analytics ikke er en ø, der står alene: det data, som benyttes i modellerne har en oprindelse og som man så malerisk siger om matematiske modeller "garbage in, garbage out!".

God læse- og lærelyst

– Sune Lauth Gadegaard

# 2 Introduktion til Python og en let indføring i brugen af Pyomo

For at blive i stand til at modellere og ikke mindst løse matematiske optimeringsproblemer er det altafgørende at have adgang til et effektivt software set-up. Generelt set er der behov for følgende delelementer for at opnå denne effektivitet

1. Et *programmeringssprog*, som sætter brugeren i stand til at kommunikere med en computer.
2. Et *algebraisk modelleringssprog*, der sætter brugeren i stand til at oversætte matematiske modeller til et sprog computeren kan fortolke.
3. En *solver*, der som input tager en model og som output producerer en løsning af tilfredsstillende kvalitet (hvis en sådan kan findes inden for den afsatte beregningstid).

Der findes mange forskellige løsninger på disse tre ovenstående punkter. For eksempel har IBM udviklet en solver kaldet CPLEX, som kommer med sit eget dedikerede algebraiske modelleringssprog kaldet `OPL` og CPLEX kommer også med en API til Python kaldet `DOcplex`. En anden solver-producent kaldet Gurobi kommer med en dedikeret API til Python kaldet `gurobipy`. Både CPLEX, Gurobi og mange andre solver-producenter kommer også med API'er til andre programmeringssprog som for eksempel Java, `C`, `C++`, `C#`. Et problem med disse ofte meget effektive (hurtige!) API'er er, at de er dedikerede til netop én specifik solver: Et program skrevet til at benytte CPLEX kan ikke umiddelbart hurtigt omskrives til at bruge en af de andre solvere. Dette kan være et problem, da disse solver-producenter naturligvis sælger deres solvere med licens; skulle ens IT-infrastruktur ændres så man ikke længere har licens til CPLEX men til Gurobi i stedet, så skal man principielt genskrive hele sin kodebase.

I det følgende tages en anden tilgang hvor et Python-baseret open-source modelleringssprog udviklet specifikt til matematisk optimering vil blive introduceret.

Dette modelleringssprog hedder Pyomo. Pyomo er så at sige et lag, man lægger mellem udvikleren og solveren, som giver mulighed for at udskifte solveren med et "snuptag": Har man før brugt CPLEX og nu vil skifte til Gurobi ændres blot følgende ene linje

```
solver = pyomo.SolverFactory('cplex')
```

til

```
solver = pyomo.SolverFactory('gurobi')
```

Den resterende del af dette kapitel bruges til først at give en guide til installation af både Python, Pyomo og en solver. Dernæst følger en generel introduktion til programmering og særligt til programmering i Python. Denne del kan ikke stå alene som eneste oplæring i Pythonprogrammering, og læseren opfordres til at benytte nogle af de utallige gode tutorials til Python der ligger frit tilgængeligt på nettet. Herefter følger en introduktion til brugen af den specifikke pakke Pyomo. Slutteligt i indeværende kapitel er en (ufuldstændig) liste (se Tabel 2.6) med en række Pythonpakker, der kan være interessante for læseren af denne bog.

Den opmærksomme læser vil formodentlig lægge mærke til, at kodeeksemplerne i denne bog benytter *engelsk* som sprog på trods af at resten af bogen er skrevet på dansk. Dette skyldes, at overraskende mange programmer ikke understøtter de danske bogstaver (æ, ø og å). Derfor er det en god vane at få ind, at man koder på engelsk for at undgå ubehagelige fejl, som ofte er meget svære at gennemskue, men som blot kommer fra, at man har benyttet de tre danske bogstaver. Derfor er kodeeksemplerne og kommentarerne i disse alle lavet på engelsk.

## 2.1 Installation af Python og Pyomo

For at benytte Pyomo skal der først og fremmest være installeret en funktionsdygtig og opdateret version af Python på maskinen. Mens dette skrives, er den nyeste stabile version af Python version 3.10.6.

### 2.1.1 Installation af Python på Windows 10

For at installere Python på en Windows 10 maskine, skal man gå til https://www.python.org/downloads/windows/ og vælge den nyeste stabile version

af Python. I bunden af den næste side, skal man vælge den distribution der passer til maskinen. Dette vil typisk være "Windows installer (64)'". Når denne fil er hentet, dobbeltklikkes på filen og man skal herefter blot følge instruktionerne. OBS: Man skal sørge for, at der på den første side i installeren er sat et tjek i "Add Python 3.10 to PATH". Efter endt installation, kan man tjekke om alt er gået som det skal ved at åbne en command-promt og skrive `python3`. Hvis Python er korrekt installeret, åbnes Pythons interaktive interpreter. Skriv her efter for eksempel `2+2` og se, at Python er i stand til at udregne dette.

### 2.1.2 Installation af Python på Mac

På langt de fleste Mac-computere er der allerede installeret en version af Python. Spørgsmålet er nu, om det er den rigtige. En simpel måde at teste dette på er ved at åbne en terminal og køre kommandoen `python3 --version`. Hvis en version nyere end Python 3.9 er installeret, er du klar til at fortsætte. Ellers bør du opdatere din Python installation.

Er læseren Mac-bruger og endnu ikke erfaren bruger af package-manageren Homebrew, vil forfatteren af dette materiale hermed give en varm anbefaling. Ved at besøge `https://brew.sh/` kan man se hvorledes `Homebrew` installeres. Homebrew giver brugeren en særdeles strømlinet måde at installere og holde pakker og programmer opdateret på. Alt foregår fra terminalen, så det er hurtigt og effektivt. `Homebrew` kan gå hen og blive smart når du senere skal installere en solver.

### 2.1.3 Installation af Pyomo

Med installationen af Python kom også programmet `pip`. For at være sikker på at `pip` er korrekt installeret kan man åbne en terminal og køre kommandoen `python3 -m pip -version`. Hvis `pip` mod forventninger ikke er installeret, bedes dette udbedres inden der forsættes.

Akronymet `pip` står for "Pip installs packages". Ved hjælp af `pip` kan man installere nye pakker, som man ønsker at bruge i Python. For at forstå dette koncept, skal man vide, at Python er et programmeringssprog med et stort community bag, som konstant udvikler nye *pakker*, som giver brugeren af Python nye funktionaliteter. Generelt er det sådan i Python, at uanset hvad man ønsker at kode, så er der nogen, som har gjort det før dig, og der findes med stor sandsynlighed også en pakke til dit problem.

Her er fokus på pakken, som implementerer Pyomo, og denne installeres på følgende måde:

1. Åben en terminal (Mac) eller en command-promt (Windows)
2. Kør kommandoen `pip install pyomo`
3. Du har nu installeret Pyomo i dit system

### 2.1.4 Installering af solverne CBC og GLPK

To af de bedste free-ware solvere er *c*oin-or's *b*ranch-and-*c*ut solver kaldet `CBC` og *G*NU projektets *l*ineær *p*rogrammerings*k*it kaldet `glpk`. Disse kan installeres på følgende vis

Windows: Installation på Windows kan gøres som følger

CBC: For at installere `CBC` kan man benytte sig af det faktum at installation af et pakken `PuLP` også installerer `CBC`. Dermed kan man installere `PuLP` ved at køre følgende kommando i en command-promt "`pip install pulp`".

glpk: Installation af `glpk` er en kende mere tricky. Det anbefales at man følger vejledningen via dette link.

Mac: Installation på Mac er relativt overkommeligt såfremt man har installeret `Homebrew`:

CBC: I en terminal køres kommandoen "`brew install cbc`"

glpk: I en terminal køres kommandoen "`brew install glpk`"

Succesful installation af én af de to ovenfor anførte solvere er tilstrækkeligt for at løse alle opgaver og cases hørende til denne bog.

### 2.1.5 Installering af CPLEX eller Gurobi

Man vil opdage, at mange optimeringsproblemer, som involverer heltallige variabler, er særdeles svære at løse og dermed også tidskrævende. Dette er sandt selv med hvad der er alment accepteret som to af de bedste open-source solvere, nemlig `glpk` og `cbc`. Inden for området *machine learning* er de bedste open-source implementeringer af de brugte algoritmer (næsten) lige så gode som de

kommercielle. Ligeledes har store kommercielle firmaer som Microsoft og Google valgt at gøre deres software gratis tilgængeligt og i mange tilfælde også lavet softwaren open-source. Dette er desværre ikke tilfældet inden for matematisk optimering: de store kommercielle softwareudviklere har ikke gjort deres software gratis tilgængeligt og deres software er *meget* bedre end de bedste open-source implementeringer. Man vil altså i langt de fleste tilfælde se et markant speed-up af løsningen af sine modeller, hvis man skifter fra en open-source til en kommerciel solver. Desværre kan licenserne til disse være kostbare.

Heldigvis tilbyder de to solvere CPLEX og Gurobi hvad de kalder "Academic licenses". Det vil sige, at hvis man er indskrevet på eller på anden måde er affilieret med en videregående uddannelse, så kan man få tildelt et licens uden omkostninger. Hvis læseren finder dette interessant, kan man søge på hhv. "cplex academic initiative" eller "gurobi academic licens", hvorved man vil blive guidet til hvordan man opnår et licens, hvordan solveren installeres osv.. Man skal være særligt opmærksom på, at solveren skal tilføjes systemts path-variabel for at Pyomo senere kan gøre brug af dem.

### 2.1.6 Valg af IDE

Der er et hav af værktøjer til udvikling af Pythonkode og mange af dem er både rigtig gode og gratis. Der er ingen restriktioner på hvilket IDE (integrated development environment) man kan benytte i dette kursus, og læseren skal blot vælge én som falder i dennes smag. To populære valg er *PyCharm* (som forfatteren benytter) og *Spyder*. En simpel google-søgning på "Best free python IDE 2022" vil producere en række sider, som diskuterer for og i mod for de forskellige gratis IDE'er.

## 2.2 Introduktion til programmering i Python

#### Check this out

##### List of coursesiover

Som før nævnt er Python et programmeringssprog og Pyomo er et Python-baseret modelleringssprog specielt udviklet til optimering. Dette afsnit giver en særdeles kort og overfladisk introduktion til programmering og programmering i Python og en mere detaljeret, omend stadig overfladiske, introduktion til Pyomo følger i sektion 2.4.

### 2.2.1 Hvad er programmering og hvad er Python

Python er et såkaldt *interpreted*, *object oriented*, *high level* programmeringssprog. At Python er high-level betyder, at afstanden fra udvikleren til de indre mekanismer og detaljer i computeren er relativt stor. Det vil samtidig sige, at udvikleren ikke nødvendigvis behøver at have nogen dybere forståelse for hvordan for eksempel hukommelse håndteres af systemet. På den måde er niveauet krævet af en nybegynder relativt lavt og man kan ofte overraskende hurtigt lære at udvikle endog meget komplicerede programmer.

Ved at påstå, at man hurtigt kan udvikle komplicerede programmer, er det implicit antaget, at læseren ved hvad et *program* er. Dette er dog ikke nødvendigvis givet! Et (computer) program er en mængde af *instruktioner*, som en computer bruger til at udføre en bestemt funktion. Man kan tænke på et program som en opskrift: programmet foreskriver *hvad* der skal gøres og i hvilken *rækkefølge* det gøres.

Det specifikke programmeringssprog giver udvikleren en måde at kommunikere med computeren. På samme måde som menneskesprog giver os mennesker en måde at kommunikere med hinanden på. Og som det er med menneskesprog som dansk, tysk, kinesisk og Khoisan, skal den måde man udtrykker sig på indeholde sætninger, som indeholder ord fra det rigtige sprog og som følger den korrekte syntaks og morfologi.

Et program er således en række instruktioner udtrykt i et bestemt sprog, som computeren *udfører* (execute på engelsk). De fleste programmer kræver mange på hinanden følgende instruktioner (det vil sige, at flere instruktioner følger efter hinanden: "åben datafil" → "læs datafil" → "rens data" osv.) og sådan et program kan blive særdeles uoverskueligt at læse for et menneske, hvis det blot er skrevet "ud i et". En måde at gøre op med dette problem på er, at man opdeler programmet i *funktioner*, som her kan tænkes som delelementer af en samlet mængde af instruktioner. Hvis man for eksempel ønsker at skabe sig et overblik over antallet af frugter på lager (det kan for nemheds skyld antages, at grønthandleren kun forhandler æbler, pærer og bananer), så kunne dette skrives op som følgende funktionsopdelte program i *pseudokode*

```
num_apples = count_apples ( )
num_pears = count_pears ( )
num_bananas = count_bananas ( )
total_fruit = num_apples+num_pears+num_bananas
print ( total_fruit )
```

Dette program har oprettet fire *variabler* kaldet `num_apples`, `num_pears`, `num_bananas` og `total_fruit`, som benyttes til at gemme værdier i. Der udover benytter koden sig af tre funktioner (`count_apples()`, `count_pears()` og `count_bananas()`), som udfører det hårde arbejde med at tælle frugterne. Til sidst lægges alle antallene sammen og gemmes i variable `total_fruit`, som slutteligt printes ved hjælpe af funktionen `print()`. Man skal her forestille sig, at der ligger adskille instruktioner gemt i de fire brugte funktioner. Ved at opdele programmer i funktioner, som *kalder* hinanden, kan man lave programmer, der er meget nemmere at læse for mennesker, og hvor *koden* (som instruktionerne også kaldes) senere kan genbruges i andre sammenhænge. For eksempel kunne man sagtens forestille sig, at det var godt givet ud, at isolere koden til `print()` funktionen, da man formegentlig kommer til at skulle printe andre ting i andre programmer.

## 2.3 Opbygning af Pythonkode og det første program

Pythonkode bliver mest elegant, hvis det opbygges i en række funktioner, som har hver sin funktionalitet. En speciel funktion er funktionen `main` der, som navnet angiver, er hoved-funktionen. Det er kutyme, at `main`-funktionen er den eneste funktion man kalder direkte, og så sørger `main`-funktionen for at kalde alle andre funktioner, som skal benyttes for at programmet har den ønskede funktionalitet. Al anden kode bør defineres i funktioner!

Lad os nu introducere vores første rigtige (korrekt vokabular og syntaks) program skrevet i Python (se Kodeeksempel 2.1).

**Kodeeksempel 2.1: Implementering af en første funktion i Python. Implementerer det klassiske "Hello world!" program.**

```
# Definition of the hello_world function
def my_first_function():
  print("Hello world!")

# Definition of the main function
def main():
  my_first_function()

# Run the main function
if __name__ == '__main__':
  main()
```

Dette program skal læses nedefra og opefter: Først kaldes `main`-funktionen. Læg her mærke til den lidt besynderlige notation brugt før `main`-funktionen kaldes, nemlig `if __name__=='__main__':`. Variablen `__name__` er en særlig variabel i Python, som sættes lig med `'__main__'` så snart koden køres (såfremt det ikke er kode der importeres). Dermed vil koden inden i `if`-statementet udføres da `__name__` er lig med `'__main__'`. Hvis man vil vide mere om denne konstruktion og hvorfor den er smart, kan man eksempelvis læse videre her.

For at forstå, hvad `main`-funktionen gør, flyttes fokus nu til linjen `def main():`. Denne linje definerer en ny funktion, som hedder `main`. Læg mærke til, at selve definitionen af de instruktioner, som `main`-funktionen skal udføre, er *indenteret* eller indrykket med et "tab". Læg yderligere mærke til, at linjen `def main():` *slutter med et kolon* - alle definitioner af funktioner/-procedurer skal afsluttes med et kolon. Selve definitionen af `main`-funktionen er her relativt simpel, da den blot *kalder* en anden funktion, nemlig funktionen `my_first_function`. For at gennemskue hvad denne funktion gør, bør læseren finde funktionens definition: `def my_first_function():`. Den følgende indrykkede linje består af endnu et funktionskald til `print("Hello world!")`. Og nu må læseren undre sig, da der ikke er nogen definition af denne funktion. Hvis man kører koden, vil man se, at der bliver printet "Hello world!" til Python konsollen. Dette er noget af det smukke ved et såkaldt high-level sprog som Python: `print`-funktionen er defineret af udviklerne af Python og man kan med glæde blot benytte sig af denne funktionalitet uden at bekymre sig om, hvorledes "Hello world!" kom fra koden til konsollen.

Funktionen `print` har et karakteristika, som de funktioner, der er defineret i dette eksempel ikke har: den tager et *argument*. Et argument til en funktion, er, så at sige, funktionens input. Muligheden for at give en funktion et argument er dog ikke forbeholdt på forhånd definerede funktioner; man kan selv definere funktioner, som tager argumenter. Tag for eksempel Pythonkode i Kodeeksempel 2.2:

**Kodeeksempel 2.2: Implementering af en funktion som tager et argument. Her er argumentet en `string`.**

```
# Definition of the "Hello" function function
def hello( name : str ):
  print("Hello " + name + ". Have a nice day!")

# Definition of the main function
def main():
```

```
  hello("Niels")
  hello("Peter")

# Run the main function
if __name__ == '__main__':
  main()
```

Her er `my_first_function()` skiftet ud med en funktion kaldet `hello`, som her tager et argument, som her kaldes `name`. Desuden er det blevet indikeret til funktionen at argumentet `name` skal være af *typen* `str`. Typen `str` er en såkaldt `string`, som bruges til at opbevare ord og sætninger (modsat tal og andre objekter). I under-sektion 2.3.1 gennemgås nogle af de mest almindelige typer og brugen af variabler i flere detaljer. Outputtet af ovenstående kode vil være

```
Hello Niels. Have a nice day!
Hello Peter. Have a nice day!
```

Læg mærke til, at man på den måde kan genbruge koden implementeret i funktionen `hello( name : str )` og blot ændre argumentet, som bliver givet til funktionen.

Lige som funktioner kan tage argumenter, kan de også *returnere* objekter. Det kan for eksempel være smart, at en funktion i stedet for at printe et resultat kan returnere dette resultat, så det kan bruges som input til andre funktioner (tænk på eksemplet hvor der blev talt frugter. Her blev antallet af frugter *returneret*). For at give et eksempel på en funktion, som både tager et argument og returnere en værdi betragt følgende Pythonkode

**Kodeeksempel 2.3: Implementering af en funktion som tager et argument (heltal) og returnerer en værdi af typen `bool`. Illustrerer brugen af "if-else"-konstruktionen.**

```
# Test whether argument is even or not. Returns truth value
def is_even( number : int ) -> bool:
  if (number % 2) == 0:
    return True
  else:
    return False

# Definition of the main function
def main():
  is_it_even = is_even( 42 )
  print ( is_it_even )

# Run the main function
```

```
if __name__ == '__main__':
  main()
```

I Kodeeksempel 2.3 tager funktionen `is_even( number : int )` et heltal (`int`) som argument og returnerer et objekt af typen `bool` (kort for `boolean`, altså en variabel, som kan tage værdierne `True` og `False`). Returneringstypen indikeres med en pil (`->`) efterfulgt af typen (`bool`). Funktionen returnerer `True` hvis argumentet er et lige tal (deleligt med 2, eller ækvivalent hvis resten ved division med 2 er lig med 0) og *ellers* returneres `False`. Det vil altså sige, at funktionen `is_even()` tager et helal (int) som argument, tjekker om tallet er lige, og hvis dette er tilfældet returneres `True`. Hvis det derimod ikke er sandt, returneres `False`. Man skal her notere sig, at funktionen kun returnerer noget, hvis eksekveringen af koden når et `return`-statement. Så snart koden når et return-statement forlades funktionen. Det vil sige, at man skal placere sine return-statements med forsigtighed!

Læseren bør også lægge mærke til "if-else" konstruktionen, som benyttes i `is_even( )`-funktionen. Dette er en særdeles brugt konstruktion, der skal læses som følger: Hvis det logiske udsagn der følger efter `if` er sandt (`True`), så eksekveres koden indenteret under `if`. Hvis udsagnet derimod er falskt (`False`), så eksekveres koden indenteret under `else`. Mere generelt kan en "if-else" konstruktion udvides til en "if-else if-else" konstruktion som følger

```
# A silly Python function, illustrating if, else if, else
def silly_func(num: int) -> str:
  if num < 0:
    return "negative"
  elif num <= 5:
    return "0 <= num <= 5"
  elif num <= 10:
    return "5 < num <= 10"
  else:
    return "num > 10"
```

Udtrykket `elif` er kort form af "else if". På den måde, kan man opbygge en række tjek, som en sekvens af `if`, `elif` og `else`, hvor den sidste `else` så at sige, håndtere alle de tilfælde, som ikke er dækket af `if` og `elif`. Man skal tænke disse strukturer som en si, som bliver finere og finere: I funktionen `silly_func` tjekkes først om argumentet `num` er mindre end 0, hvis dette er tilfældet, returneres sætningen "negative". Ellers tjekkes om `num` er mindre end

5. I så tilfælde udprintes sætningen "0 <= num <= 5". Læg mærke til, at selvom der kun tjekkes om `num` er mindre end eller lig med 5, vides det også, at `num` er større end eller lig med 0. Dette skyldes, at ellers havde den første del af "if-elif-else" konstruktionen været aktiveret. Det anbefales læseren at implementere `silly_func` i Python og kalde den fra en `main` funktion for eksempel med argumenterne -1, 2, 5, 9 og 100. Dette kan være en god måde at danne sig et overblik over, hvordan disse konstruktioner fungerer.

### 2.3.1 Typer og variabler

I det foregående afsnit blev en række forskellige *typer* allerede introduceret. Nærmere bestemt typerne `string`, `int` og `bool`. Men disse typer blev brugt uden nogen videre definition og ved blot at hinte til ideen om en `variabel`. For at forstå typer skal variabler først defineres mere formelt

**Definition 2.1** (Variabel)**.** Variabler bruges til at gemme information, som et program kan tilgå og manipulere. Variabler giver også en måde hvor data kan navngives med beskrivende navne, så programmer bliver nemmere at forstå for en menneskelig læser.

Det kan være instruktivt at tænke på variabler som containere, der kan indeholde information. Deres eneste formål er, at de skal navngive og gemme information/data i computerens hukommelse. Variablerne kan benyttes i hele deres *scope*.

På den måde er der altså en forskel på en såkaldt *beslutningsvariabel* kendt fra optimeringsproblemer og en programmeringsvariabel. De variabler man benytter sig af i et computerprogram bliver tildelt værdier, og er altså ikke noget man lader en "black-box" solver finde værdier til.

Som nævnt findes der en række forskellige *typer* af variabler i Python og man kan som udvikler selv definere nye typer efterhånden, som man skal bruge dem. I Tabel 2.1 ses en oversigt og nogle få meget brugte typer.

### 2.3.2 Beskrivende navne på variabler og funktioner

Variabler bruges ofte til at sektionere programmer og til at holde styr på mellemregninger. Det kan være godt for læsbarheden af koden, hvis man benytter sig af beskrivende navne på variabler (og funktioner). På den måde gør man det nemmere selv at genbesøge sin kode og det gør det muligt for andre, at benytte og vedligeholde den. For at illustrere dette, antag at en Pythonudvikler ønsker at

Tabel 2.1: Oversigt over nogle af de mest brugte typer i Python.

| Type | Beskrivelse |
|---|---|
| `bool` | Type, som kun kan indeholde to værdier, nemlig `True` og `False`. |
| `int` | Type, som kan indeholde heltalsværdier. Det vil sige, værdier fra mængden $\mathbb{Z}$. |
| `float` | Type, som kan indeholde kommatal. Det vil sige tal som 2.7, 4.40604571360 og 42.0 |
| `str` | Type, som kan indeholde en sekvens af bogstaver. For eksempel `'a'`, `'Hello world'` og `'42'`. Denne type kaldes en *string*. |
| `list` | Type, som kan indeholde en række enheder. For eksempel er `['James Bond','banana','apple']` en Python liste af typen `str` og `[23,34,45]` er en liste af hele tal. Læg særligt mærke til, at en liste omslutter sine elementer med firkantede parenteser. |
| `set` | Type, som kan indeholde en række af *unikke* elementer. For eksempel er `{2,5,9}` en mængde af tallene 2, 5 og 9. Læg mærke til, at der er tale om unikke elementer, så `{2,2,3,4,4,6,6}={2,3,4,6}`. Man bruger modsat lister Tuborg-parenteser til at beskrive et set i Python. Mængder, eller sets, kan være særligt smarte fordi de tillader mængdeoperationer som forenings- og snitmængder osv.. |
| `dict` | En dict er en speciel type, som bruges meget i Python. `dict` er kort for *dictionary* og en sådan består af en samling af såkaldte "key-value"-par. Hver key-value par linker key'en til de tildelte værdier. For at give et eksempel, betragt `car = { 'brand':'fiat', 'model':'multipla', 'year':1995, 'cool':False }`. Man kan altså gemme mange slags typer i en `dict` og man kan efterfølgende tilgå hvert element i `dict`'en som følger `car['cool']`. |

indlæse data til et *uncapacitated lot sizing problem* (se kapitel 5). Antag envidere at en funktion `readData()`, der læser dataet fra en fil, er implementeret og betragt Pythonkoden i Kodeeksempel 2.4.

Kodeeksempel 2.4: Eksempel på dårlig navngivning af variabler og funktioner

```
datafile = './datafile'
data =  readData(datafile)
```

Der er tale om så generiske navne på de involverede variabler og funktioner, at der kunne være tale om et hvilket som helst program. Man kunne endda tale om, at `data` formegentlig er det dårligste navn til en variabel der overhovedet findes, da alle variabler vil indeholde en eller anden form for data. I Kodeeksempel 2.5 illustreres mere beskrivende navne til variabler og funktioner

Kodeeksempel 2.5: Eksmepel på bedre navngivning af variabler og funktioner

```
uncapLotSizingDataFile = './datafile'
```

```
uncapLotSizingData =  readUncapLotSizingData(
   uncapLotSizingDataFile)
```

Dette kræver lidt mere arbejde under implementeringen, men gør livet meget nemmere siden hen.

## 2.4 Introduktion til Pyomo

Pyomo er som nævnt en pakke i Python, som implementerer et algebraisk modelleringssprog, der kan benyttes til at implementere matematiske programmeringsmodeller samt til at *interface* med en række forskellige solvere. Der er række ting, som kan fremhæves ved Pyomo:

1. I dag benyttes Python i meget stort omfang inden for data science og business intelligence. Pyomo giver Pythonbrugeren mulighed for også at implementere optimeringsmodeller.
2. Pyomo stiller en relativt letlæseligt syntaks til rådighed for brugeren, der ønsker at implementere og løse matematiske programmeringsmodeller.
3. Pyomo er ikke begrænset til lineære modeller som en anden populær optimeringspakke i Python, nemlig `PuLP`, er.
4. Pyomo gør det nemt at interface med en lang række af de mest udbredte solvere. Dermed leverer Pyomo en relativ agil infrastruktur.

Disse punkter kombineret gør, at Pyomo er valgt som modelleringssprog i denne bog. Forfatteren vil gerne gøre opmærksom på, at dette er et tilvalg af Pyomo mere end det er et fravalg af andre pakker. Andre pakker kan være akkurat lige så gode, men Pyomo har altså virket mest attraktiv for forfatteren.

### 2.4.1 Opbygning af Python kode til løsning af et optimeringsproblem

For at benytte en standardiseret opbygning af Pythonkoden vil denne sektion beskrive hvorledes denne bog forslår at opbygge koden. I Kodeeksempel 2.6 ses et eksempel på opbygning af koden til løsning af en model i Python.

Først importeres Pyomopakken således af funktioner og objekter fra denne pakke er til rådighed. Dernæst defineres fem funktioner, som her har fået generiske navne: `readData`, `buildModel`, `solveModel`, `displaySolution` og

Kodeeksempel 2.6: Eksempel på opbygning af koden til løsning af optimeringsproblemer

```
import pyomo.environ as pyomo
def readData(filename:str)-> dict:
  # implementation of function

def buildModel(data:dict)->pyomo.ConcreteModel():
  # Implementation of function

def solveModel(model:pyomo.ConcreteModel()):
  # Implementation of function

def displaySolution(model:pyomo.ConcreteModel()):
  # Implementation of function

def main():
  data = readData('pathToTheDataFile')
  model = buildModel(data)
  solveModel(model)
  displaySolution(model)

if __name__ == '__main__':
  main()
```

`main`. Funktionen `main` kalder de andre funktioner i den rigtige rækkefølge. Først indlæses data fra en fil som indeholder dataet, der skal bruges til at bygge modellen. Dataet gemmes i variablen `data` (som nævnt er et dårligt, men generisk navn). Herefter kaldes `buildModel` med argumentet `data` - dermed får funktionen `buildModel` adgang til det data, der definerer instansen. Funktionen `buildModel` returnerer en såkaldt `ConcreteModel` som er et Pyomoobjekt, der indeholder den model, der er bygget i `buildModel`-funktionen. Den gemte model gemmes i variablen kaldet `model`. Modellen gives herefter som argument til funktionen `solveModel`, som implementerer koden, der skal til for at løse modellen. Herefter kaldes funktionen `displaySolution` med `model` som argument. Formålet med denne funktion er at printe/illustrere løsningen på en for et menneske læselig facon. Som det fremgår af Kodeeksempel 2.6 er den egentlige implementering af de fire centrale funktioner udeladt, da denne afhænger af det specifikke problem man vil løse.

I de følgende sektioner vil en række eksempler blive illustreret hvor funktionerne oven for bliver implementeret. Funktionen `main` forbliver den samme og navnene på funktioner forbliver ligeledes uforanderlige for at øge læsbarheden.

### 2.4.2 Et første Pyomo eksempel

I dette lille eksempel bliver følgende lineære program implementeret og løst:

$$\begin{aligned} \max\ & 20x_1 + 10x_2 \\ \text{s.t.:}\ & x_1 + 2x_2 \leq 120 \\ & x_1 + x_2 \leq 90 \\ & x_1 \leq 70 \\ & x_2 \leq 50 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Dette lineære program kan også skrives på formen

$$\begin{aligned} \max\ & cx \\ \text{s.t.:}\ & Ax \leq b \\ & x_1 \leq 70 \\ & x_2 \leq 50 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Hvor $c = (20, 10)$, $b = (120, 90)$ og

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$$

For at gøre kodeeksemplerne mere overskuelige benytter vi denne sidste opskrivning.

Da datamængden er relativt lille her, vil dataet blot blive defineret direkte i `readData`-funktionen. Dette illustreres i Kodeeksempel 2.7

**Kodeeksempel 2.7: Implementering af `readData`-funktionen for eksemplet i under-sektion 2.4.2**

```
def readData(filename: str) -> dict:
  # implementation of readData
  data = dict()
  data['c'] = [20, 10]
  data['A'] = [[1, 2], [1, 1]]
  data['b'] = [120, 90]
  data['boundX1'] = 70
  data['boundX2'] = 50
  return data
```

Det næste der skal gøres er at implementere `buildModel`-funktionen, hvilket er det mest centrale punkt i dette eksempel. Implementeringen af denne funktion kan ses i Kodeeksempel 2.8.

**Kodeeksempel 2.8: Implementering af `buildModel`-funktionen for eksemplet i under-sektion 2.4.2**

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Create the model object
  model = pyomo.ConcreteModel()
  # Copy the data to the model object
  model.c = data['c']
  model.A = data['A']
  model.b = data['b']
  model.boundX1 = data['boundX1']
  model.boundX2 = data['boundX2']
  # Define the variables
  model.x1 = pyomo.Var(within=pyomo.NonNegativeReals, bounds=(0,
     model.boundX1))
  model.x2 = pyomo.Var(within=pyomo.NonNegativeReals, bounds=(0,
     model.boundX2))
  # Define the objective function
  model.obj = pyomo.Objective(expr=model.c[0]*model.x1 + model.c
     [1]*model.x2, sense=pyomo.maximize)
  # Add the constraints
  model.cst1 = pyomo.Constraint(expr=model.A[0][0]*model.x1 +
     model.A[0][1]*model.x2 <= model.b[0])
  model.cst2 = pyomo.Constraint(expr=model.A[1][0]*model.x1 +
     model.A[1][1]*model.x2 <= model.b[1])
  return model
```

Implementeringen af `buildModel`-funktionen kræver en række kommentarer. Først og fremmest er det første, der gøres, at oprette en variabel kaldet `model` og denne sættes lig med `pyomo.ConcreteModel()`. Det betyder, at variablen `model` herefter indeholde en, indtil videre, tom Pyomomodel. Man kan tænke på denne model som følgende optimeringsproblem

$$\min$$

$$\text{s.t.:}$$

Som default er en Pyomomodel et minimeringsproblem og man skal, som senere illustreret, huske at ændre dette i fald man ønsker at maksimere.

Efter initialisering af `model`-variablen kopieres dataet fra `data`-dictionary'en over i modellen. Den første linje af denne del af koden består i oprettelsen af en variabel (ikke beslutningsvariabel) i `model`-objektet, der hedder `c`. Denne sæt-

tes lig med objektfunktionskoefficienterne, der er gemt i `data`-variablen. Samme procedure benyttes til at kopiere data til `model`-objektet.

Efter kopiering af data oprettes modellens to variabler. En variabel defineres ved at specificere hvilken type variablen er ved at sætte `within` lig med den ønskede type. Endvidere kan man specificere en nedre og en øvre grænse for variablen hvis man kender denne ved at sætte `bounds` lig med disse. Ved at specificere grænser på variablerne undgås det, at tilføje begrænsningerne $x_1 \leq 70$ og $x_2 \leq 50$ eksplicit.

Når variablerne er defineret, kan objektfunktionen defineres. Der oprettes endnu en variabel i `model`-objektet, her kaldet `obj`. Denne sættes lig med et `pyomo.Objective` objekt. I dette objekt specificeres et *expression* som sættes lig med den faktiske objektfunktion. Her skal man være opmærksom på, at en Pythonliste (som objektfunktionskoefficienterne er gemt i) er indekseret fra 0 og ikke fra 1. Efter at objektfunktionen er defineret, specificeres det, at denne skal maksimeres ved at sætte objektfunktionens `sense` lig med `pyomo.maximize`.

Slutteligt tilføjes de to begrænsninger én ad gangen ved hjælp af et objekt af typen `pyomo.Constraint`. Når disse er tilføjet returneres den nu færdigbyggede model.

Med modellen implementeret skal den løses hvorfor `solveModel` skal implementeres (se Kodeeksempel 2.9).

**Kodeeksempel 2.9: Implementering af `solveModel`-funktionen for eksemplet i under-sektion 2.4.2**

```
def solveModel(model: pyomo.ConcreteModel()):
  # Set the solver
  solver = pyomo.SolverFactory('glpk')
  # Solve the model
  solver.solve(model, tee=True)
```

Først defineres en variabel kaldet `solver` og denne specificeres til en specifik solver fra Pyomos `SolverFactory`. I dette eksempel benyttes solveren `glpk` (husk at vælg en solver, som er installeret på det pågældende system). Dernæst løses modellen ved at kalde `solve`-funktionen fra `solver` variabel. Denne funktion tager modellen som argument og i dette tilfælde er en option kaldet `tee` sat til `True`. Det betyder, at solveren skriver til Pythonprompten løbende, så man kan følge med i, hvad den laver.

Efter solveren har løst modellen, er det tid til at tilgå løsningen. Dette gøres ved at implementere `displaySolution`-funktionen (se Kodeeksempel 2.10).

**Kodeeksempel 2.10: Implementering af `displaySolution`-funktionen for eksemplet i undersektion 2.4.2**

```
def displaySolution(model:pyomo.ConcreteModel()):
  # Print solution information to prompt
  print('Optimal objective function value =',
    pyomo.value(model.obj))
  print('Optimal value for x1 =',
    pyomo.value(model.x1))
  print('Optimal value for x2 =',
    pyomo.value(model.x2))
  print('Slack for cst1 =', model.cst1.uslack())
  print('Slack for cst2 =', model.cst2.uslack())
```

De tre første kommandoer i funktionen er nærmest identiske, og kun den første vil derfor blive gennemgået. Her printes (vha. `print`-funktionen) en string, der fortæller at her printes den optimale objektfunktionsværdi. I samme print-statement tilføjes værdien af `model.obj` variablen. For at tilgå dennes værdi (ikke blot variable i sig selv), bruges funktionen fra Pyomo kaldet `value`. Som navnet indikerer, returnerer denne funktion *værdien* af variablen.

I de to sidste linjer printes slacket for de to begrænsninger. Her bruges funktionen `uslack()`, som returnerer slacket til begrænsningens øvre grænse (uslack er kort for upper slack). Hvis man har begrænsninger, som er af typen *større end eller lig med* benyttes `lslack()` i stedet for at få slacket til den nedre grænse (lslack er kort for lower slack). Outputtet fra `displaySolution`-funktionen er givet nedenfor:

```
Optimal objective function value = 1600.0
Optimal value for x1 = 70.0
Optimal value for x2 = 20.0
Slack for cst1 10.0
Slack for cst2 0.0
```

### 2.4.3 Et eksempel med flere variabler

Det skulle være klart fra eksemplet i Afsnit 2.4.1 hvordan en variabel defineres, hvorledes en objektfunktion tilføjes og hvorledes begrænsninger defineres og tilføjes.

I dette eksempel vil fokus være på hvorledes summer med flere variabler kan implementeres i Pyomo. Forestiller man sig, at et optimeringsproblem har 100, 1000 eller endnu flere variabler, vil det være meget omstændeligt at skulle opskrive eksempelvis en objektfunktion besående af en sum af disse mange variabler

og deres tilhørende koefficienter. Pyomo stiller heldigvis en funktion til rådighed kaldet `sum` som automatiserer denne proces. Funktionen `sum` har følgende syntaks: `sum( summand[i] for i in someSet )`. For at illustrere syntaksen misbruges notationen en smule og det haves at

$$\sum_{i \in I} a_i =: \texttt{sum(a[i] for i in I)}$$

For sammensatte summer haves blot

$$\sum_{i \in I} \sum_{j \in J} a_{ij} =: \texttt{sum(a[i][j] for i in I for j in J)}$$

Denne funktionalitet er særdeles vigtig for at kunne implementere større optimeringsmodeller.

En anden funktionalitet er forbundet med oprettelsen af variabler. I eksemplet i Afsnit 2.4.1 oprettedes variablerne én af gangen. Men har man igen tusindvis af variabler, er dette ikke en farbar vej. I stedet kan man udnytte, at en hel liste af variabler kan defineres over en mængde/range/dictionary. I eksemplet i dette afsnit definerer vi en liste af variabler over et range. Men først skal problemet der skal løses defineres.

Her er tale om et såkaldt *knapsack*-problem givet ved

$$\begin{aligned} \max \quad & \sum_{i \in I} p_i x_i \\ \text{s.t.:} \quad & \sum_{i \in I} c_i x_i \leq B \\ & x_i \in \{0, 1\}, \qquad \forall i \in I \end{aligned}$$

Her er $I$ en mængde af *items* som man skal vælge imellem. Hvert item $i \in I$ har en *profit* givet ved $p_i$ og en omkostning $c_i$. Variablerne $x_i$ er binære og antager værdien 1 hvis og kun hvis item $i$ vælges. Formålet er således, at udvælge en delmængde af $I$ som samlet har den højeste profit under bibetingelse af, at budgettet på $B$ enheder overholdes. Dette konkrete eksempel har 10 items med data specificeret i Tabel 2.3. I dette eksempel, er `main`- og `solveModel`-funktionerne identiske med funktionerne med samme navne i Afsnit 2.4.2. Derfor genimplementeres disse ikke her. Det betyder, at `readData`-funktionen skal implementeres hvilket er gjort i Kodeeksempel 2.11.

Tabel 2.3: Data til knapsack eksemplet fra Afsnit 2.4.3.

| Items | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Profit | 26 | 98 | 85 | 98 | 82 | 76 | 75 | 46 | 33 | 34 |
| Omkostning | 20 | 34 | 48 | 52 | 74 | 50 | 37 | 20 | 19 | 25 |
| Budget | 125 | | | | | | | | | |

Kodeeksempel 2.11: Implementering af `readData`-funktionen for eksemplet i under-sektion 2.4.3

```
def readData(filename: str) -> dict:
  # implementation of readData
  data = dict()
  data['p'] = [26,98,85,98,82,76,75,46,33,34]
  data['c'] = [20,34,48,52,74,50,37,20,19,25]
  data['B'] = 125
  return data
```

Her defineres de tre data-elementer i problemet nemlig profitten, omkostningerne og budgettet. Disse gemmes i en dictionary kaldet `data`.

Herefter kan `buildModel`-funktionen implementeres, da data nu er tilgængeligt. Dette gøres i Kodeeksempel 2.12.

Kodeeksempel 2.12: Implementering af `buildModel`-funktionen for eksemplet i under-sektion 2.4.3

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Create the model object
  model = pyomo.ConcreteModel()
  # Copy the data to the model object
  model.p = data['p']
  model.c = data['c']
  model.B = data['B']
  # Create a range from 0 to (and not including) the length of p
  model.items = range(0,len(data['p']))
  # Define the variables
  model.x = pyomo.Var(model.items, within=pyomo.Binary, bounds
     =(0,1))
  # Define the objective function
  model.obj = pyomo.Objective(expr=sum(model.p[i]*model.x[i] for
      i in model.items), sense=pyomo.maximize)
  # Add the constraint
  model.budgetCst = pyomo.Constraint ( expr=sum( model.c[i]*
     model.x[i] for i in model.items) <= model.B)
  return model
```

Som i forrige afsnit, oprettes først et `ConcreteModel` objekt som gemmes i `model`-variablen. Herefter kopieres data fra `data`-dictionary'en over i `model`-variablen. For at have et range at definere variablerne over, oprettes en variabel i `model`-objektet her kaldet `itmes`. Denne variabel holder et range som går fra 0 til (og ikke med) længden (`len()`-funktionen benyttes) af listen der indeholder profitten på hvert af de 10 items. Det vil sige, at `model.items=range(0,10)`. Læg mærke til det elegante i, at dette range (`model.items`) afhænger af størrelsen på det data, der gives som argument til `buildModel`-funktionen. Havde længden af listen gemt i `data['p']` bestået af 20 værdier, havde vi i stedet haft, at `model.item=range(0,20)`. På den måde er modelopbygningen gjort uafhængig af dataet i den forstand, at vi kan komme med hvilket som helst knapsack problem af en hvilken som helst størrelse og løse det med denne implementering.

Efter data er tilføjet til `model`-objektet skal beslutningsvariablerne tilføjes. Dette gøres ved at tilføje en variabel for hvert element i det range der blev gemt i `model.items`. I Kodeeksempel 2.8 tilføjedes én variabel af gangen med syntaksen

```
model.x1 = pyomo.Var(within=pyomo.NonNegativeReals, bounds=(0,
    model.boundX1))
```

Nu tilføjes altså alle nødvendige variabler på én gang med syntaksen

```
model.x = pyomo.Var(model.items, within=pyomo.Binary, bounds
    =(0,1))
```

Her har `pyomo.Var()` funktionen fået et ekstra argument, der specificere hvor mange variabler, der skal oprettes. I dette tilfælde skal der altså være en variabel for hvert element i ranget `model.items`. På denne måde, er der adgang til indekserede variabler, således at man eksempelvis kan tilgå variablen $x_2$ ved `model.x[2]`. Læg mærke til, at variablerne er defineret til at være af typen `pyomo.Binary` da variabler kun kan tage værdier i $\{0, 1\}$.

Muligheden for indeksering af variabler betyder, at objektfunktionen kan tilføjes på elegant vis ved at benytte den føromtalte `sum`-funktion i Python. Objektfunktionen består altså af summen over `model.items` af ledende givet ved produktet mellem `model.p[i]` og `model.x[i]`. Læg mærke til, at objektfunktionens `sense` igen er specificeret til maksimering.

Efter at objektfunktionen er tilføjet, kan modellens ene begrænsning tilføjes. Igen benyttes `sum`-funktionen til at formulere venstresiden af begrænsningen. Slutteligt returneres `model` objektet.

For at afslutte dette eksempel implementeres `displaySolution`-funktionen for, at det bliver muligt at inspicere en optimal løsning. Implementeringen af `displaySolution` starter med, at den optimale objektfunktionsværdi printes sammen med en beskrivende tekst (se Kodeeksempel 2.13).

**Kodeeksempel 2.13: Implementering af `displaySolution`-funktionen for eksemplet i undersektion 2.4.3**

```
def displaySolution(model:pyomo.ConcreteModel()):
  # Print solution information to prompt
  print('Optimal objective function value =', pyomo.value(model.
     obj))
  print('Optimal values for the x-variables')
  for i in model.items:
    print('x[{}]={}'.format(i,pyomo.value(model.x[i])))
```

Dernæst printes en linje til prompten, som anviser, at herefter følger de optimale værdier for $x$-variablerne. For at printe disse værdier til brugeren, benyttes et *for loop*, som gennemløber alle items $i$ i `model.items`. For hvert item printes hvilken $x$-variabel, der er tale om og hvilken værdi denne antager i en optimal løsning. Her er `.format()`-funktionen brugt til at indsætte værdierne for `i` og `pyomo.value(model.x[i])` i den string, der printes. Hvorledes outputtet præcist er formatteret, er mindre vigtigt. Outputtet for denne specifikke funktion er givet ved

```
Optimal objective function value = 275.0
Optimal values for the x-variables
x[0]=0.0
x[1]=1.0
x[2]=0.0
x[3]=1.0
x[4]=0.0
x[5]=0.0
x[6]=0.0
x[7]=1.0
x[8]=1.0
x[9]=0.0
```

Tabel 2.5: Data til $p$-median-eksemplet fra Afsnit 2.4.3.

| | | Efterspørgselspunkter | | | | |
|---|---|---|---|---|---|---|
| | $c_{ij}$ | 1 | 2 | 3 | 4 | 5 |
| Lokationer | 1 | 10 | 17 | 20 | 15 | 14 |
| | 2 | 10 | 20 | 11 | 10 | 12 |
| | 3 | 16 | 15 | 19 | 11 | 11 |
| | 4 | 13 | 19 | 11 | 20 | 12 |
| | 5 | 15 | 19 | 13 | 20 | 14 |
| | $p = 2$ | | | | | |

## Et eksempel med mange variabler og mange begrænsninger

I dette eksempel implementeres et såkaldt *p-median problem*. Problemet kan beskrives som følger: En mængde af mulige placeringer for faciliteter $I$, med $|I| = n$, samt en mængde af efterspørgselspunkter $J$, med $|J| = m$, er givet. Der skal så placeres $p$ faciliteter på de $n$ mulige placeringer på en sådan måde, at omkostningen ved at servicerer de $m$ efterspørgselspunkter minimeres. Det antages, at omkostningen ved at servicerer efterspørgselspunkt $j$ fra en facilitet placeret på lokation $i$ er givet ved $c_{ij} > 0$. Samtidig antages det, at hvis der ikke placeres en facilitet på lokation $i$ så kan ingen efterspørgselspunkter serviceres fra denne lokation.

Dette problem kan formuleres som følger: Lad $y_i$, for $i \in I$, være en binær variabel som antager værdien 1 hvis og kun hvis en facilitet placeres på lokation $i$. Lad endvidere $x_{ij}$, for $i \in I$ og $j \in J$, være en binær variabel, som er lig med 1 hvis og kun hvis efterspørgselspunkt $j$ serviceres fra lokation $i$. Da kan problemet formuleres som (se kapitel 4 for en mere uddybende forklaring af problemet)

$$\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{i \in I} x_{ij} = 1, && \forall j \in J && (2.1) \\
& x_{ij} \leq y_i, && \forall i \in I, j \in J && (2.2) \\
& \sum_{i \in i} y_i = p, \\
& x_{ij}, y_i \in \{0, 1\}, && \forall i \in I, j \in J
\end{aligned}$$

Det specifikke instans, der skal løses her, er givet i Tabel 2.5.

Igen er `main`-funktionen og `solveModel`-funktionen identiske med eksemplet i Afsnit 2.4.1. Derfor er første skridt at implementere `readData()`-funktionen, som igen blot hard-koder dataet i stedet for at læse dette fra en fil (se Kodeeksempel 2.14).

**Kodeeksempel 2.14: Implementering af `readData`-funktionen for eksemplet i sektion 2.4.3**

```
def readData(filename: str) -> dict:
  # implementation of function
  data = dict()
  data['p'] = 2
  data['n'] = 5
  data['m'] = 5
  data['c'] = [[10, 17, 20, 15, 14],
        [10, 20, 11, 10, 12],
        [16, 15, 19, 11, 11],
        [13, 19, 11, 20, 12],
        [15, 19, 13, 20, 14]]
  return data
```

Herefter implementeres `buildModel`-funktionen, som er det springende punkt i dette afsnit, da der her bliver tilføjet grupper af begrænsninger. Implementeringen ses i Kodeeksempel 2.15.

Som med de andre eksempler oprettes først et `ConcreteModel`-objekt hvortil data kopieres og i dette eksempel oprettes to ranges, nemlig `model.locations` og `model.customers` (customers bruges som oversættelses af efterspørgselspunkter).

Som det næste defineres de to mængder af variabler. Først defineres $x$-variablerne og `pyomo.Var()`-funktionen har fået endnu et argument, således at der oprettes en $x$-variabel for hver *kombination* af element i `model.locations` og `model.customers`. Man skal være opmærksom på, at når man tilgår variabler med mere end én dimension i Pyomo, så er det ikke som med en Python-matrix. Hvis `A` er en matrix i Python defineret som en list af lister, så vil man kunne tilgå den $i$'te række og $j$'te søjle via `A[i][j]`. Men med Pyomo-variabler, tilgås det $(i, j)$'te element af en variabel kaldet `x` via `x[i,j]`!

Denne lille krølle med at tilgå det $(i, j)$'te element i en beslutningsvariabel ses brugt ved tilføjelsen af objektfunktionen. Læg her mærke til, at der summeres over `model.c[i][j]` ganget med `model.x[i,j]` (`c` er en liste af lister og `x` er en beslutningsvariabel). Læg også mærke til, at der summeres over både `i in model.locations` og `j in model.customers`.

Kodeeksempel 2.15: Implementering af `buildModel`-funktionen for eksemplet i sektion 2.4.3

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Create the model object
  model = pyomo.ConcreteModel()
  # Copy the data to the model object
  model.p = data['p']
  model.n = data['n']
  model.m = data['m']
  model.c = data['c']
  model.locations = range(0, model.n)
  model.customers = range(0,model.m)
  # Define the x-variables
  model.x = pyomo.Var(model.locations, model.customers, within=
     pyomo.Binary, bounds=(0, 1))
  # Define the y-variables
  model.y = pyomo.Var(model.locations, within=pyomo.Binary,
     bounds=(0,1))
  # Define the objective function
  model.obj=pyomo.Objective(expr=sum(model.c[i][j]*model.x[i, j]
      for i in model.locations for j in model.customers))
  # Add the "service all" constraints
  model.serviceAll = pyomo.ConstraintList()
  for j in model.customers:
    model.serviceAll.add(expr=sum( model.x[i, j] for i in model.
       locations)==1)
  # Add the indicator constraints
  model.indicators = pyomo.ConstraintList()
  for i in model.locations:
    for j in model.customers:
      model.indicators.add(expr=model.x[i, j] <= model.y[i])
  # Add cardinality constraint
  model.cardinality=pyomo.Constraint(expr=sum(model.y[i] for i
     in model.locations)==model.p)
  return model
```

Efter at objektfunktionen er tilføjet kan første gruppe af begrænsninger tilføjes. Det drejer sig om begrænsningerne (2.1), som sørger for, at hver efterspørgselspunkt bliver serviceret. Det første man skal gøre her, er at definerer en ny variabel i `model`-objektet, som skal bruges til at gemme en liste af begrænsninger (i dette tilfælde én begrænsning for hvert efterspørgselspunkt). Denne variabel skal være af typen `pyomo.ConstraintList`. Den første `ConstraintList` der defineres er altså `model.serviceAll`. Herefter gennemløbes det ranget `model.customers` og der tilføjes (vha. `add()`-funktionen) en begrænsning til `model.serviceAll` for hvert element i `model.customers`. Hver begrænsning er givet ved en sum over alle $x_{ij}$ variablerne for fastholdt $j$ og højreside er blot et ettal.

Når disse begrænsninger er tilføjet, tilføjes alle indikatorbegrænsningerne (2.2). Disse tilføjes ligeledes ved at oprettet en ny variabel, som holder en `ConstraintList`. Da der her skal tilføjes en begrænsning for hver kombination af lokationer og efterspørgselspunkter bruges to nestede for-loops til dette.

Slutteligt tilføjes kardinalitetsbegrænsingen, der sørger for, at der vælges præcis $p$ lokationer til faciliter og `model`-objektet returneres til sidst.

Efter at `buildModel`-funktionen er implementeret, skal `displaySolution` udfyldes. Ønsket er her, at objektfunktionsværdien udskrives, hvorefter en liste med $y$-variablernes værdi printes og slutteligt gives en oversigt over hvilke lokationer hvert efterspørgselspunkt serviceres fra. Dette er implementeret i Kodeeksempel 2.16.

**Kodeeksempel 2.16: Implementering af `displaySolution`-funktionen for eksemplet i sektion 2.4.3**

```
def displaySolution(model: pyomo.ConcreteModel()):
  # Print solution information to prompt
  print('Objective function value =', pyomo.value(model.obj))
  print('Optimal values for the y-variables')
  for i in model.locations:
    print('y[{}]={}'.format(i, pyomo.value(model.y[i])))
  for j in model.customers:
    for i in model.locations:
      xVal = pyomo.value(model.x[i, j])
      if xVal == 1.0:
        print ('Customer', j, 'is serviced from location', i)
```

I denne implementering loopes der igennem $y$-variablerne, og deres værdier udskrives. Herefter udskrives en oversigt over relationen mellem efterspørgselspunkterne

og lokationerne. Læg mærke til, at der er $5 \times 5 = 25$ $x$-variabler i denne model og det er i princippet kun 5 af dem, der er særligt interessant at kende værdien af, nemlig de fem der antager værdien 1. For at få et overblik over relationerne gennemløbes de 25 par af $i$ og $j$ og det tjekkes om den pågældende kombination af $i$ og $j$ har en $x_{ij}$ variabel som antager værdien 1 i en optimal løsning (`if xVal == 1`). Hvis dette er tilfældet, vides det, at efterspørgselspunkt $j$ serviceres af lokation $i$ og dette printes derfor til prompten. Funktionens endelig output er gengivet nedenfor

```
Objective function value = 57.0
Optimal values for the y-variables
y[0]=0.0
y[1]=1.0
y[2]=1.0
y[3]=0.0
y[4]=0.0
Customer 0 is serviced from location 1
Customer 1 is serviced from location 2
Customer 2 is serviced from location 1
Customer 3 is serviced from location 1
Customer 4 is serviced from location 2
```

## 2.5 Pakker som kan have interesse for "*the prescriptive analyst*"

I Tabel 2.6 er listet en række Pythonpakker, som kan have interesse for læseren af denne bog. Listen er absolut ikke udtømmende og er blot et lille udsnit af de pakker forfatteren af bogen ofte har benyttet sig af. Tabel 2.6 er dermed blot tænkt som en lille hjælp, til den interesserede læser.

Hvis man er interesseret i at benytte disse pakker, er der på nettet utallige eksempler og tutorials, og man skal blot benytte sin favoritsøgemaskine til at finde disse.

**Tabel 2.6: Ufuldstændig liste af pakker som kan være interessant for "the prescriptive analyst".**

| Pakker | Kort forklaring og link. |
|---|---|
| Matplotlib | Kan bruges til at visualisere grafer, plots og meget mere. |
| Numpy | Uundværlig pakker hvis man laver data-science i nogen form. Implementerer effektive rutiner til at håndtere store data-mængder. |
| PuLP | Et andet algebraisk modelleringssprog, som minder en del op Pyomo. Har en lidt anden syntaks, og som navnet antyder, håndterer denne pakker kun *lineære* blandede heltalsprogrammer. |
| statsmodels | Implementerer en kaskade af klasser og funktioner til at estimere en række statistiske modeller og til at lave statistiske tests. Endvidere, implementerer statsmodels også en række forecasting teknikker deriblandt ARIMA og SARIMAX modellerne, som ofte bruges til forecasting baseret på tidsserier. |
| json | Dette er en pakke, som implementerer rutiner til at læse data fra json-filer og til at gemme data i json-filer. Json formatet (som er en forkortelse af JavaScript Object Notation) er et format, som egner sig særligt godt til at gemme key-value par, som er den måde en dict i Python virker. |

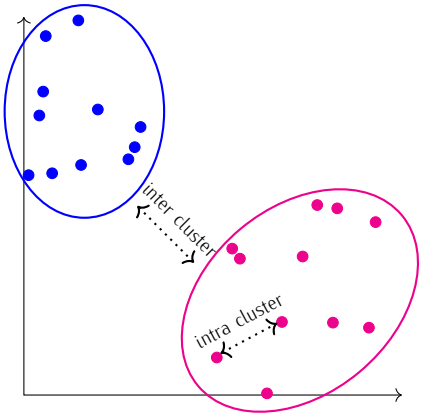
# 3 Clustering

## 3.1 Hvad er clustering

Disciplinen *clustering* består, ikke overraskende, i at clustre, eller på dansk *gruppere*, data i mindre grupper eller clustre. For at behandle emnet mere stringent, vil vi starte med at definere en cluster.

**Definition 3.1.** En cluster, eller gruppe, er en samling af dataobjekter, som er tilsvarende andre objekter i samme gruppe og som er forskellige/anderledes fra objekter i andre gruppe. Se Figur 3.1 for en illustration af en gruppering af dataobjekter.

Clustering er en teknik, som falder ind under den del af statistisk læring/-maskinlæring, som hedder *unsupervised learning* og er dermed faktisk ikke traditionelt en del af det man vil betragte som prescriptive analytics. Derimod er clustering en meget brugt teknik indenfor *prædiktiv* og *explorative* analytics. Vi vil dog alligevel gennemgå nogle clustering-teknikker i dette kursus, da man ofte kan bruge en form for clustering til at behandle data, før man begynder at bruge sine teknikker fra prescriptive analytics. Ofte, når datasæt er store, kan det være en fordel at gruppere dataobjekter for at forsimple sine modeller så de kan optimeres. Man kunne for eksempel forestille sig en "prescriptive analytics"-analytiker, som planlægger ruterne for et renovationsfirma som samler husholdningsaffald ind i et område af Danmark. Der vil typisk være rigtig, rigtig mange husstande. I Skanderborg Kommune er der for eksempel




**Figur 3.1: Illustration af clustering af et datasæt. Dataobjekterne er grupperet i to grupper en blå og en magenta.**

omkring 25.000 husstande og givet at nogle af de bedste problemtilpassede algoritmer ikke kan løse ruteplanlægningsproblemer med mere end et par tusinde efterspørgselspunkter, skal man bearbejde sådan et datasæt, inden man giver sig til at optimere på det.

### 3.1.1 Clustering vs. classification

En anden teknik, som ofte bliver forvekslet med clustering, er "classification". Classification er også en maskinlæringsteknik, men den falder ikke ind under unsupervised learning som clustering, men derimod *supervised learning*. Forskellen ligger i, at når man bruger classification, kender man på forhånd nogle grupper, som man ønsker at inddele dataobjekter efter. Et meget brugt eksempel er spamfiltre på email: givet at en ny mail tikker ind, skal den klassificeres som enten spam eller valid. Det vil altså sige, at vi kender de mulige grupper, som en email kan komme i. Andre eksempler er diagnosticering af Alzheimers ved hjælp af classification anvendt på OCT-scanninger af øjne (se Tian m.fl. (2021)), tildeling af kredit i banker baseret på forbrugsmønstre og korttransaktioner (se Khandani m.fl. (2010)), og mange flere.

Clustering derimod, er en teknik, hvor grupperne ikke kendes på forhånd. Det er derfor en mere *explorative* teknik, hvor man ønsker at få overblik over, *hvorvidt* det er muligt at gruppere dataobjekter i ensartede grupper. Så kunne man spørge sig selv, hvorfor det er interessant at gruppere dataobjekter, hvis man ikke kender de grupper de skal i! Det, som gør clustering så stærkt et værktøj er, at det kan bruges til at finde *skjulte sammenhænge*, som ikke er åbenlyse for mennesker *a priori*. Clustering kan derfor både være et "stand alone" værktøj til databehandling og være såkaldt første skridt til en mere avanceret algoritme. Som eksempler på områder, hvor clustering er anvendeligt kan nævnes: inden for marketing kan clustering bruges til at segmentere kunder sådan, at man kan lave målrettede reklamer, inden for recommender systems (de algoritmer der siger: "du har før set XX, vi tror godt du vil kunne lide YY" (tænk Netflix, Spotify, Zalando, osv.)) bruges clustering til at bestemme grupper af brugere som er lig dig, clustering bruges til at finde særegne mønstre i billeder til for eksempel ansigtsgenkendelse, og clustering kan bruges inden for forskning i biologiske systemer for at blotlægge nogle underliggende mønstre.

## 3.2 Tilgange til Clustering

En clustering algoritme er en procedure der, som input, tager et datasæt, som ikke er grupperet og som output giver 2 eller flere grupper og en entydig afbildning fra dataobjekterne til grupperne. Det vil sige, en clustering algoritme skal angive, hvilken gruppe hvert dataobjekt tilhører. I det følgende præsenteres en række tilgange til clustering: klassisk $k$-means, lokationsbasrede modeller, en graf-opdelingsmodel, samt en model, som minimerer den største diameter i en cluster.

### 3.2.1 $k$-means clustering

Målet med en $k$-means clustering algoritme er at opdele $n$ dataobjekter i $k$ grupper på en sådan måde, at hvert dataobjekt tilhører en og kun én gruppe. Målet er at danne grupperne således, at den total kvadrerede afstand mellem dataobjekterne i grupperne minimeres. Mere formelt kan vi opskrive $k$-means problemet på følgende måde: Antag, at der eksisterer $n$ dataobjekter her kaldet $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. Hvert dataobjekt er $d$-dimensionelt, det vil sige, $\mathbf{x}_i \in \mathbb{R}^d$ for all $i = 1, \ldots, n$. I eksemplet i Figur 3.1 er dataobjekterne todimensionelle; hvert objekt holder information om en $x$-koordinat og en $y$-koordinat. Principielt kunne $d$ være meget stor; et dataobjekt kunne holde information om min placering, det vil sige $(x, y, z)$-koordinater, hvilken dag og hvilket tidspunkt dataet er observeret, tidligere placering og så videre.

Målet for en $k$-means algoritme er nu at opdele de $n$ dataobjekter i $k < n$ grupper, her kaldet $S_1, S_2, \ldots, S_k$ sådan at $S_i$ og $S_j$ ikke har nogle dataobjekter til fælles og sådan at foreningen af alle $S_i$'erne indeholder alle dataobjekterne. Vi vil lade $\mathcal{S}$ være mængden af alle grupperne. Projektet er nu, at lave grupper, hvor den gennemsnitlige kvadrerede afstand mellem dataobjekterne i grupperne minimeres. Eller mere matematisk

$$\min_{\mathcal{S}} \sum_{i=1}^{k} \frac{1}{|S_i|} \sum_{\mathbf{x}_i, \mathbf{x}_j \in S_i} ||\mathbf{x}_i - \mathbf{x}_j||^2 \tag{3.1}$$

Man kan her notere sig, at $k$-means problemet ligner det almindelige *least squares* problem kendt fra kurser i statistik. Dog gælder her, at vi kun måler på afstandene *internt* i grupperne. Vi er så at sige ligeglade med afstandene mellem grupperne.

Man skal lægge mærke til, at antallet af grupper er en eksogent givet parameter. Det vil altså sige, at man på forhånd skal angive antallet af grupper, man ønsker

at opdele dataobjekterne i. Dette kan være problematisk, da man ofte bruger $k$-means til at betragte data, som man ikke ved meget om på forhånd. Man kan dog anvende sin $k$-means algoritme for forskellige værdier af $k$ og så sammenligne resultaterne man opnår.

Det er desværre et såkaldt $\mathcal{NP}$-hårdt[1] problem at løse (3.1). Derfor vil man oftest bruge en *heuristik* til at håndtere $k$-means problemet. En meget kendt algoritme er Lloyds algoritme, som iterativt forsøger at opdele dataet i grupper baseret på kvalificerede gæt på præsenterende punkter. De kvalificerede gæt, som benyttes i Lloyds algoritme er de såkaldte tyngdepunkter (centre of gravity) i grupperne. Givet en gruppe af dataobjekter $\mathcal{S}_i = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \ldots, \mathbf{x}_{i_{|\mathcal{S}_i|}}\}$ er gruppens tyngdepunkt eller center givet ved

$$m_i = \frac{1}{|S_i|} \sum_{j=1}^{|S_i|} \mathbf{x}_{i_j} = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}$$

Det vil sige, at $m_i$ angiver *gennemsnits*-punktet af dataobjekterne i gruppen $\mathcal{S}_i$. For at illustrere dette, tag en gruppe som består af data objekterne $\{(0,0),(1,0),(0,1)\}$ hvorved tyngdepunktet bliver $m = (\frac{1}{3}, \frac{1}{3})$ (læseren opfordres til at plotte de tre punkter i gruppen samt tyngdepunktet $m$ for at få en forståelse af den underliggende idé).

Lloyds algoritme kan kort opsummeres på følgende vis:

Step 0: Lad $m_1, m_2, \ldots, m_k$ være et gæt på centrene af de endelige grupper. Det vil sige, $m_i$ *repræsenterer* gruppe $S_i$.

Step 1: For hvert af de $n$ dataobjekter, $\mathbf{x}_j$, udregn nu afstanden mellem $\mathbf{x}_j$ og hver af de $k$ centre $m_i$. Lad disse afstande være $d_{ij}$.

Step 2: Tildel nu hvert dataobjekt $\mathbf{x}_j$ til den gruppe $i$ hvor $d_{ij}$ er mindst. Det vil sige, hvis $d_{j1} \leq d_{ji}$ for alle $i = 2, \ldots, k$ så skal dataobjekt $\mathbf{x}_j$ tildeles gruppe $S_1$ og så fremdeles.

Step 3: Udregne nu nye centre $m_i$ baseret på de nu opdaterede grupper:

$$m_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}$$

[1] Dette betyder, løst sagt, at problemet kan være meget svært at løse eksakt i praksis. For mere omkring $\mathcal{NP}$-hårdhed, kan man konsultere den tilhørende wikipedia side.

**Tabel 3.1: Et lille datasæt bestående af 20 objekter som hver indeholder en x- og en y-koordinat.**

| Objekt | x | y | Objekt | x | y | Objekt | x | y | Objekt | x | y |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 14 | 16 | $x_6$ | 20 | 8 | $x_{11}$ | 21 | 14 | $x_{16}$ | 14 | 23 |
| $x_2$ | 2 | 10 | $x_7$ | 3 | 25 | $x_{12}$ | 4 | 14 | $x_{17}$ | 22 | 19 |
| $x_3$ | 6 | 6 | $x_8$ | 13 | 14 | $x_{13}$ | 3 | 22 | $x_{18}$ | 1 | 10 |
| $x_4$ | 18 | 6 | $x_9$ | 25 | 9 | $x_{14}$ | 8 | 8 | $x_{19}$ | 4 | 24 |
| $x_5$ | 6 | 23 | $x_{10}$ | 0 | 13 | $x_{15}$ | 6 | 4 | $x_{20}$ | 20 | 12 |

Hvis mindst et center $m_i$ ændres fra forrige iteration, gå da tilbage til **Step 1**. Ellers returner de nuværende grupper.

Man siger, at algoritmen er *konvergeret*, når grupperne dannet i **Step 1** ikke har ændret sig, i to på hinanden følgende iterationer af algoritmen. Når det sker, står man med sine grupper.

Det er i sagens natur ikke helt indlysende hvordan man i **Step 0** skal komme frem til et fornuftigt gæt på centrene $m_i$, $i = 1, \ldots, k$. Én måde er, at man blot vælger $k$ dataobjekter tilfældigt fra datasættet og lader disse være centrene i **Step 0**. Denne metode kaldes en Forgy-initialisering. En anden metode, som kaldes *farthest point* metoden, vælger først et dataobjekt på tilfældig vis. Dette punkt bruges som $m_1$. Dernæst vælges et dataobjekt, som er længst fra $m_1$. Dette bruges som $m_2$ nu vælges, blandt de resterende dataobjekter, et tredje dataobjekt, som er længst fra $m_1$ og $m_2$. Dette objekt sættes lig $m_3$. Således fortsættes til man har udtrukket $k$ dataobjekter.

*Eksempel* 3.1 ($k$-means clustering). I det følgende lille eksempel vil vi prøve at se en iteration af $k$-means algoritmen beskrevet ovenfor. For at benytte algoritmen skal vi bruge det data, som er at finde i Tabel 3.1. Datasættet består af 20 objekter, som hver indeholder en x-koordinat og en y-koordinat. Målet er at lave to grupper fra datasættet.

Det første $k$-means algoritmen foreskriver er, at vi skal initialisere algoritmen med to gæt på centre for de endelige grupper (se **Step 0**). For at gøre det enkelt, vælger vi her $m_1 = \mathbf{x}_1$ og $m_2 = \mathbf{x}_2$.

Det næste vi skal gøre, jf. **Step 1**, er at beregne afstandene mellem $m_1$ og $m_2$ og de resterende punkter hvilket er gjort i den følgende tabel:

| $d_{ij}$ | $\mathbf{x}_1$ | $\mathbf{x}_2$ | $\mathbf{x}_3$ | $\mathbf{x}_4$ | $\mathbf{x}_5$ | $\mathbf{x}_6$ | $\mathbf{x}_7$ | $\mathbf{x}_8$ | $\mathbf{x}_9$ | $\mathbf{x}_{10}$ | $\mathbf{x}_{11}$ | $\mathbf{x}_{12}$ | $\mathbf{x}_{13}$ | $\mathbf{x}_{14}$ | $\mathbf{x}_{15}$ | $\mathbf{x}_{16}$ | $\mathbf{x}_{17}$ | $\mathbf{x}_{18}$ | $\mathbf{x}_{19}$ | $x_{20}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_1$ | 0 | 13.4 | 12.8 | 10.8 | 10.6 | 10.0 | 14.2 | 2.2 | 13.0 | 14.3 | 7.3 | 10.2 | 12.5 | 10.0 | 14.4 | 7.0 | 8.5 | 14.3 | 12.8 | 7.2 |
| $m_2$ | 13.4 | 0 | 5.7 | 16.5 | 13.6 | 18.1 | 15.0 | 11.7 | 23.0 | 3.6 | 19.4 | 4.5 | 12.0 | 6.3 | 7.2 | 17.7 | 21.9 | 1.0 | 14.1 | 18.1 |

I **Step 2** kan vi nu nemt, ved inspektion, finde ud af hvilken gruppe hvert dataobjekt skal i og vi ender med grupperne:

$$S_1 = \{\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{11}, \mathbf{x}_{16}, \mathbf{x}_{17}, \mathbf{x}_{19}, \mathbf{x}_{20}\}$$
$$S_2 = \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_{10}, \mathbf{x}_{12}, \mathbf{x}_{13}, \mathbf{x}_{14}, \mathbf{x}_{15}, \mathbf{x}_{18}\}$$

Det sidste step i første iteration er nu, at vi skal genberegne centrene af de grupper vi har lavet. Dette gøres for gruppe $S_1$ på følgende måde:

$$m_1 = \frac{1}{|S_1|}\sum_{\mathbf{x}\in S_1}\mathbf{x} = \frac{1}{12}\left(\begin{pmatrix}14\\16\end{pmatrix} + \begin{pmatrix}18\\6\end{pmatrix} + \begin{pmatrix}6\\23\end{pmatrix} + \cdots + \begin{pmatrix}20\\12\end{pmatrix}\right)$$
$$= \frac{1}{12}\begin{pmatrix}180\\193\end{pmatrix} \approx \begin{pmatrix}15\\16.083\end{pmatrix}$$

På samme vis kan vi beregne $m_2$ til at være punktet $m_2 = \begin{pmatrix}3.75\\10.875\end{pmatrix}$. Proceduren fortsætter nu med disse nye centre og går tilbage til **Step 1**, hvor afstandene til $m_1$ og $m_2$ beregnes på ny.

---

**Opgave 3.1:** Plot datasættet fra Tabel 3.1 med dit favorit graf-værktøj (fx Excel, R, Python).

**Opgave 3.2:** Færdiggør $k$-means algoritmens iterationer. Det vil sige, gentag **Step 0**–**Step 3** på side 36 indtil grupperne $S_1$ og $S_2$ ikke ændres mere.

**Opgave 3.3:** Plot de forskellig clustre/grupper, der dannes i $k$-means algoritmens iterationer.

**Opgave 3.4:** Hvad sker der hvis man vælger to andre startpunkter for algoritmen? Er vi garanteret at få de samme grupper når algoritmen terminerer?

---

### 3.2.2 Lokationsbaseret tilgang til clustering

En af udfordringerne ved $k$-means tilgangen er, at det ikke er en ret agil metode, i og med, at man ikke har nemt ved at ændre ved antagelserne; man kan ikke sætte et maksimum eller minimum på antallet af elementer i hver gruppe, man skal "nøjes" med at benytte Euklidisk afstand mellem dataobjekter osv.. I dette afsnit

vil vi tage en mere generel metode i betragtning: lokaliseringsbaseret clustering. Og vi vil benytte *integer linear programming* (ILP eller blot IP) til det.

Det vil igen være en antagelse, at der er $n$ dataobjekter, som skal grupperes i $k$ grupper, hvor $k < n$. Inden for lokationsbaseret clustering vil vi benytte to sæt af *binære* variabler:

$$y_i = \begin{cases} 1, & \text{hvis dataobjekt } \mathbf{x}_i \textit{ repræsenterer} \text{ en gruppe} \\ 0, & \text{ellers} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{hvis dataobjekt } \mathbf{x}_j \text{ er i den gruppe, som dataobjekt } \mathbf{x}_i \text{ repræsenterer} \\ 0, & \text{ellers} \end{cases}$$

Vi kan fra denne definition allerede deducere en relation mellem $y_i$ og $x_{ij}$: hvis $y_i = 0$ så skal $x_{ij} = 0$. Dette følger fra, at hvis dataobjekt $\mathbf{x}_i$ *ikke* repræsenterer en gruppe, så kan dataobjekt $\mathbf{x}_j$ ikke være i en gruppe, som er repræsenteret af $i$ (for den findes ikke).

Ligesom med $k$-means ønsker vi at minimere en form for afstandsmål mellem elementerne i grupperne. For at gøre dette, lader vi $c_{ij}$ være et mål for afstanden mellem dataobjekterne $\mathbf{x}_i$ og $\mathbf{x}_j$. Det kunne, som i tilfældet med $k$-means, være den Euklidiske afstand, men kunne også være mange andre. For eksempel kunne Pearson korrelationen, Manhattan normen, Tchebycheff normen eller et hvilket som helst andet tænkeligt afstandsmål bruges. Målet for den lokationsbaserede clustering er at minimere den totale afstand fra hvert dataobjekt til det dataobjekt, som repræsenterer den gruppe, hvortil de er tildelt. Dette gøres matematisk med objektfunktionen

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}.$$

Vi kan nu gå videre til begrænsningerne til modellen. Som ovenfor nævnt haves relationen: $y_i = 0 \Rightarrow x_{ij} = 0$. Denne relation kan simpelt opskrives som $x_{ij} \leq y_i$ for alle $i, j = 1, \ldots, n$. For at se, at dette er korrekt, betragtes tilfældet $y_i = 0$. I dette tilfælde reduceres uligheden til $x_{ij} \leq 0$ hvilket betyder at $x_{ij} = 0$, thi $x_{ij}$ er binær. Hvis, på den anden side, $y_i = 1$ haves at $x_{ij} \leq 1$ hvilket altid er tilfældet, hvorfor uligheden ikke begrænser $x_{ij}$ i dette tilfælde.

Endvidere skal modellen sørge for, at hvert dataobjekt tildeles *præcis én* gruppe.

Dette gøres med begrænsningerne:

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n.$$

Det sidste, der mangler er, at der skal vælges præcis $k$ grupper. I og med at $y_i$-variablerne angiver om et dataobjekt er repræsentant for en gruppe eller ej, kan vi blot sørge for, at der bliver valgt præcis $k$ repræsentanter. Dette gøres med begrænsningen:

$$\sum_{i=1}^{n} y_i = k.$$

Med disse begrænsninger, er modellen komplet og vi kan opstille det fulde IP, som bruges til at gruppere dataobjekterne:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.:} \quad & \sum_{i=1}^{n} x_{ij} = 1, && \forall j = 1, \ldots, n \\
& x_{ij} \leq y_i, && \forall i, j = 1, \ldots, n \\
& \sum_{i=1}^{n} y_i = k, \\
& x_{ij} \in \{0, 1\}, && \forall i, j = 1, \ldots, n \\
& y_i \in \{0, 1\}, && \forall i = 1, \ldots, n
\end{aligned}$$

Man bør notere sig, at den lokationsbaserede metode ikke har mulighed for at finde centre, der ligger "mellem" dataobjekterne da man vælger repræsentanterne for grupperne blandt dataobjekterne. Det gør naturligvis metoden lidt mere udsat for outliers. På den anden side, har vi nu mulighed for at sætte for eksempel *minimum og maksimum antal på hver gruppe* ($a_i y_i \leq \sum_{j=1}^{n} x_{ij} \leq b_i y_i$, for alle $i = 1, \ldots, n$), tilføje information om to objekter, $\mathbf{x}_j$ og $\mathbf{x}_k$, der *ikke skal i samme gruppe* ($x_{ij} + x_{ik} \leq 1$ for alle $i = 1, \cdots, n$), at to objekter *skal i samme gruppe* ($x_{ij} = x_{ik}$ for alle $i = 1, \ldots, n$), samt mange andre begrænsninger.

Objektfunktionen i programmet ovenfor minimerer den totale afstand, der er mellem dataobjekterne og de repræsentanter de er tildelt. Dette har en tendens til at negligere enkelte punkter langt væk fra andre clustre. Hvis man vil have en

mere "fair" konstruktion af sine grupper, kan man i stedet minimere den maksimale afstand fra et dataobjekt til dens repræsentant; Afstanden, $\rho_j$, fra dataobjekt $\mathbf{x}_j$ til dens repræsentant er givet ved (hvorfor?)

$$\rho_j = \sum_{i=1}^{n} c_{ij} x_{ij}.$$

Det vil sige, at hvis objektfunktionen ændres til

$$\min \max_{j=1,\ldots,n} \{\rho_j\}$$

så minimeres den maksimale afstand fra et dataobjekt til dens tilhørende gruppe-repræsentant. Dette er åbenlyst en ikke-lineær objektfunktion, og der skal gøres noget for at få det til at passe i vores nuværende framework. En måde det kan gøres på er ved at introducere en ny variabel $\rho^{\max}$, som skal antage værdien af den længste afstand mellem et dataobjekt og dens gruppe. I og med $\rho^{\max}$ er den længste afstand, skal den være større end eller lig med alle de individuelle afstande:

$$\rho_j \leq \rho^{\max}, \quad \forall j = 1, \ldots, n$$

Hvis nu objektfunktionen ændres til $\min \rho^{\max}$ så vil vi presse $\rho^{\max}$ så langt ned som mulig, og det første der stopper $\rho^{\max}$ nedad til er den største afstand fra et dataobjekt til dens gruppe. Det vil sige, i en optimal løsning er $\rho^{\max} = \max_{j=1,\ldots,n} \rho_j$ hvilket er hvad vi ønskede. Vi ender altså med problemet

$$\begin{aligned}
&\min\ \rho^{\max} \\
&\text{s.t.:}\ \sum_{i=1}^{n} x_{ij} = 1, && \forall j = 1, \ldots, n \qquad (3.2)\\
&\quad x_{ij} \leq y_i, && \forall i, j = 1, \ldots, n \\
&\quad \sum_{i=1}^{n} y_i = k, \\
&\quad \rho_j = \sum_{i=1}^{n} c_{ij} x_{ij}, && \forall j = 1, \ldots, n \\
&\quad \rho_j \leq \rho^{\max}, && \forall j = 1, \ldots, n \\
&\quad x_{ij} \in \{0, 1\}, && \forall i, j = 1, \ldots, n \\
&\quad y_i \in \{0, 1\}, && \forall i = 1, \ldots, n \qquad (3.3)
\end{aligned}$$

*Eksempel* 3.2 (Lokationsbaseret clustering ). Vi fortsætter her eksemplet fra Eksempel 3.1, og laver to grupper, hvor den totale afstand mellem dataobjekterne og deres respektive repræsentanter minimeres. For at gøre dette benyttes den Euklidiske afstand mellem dataobjekterne til at bestemme omkostningskoefficienterne $c_{ij}$. Det vil sige $c_{ij}$ er givet ved

$$c_{ij} = \sqrt{(\mathbf{x}_i^x - \mathbf{x}_j^x)^2 + (\mathbf{x}_i^y - \mathbf{x}_j^y)^2}$$

hvor $\mathbf{x}_i^x$ og $\mathbf{x}_i^y$ henholdsvis betyder $x$- og $y$-koordinatet for dataobjekt $\mathbf{x}_i$. Modellen som vi ender med har i alt $20 \times 20 = 400$ $x_{ij}$-variabler og 20 $y_i$ variabler og $20 + 20 \times 20 + 1 = 421$ begrænsninger (heltalsbegrænsningerne fratrukket) og kan implementeres i Pyomo som vist i Kodeeksempel 3.1.

**Kodeeksempel 3.1: Python implementering af en lokationsbaseret clustering model. Pyomo er brugt som modelleringssprog.**

```
import pyomo.environ as pyomo

# The data is assumed to be stored in a dictionary called '
   clusterData'
# Assume the following data-keys are defined:
# 'nrObjects', 'nrGroups', 'c'

# First data is copied to model
model = pyomo.ConcreteModel()
model.objects = range(0, clusterData['nrObjects'])
model.c = clusterData['c']
model.nrGroups = clusterData['nrGroups']

# Next, define variables
model.x = pyomo.Var(model.objects, model.objects, within=pyomo.
   Binary)
model.y = pyomo.Var(model.objects, within=pyomo.Binary)

# Set up objective function
model.obj = pyomo.Objective(expr=sum(model.c[i][j] * model.x[i,
   j] for i in model.objects for j in model.objects),
sense=pyomo.minimize)

# Set up 'sum to one' constraints
model.one_group = pyomo.ConstraintList()
for j in model.objects:
  model.one_group.add(sum(model.x[i, j] for i in model.objects)
     == 1.0)

# Set up generalized upper bounds
```

```
model.GUB = pyomo.ConstraintList()
for i in model.objects:
  for j in model.objects:
    model.GUB.add(model.x[i, j] <= model.y[i])

# Add cardinality constraint
model.cardinality = pyomo.Constraint(expr=sum(model.y[i] for i
   in model.objects) == model.nrGroups)

# Solve and print solution information
solver = pyomo.SolverFactory('glpk')
results = solver.solve(model, tee=True)
results.write()
```

Man kan lige notere sig, at denne model ikke ville kunne løses med Excels standard Solver, da Solver har en begrænsning på 200 variabler og 100 begrænsninger.

De grupper som dannes af denne model er

$$S_1 = \{x_2, x_3, x_5, x_7, x_{10}, x_{12}, x_{13}, x_{14}, x_{15}, x_{18}, x_{19}\}$$

og

$$S_2 = \{x_1, x_4, x_6, x_8, x_9, x_{11}, x_{16}, x_{17}, x_{20}\}.$$

Det ses, at disse grupper er markant anderledes end de grupper der blev fundet (i første iteration) med $k$-means algoritmen i Eksempel 3.1. Ikke desto mindre, er dette også en gruppering af dataobjekterne i to grupper.

---

**Opgave 3.5:** Bekræft resultatet fra Eksempel 3.2 ved at implementere din egen model.

**Opgave 3.6:** Plot grupperne fundet i **Opgave 3.5** og sammenlign disse med grupperne fundet i **Opgave 3.3**.

**Opgave 3.7:** Implementer min-max clustering modellen (3.2)–(3.3) og sammenlign de nye grupper med de to grupper fundet i Eksempel 3.2.

---

### 3.2.3 Clustering med minimering af cluster-diameteren

I de lokationsbaserede clustering modeller, var fokus på at finde en repræsentant for et cluster blandt dataobjekterne, for så at gruppere objekterne således, at den

største afstand eller den totale afstand fra dataobjekterne til deres repræsentant i clustrene blev minimeret. I det følgende, vil vi tage en anden naturlig tilgang: minimering af diameteren i en cluster. Med diameteren i en cluster menes den største afstand mellem to dataobjekter i samme cluster. Vi vil som ovenfor antage, at $c_{ij}$ er et mål for afstanden mellem dataobjekterne $\mathbf{x}_i$ og $\mathbf{x}_j$ og vi vil igen antage, at vi ønsker dataobjekterne opdelt (clustret) i $k$ forskellige grupper. Til dette formål defineres nu variabler $x_{il}$ givet ved

$$x_{il} = \begin{cases} 1, & \text{hvis dataobjekt } \mathbf{x}_i \text{ er i cluster } l \\ 0, & \text{ellers} \end{cases}$$

Derudover vil vi introducere en variabel $D_l$ for hvert cluster, $l = 1, \ldots, k$, som angiver en øvre grænse for diameteren i cluster $l$. Det vil sige, at $D_l$ skal tage en værdi som er mindst lige så stor som den største afstand, der er mellem 2 dataobjekter i cluster $l$. Slutteligt vil vi indføre variablen $D^{\max}$ som skal tage værdien af den største diameter over de $k$ clustres diametre. Det vil sige, at $D^{\max} = \max_{l=1,\ldots,k}\{D_l\}$.

Vi vil starte modellen med at opstille objektfunktion, som indlysende er givet ved

$$\min \; D^{\max}.$$

Altså, vi ønsker at minimere den største diameter over alle clustrene. Det næste, der skal modelleres, er, at $D^{\max}$ skal antage den rigtige værdi i optimum. Dette gøres ved at kræve, at følgende uligheder er opfyldt for alle clustrene

$$D^{\max} \geq D_l, \qquad \forall l = 1, \ldots, k$$

Med disse begrænsninger sørges for, at $D^{\max}$ altid er større end alle diametrene i clustrene, hvorved $D^{\max}$ naturligvis også er større end den største. Da der samtidig minimeres, vil $D^{\max}$ aldrig være strengt større end den største diameter i en optimal løsning, hvorfor $D^{\max}$ antager den korrekte værdi.

Når nu værdien af $D^{\max}$ er fastsat, skal hver $D_l$ naturligvis også antage den korrekte værdi. Igen skal vi sørge for, at $D_l$ er større end eller lig med den største afstand mellem to dataobjekter i cluster $l$. Dette kan gøres ved at benytte følgende uligheder for hvert par af dataobjekter og cluster:

$$D_l \geq c_{ij} x_{il} x_{jl}, \qquad \forall l = 1, \ldots, n, \;\; i, j = 1, \ldots, n : i \neq j$$

For at indse, at dette er korrekt, deles analysen op i to tilfælde

$x_{il} = 0 \vee x_{jl} = 0$: Hvis enten dataobjekt $i$ eller dataobjekt $j$ (eller begge) *ikke* er i cluster $l$ skal afstanden mellem disse to dataobjekter ikke influere på denne clusters diameter. Ved indsættelse af værdierne for $x$-variablerne fås

$$\begin{aligned} & D_l \geq c_{ij} x_{il} x_{jl} = c_{ij} \cdots 0 \cdot x_{jl} = 0 \\ & \vee \\ & D_l \geq c_{ij} x_{il} x_{jl} = c_{ij} \cdots x_{il} \cdot 0 = 0 \\ & \vee \\ & D_l \geq c_{ij} x_{il} x_{jl} = c_{ij} \cdots 0 \cdot 0 = 0 \end{aligned}$$

Altså reduceres ulighederne blot til, at diameteren i cluster $l$ skal være større end 0, hvilket er trivielt sandt, og dermed ikke nogen restriktion for modellen.

$x_{il} = x_{jl} = 1$: I dette tilfælde er både dataobjekt $i$ og $j$ i cluster nummer $l$. Uligheder skal derfor sørge for, at $D_l$ (cluster $l$'s diameter) er større end eller lig med afstanden mellem dataobjekt $i$ og $j$. Ved indsættelse af værdien for $x$-variablerne fås

$$D_l \geq c_{ij} x_{il} x_{jl} = c_{ij} \cdots 1 \cdot 1 = c_{ij}$$

Hermed er det netop garanteret, at hvis dataobjekt $i$ og $j$ begge er i cluster $l$, så vil diameter-variablen antage en værdi, som er mindst lige så stor, som afstanden mellem disse to objekter.

Da disse uligheder skal gælde for alle par af dataobjekter og alle clustre, haves det, at hver $D_l$ vil antage en værdi, som er større end alle parvise afstande mellem objekterne i cluster $l$. Dermed er $D_l$ også større end den største afstand internt i clustret, hvorved argumentet er tilendebragt.

Slutteligt skal hvert dataobjekt i præcis ét cluster, hvilket klares med de sædvanlige "vælg én blandt mange"-begrænsninger:

$$\sum_{l=1}^{k} x_{il} = 1, \qquad \forall i = 1, \ldots, n.$$

Dermed kan det blandede heltalsprogram for minimering af den største diameter

i en cluster opskrives som følger

$$
\begin{aligned}
\min\ & D^{\max} \\
\text{s.t. } & D^{\max} \geq D_l, && \forall l = 1, \ldots, k \\
& D_l \geq c_{ij} x_{il} x_{jl}, && \forall l = 1, \ldots, k,\ i, j = 1, \ldots, n : i \neq j \\
& \sum_{l=1}^{k} x_{il} = 1, && \forall i = 1, \ldots, n \\
& x_{il} \in \{0, 1\}, && \forall i = 1, \ldots, n,\ l = 1, \ldots, k
\end{aligned} \tag{3.4}
$$

Selvom denne formulering ser tilforladelig ud, med sine få grupper af begrænsninger, er der for det første rigtig mange begrænsninger til stede i ulighederne (3.4). Dernæst er begrænsningerne (3.4) også ikke-lineære, da de indeholder produktet af to variabler. Dette er dog ikke katastrofalt, da dette let kan lineariseres til de lineære begrænsninger

$$
\begin{aligned}
& D_l \geq c_{ij} \left( x_{il} + x_{jl} - 1 \right), && \forall l = 1, \ldots, k,\ i, j = 1, \ldots, n : i \neq j \\
& D_l \geq 0, && \forall l = 1, \ldots, k
\end{aligned}
$$

Det vil sige, at clustering problemet med fokus på minimering af den største diameter på tværs af alle clustre, kan opskrives som det følgende *lineære* blandede heltalsprogram

$$
\begin{aligned}
\min\ & D^{\max} \\
\text{s.t. } & D^{\max} \geq D_l, && \forall l = 1, \ldots, k \\
& D_l \geq c_{ij} \left( x_{il} + x_{jl} - 1 \right), && \forall l = 1, \ldots, k,\ i, j = 1, \ldots, n : i \neq j \\
& D_l \geq 0, && \forall l = 1, \ldots, k \\
& \sum_{l=1}^{k} x_{il} = 1, && \forall i = 1, \ldots, n \\
& x_{il} \in \{0, 1\}, && \forall i = 1, \ldots, n,\ l = 1, \ldots, k
\end{aligned}
$$

---

**Opgave 3.8:** Ændre programmet for minimering af den største diameter, til at minimere den gennemsnitlige diameter over clustrene. Argumenter for, at dette også minimerer summen af cluster-diametre.

**Opgave 3.9:** Løs clustering problemet med minimering af den maksimale cluster-diameter for datasættet i Eksempel 3.1.

**Opgave 3.10:** Minimering af cluster-diametre fokuserer på *intra-cluster*-afstanden som illustreret i Figur 3.1. Foreslå en model, som fokuserer på at maksimere *inter-cluster*-afstanden i stedet.

### 3.2.4 Graf-opdeling

I det foregående har vi betragtet dataobjekter bestående af en række data-indgange, som definerede objektet. Ud fra data-indgangene har vi forsøgt at finde dataobjekter, som lignede hinanden for så at laver grupper, der var så homogene som muligt. I dette afsnit vil vi betragte dataobjekter, som er beskrevet ved deres *relation* til andre dataobjekter. Vi vil ligesom i det ovenstående betragte $n$ dataobjekter: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. Vi går nu ud fra, at hvert dataobjekt består af $n$ indgange, én for hvert dataobjekt. Det vil sige, at $\mathbf{x}_i$ er en vektor med $n$ indgange: $\mathbf{x}_i = (\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{n,n})$. Hver indgang i vektoren $\mathbf{x}_i$ angiver om $\mathbf{x}_i$ er i relation til et andet dataobjekt. Mere formelt kan det skrives som

$$\mathbf{x}_{ij} = \begin{cases} 1, & \text{hvis dataobjekt } \mathbf{x}_i \text{ er i relation til dataobjekt } \mathbf{x}_j \\ 0, & \text{ellers} \end{cases}$$

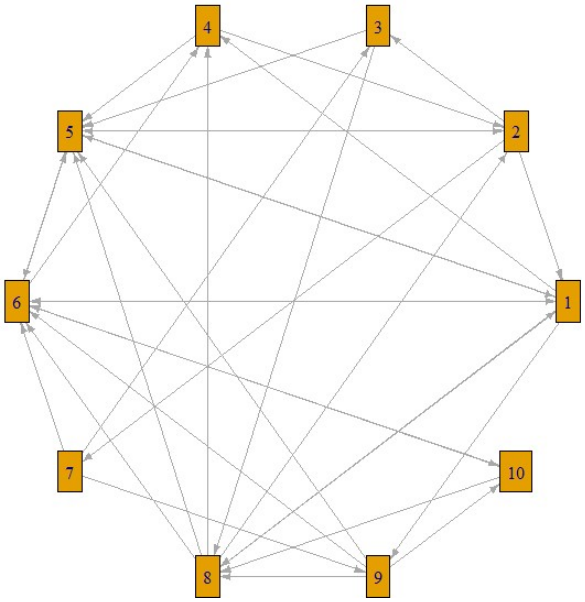
For at anskueliggøre hvad disse dataobjekter kunne være, kan man forestille sig et socialt netværk som for eksempel Facebook, Twitter, eller LinkedIn. Hver bruger, lad os kalde dem $i$, vil her være et dataobjekt, $\mathbf{x}_i$, hvor $\mathbf{x}_{ij} = 1$, hvis bruger $i$ er *connected* med bruger $j$. Den slags data kan være meget nyttigt i forbindelse med for eksempel markedsføring på platformen, eller når platformen foreslår andre brugere som mulige nye connections.

Grunden til, at man kalder denne type problemer for graf-opdelingsproblemer skyldes, at man kan repræsentere dataet som en graf. For at få syn for sagen, skal vi nu betragte et lille eksempel.

*Eksempel* 3.3. Antag, at vi har et (lille) netværk bestående af 10 brugere. I netværket kan en bruger *følge* en anden bruger. Hvis dette er tilfældet markeres det med et 1-tal i Tabel 3.3. Man kan her notere sig, at det ikke er som på Facebook eller på LinkedIn, hvor en relation er gensidig; vi har for eksempel at bruger 1 følger bruger 4, men ikke omvendt. Det er umiddelbart ikke så nemt at overskue, hvorledes relationerne hænge sammen ved blot at betragte Tabel 3.3.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 6 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 7 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 8 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

Tabel 3.3: 0-1 repræsentation af realtionerne.




Figur 3.2: Graf-repræsentation af relationerne.

Det kan derfor ofte være belejligt at repræsentere data som en graf, hvilket er gjort i Figur 3.2. Her repræsenterer *knuderne* i grafen brugerne, mens *kanterne*/pilene repræsenterer relationerne. Der går en kant *fra* bruger $i$ *til* bruger $j$ hvis $\mathbf{x}_{ij} = 1$. Her er valgt at hvis $\mathbf{x}_{ij} = \mathbf{x}_{j,i} = 1$ så repræsenteres det med en kant som har spids i begge ender: $\leftrightarrow$.

Ofte ønsker man at opdele en graf i nogle komponenter hvor man ændrer så få relationer som muligt; fjerner så få eksisterende relationer og indfører så få nye relationer som muligt.

Det vil sige, at i grafopdelingsproblemet ønskes det at skabe en ny graf, som ligner den graf, der defineres af $\{\mathbf{x}\}_{ij=1}^{n}$, så meget som muligt og således, at den nye graf består af en ikke-sammenhængende *delkomponenter*.

Dette kan gøres ved hjælp af en IP-model, hvor man benytter følgende mængde af variabler:

$$z_{ij} = \begin{cases} 1, & \text{hvis dataobjekt } \mathbf{x}_i \text{ er relation til dataobjekt } \mathbf{x}_j \text{ i den nye graf} \\ 0, & \text{ellers.} \end{cases}$$

Hvis to dataobjekter er i samme komponent (eller gruppe om man vil), vil vi antage at de er i relation til hinanden. Det vil sige, at $z_{ij} = z_{ji}$.

Objektfunktionen er nu at maksimere antallet af relationer, som stadig eksisterer efter opdelingen, og minimere antallet af nye relationer, der indføres i den nye

graf. Dette kan gøres ved, at betragte følgende objektfunktion:

$$\max w_1 \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{x}_{ij} z_{ij} - w_2 \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - \mathbf{x}_{ij}) z_{ij}$$

Hvis $\mathbf{x}_i$ og $\mathbf{x}_j$ er i relation til hinanden ($\mathbf{x}_{ij} = 1$) *og* disse to objekter placeres i samme gruppe ($z_{ij} = 1$) så tæller dette op i objektfunktionen. Hvis, på den anden side, $\mathbf{x}_{ij} = 0$ og $\mathbf{x}_i$ og $\mathbf{x}_j$ placeres i samme gruppe ($z_{ij} = 1$), så indføres en relation, som ikke var der oprindeligt. Da $\mathbf{x}_{ij} = 0$ medfører at $1 - \mathbf{x}_{ij} = 1$ betyder det, at det trækker ned i objektfunktionen. I objektfunktionen vægtes det forskelligt at fjerne relationer ($w_1$) og tilføje relationer $w_2$. Dette giver mulighed for at styre, hvad modellen skal have mest fokus på.

Det næste vi skal sørge for er, at der rent faktisk bliver lavet grupper. Det kan man gøre ved hjælp af såkaldte trekants-relationer: hvis $\mathbf{x}_i$ er i gruppe med $\mathbf{x}_j$ og $\mathbf{x}_j$ er i gruppe med $\mathbf{x}_k$, så skal $\mathbf{x}_i$ også være i gruppe med $\mathbf{x}_k$. Dette kan modelleres ved hjælpe af følgende mængde af begrænsninger:

$$z_{ij} + z_{jk} - 1 \leq z_{ik}, \quad \forall i, j, k = 1, \ldots, n : i \neq j, j \neq k, i \neq k \tag{3.5}$$

For at overbevise sig selv om, at dette er rigtigt, kan vi betragte fire forskellige tilfælde:

$z_{ij} = z_{jk} = 0$: I dette tilfælde er $\mathbf{x}_i$ og $\mathbf{x}_j$ *ikke* i gruppe med hinanden og $\mathbf{x}_j$ og $\mathbf{x}_k$ er *ikke* i gruppe med hinanden. Derfor kan vi ikke sige noget om relationen mellem $\mathbf{x}_i$ og $\mathbf{x}_k$. Hvis vi indsætter værdien af $z_{ij}$ og $z_{jk}$ i ulighederne (3.5) får vi $z_{i,k} \geq z_{ij} + z_{jk} - 1 = -1$ hvilket altid er rigtigt (hvorfor), hvorved vi opnår at $z_{ik}$ er ubegrænset af ulighederne i dette tilfælde.

$z_{ij} = 1$ og $z_{jk} = 0$: Her haves, at $\mathbf{x}_i$ og $\mathbf{x}_j$ er i samme gruppe og $\mathbf{x}_j$ og $\mathbf{x}_k$ *ikke* er i samme gruppe. Det betyder, at $\mathbf{x}_i$ og $\mathbf{x}_k$ ikke skal tvinges til at være i samme gruppe; altså at $z_{ik}$ ikke skal tvinges til at antage værdien 1. Indsættes værdierne for $z_{ij}$ og $z_{jk}$ i ulighederne (3.5) fås: $z_{ik} \geq z_{ij} + z_{jk} - 1 = 1 - 0 - 1 = 0$. Altså tvinges $z_{ik}$ ikke til at antage værdien 1.

$z_{ij} = 0$ og $z_{jk} = 1$: Samme argument som for forrige punkt.

$z_{ij} = z_{jk} = 1$: Nu er $\mathbf{x}_i$ og $\mathbf{x}_j$ i samme gruppe og $\mathbf{x}_j$ og $\mathbf{x}_k$ er i samme gruppe. Det vil sige, at vi nu skal sørge for, at $\mathbf{x}_i$ og $\mathbf{x}_k$ kommer i samme gruppe. Vi indsætter igen værdierne og ser, hvad der sker: $z_{ik} \geq z_{ij} + z_{jk} - 1 \leq z_{ik} = 1 + 1 - 1 = 1$. Altså sørger vi for, at $\mathbf{x}_i$ og $\mathbf{x}_k$ kommer i samme gruppe.

Dermed er der netop givet et *bevis* for, at vores model gør det den skal med hensyn til at lave konsistente grupper. En yderligere mængde betingelser, der ofte introduceres til grafopdelingsproblemer såkaldte *kardinalitetsbegrænsninger*. Det vil sige, en række begrænsninger, som sætter et minimum/maksimum på antallet af medlemmer i grupperne. Lader man $u$ være en øvre grænse og $l$ være en nedre grænse på gruppernes størrelse, kan disse begrænsninger opskrives som

$$l - 1 \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ij}, \quad \forall i = 1, \ldots, n \tag{3.6}$$

$$u - 1 \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ij}, \quad \forall i = 1, \ldots, n. \tag{3.7}$$

På højresiden af begrænsningerne (3.6) og (3.7) tælles hvor mange dataobjekter der er i gruppe med dataobjekt $i$, mens venstresiden angiver en henholdsvis nedre grænse og øvre grænse på dette antal (hvorfor trækkes 1 fra henholdsvis den nedre og den øvre grænse for gruppestørrelserne?).

Alt i alt leder dette frem til graf-opdelingsproblemet givet ved:

$$\begin{aligned}
\max\ & w_1 \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{x}_{ij} z_{ij} - w_2 \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - \mathbf{x}_{ij}) z_{ij} \\
\text{s.t.:}\ & z_{ij} + z_{jk} - 1 \leq z_{ik}, && \forall i, j, k = 1, \ldots, n \\
& l - 1 \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ij}, && \forall i = 1, \ldots, n \\
& u - 1 \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ij}, && \forall i = 1, \ldots, n \\
& z_{ij} \in \{0, 1\}, && \forall i, j = 1, \ldots, n
\end{aligned}$$

Man kan genopdage en pointe nævnt tidligere, nemlig den lidt åbenlyse sammenhæng, at hvis $\mathbf{x}_i$ er i gruppe med $\mathbf{x}_j$, ja så må $\mathbf{x}_j$ også være i gruppe med $\mathbf{x}_i$. Det vil sige, at man kan hjælpe sin solver ved at angive, at $z_{ij} = z_{ji}$. På den måde kan den *substituere* $z_{ji}$ med $z_{ij}$ for på den måde at halvere antallet af variabler i modellen.

Man kan naturligvis også modellere dette direkte ved kun at definerer variablerne $z_{ij}$ for *alle ordnede par* af data objekter $\mathbf{x}_i$ og $\mathbf{x}_j$. Dette vælger vi dog elegant at undgå i denne bog.

---

**Opgave 3.11:** Find *præferencematricen* via link på din uddannelses learning management system og visualiser matricen ved hjælp af en graf, som har en kant/pil fra en studerende $i$ til studerende $j$, hvis studerende $i$ ønsker at være i gruppe med studerende $j$.

**Opgave 3.12:** Opstil et graf-opdelingsproblem baseret på relationerne angivet i *præferencematricen* og hvor der maksimalt er 5 og minimalt er tre i hver gruppe. Løs problemet og visualiser grupperne der dannes. Hvor mange ønsker blev opfyldt? Hvem fik flest ønsker opfyldt og hvor mange? Hvem fik færrest og hvor mange?

**Opgave 3.13:** Lad $z^*$ være den optimale objektfunktionsværdi fra **Opgave 3.12**. Tilføj en betingelse i programmet, som kræver, at

$$w_1 \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{x}_{ij} z_{ij} - w_2 \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - \mathbf{x}_{ij}) z_{ij} = z^*.$$

Maksimer nu det mindste antal ønsker en studerende får opfyldt under den førnævnte bibetingelse.

---

# 4 Lokationsplanlægning og netværksdesign

## 4.1 Hvad er lokaliseringsplanlægning og netværksdesign?

Lokaliseringsplanlægning og network design beskæftiger sig med hvor *faciliteter* skal placeres og hvorledes *netværket*/infrastrukturen mellem faciliteterne skal etableres. Dette er i sagens natur *strategiske* planlægningsproblemer idet planlægning af placeringen af for eksempel nye produktionsfaciliteter, sygehuse, brandstationer og lignende har en særdeles lang tidshorisont (se evt. Handfield og Bozarth (2016) for en definition på strategisk planlægning).

I sin grundessens består lokationsplanlægning i at finde optimale placeringer for nogle faciliteter med henblik på at servicere en form for efterspørgsel. Vi vil her bruge ordet *facilitet* om hvad end det nu er der skal placeres; det vil sige, at facilitet skal forstås bredt. Når man sige "en optimal" placering er det naturligvis i forhold til en objektfunktion som udtrykker en beslutningstagers præferencer og under hensyntagen til en række begrænsninger.

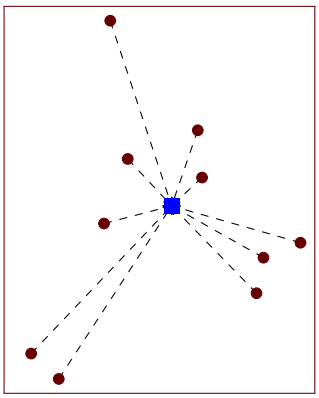
Lokationsplanlægning er anvendeligt inden for mange discipliner såsom teoretisk matematik, økonomi, geografi, logistik og ingeniørvidenskab og har derfor tiltrukket sig megen opmærksomhed i litteraturen.

Et af de første lokationsplanlægningsproblemer, som blev studeret i litteraturen, er det såkaldte Weber problem, som består i at placere én facilitet blandt $n$ efterspørgselspunkter i en Euklidisk plan, således at summen af afstandene fra efterspørgselspunkterne til faciliteten minimeres. Det er nærliggende at visualisere Weber problemet som i Figur 4.1.

I løbet af 1960'erne, samtidig med udviklingen af algoritmer til at løse lineære programmeringsproblemer, begyndte fundamentet for det man i dag betragter som lokaliseringsplanlægning at tage sin form. Et af de første åbenlyse problemer som blev studeret, var Weber problemet hvor man ikke nøjes med at placere én facilitet med derimod et arbitrært antal. Dette problem blev døbt *multi-Weber problemet*

eller det *planære p-median problem* hvor $p$ angiver antallet af faciliteter som skal placeres og blev blandt andet studeret af Cooper (1963).

Senere udkom to centrale artikler Hakimi (1964) og Hakimi (1965), som beskrev hvorledes faciliteter kan placeres i et netværk/på en graf således at summen af afstandene fra knuderne i grafen til de placerede faciliteter minimeres. Dette problem kaldes i dag for et *p-median* problem. Hakimi (1965) udvidede modellen til også at betragtet problemet hvor den *maksimale* afstand fra en knude til den nærmeste facilitet skulle minimeres. Dette kaldes i dag for et *p-center* problem.



**Figur 4.1: Illustration af Weber problemet.**

Det lykkedes Hakimi at vise, at selvom man kan placere faciliteterne langs kanterne i grafen, så vil der, for $p$-median problemet, altid være en optimal løsning hvor faciliteterne placeres i knuderne. Dette resultat har haft stor betydning for mange lokaliseringsproblemer i det det betyder, at mange lignende problemer kan opskrives ved hjælp af et (blandet) heltalsproblem, da der kun er et endeligt antal mulige steder at placere faciliteterne som skal undersøges.

I 1965 formulerede Balinski (1965) det, som kom til at blive kendt som det ubegrænsede lokaliseringsproblem, eller på engelsk *the uncapacitated facility location problem (UFLP)*. Dette problem har haft enorm betydning for modellerne inden for lokaliseringsplanlægning, da det er særdeles generelt, og kan udvides i det nærmest uendelige, for at tage højde for mange andre faktorer, end de som Balinski betragtede i 1965.

I det følgende betragtes kun problemer, som falder inden for kategorien diskrete lokaliseringsproblemer. Det vil, løst sagt, sige problemer, som kan opskrives som blandede heltalsproblemer.

Vi vil starte med at give en matematisk beskrivelse af $p$-median og $p$-center problemerne, da disse er nogle af de mest basale diskrete lokaliseringsproblemer. Herefter går vi videre til lokaliseringsproblemer med faste omkostninger, efterfulgt af såkaldte *covering* problemer. Slutteligt introduceres det simple at formulere, men svære at løse, *fixed charge transportation problem* og fixed charge minimum cost network flow problem. Begge de to sidste problemer er såkaldte

netværksplanlægningsproblemer.

Vi vil i den del af bogen, som omhandler lokaliseringsproblemer fast bruge følgende notation. Vi lader $\mathcal{I}$ være en mængde af mulige lokationer for faciliteter og vi vil lade $\mathcal{J}$ være en mængde af efterspørgselspunkter, som skal serviceres fra faciliteterne. Endvidere lader vi $d_j > 0$ være antallet af efterspurgte enheder af et produkt hos efterspørgselspunkt $j \in \mathcal{J}$. Vi vil endvidere fast benytte følgende variabler:

$$y_i = \begin{cases} 1, & \text{hvis der placeres en facilitet på lokation } i \in \mathcal{I} \\ 0, & \text{ellers} \end{cases}$$

$x_{ij}$ = andelen af efterspørgselspunkt $j$'s efterspørgsel som serviceres fra lokation $i$

Med denne notation er vi klar til at formulere nogle af de mest centrale lokaliseringsplanlægningsproblemer.

## 4.2 $p$-median og $p$-center problemet

I dette afsnit opstilles to meget centrale problemer indenfor lokaliseringsplanlægningen. En grundlæggende antagelse vil være, at der skal placeres $p \geq 1$ faciliteter blandt de $\mathcal{I}$ mulige lokationer og vi vil antage, at dette antal, $p$, er eksogent givet. Vi vil her lade $c_{ij} \geq 0$ være omkostningen forbundet med at servicere al efterspørgsel hos $j \in \mathcal{J}$ fra en facilitet placeret ved lokation $i \in \mathcal{I}$.

### 4.2.1 $p$-median problemet

I $p$-median problemet ønskes det, at summen af afstandene fra efterspørgselspunkterne til de faciliteter som de serviceres fra minimeres. I denne formulering lægges der op til, at hvert efterspørgselspunkt skal serviceres fra en og kun én facilitet. Det betyder, at vores $x_{ij}$ variabler enten skal være lig 1 (100% af efterspørgslen hos $j$ håndteres hos $i$) eller 0 (intet af $j$'s efterspørgsel serviceres fra $i$). Det vil altså sige, at i $p$-median problemet vil vi indføre en begrænsning på $x_{ij}$-variablerne som kræver at disse bliver *binære*. Når dette er gjort, kan vi gå videre med at opstille en objektfunktion for problemet; Hvis $j$'s efterspørgsel serviceres fra facilitet $i$ indtræffer en omkostning på $c_{ij}$, og hvis ikke, så haves ingen omkostninger forbundet med parret $(i, j)$. Det vil sige, at vi får en klassisk

sum-objektfunktion givet ved:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij}.$$

For at sikre at alle kunder serviceres, indføres begrænsningerne

$$\sum_{i \in \mathcal{I}} x_{ij} = 1, \qquad \forall j \in \mathcal{J} \tag{4.1}$$

Disse begrænsninger sørger præcist for, at for hvert efterspørgselspunkt $j$ skal der være en begrænsning, som vælger præcis en $x_{ij}$ variable til at være lig 1 og resten til at tage værdien 0. Dette er netop hvad begrænsningerne (4.1) gør.

Begrænsningerne (4.1) er desværre ikke nok til at sørge for, at et efterspørgselspunkt kun kan serviceres fra en lokation, hvis man har valgt at placere en facilitet der. Med andre ord, skal vi have formuleret implikationen $y_i = 0 \Rightarrow x_{ij} = 0$. Dette kan vi gøre med følgende mængde af begrænsninger:

$$x_{ij} \leq y_i, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}.$$

Disse begrænsninger sørger netop for, at hvis $y_i = 0$, så er $x_{ij} = 0$. På den anden side, hvis $y_i = 1$ så er $x_{ij}$ ikke begrænset af $y_i$ og modellen kan således vælge at servicere efterspørgselspunkt $j$ fra alle åbne faciliteter. Slutteligt skal vi sørge for, at der vælges præcis $p$ faciliteter i den endelige løsning:

$$\sum_{i \in \mathcal{I}} y_i = p. \tag{4.2}$$

Opsummeret kan $p$-median problemet altså opskrives som følger

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij} & & \\
\text{s.t.:} \quad & \sum_{i \in \mathcal{I}} x_{ij} = 1, & & \forall j \in \mathcal{J} \\
& x_{ij} \leq y_i, & & \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& \sum_{i \in \mathcal{I}} y_i = p, & & \\
& x_{ij} \in \{0, 1\}, & & \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& y_i \in \{0, 1\}, & & \forall i \in \mathcal{I}
\end{aligned} \tag{4.3}$$

Man bør her notere sig, at hvis man er givet en konstellation af åbne faciliteter givet ved $\bar{y}$, så vil løsningen til $x_{ij}$-variablerne (givet $y = \bar{y}$ er fast) altid være en 0-1 løsning, også selvom man fjerner heltalskravet til $x_{ij}$-variablerne. Dette skyldes, at modellen altid blot vil vælge at servicere efterspørgselspunkt $j$ fuldt fra den åbne facilitet hvor $c_{ij}$ er mindst. Tilføjes kapacitetsbegrænsninger til problemet, er dette ikke tilfældet mere, da man risikerer at overstige faciliteternes kapaciteter ved at lave en sådan tildeling. I tilfældet hvor der ikke er kapacitetsbegrænsninger kan (4.3) derfor ændres til $x_{ij} \geq 0$ for alle $i \in \mathcal{I}$ og $j \in \mathcal{J}$. Denne observation er ikke helt ligegyldig i forhold til beregningstider: husk, at et branch and bound træ kan (i worst case) have i størrelsesordenen $\mathcal{O}(2^m)$ knuder, hvor $m$ er antallet af binære variabler. Derfor er der stor forskel på, om $m = |\mathcal{I}|$ eller $m = |\mathcal{I}| \times |\mathcal{J}|$.

Man bør nu gå tilbage til sine noter omkring clustering og sammenligne $p$-median modellen med den lokationsbaserede clustering model.

### 4.2.2 $p$-center problemet

$p$-center problemet fokuserer ikke, som $p$-median problemet, på at minimere den total omkostning ved placeringen af nye faciliteter. I stedet fokuseres der på at minimere den maksimale omkostning/afstand fra et efterspørgselspunkt til den facilitet, som punktet er tildelt. Det vil sige, hvor $p$-median problemet fokuserer på systemets *effektivitet* fokuserer $p$-center problemet på *retfærdighed* i den endelige placering af lokationer. Vi vil her fortolke $c_{ij}$ som en omkostning efterspørgselspunktet $j$ har for at blive serviceret fra lokation $i$. Ved hjælp af de samme variabler, som blev brugt ovenfor, kan $p$-center problemet modelleres som følgende heltalsproblem:

$$
\begin{aligned}
\min \ & \max_{j \in \mathcal{J}} \left\{ \sum_{i \in \mathcal{I}} c_{ij} x_{ij} \right\} & & \\
\text{s.t.:} \ & \sum_{i \in \mathcal{I}} x_{ij} = 1, & & \forall j \in \mathcal{J} \\
& x_{ij} \leq y_i, & & \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& \sum_{i \in \mathcal{I}} y_i = p, & & \\
& x_{ij} \in \{0, 1\}, & & \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& y_i \in \{0, 1\}, & & \forall i \in \mathcal{I}
\end{aligned}
$$

Objektfunktionen for $p$-center problemet er åbenlyst ikke-lineær, men kan relativt let *linariseres* (se **Opgave 4.2**).

**Opgave 4.1:** Reflekter over hvorfor de lokationsbaserede clustering modeller hed netop *lokationsbaserede*.

**Opgave 4.2:** Omskriv $p$-center problemet vha. nye variabler $\rho_j$ og $\rho^{\max}$, på samme vis som blev gjort i kapitel 3.

**Opgave 4.3:** Eliminer $\rho_j$ variablerne fra din omskrivning i **Opgave 4.2** således, at $p$-center problemet er formuleret udelukkende med $y$, $x$ og $\rho^{\max}$-variabler.

**Opgave 4.4:** For $p$-median problemet blev det postuleret, at det ikke var nødvendigt at inkludere heltalsbegrænsningen på $x_{ij}$-variablerne og at disse kunne erstattes af begrænsningerne $x_{ij} \geq 0$. Hvorfor er begrænsningen $x_{ij} \leq 1$ ikke nødvendig?

## 4.3 Lokaliseringsplanlægning med faste åbningsomkostninger

I $p$-median og $p$-center problemerne var det en antagelse, at antallet af faciliteter som skulle placeret var eksogent givet. I mange tilfælde, er det ikke antallet af faciliteter en beslutningstager bekymrer sig for, men snarer omkostningen ved at etablere dem. Man kan notere sig, at givet at $c_{ij}$ er ikke-negative, vil den optimale omkostning i $p$ median problemet være en ikke-stigende kurve som en funktion af antallet af $p$ (hvorfor?). Omvendt vil omkostningen forbundet med drift og etablering af faciliteter være en ikke-aftagende funktion af $p$. Noget sådan generelt kan ikke siges om de total omkostninger. Funktionen for de totale omkostning (service + faste) er ikke garanteret at "opføre sig pænt" som det ses i Figur 4.2.

For at gøre fremstillingen mere stringent indføres nu endnu en parameter, nemlig den faste omkostning $f_i > 0$ ved at åbne en facilitet ved lokation $i \in \mathcal{I}$. Vi vil nu præsentere det såkaldte ubegrænsede lokaliseringsproblem (UFLP) som på mange måder er lig $p$-median problemet. Man skal stadig servicere alle efterspørgselspunkter og sørge for, at serviceringen kun sker fra faciliteter, som rent faktisk er åbne. Hvad der adskiller UFLP fra $p$-median problemet er, at *kardinalitetsbegrænsningen* (4.2) fjernes for, at omkostningen mellem serviceomkostninger og faste omkostninger balanceres af objektfunktionen i stedet. Matematisk kan

UFLP beskrives ved følgende blandede heltalsprogram:

$$\begin{aligned}
\min \quad & \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \\
\text{s.t.:} \quad & \sum_{i\in\mathcal{I}} x_{ij} = 1, && \forall j \in \mathcal{J} \\
& x_{ij} \leq y_i, && \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& x_{ij} \geq 0, && \forall i \in \mathcal{I}, j \in \mathcal{J} \\
& y_i \in \{0,1\}, && \forall i \in \mathcal{I}
\end{aligned} \tag{4.4}$$

Til trods for ligheden med $p$-median problemet, er UFLP ofte i praksis meget svære at løse til optimalitet, da de to dele af objektfunktionen har meget forskellige absolutte størrelser. Man kan notere sig, at der er $|\mathcal{I}| \times |\mathcal{J}|$ begrænsninger af typen $x_{ij} \leq y_i$. Dette kan potentielt blive et problem, hvis antallet af lokationer og/eller antallet af efterspørgselspunkter bliver stort, da det kan bruge en frygtelig masse af en computers hukommelse blot at lagre sådan en model. Det er dog muligt at undgå disse mange begrænsninger ved at *aggregere* dem. Hvis vi summer begrænsningerne sammen over $j \in \mathcal{J}$ for alle $i \in \mathcal{I}$ får vi begrænsningerne

$$\sum_{j\in\mathcal{J}} x_{ij} \leq \sum_{j\in\mathcal{J}} y_i = |\mathcal{J}| y_i, \quad \forall i \in \mathcal{I} \tag{4.5}$$

Disse begrænsninger gør det samme (for heltallige $y$) som begrænsningerne (4.4): Hvis $y_i = 0$ medfører det, at $x_{ij}$-variablerne skal være lig 0 for alle $j \in \mathcal{J}$. Altså, hvis en facilitet ikke er åben, kan man ikke tildele noget efterspørgsel til den. Hvis på den anden side $y_i = 1$ så står der er, at $\sum_{j\in\mathcal{J}} x_{ij} \leq |\mathcal{J}|$ hvilket altid er korrekt. På den måde, er $|\mathcal{I}| \times |\mathcal{J}|$ begrænsninger blevet til $|\mathcal{I}|$ begrænsninger. Det har dog den bagside, at *LP relakseringen* bliver meget svagere, og den formulering hvor man bruger (4.5) frem for (4.4) er i litteraturen følgeligt kendt som *den svage formulering*.

### 4.3.1 Kapacitetsbegrænsninger

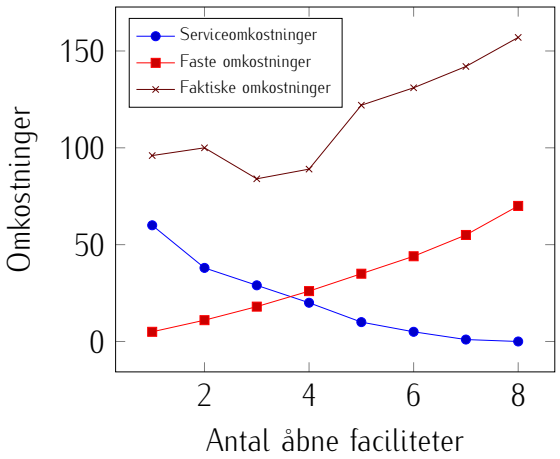
I mange praktiske tilfælde vil der være en *kapacitetsbegrænsning* på faciliteterne; maksimal produktion per periode, maksimal lagerplads, osv.. Derfor arbejdes der ofte med det, der hedder det *kapacitetsbegrænsede lokaliseringsproblem* (på engelsk Capcitated Facility Location Problem (CFLP)). Dette er i sin enkelthed blot UFLP som blev formuleret ovenfor med den ekstra feature, at kapaciteten i hver facilitet er begrænset opadtil. I det følgende, lad $s_i > 0$ være den maksimale kapacitet

på lokation $i \in \mathcal{I}$, hvis en facilitet åbnes her. Kapacitetsbegrænsningerne kan nu formuleres på følgende vis

$$\sum_{j \in \mathcal{J}} d_j x_{ij} \leq s_i y_i, \quad \forall i \in \mathcal{I}. \tag{4.6}$$

Man bør her sammenligne (4.5) med (4.6). De aggregerede begrænsninger (4.5) kan betragtes som kapacitetsbegrænsninger for et CFLP, hvor hvert efterspørgselspunkt har en efterspørgsel på én enhed ($d_j = 1$ for alle $j \in \mathcal{J}$), og hvor hver facilitet har en kapacitet på $|\mathcal{J}|$ enheder. Altså er CFLP en *generalisering* af UFLP, i den forstand, at hvis der eksisterer en effektiv måde at løse et CFLP på, så kan denne metode også bruges til at løse et UFLP på effektiv vis. Rent faktisk, er begrænsningerne (4.4) ikke længere nødvendige, da (4.6) sørger for, at ingen efterspørgselspunkter kan serviceres fra en facilitet som ikke er etableret. Det vil dog ofte være tilfældet, at (4.4) kan hjælpe til at styrke LP-relakseringen af problemet hvorved det kan løses hurtigere.

Det er væsentligt, at når der er kapacitetsbegrænsninger til stede, gælder der ikke længere at man kan fjerne heltalskravet fra $x_{ij}$-variablerne, og så stadig få en heltalsløsning ud. Her bliver man derfor nødt til at afgøre med sig selv (og beslutningstagerne), om det er brugbart i en løsning, at en kunde serviceres fra mere end en facilitet eller ej. Hvis man kræver, at hver kunde skal serviceres fra en og kun én facilitet, så kaldes problemet et *single source* CFLP (SS-CFLP) og er endog meget svære at løse end det almindelige CFLP.




**Figur 4.2: Minimale serviceomkostninger, faste omkostninger og total omkostninger som funktion af antallet af åbne faciliter.**

## 4.4 Covering-problemer

I de foregående afsnit har der været fokuseret på det som i litteraturen bliver betragtet som *location-allocation* modeller: Nogle faciliteter er blevet placeret (location) mens efterspørgslen er blevet allokeret (allocation). I det følgende vil vi betragte en anden type lokaliseringsproblemer, nemlig covering-problemer. Når man beslutter hvor en facilitet, som yder en service skal placeres (for eksempel

hvor en ambulance skal holde og vente på nødopkald), vil det ofte være tilfældet, at kunder/et efterspørgselspunkter kun kan modtage denne service, hvis de er inden for en hvis afstand af den nærmeste facilitet (for eksempel, at ambulancen kan ankomme inden for 10 minutter). Denne type problemer er lige netop covering-problemer. Man kan bekvemt forestille sig, at en facilitet placeret på en specifik lokation dækker (covers) et bestemt område, og at alle kunder inden for dette område vil blive dækket af faciliteten, såfremt den etableres.

Generelt set er der to slags covering problemer, nemlig *set covering* og *maximal covering* problemer. Når man taler om et *set covering* problem, minimeres de totale omkostninger, som opstår ved at placere en række faciliteter, der skal *dække alle kunder*. Hvis man er i det specielle tilfælde, at placeringen af faciliteterne har den samme omkostning uanset hvilken lokation man betragter, svarer dette til at minimere antallet af faciliteter, der skal til for at dække alle kunder. Når man har analyseret sin løsning til *set covering* problemet vil man ofte observere, at relativt få faciliteter kan dække en høj procentdel af kunderne, og at hvis man ønsker fuld dækning, kræver det placering af et stort antal faciliteter. Da placeringen af så mange faciliteter kan være umuligt (for eksempel på grund af budgetbegrænsninger), betragter man derfor ofte også den anden variant af covering problemet, nemlig *maximal covering* problemet. Dette problem maksimerer antallet af kunder, som dækkes, givet at der er et maksimalt antal faciliteter som kan placeres.

Vi vil i dette afsnit forsat bruge variablerne $y_i$, som angiver om en facilitet placeres på lokation $i \in \mathcal{I}$ eller ej og vi fortsætter også brugen af $f_i$ som en fast omkostning ved at etablere en facilitet på lokation $i \in \mathcal{I}$. Her indføres dog også to nye *parametre*:

$$a_{ij} = \begin{cases} 1, & \text{hvis efterspørgselspunkt } j \in \mathcal{J} \textit{ kan} \text{ dækkes fra lokation } i \in \mathcal{I} \\ 0. & \text{ellers} \end{cases}$$

$b_j =$ antallet af faciliteter, som skal dække efterspørgselspunkt $j \in \mathcal{J}$.

Endvidere fortsættes brugen af $p$, som det maksimale antal faciliteter, som kan placeres (lige som i $p$-median problemet).

### 4.4.1 Set covering problemet

I set covering problemet ønsker man at dække alle kunder, og at gøre dette til en minimal omkostning. Da variablerne er på plads ($y_i = 1$ hvis og kun hvis en

facilitet placeres ved lokation $i \in \mathcal{I}$) kan vi starte med at definere objektfunktionen for set covering problemet som følger:

$$\min \sum_{i \in \mathcal{I}} f_i y_i.$$

Med denne objektfunktion minimeres de total omkostninger forbundet med at placere faciliteter. Hvis der ikke er andre begrænsninger, end at $y_i \in \{0, 1\}$ vil $y_i = 0$ for alle $i \in \mathcal{I}$ og problemet er ikke særligt interessant. Derfor skal en mængde af begrænsninger, som sørger for, at alle kunder bliver dækket (covered) introduceres. Der skal altså for hver kunde $j \in \mathcal{J}$ åbnes $b_j$ faciliteter, som er inden for en afstand hvor faciliteterne kan dække kunde $j$. Spørgsmålet er så, hvordan man kan identificere, om en facilitet på lokation $i$ har potentialet til at dække kunde $j$? Det er netop angivet med parameteren $a_{ij}$. Det vil sige, at man skal introducere begrænsninger af følgende form

$$\sum_{i \in \mathcal{I}} a_{ij} y_i \geq b_j, \quad \forall j \in \mathcal{J}. \tag{4.7}$$

Alternativt, kan man definere en mængde $\mathcal{I}_j = \{i \in \mathcal{I} \ : \ a_{ij} = 1\}$ af alle lokationer hvorfra kunde $j$ vil kunne dækkes fra og så skrive begrænsningerne som følger

$$\sum_{i \in \mathcal{I}_j} y_i \geq b_j, \quad \forall j \in \mathcal{J}. \tag{4.8}$$

De to mængder af begrænsninger, (4.7) og (4.8), er blot to måder at skrive det samme på.

Til slut skal man også sørge for, at der ikke åbnes mere end $p$ faciliteter og her bruges den kendte begrænsning

$$\sum_{i \in \mathcal{I}} y_i \leq p.$$

Det vil sige, at man alt i alt ender med programmet

$$\min \sum_{i \in \mathcal{I}} f_i y_i \tag{4.9}$$

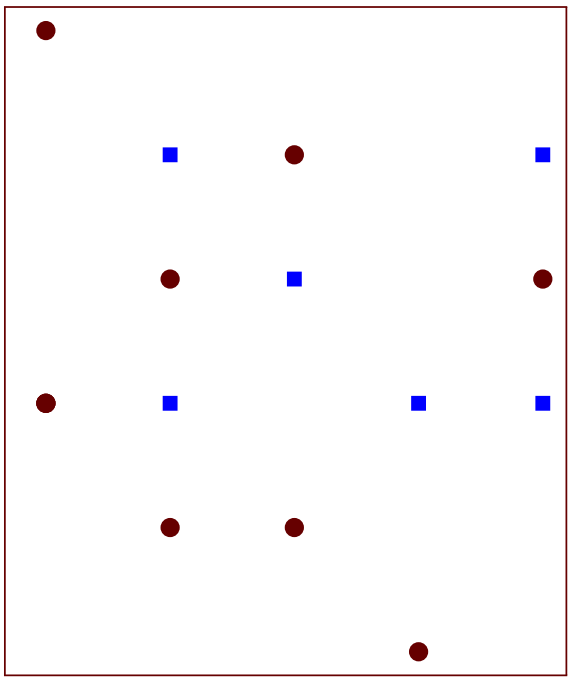
$$\text{s.t.:} \sum_{i \in \mathcal{I}} a_{ij} y_i \geq b_j, \qquad \forall j \in \mathcal{J} \tag{4.10}$$

$$\sum_{i \in \mathcal{I}} y_i \leq p,$$

$$y_i \in \{0, 1\}, \qquad \forall i \in \mathcal{I} \tag{4.11}$$

| | Lokationer | | | | | |
|---|---|---|---|---|---|---|
| kunder | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 22 | 10 | 20 | 28 | 10 | 22 |
| 2 | 30 | 22 | 45 | 40 | 22 | 10 |
| 3 | 30 | 22 | 45 | 40 | 22 | 10 |
| 4 | 14 | 32 | 36 | 22 | 20 | 14 |
| 5 | 20 | 45 | 41 | 22 | 32 | 28 |
| 6 | 42 | 14 | 41 | 50 | 28 | 32 |
| 7 | 14 | 32 | 10 | 10 | 20 | 32 |
| 8 | 22 | 10 | 32 | 32 | 10 | 10 |
| 9 | 30 | 22 | 45 | 40 | 22 | 10 |
| 10 | 22 | 30 | 42 | 32 | 22 | 10 |

Tabel 4.1: Afstandsmatrice mellem kunder og lokationer.



Figur 4.3: Set covering problem hvor kunder er markeret med røde cirkler og mulige lokationer er markeret med blå kvadrater.

*Eksempel* 4.1. I dette lille eksempel skal vi betragte et covering problem som vist i Figur 4.3. Afstandsmatrisen er givet i Tabel 4.1. Det antages, at en kunde ikke kan serviceres af en facilitet, som ligger strengt mere end 20 afstandsenheder væk, at der må placeres lige så mange faciliter som det ønskes ($p = 6$) og at $b_j = 1$ for alle kunderne. Vi vil også antage, at $f_i = 1$ for alle lokationerne.

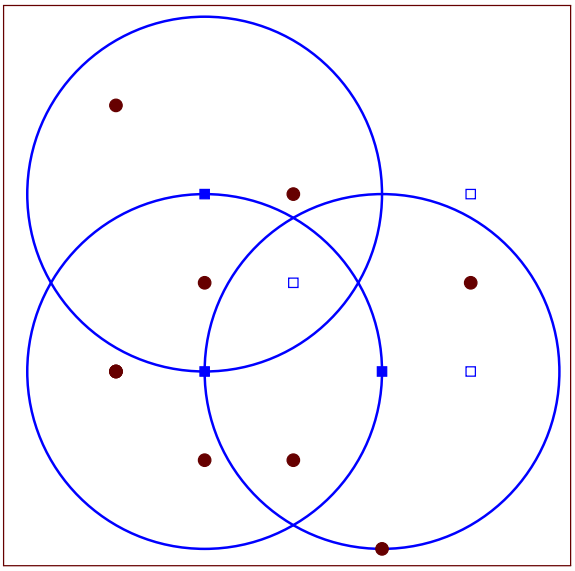
For kunde 1, kan vi se, at $a_{1,1} = a_{4,1} = a_{6,1} = 0$ og $a_{2,1} = a_{3,1} = a_{5,1} = 1$. Altså kommer begrænsningen (4.10) for $j = 1$ til at se ud som følger

$$y_2 + y_3 + y_5 \geq 1.$$

Vi kan fortolke denne begrænsning som følger: hvis kunde $j = 1$ skal dækkes inden for en radius af 20 afstandsenheder, skal der etableres faciliteter på mindst én af lokationerne 2, 3 eller 5. Ligeledes får vi, for kunde $j = 2$, at $a_{1,2} = a_{2,2} = a_{3,2} = a_{4,2} = a_{5,2} = 0$ og at $a_{6,2} = 1$. Det leder til begrænsningen

$$y_6 \geq 1$$

hvilket betyder, at vi allerede nu kan se, at der skal placeres en facilitet ved lokation 6. Løses programmet (4.9)–(4.11) for dette datasæt fås en løsning med $y_1 = y_2 = y_6 = 1$ og $y_3 = y_4 = y_5 = 0$. Løsningen kan illustreres ved



**Figur 4.4: Løsning til set covering problemet fra Eksempel 4.1. Bemærk, at der er flere kunder, som er dækket af mere end en facilitet.**

Figur 4.4, hvor cirklerne har en radius på 20 afstandsenheder og de "udfyldte faciliteter" er de faciliteter som skal placeres for at opnå fuld dækning med færrest faciliteter.

---

**Opgave 4.5:** Opskriv hele $(a_{ij})_{i\in\mathcal{I},j\in\mathcal{J}}$-matricen fra Eksempel 4.1.

**Opgave 4.6:** Opskriv set covering problemet fra Eksempel 4.1 eksplicit. Det vil sige, opskriv problemet *uden* brug af sum-tegn ($\sum$) og "for alle"-tegn ($\forall$).

**Opgave 4.7:** Løs set covering problemet fra Eksempel 4.1 med grænsen for dækning sat ned fra 20 enheder til 19 enheder.

---

### 4.4.2 Maximal set covering problemet

Som nævnt før, er det ikke altid muligt at dække alle kunder i set covering problemet. Dette kan skyldes, at der ikke er økonomi til at etablere så mange faciliteter, som det kræver eller simpelthen, at kravet for dækning er for højt (for få ettaller i $a_{ij}$-matricen). Derfor kan det være gunstigt at betragte en mere pragmatisk model, som i stedet for at minimere omkostningerne ved at dække al efterspørgslen, maksimerer den efterspørgsel, som kan dækkes givet et bestemt budget.

For at benytte en sådan model, indføres nu nye binære variabler $z_j$, $j \in \mathcal{J}$, som defineres som følger:

$$z_j = \begin{cases} 1, & \text{hvis kunde } j\text{'s efterspørgsel dækkes af mindst } b_j \text{ faciliteter} \\ 0, & \text{ellers} \end{cases}$$

Med disse nye variabler ved hånden, kan det at maksimere antallet af kunder hvis efterspørgsel dækkes, formuleres med følgende objektfunktion

$$\max \sum_{j \in \mathcal{J}} z_j. \tag{4.12}$$

Begrænsningerne (4.10) kræver at alle kunder skal dækkes fra $b_j$ faciliteter og bør derfor omformuleres til

$$\sum_{i \in \mathcal{I}} a_{ij} y_i \geq b_j z_j, \quad \forall j \in \mathcal{J}. \tag{4.13}$$

Dermed bliver det ikke et krav at alle kunders efterspørgsel skal dækkes. Med objektfunktionen (4.12) vil en solver forsøge at sætte så mange $z_j$ variabler til 1 som muligt, men det kan kun lade sig gøre, hvis antallet af faciliteter, som er åbne og dækker kunde $j$ er større end eller lig med $b_j$ (jf. (4.13)). Som I tilfældet med set covering problemet, skal der også være en kardinalitetsbegrænsning på antallet af åbne faciliteter, hvilket kan formuleres som

$$\sum_{i \in \mathcal{I}} y_i \leq p.$$

Dette leder frem til maximal covering problemet

$$\begin{aligned} \max \quad & \sum_{j \in \mathcal{J}} z_j & & \\ \text{s.t.:} \quad & \sum_{i \in \mathcal{I}} a_{ij} y_i \geq b_j z_j, & & \forall j \in \mathcal{J} \\ & \sum_{i \in \mathcal{I}} y_i \leq p & & \\ & y_i \in \{0,1\}, & & \forall i \in \mathcal{I} \\ & z_j \in \{0,1\}, & & \forall j \in \mathcal{J}. \end{aligned} \tag{4.14}$$

**Opgave 4.8:** Hvad er det højeste antal kunder, som kan dækkes, hvis grænsen for dækning sættes fra 20 ned til 10 i Eksempel 4.1?

**Opgave 4.9:** Hvordan skal begrænsningen (4.14) ændres hvis man, i stedet for et maksimalt antal faciliteter, ønsker at indføre en budgetbegrænsning på etableringen af faciliteter?

**Opgave 4.10:** I stedet for at maksimere *antallet* af dækkede kunder, ønskes det at maksimere den totale *mængde* efterspørgsel, som dækkes (lad $d_j$ være mængden af efterspørgsel hos kunde $j$). Formuler denne version af max-covering problemet.

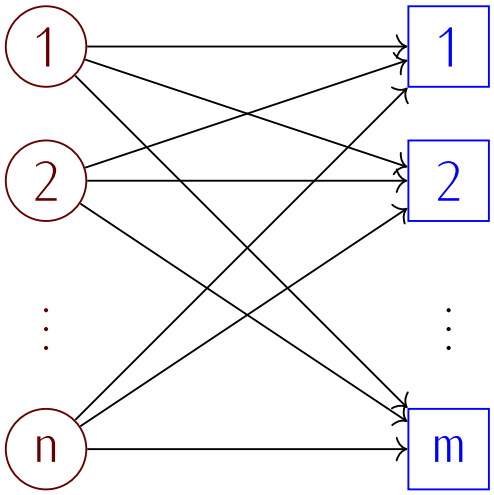
**Opgave 4.11:** Formuler max-covering problemet som et $p$-median problem.

## 4.5 Netværksdesign problemer

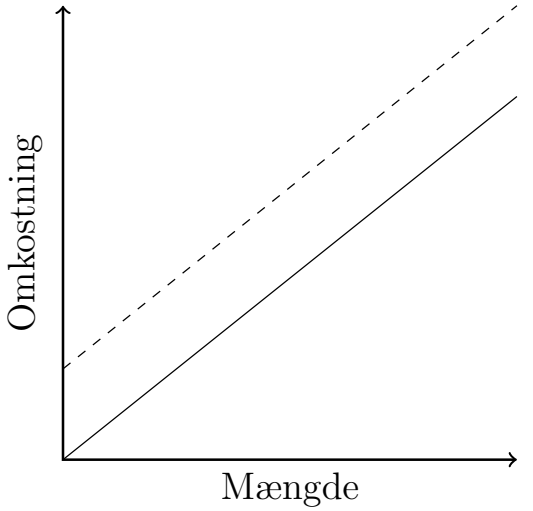
Netværksdesign problemer beskæftiger sig med etablering af netværk eller infrastruktur. Man bør ikke forfalde til den tanke, at "infrastruktur" her nødvendigvis refererer til vejnetværk. Netværksplanlægning foregår også i forbindelse med planlægning af mange andre områder som for eksempel planlægning af rørledninger, placering af ledere på printplader, planlægning af flows i en supply chain og mange andre.

### 4.5.1 Fixed charge transportation problem

Til at starte denne del om Netværksdesign betragtes et meget simpelt eksempel på et netværksplanlægningsproblem, nemlig det som kaldes et *fixed charge transportation problem* (FCTP). Som navnet afslører, er dette problem nært beslægtet med det lineære transportproblem. For at denne bog står selvstændigt, vil det lineære transportproblem her blive gennemgået: Det antages, at der findes en mængde af fabrikker $\mathcal{I}$ som hver har et positiv forsyning af et produkt på $s_i$ enheder. Der er endvidere en mængde bestående af aftagere, $\mathcal{J}$, som hver efterspørger $d_j$ enheder af produktet (det antages as $\sum_{i\in\mathcal{I}} s_i \geq \sum_{j\in\mathcal{J}} d_j$). For hver enhed af kunde $j$'s efterspørgsel der efterkommes fra fabrik $i$, pådrages en omkostning på $c_{ij}$ kroner. Målet er nu at finde en transportplan som sørger for, at hver kunde får sin efterspørgsel dækket og sådan at hver fabriks kapacitet respekteres. For




(a) Illustration af det lineære transportproblem.




(b) Omkostningsstruktur med (– – –) og uden (–) faste omkostninger.

**Figur 4.5: TV: Den todelte graf illustrer transportproblemet. TH: Omkostningsstruktur med og uden faste omkostninger.**

at gøre dette, introduceres en *kontinuert* variabel $g_{ij}$ som angiver hvor mange enheder af produktet der transporteres fra fabrik $i$ til kunde $j$. En formulering af det lineære transportproblem er således

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}g_{ij} \tag{4.15}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} g_{ij} \geq d_j, \qquad \forall j \in \mathcal{J}$$

$$\sum_{j\in\mathcal{J}} g_{ij} \leq s_i, \qquad \forall i \in \mathcal{I}$$

$$g_{ij} \geq 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{4.16}$$

Det lineære transportproblem kan illustreres ved en todelt graf som vist i Figur 4.5a. Hver $g_{ij}$-variabel svarer til mængden der sendes over en af *kanterne* i grafen. Af samme grund forkorter man ofte sin lingo en smule og erstatter "mængden, som sendes fra udbyder $i$ til aftager $j$" til "flowet på kanten $(i, j)$".

Formuleringen (4.15)–(4.16), af det lineære transportproblem, antager implicit at der allerede er infrastruktur, som kan benyttes og at der ikke er nogen vedligeholdelses omkostninger forbundet med at transportere fra $i$ til $j$, som ikke er *proportionale* med mængden der transporteres.

I Figur 4.5b angiver den optrukne linje den omkostningsstruktur, som det lineære transportproblem benytter: Der er en lineær relation mellem mængden der transporteres fra $i$ til $j$ og omkostningen som indtræffer. Ofte vil der dog være faste omkostninger forbundet med en sådan transport (etableringsomkostninger, opstartsafgifter, vedligehold af infrastruktur osv.). Dette er illustreret ved den stiplede linje i Figur 4.5b, hvor der så at sige betales en "lump sum", som er uafhængig af mængden for at benytte infrastrukturen mellem $i$ og $j$. Dette kan modelleres på følgende måde: Indfør en mængde nye binære variabler $y_{ij}$ defineres som

$$y_{ij} = \begin{cases} 1, & \text{hvis } g_{ij} > 0 \\ 0, & \text{ellers} \end{cases}$$

og lad $f_{ij} > 0$ være den faste omkostning forbundet med at sende et positivt flow på kanten $(i, j)$. Da skal først objektfunktionen opdateres til at indeholde de faste omkostninger:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} g_{ij} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} f_{ij} y_{ij}.$$

Som tidligere nævnt skal vi nu have modelleret relationen $g_{ij} > 0 \Rightarrow y_{ij} = 1$. Man kan her bruge et trick med at *negere* det førnævnte udsagn, for at få et ækvivalent:

$$\neg(g_{ij} > 0 \Rightarrow y_{ij} = 1) = (g_{ij} = 0 \Leftarrow y_{ij} = 0).$$

Udtrykket til højre har vi set før i forbindelse med lokaliseringsproblemerne. Vi skal dog her være opmærksom på, at $g_{ij}$ ikke længere er begrænset oppefra af 1, men derimod af $d_j$ og $s_i$. Det vil sige, at vi kan modellere udsagnet $g_{ij} > 0 \Rightarrow y_{ij} = 1$ på følgende vis:

$$g_{ij} \leq M y_{ij}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{4.17}$$

Hvor $M$ er tilstrækkeligt stor til ikke at begrænse $g_{ij}$ når $y_i = 1$, men stadig så lille som muligt. Med opdateringen af objektfunktionen og begrænsningerne

(4.17) kan FCTP problemet opskrives som følger:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}g_{ij} + \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} f_{ij}y_{ij} \tag{4.18}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} g_{ij} \geq d_j, \qquad \forall j \in \mathcal{J}$$

$$\sum_{j\in\mathcal{J}} g_{ij} \leq s_i, \qquad \forall i \in \mathcal{I}$$

$$g_{ij} \leq My_{ij}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J} \tag{4.19}$$

$$g_{ij} \geq 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}$$

$$y_{ij} \in \{0,1\}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{4.20}$$

Omend det ser uskyldigt ud at tilføje nogle faste omkostninger, er de beregningsmæssige besværligheder dette medfører ikke negligerbare! Det er, med software som CPLEX muligt at løse almindelige lineære transportproblemer med flere tusinde udbydere og aftager i løbet af, om ikke sekunder, så minutter på en almindelig privatcomputer. Med hensyn til versionen med faste omkostninger, kan et lille problem med $|\mathcal{I}| = |\mathcal{J}| = 20$ tage dage at løse.

---

**Opgave 4.12:** Gå tilbage og betragt det kapacitetsbegrænsede lokaliseringsproblem (CFLP) som var defineret som

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} x_{ij} = 1, \qquad \forall j \in \mathcal{J}$$

$$\sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i, \qquad \forall i \in I$$

$$x_{ij} \geq 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}$$

$$y_i \in \{0,1\}, \qquad \forall i \in \mathcal{I}.$$

Antag nu, at der på nuværende tidspunkt ikke findes infrastruktur mellem faciliteterne $\mathcal{I}$ og efterspørgselspunkterne $\mathcal{J}$. Derimod vil der tilgå en fast omkostning på $b_{ij}$ kroner hvis en kunde $j$ skal serviceres fra facilitet $i$ *uafhængigt* af hvilken andel af $j$'s efterspørgsel facilitet $i$ varetager. Opdater CFLP til at kunne håndtere disse nye faste omkostninger.

**Opgave 4.13:** I formuleringen, (4.18)–(4.20), af FCTP er begrænsningerne (4.19) af type *big-M*. Det er velkendt, at store værdier af $M$ kan give problemer når man løser IP'er. Derfor, find den mindste brugbare værdi af $M$ for hver par af $i$ og $j$. *Hint: hvad er det største $g_{ij}$ kan blive?*

**Opgave 4.14:** Lav substitutionen $g_{ij} = d_j z_{ij}$, hvor $z_{ij} \geq 0$ i formuleringen for FCTP. Hvorledes kan variablerne $z_{ij}$ fortolkes? Sammenlig formuleringen hvor $g_{ij}$ er substitueret med $d_j z_{ij}$ med formuleringen af et CFLP. Hvilke lighedspunkter og afvigelser er der mellem disse to programmer?

### 4.5.2 Fixed charge network flow problemer

Vi skal nu betragte et såkaldt *network flow problem*, hvor der senere bliver tilføjet faste omkostninger på at benytte kanter i en graf. Et minimum cost network flow problem, er et problem defineret over en graf $G = (V, E)$ hvor $V$ er knuderne og $E$ er kanterne i grafen. Der er for hver kant $(i, j) \in E$ en omkostning forbundet med at sende én enhed af et givet produkt hen over denne kant. Vi kalder denne enhedsomkostning $c_{ij}$ og vil vil antage, at $c_{ij} > 0$. Blandt knuderne i $V$ er der to specielle knuder; en såkaldt *source-knude*, $s \in V$, og en *sink-knude*, $t \in V$. Source-knuden, $s$, har en produktion af et produkt, som skal sendes gennem grafens knuder, via kanterne, til sink-knuden, $t$, som har en efterspørgsel på $d_t$ enheder af produktet. Vi vil her antage, at produktionen hos $s$ er stor nok til at imødekomme denne efterspørgsel. Dette *flow* af varer skal naturligvis sendes således, at omkostningen minimeres. De resterende knuder $V \setminus \{s, t\}$ kaldes transshipment knuder, da varerne blot rejser igennem disse knuder (her betyder $V \setminus \{s, t\}$ "mængden $V$ *fraregnet* knuderne $s$ og $t$").

For at modellere dette problem indfører vi ikke-negative, kontinuerte variabler $g_{ij}$ for alle kanter $(i, j)$ i $E$ defineret som følger:

$$g_{ij} = \text{Antal enheder af produktet som sendes fra } s \text{ til } t \text{ via kanten } (i, j).$$

Med denne definition, kan vi nemt opstille objektfunktionen

$$\min \sum_{(i,j)\in E} c_{ij} g_{ij}.$$

Når vi betragter source-knuden, hvorfra vi ved at $d_t$ enheder skal sendes (hvorfor?), ved vi, at summen af det flow som forlader knude $s$ skal være lig med $d_t$. Vi ved

også, at flowet ind i vores source-knude skal være lig med nul. Derfor kan vi opskrive følgende *flow-begrænsninger* for source-knuden:

$$\sum_{j:(s,j)\in E} g_{sj} = d_t$$
$$\sum_{i:(i,s)\in E} g_{is} = 0 \tag{4.21}$$

På samme vis kan man argumentere for flowet ind og ud af sink-knuden: Det samlede flow ind i sink-knuden skal matche efterspørgslen, og der skal ikke være noget flow ud:

$$\sum_{i:(i,t)\in E} g_{it} = d_t$$
$$\sum_{j:(t,j)\in E} g_{tj} = 0 \tag{4.22}$$

Det vi mangler nu er de såkaldte *flow conservation*-begrænsninger for transshipment knuderne. Disse skal sørge for at *in-flowet* i en transshipment knude skal være lig med *out-flowet*. Dette kan opskrives matematisk på følgende vis:

$$\sum_{i:(i,j)\in E} g_{ij} = \sum_{i:(j,i)\in E} g_{ji}, \quad \forall j \in V \setminus \{s,t\}$$

Man kan notere sig, at i og med at $c_{ij} > 0$, og det ikke er nødvendigt at sende noget ind i source-knuden og ud af sink-knuden, kan man faktisk undlade begrænsningerne (4.21) og (4.22). Med alle delelementer på plads, kan det fulde program for minimum cost network flow problemet opskrives som

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij} g_{ij} \\
\text{s.t.:} \quad & \sum_{j:(s,j)\in E} g_{sj} = d_t, \\
& \sum_{i\,(i,t)\in E} g_{it} = d_t, \\
& \sum_{i:(i,j)\in E} g_{ij} = \sum_{i:(j,i)\in E} g_{ji}, && \forall j \in V \setminus \{s,t\} \\
& g_{ij} \geq 0. && \forall (i,j) \in E
\end{aligned}$$

Minimum cost network flow problemets objektfunktion består, som transportproblemet, af en sum af omkostningsfunktioner, som er lineære i mængden der flyttes over en kant. Der vil ofte være faste omkostninger forbundet med at flytte varer fra et knudepunkt til et andet i form af, for eksempel, udgifter til fortoldning og/eller anden bureaukrati. Derfor udvides minimum cost network flow problemet til at indeholde faste omkostninger $f_{ij}$, som indtræffer når en kant i grafen benyttes. Udvidelsen er meget lig den for transportproblemet, og vi vil derfor indføre binære variabler $y_{ij}$ for hver kant $(i, j) \in E$ givet ved

$$y_{ij} = \begin{cases} 1, & \text{hvis kanten } (i, j) \text{ bruges til at sende flow fra } s \text{ til } t \\ 0, & \text{ellers} \end{cases}$$

Vi skal nu lave samme relation, som var mellem $g_{ij}$ og $y_{ij}$ for transportproblemet, nemlig at $g_{ij} > 0 \Rightarrow y_{ij} = 1$. Dette gøres naturligvis med big-M begrænsninger af formen

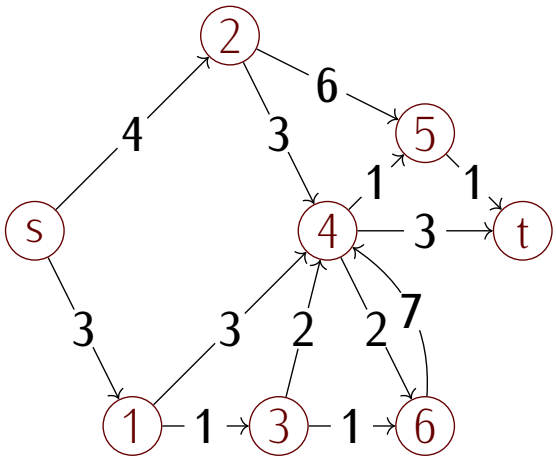
$$g_{ij} \leq M y_{ij}, \quad \forall (i, j) \in E \tag{4.23}$$

og ligesom med transportproblemet bliver objektfunktionen udvidet med et led, som indeholder disse nye fase omkostninger:

$$\min \sum_{(i,j)\in E} c_{ij} g_{ij} + \sum_{(i,j)\in E} f_{ij} y_{ij}.$$

*Eksempel* 4.2. I Figur 4.6 ses en graf med 8 knuder og 12 kanter. Der skal sendes 4 enheder flow fra source-knuden $s$ til sink-knuden $t$, således at omkostningerne minimeres. Omkostningerne ved at sende en enhed flow på kanterne, er ligeledes angivet i grafen med fed skrift på kanterne.

En optimal løsning til dette problem er at sende flowet på stien $(s) \to (1) \to (4) \to (5) \to (t)$ med en omkostning på 32. Hvis man nu antager, at der er en fast omkostning på $f_{ij} = 10$ forbundet med brug af enhver kant i grafen, vil løsningen til minimum cost network flow problemet være lige $32 + 4 \times 10 = 82$ da der er fire kanter i spil i løsningen. En optimal løsning til minimum cost network flow problemet med faste omkostning har derimod en optimal objektfunktionsværdi på 66, og består af stien $(s) \to (1) \to (4) \to (t)$, hvorved antallet af kanter, som bruges i løsningen, reduceres.




Figur 4.6: Illustration af grafen brugt i Eksempel 4.2

**Opgave 4.15:** Bestem den mindste valide værdi af $M$ i begrænsningerne (4.23).

**Opgave 4.16:** Udvid minimum cost network flow problemet til at have kapacitetsbegrænsninger på kanterne. Det vil sige, at der ikke kan sendes flere end $u_{ij}$ enheder på kanten $(i, j) \in E$.

**Opgave 4.17:** Udvid minimum cost network flow problemet således, at hvis der sendes flow på kanten $(i, j)$ så skal dette flow være større end eller lig med $l_{ij}$. *Hint: Modeller implikationen* $g_{ij} > 0 \Rightarrow g_{ij} \geq l_{ij}$ *ved hjælp af* $y_{ij}$ *variablerne.*

**Opgave 4.18:** Antag nu, at der findes to source-knuder $s_1$ og $s_2$ og to sink knuder $t_1$ og $t_2$ i grafen $G$, hvor produktionen hos $s_1$ skal sendes til $t_1$ og produktionen hos $s_2$ skal sendes til $t_2$. Omkostningen per enhed, der sendes på kanten $(i, j)$ afhænger ikke af om det er produceret til $t_1$ eller $t_2$ (det vil sige omkostningen per enhed er $c_{ij}$). Hvis en kant bruges ($y_{ij} = 1$) og den faste omkostning er indtruffet, kan den bruges til at sende til både $t_1$ og $t_2$ uden ekstra omkostninger. Modeller dette nye fixed charge network flow problem med to source-sink par. *Hint: indfør variabler* $g^1_{ij}$ *til at holde styr på flowet mellem* $s_1$ *og* $t_1$ *og variabler* $g^2_{ij}$ *til at holde styr på flowet mellem* $s_2$ *og* $t_2$.

# 5 Produktionsplanlægning

I kurser om eksempelvis Operations Management, undervises der i produktionsplanlægning med udgangspunkt i MRP og MPS. Men set fra et optimeringsbaseret synspunkt, er disse tilgange "blot" heuristikker uden nogen garanti for hvor gode produktionsplaner, der bliver udarbejdet. Vi vil derfor i dette kapitel fokusere på, hvorledes *optimale* produktionsplaner kan udarbejdes med udgangspunkt i bill of materials (BOM) og data omkring lead time, produktionskapaciteter, produktionspriser og lageromkostninger.

Man skal holde sig for øje, at vi i dette kapitel slet ikke behandler stokastisitet, som unægteligt er et vigtigt område at arbejde med inden for produktionsplanlægning. Vi vil dog her beskæftige os med basale, simple modeller, som læseren således kan tage udgangspunkt i, skulle mere avancerede modeller blive nødvendige. Det følgende kapitel er baseret på bogen "Production planning by Mixed Integer programming" (Pochet og Wolsey 2006), som er særdeles god at blive forstandig af.

## 5.1 Modelelementer

Der er en række modelelementer, som er til stede i alle, eller i hvert fald langt de fleste, produktionsplanlægningsproblemer. Produktionsplanlægning beskæftiger sig oftest, modsat job shop scheduling som præsenteres i kapitel 6, med at bestemme produktionspartier eller batches, særligt med fokus på *størrelsen* på sådanne batches og produktionstidspunktet, således at en forventet efterspørgsel over en endelig tidshorisont kan tilfredsstilles. Denne endelige tidshorisont kaldes også *planlægningshorisonten*. Efterspørgslen over planlægningshorisonten genereres ofte ved hjælp af *forecasts* i et make-to-stock system, eller på baggrund af ordrer i et make-to-order system. Men i virkelighedens verden kombineres de to metoder til generering af efterspørgsel således, at både eksisterende ordrer og forecasts benyttes til at fastsætte den forventede efterspørgsel.

For at opnå brugbare og økonomisk attraktive produktionsplaner, tager man oftest flere andre elementer af produktionssystemet med i betragtning. Det kunne for eksempel være adgangen til ressourcer (maskintimer, arbejdskraft, underleverandører og så videre), produktions- og lageromkostninger og andre performance indikatorer såsom service niveau.

Det simpleste produktionsplanlægningsproblem vi skal beskæftige os med i dette kapitel, er det såkaldte *single-item uncapacitated lot-sizing* problem. Dette problem består i at planlægge produktionen af ét produkt over en (diskretiseret) planlægningshorisont med efterspørgsel, som varierer over perioden. Dette problem indeholder alle de elementer, som blev beskrevet ovenfor, bortset fra, at der ikke er nogle ressource begrænsninger.

Der findes også modelleringselementer, som er til stede i nogle modeller, men slet ikke alle. Disse elementer gør som regel problemerne meget mere komplekse, og følgeligt meget sværere at løse. For at nævne to udvidelser, betragt følgende:

1. Produkterne, som skal produceres, konkurrerer om en eller flere delte ressourcer. Dette er præcis hvad man i andre kurser, og anden litteratur, måske er stødt på som *master production scheduling* (MPS).
2. Produkterne interagerer gennem flere produktniveauer; det vil sige, at nogle produkter er *output* i et produktionsstadie, men samtidig er de input i andre, eller de kan være leveret fra eksterne underleverandører. Dette skaber præcedens-begrænsninger mellem forsyningen af et produkt og forbruget heraf. Dette kendes (måske) fra *Material Requirements Planning* (MRP)

Vi vil i det følgende starte med problemet kaldet *uncapacitated lot-sizing*

## 5.2 En første simpel model - Uncapacitated lot-sizing model

Den første model vi skal beskæftige os med i dette kapitel, planlægger produktionen af et enkelt produkt, som skal igennem én enkelt proces og hvor denne proces er uden kapacitetsbegrænsninger. Det vil sige, at vi principielt har mulighed for at producere alt, der skal produceres over planlægningshorisonten i den første periode, for så at lægge denne produktion på lager, hvorfra efterspørgslen skal tilfredsstilles i efterfølgende perioder. Dette problem er i sig selv en stærkt forsimplet model og har ikke den store anvendelse i praksis, men i mange mere virkelighedsnære modeller, er denne model en sub-model, hvorfor den er oplagt

at begynde med. Som overskriften på dette afsnit indikerer, bliver dette problem omtalt som en uncapacitated lot sizing (ULS) model.

For at gøre problemet mere håndgribeligt vil vi introducere et indeks $t$, med $t = 1, \ldots, T$, til at repræsentere de (diskrete) tidsperioder, hvori produktionen skal planlægges. Her er $T$ altså den sidste tidsperiode i planlægningshorisonten. Formålet er således at planlægge hvor meget, der skal produceres i hver tidsperiode $t$ og hvor meget, der skal ligge på lager i slutningen af tidsperiode $t$ således, at efterspørgslen i hver periode kan tilfredsstilles.

Naturligvis er der mere på spil en blot at planlægge produktionen, så efterspørgslen kan imødegås: Omkostningerne forbundet med at producere og føre lager skal minimeres. Ofte udviser produktionsomkostningerne en form for *economies of scales*, som vi her vil modellere ved hjælp af en *fixed cost* del af omkostningsfunktionen. Det vil altså sige, at omkostningerne forbundet med at producere et parti af produktet kan dekomponeres i to dele: en fast omkostning, som er uafhængig af partiets størrelse og en konstant enhedsomkostning, som tilløber hver gang en ekstra enhed produceres.

For hver periode $t$, med $t \in \{1, \ldots, T\}$, har vi følgende ikke-negative parametre

$p_t$: enhedsomkostningen ved produktion i periode $t$.

$q_t$: fast produktionsomkostning i periode $t$. Vi vil antage, at $q_t$ er ikke-negativ.

$h_t$: enhedsomkostning ved lagerførelse i periode $t$.

$d_t$: efterspørgslen i periode $t$. Vi vil her antage, at efterspørgslen er heltallig og ikke-negativ.

Beslutningsvariablerne som skal bruges til denne simple ULS model er de følgende: $x_t$, $y_t$ og $s_t$. Variablerne $x_t$ angiver det producerede partis størrelse i periode $t$, og er derfor en ikke negativ, heltallig variabel. Variablerne $y_t$ er binære indikator variabler, som indikerer, om der produceres i periode $t$ ($y_t = 1$) eller ej ($y_t = 0$). Det vil sige, at vi har implikationen $x_t > 0 \Rightarrow y_t = 1$ for alle tidsperioder $t = 1, \ldots, T$. Slutteligt angiver variabler $s_t$ lagerniveauet ultimo periode $t$. Derfor er $s_t$ ligeledes en ikke-negativ heltallig variabel.

Med parametrene og variablerne defineret, kan vi gå videre med at opstille objektfunktionen for ULS modellen. Ønsket er, som nævnt, at minimere de samlede omkostninger forbundet med produktion og lagerhold, hvilket vil sige, at

objektfunktionen bliver

$$\min \sum_{t=1}^{T} (p_t x_t + q_t y_t + h_t s_t)$$

Det første led i summen angiver de variable omkostninger forbundet med produktion, mens det andet led angiver de faste omkostninger, som opstår, når et parti sættes i produktion. Slutteligt angiver det tredje og sidste led omkostningerne forbundet med at holde lager hen over de $T$ perioder i planlægningshorisonten.

En første mængde af begrænsninger i modellen er de såkaldte *efterspørgsels-begrænsninger*, som sørger for, at der samlet er lagerhold og produktion nok i hver periode til at imødekomme efterspørgslen i hver periode. Samtidig sørger disse begrænsninger for, at hvis der er en overproduktion i en periode i forhold til at imødekomme efterspørgslen, så lagerføres de overskydende enheder. Begrænsningerne ser ud som følger:

$$s_{t-1} + x_t = d_t + s_t, \quad \forall t = 1, \ldots, T \tag{5.1}$$

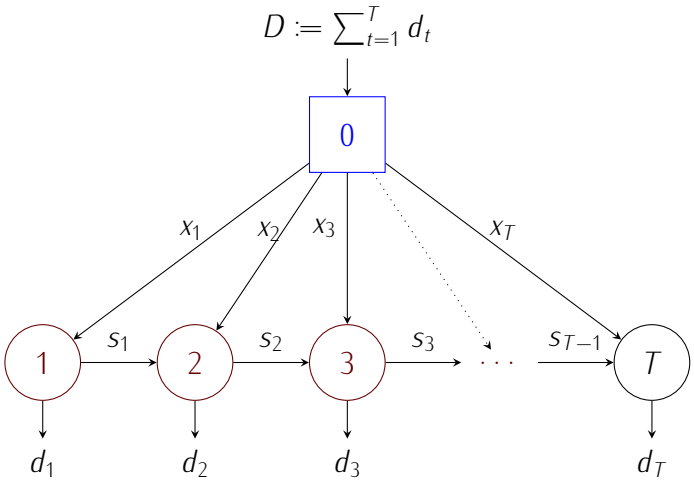
Begrænsningerne (5.1) kan altså læses fra venstre mod højre som følger: det som ligger på lager ultimo periode $t-1$ (og dermed primo periode $t$) plus det som produceres i periode $t$, skal være lig med den mængde, som efterspørges plus det, som lagerføres. Altså, det vi har på lager, plus det vi producerer skal fordeles til enten efterspørgsel eller til lager.

Som den næste mængde af begrænsninger, skal vi indføre begrænsninger, der håndhæver implikationen $x_t > 0 \Rightarrow y_t = 1$. Vi vil her benytte de sædvanlige big-$M$ begrænsninger givet ved

$$x_t \leq M_t y_t, \quad \forall t = 1, \ldots, T$$

Vi kan her notere os, at der ikke bruges ét stort $M$ men derimod et $M_t$ for hver tidsperiode.

Slutteligt har ULS modellen den antagelse, at startlageret ($s_0$) er tomt, hvorfor $s_0 = 0$. Samtidigt, kan vi på forhånd deducere, at lageret i slutningen af periode $T$ også vil være tomt, da vi ikke har nogen efterspørgsel, som skal tilfredsstilles på den anden side af planlægningshorisonten, og samtidig har det en omkostning at føre produkter på lager. Derfor haves ligeledes, at $s_T = 0$. Dermed kan vi




Figur 5.1: Illustration af en network flow fortolkning af ULS modellen

opstille ULS modellen som det lineære heltalsprogram

$$
\begin{aligned}
\min \quad & \sum_{t=1}^{T} (p_t x_t + q_t y_t + h_t s_t) \\
\text{s.t.:} \quad & s_{t-1} + x_t = d_t + s_t, && \forall t = 1, \ldots, T \\
& x_t \leq M_t y_t, && \forall t = 1, \ldots, T \\
& s_0 = s_T = 0, \\
& x_t, s_t \in \mathbb{N}_0, y_t \in \{0, 1\}, && \forall t = 1, \ldots, T
\end{aligned}
$$

Her angiver $\mathbb{N}_0$ mængden af de naturlige tal inklusiv 0, det vil sige $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$.

### 5.2.1 En network flow fortolkning af ULS modellen

Begrænsningerne (5.1) bliver også mange steder benævnt som *flow balancing* eller *flow conservation* begrænsninger. Dette skyldes, at ULS modellen kan ses som et specialtilfælde af et network flow problem, hvor hver tidsperiode bliver repræsenteret af en knude i en graf. Hver tidsperiode-knude, $t$, har så en efterspørgsel på $d_t$ enheder. Dertil tilføjes en *dummy* knude, som her kunne kaldes knude 0, som har et udbud på $D := \sum_{t=1}^{T} d_t$ enheder. Produktionsplanlægningsproblemet beskrevet som ULS kan således opstilles som et minimum cost network flow problem over en graf som illustreret i Figur 5.1.

Med denne network flow tilgang i baghovedet, kan vi også se, at hvis vi antager, at efterspørgslen er heltallig (hvilket vi har gjort, og hvilket desuden er uden tab

af generalitet), så behøver vi faktisk ikke specificere, at $x_t$ og $s_t$-variablerne skal antage heltallige værdier. Dette skyldes, at vi ved, at der findes en optimal løsning til sådan et network flow problem, hvor flow variablerne antager heltallige værdier. Når vi senere udvider modellen med flere produkter og delte ressourcer, bliver det igen nødvendigt at specificere heltalskravene på $x_t$ og $s_t$-variablerne, hvorfor vi undlader at gøre et større nummer ud af, at disse kan *relakseres*.

*Eksempel* 5.1. Virksomheden Tjase er en større producent af mælkeprodukter på det danske marked. Tjase har en afdeling, som beskæftiger sig med produktionsplanlægning og de skal netop til at lægge en produktionsplan for mælk for de kommende 10 uger. Fra forecasting afdelingen haves et forecast for de kommende 10 uger, og fra regnskabsafdelingen foreligger omkostningsestimater for produktion, opstart af produktion, samt lagerførelse af letmælk. Både efterspørgsels-forecastet og omkostningsestimaterne kan ses i følgende tabel:

| | Uge 1 | Uge 2 | Uge 3 | Uge 4 | Uge 5 | Uge 6 | Uge 7 | Uge 8 | Uge 9 | Uge 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Efterspørgsel | 100000 | 100301 | 100194 | 100597 | 100457 | 100154 | 99699 | 99538 | 99553 | 99704 |
| Produktionspris i kr/L | 4 | 4,02 | 3,93 | 3,99 | 3,96 | 3,94 | 3,90 | 3,86 | 3,76 | 3,83 |
| Opstartspris i kr | 50.000 | 50.000 | 50.000 | 500.000 | 50.000 | 50.000 | 50.000 | 50.000 | 50.000 | 500.000 |
| Lageromkostning i kr/L | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,6 | 0,6 | 0,6 | 0,6 |

Udover de i tabellen anførte værdier, haves yderligere begrænsninger til Tjases produktionsplanlægningsproblem: Lageret har en maksimal kølekapacitet på 100.000 liter mælk om ugen, og der kan maksimalt produceres 125.000 liter mælk om ugen. Disse to mængder af yderligere begrænsninger, kan opstilles ved hjælp af terminologien fra ULS modellen som følger

$$s_t \leq 100.000, \qquad \forall t = 1, \ldots, 10$$
$$x_t \leq 125.000, \qquad \forall t = 1, \ldots, 10.$$

Vi vil nu implementere en ULS model til at løse Tjases produktionsplanlægningsproblem for mælk i `OPL` og løse denne model i CPLEX Optimization studio. En Pyomo implementering af modellen kan ses i Kodeeksempel 5.1.

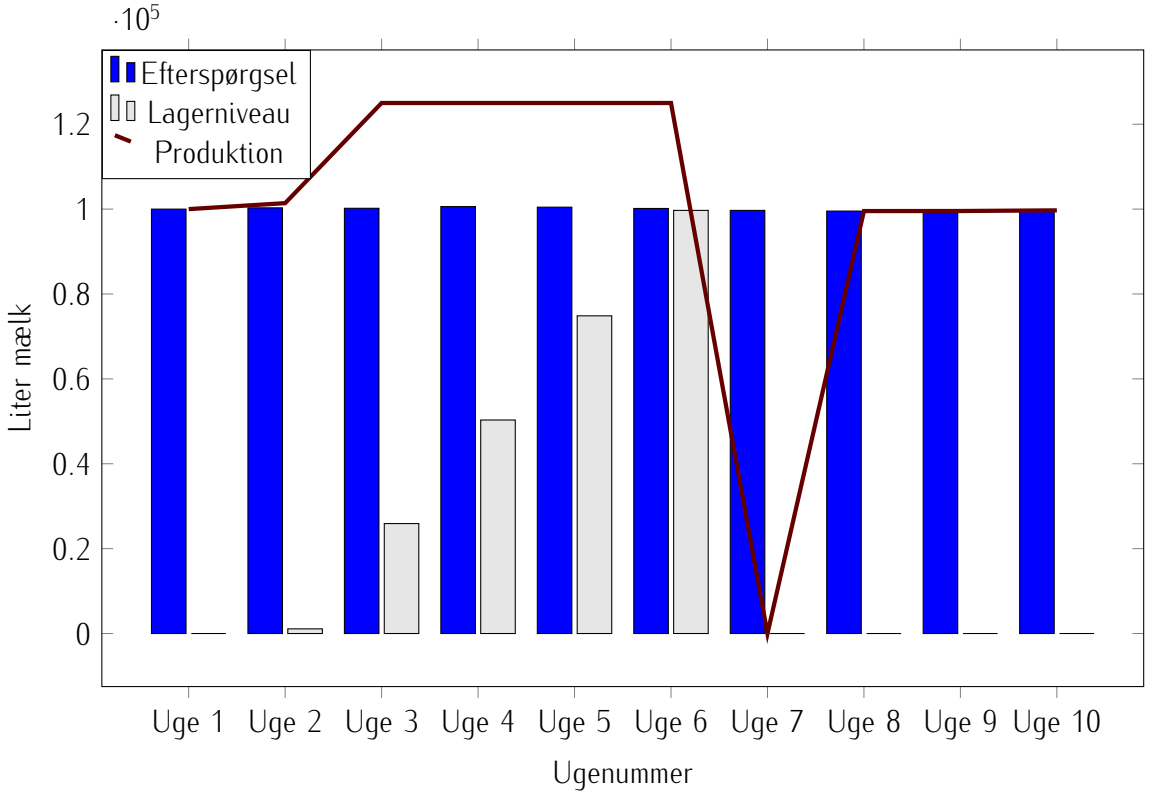
**Kodeeksempel 5.1: Python implementering ULS modellen fra Eksempel 5.1. Pyomo er brugt som modelleringssprog.**

```
# For simplicity, data is assumed stored in a dictionary called
   "ULSdata"
import pyomo.environ as pyomo  # used for modelling
# Declare the model variable
model = pyomo.ConcreteModel()
```

```
# Copy data to the model variable
model.periodNames = ULSdata['periods']
model.periods = range(0, len(model.periodNames))
model.demands = ULSdata['demands']
model.fixedCosts = ULSdata['fixedCosts']
model.inventoryCosts = ULSdata['inventoryCosts']
model.variableCosts = ULSdata['variableCosts']
# Sum the demand for a feasible "big-M"
bigM = sum(ULSdata['demands'])
# Define all the variables
model.y = pyomo.Var(model.periods, within=pyomo.Binary)
model.x = pyomo.Var(model.periods, within=pyomo.
   NonNegativeIntegers, bounds=(0, 125000))
model.s = pyomo.Var(model.periods, within=pyomo.
   NonNegativeIntegers, bounds=(0, 100000))
# Add objective function
model.obj = pyomo.Objective(
  expr=sum(model.variableCosts[t]*model.x[t]
  + model.fixedCosts[t]*model.y[t]
  + model.inventoryCosts[t]*model.s[t] for t in model.periods),
  sense=pyomo.minimize)
# Add flow conservation constraints
model.conservation = pyomo.ConstraintList()
for t in model.periods:
  if t==0:
    # Add flow conservation constraints, for first period
    model.conservation.add(expr=model.x[0]==model.s[0]+model.
       demands[0])
  else:
    model.conservation.add(model.s[t-1] + model.x[t]==model.s[t
       ]+model.demands[t])
# Add big-M constraints
model.bigM = pyomo.ConstraintList()
for t in model.periods:
  model.bigM.add(expr = model.x[t]<=bigM*model.y[t])
# Set s[0] equal to zero
model.s[0].setub(0)

# Solve the model
solver = pyo.SolverFactory('glpk')
results = solver.solve(model, tee=True)
results.write()
```

I Figur 5.2 ses også resultatet efter optimering. Det ses, at produktionen stiger i perioden fra uge 1 til uge 6 hvor lageret langsomt opbygges. I uge 7 er der således ingen produktion, og efterspørgslen imødekommes direkte og udelukkende fra lageret. Herefter stiger produktionen igen op til et niveau, der




**Figur 5.2: Produktion, efterspørgsel og lagerniveau plottet over tid. Målt i 100.000 liter.**

svarer til efterspørgslen i de pågældende uger. Den optimale objektfunktionsværdi er 4.400.725,32.

---

**Opgave 5.1:** Argumenter for, at $M_t = \sum_{t=1}^{T} d_t$ er en valid værdi for $M_t$ i ULS modellen.

**Opgave 5.2:** I denne opgave skal et godt valg af værdi til $M_t$ parametrene i ULS modellen udledes. Som hjælp kan følgende trin følges:

1. Angiv en mindste værdi for $M_T$ i ULS modellen. Det vil sige; udled en mindste øvre grænse for produktionen i den sidste periode i planlægningshorisonten.
2. Angiv en mindste værdi for $M_{T-1}$. Altså, angiv en mindste øvre grænse for produktionen i den andensidste periode.
3. Generaliser dine værdier for $M_T$ og $M_{T-1}$ til et generelt udtryk for $M_t$.

**Opgave 5.3:** I mange tilfælde er det fornuftigt at have et minimumsniveau for lagerbeholdningen. Ligeledes er det ikke urimeligt at antage, at lageret har en maksimal kapacitet. Antag, at minimumsniveauet og maksimumkapaciteten i periode $t$ er givet ved henholdsvis $\underline{s}_t$ and $\bar{s}_t$. Opstil lineære begrænsninger, som sørger for, at lagerniveauerne holder sig inden for de nedre og øvre grænser.

**Opgave 5.4:** Ofte er det muligt at have en backlog af ordre, som ikke er leveret til tiden. I de fleste tilfælde er det dyrt for en produktionsvirksomhed ikke at levere til aftalt tid, hvorfor der er en (ikke uanseelig) omkostning forbundet med backlogs. Lad $b_t$ være omkostningen forbundet med hver enhed af efterspørgslen, som er i backlog i periode $t$ og lad $r_t$ være antallet af enheder som er i backlog i periode $t$. Udvid ULS modellen til at tage højde for backlogs. *Hint: Det eneste, der skal ændres, er begrænsningerne (5.1) samt objektfunktionen. Man skal være OBS på, at værdierne for $M_t$ fundet i **Opgave 5.2** ikke gælder når der er mulighed for backlogs.*

## 5.3 Master Production Scheduling udvidelse

En første naturlig udvidelse af ULS modellen er til at inkludere mere end et produkt. Dette kan gøres ved at introducere $K$ produkter, indiceret over $k = 1, \ldots, K$. Alle disse produkter antages at have hver sin efterspørgsel givet $d_t^k \geq 0$. Læg mærke til, at vi har udvidet parameteren $d_t$ til nu at have et ekstra indeks, som holder styr på hvilket produkts efterspørgsel, der er tale om.

For at kunne planlægge produktionen af disse $K$ produkter, skal vi naturligvis også udvide omkostningsparametrene og variablerne med dette nye indeks, hvorved vi opnår følgende:

$p_t^k$: Enhedsomkostninger ved produktion af produkt $k$ i periode $t$.

$q_t^k$: Fast omkostning ved produktion af produkt $k$ i periode $t$.

$h_t^k$: Enheds omkostning ved lagerførelse af produkt $k$ i periode $t$.

$x_t^k$: Antallet af enheder af produkt $k$ produceret i periode $t$.

$y_t^k$: Binær variabel som angiver om produkt $k$ produceres i periode $t$.

$s_t^k$: Antallet af enheder af produkt $k$ på lager ultimo periode $t$.

$M_t^k$: Øvre grænse for produktionen af produkt $k$ i periode $t$

Med disse udvidelser af parametre og variabler opnås en *multi-item* ULS model givet ved

$$\begin{aligned}
\min \quad & \sum_{k=1}^{K} \sum_{t=1}^{T} \left( p_t^k x_t^k + q_t^k y_t^k + h_t^k s_t^k \right) \\
\text{s.t.:} \quad & s_{t-1}^k + x_t^k = d_t^k + s_t^k, && \forall t = 1, \ldots, T, \ k = 1, \ldots, K \\
& x_t^k \leq M_t^k y_t^k, && \forall t = 1, \ldots, T \ k = 1, \ldots, K \\
& s_0^k = s_T^k = 0, && \forall k = 1, \ldots, K \\
& x_t^k, s_t^k \in \mathbb{N}_0, y_t^k \in \{0, 1\}, && \forall t = 1, \ldots, T \ k = 1, \ldots, K
\end{aligned}$$

Med den udførlige beskrivelse af modellens elementer i forrige afsnit, er det op til læseren at fortolke denne udvidede model

Man bør notere sig, at sådan en multi-item ULS model faktisk er meget lidt interessant. Dette skyldes, at sådan som modellen er opskrevet ovenfor, optager produktionen af produkterne ingen *fælles ressourcer*. Dermed kan vi se, at multi-item ULS modellen *dekomponerer* i $K$ af hinanden uafhængige delproblemer, som hver blot er et ULS problem - altså er den optimale løsningsværdi til multi-item ULS problemet givet ved

$$Z^* = \sum_{k=1}^{K} Z_k^*$$

hvor

$$\begin{aligned}
Z_k^* = \min \quad & \sum_{t=1}^{T} \left( p_t^k x_t^k + q_t^k y_t^k + h_t^k s_t^k \right) \\
\text{s.t.:} \quad & s_{t-1}^k + x_t^k = d_t^k + s_t^k, && \forall t = 1, \ldots, T, k = 1, \ldots, K \\
& x_t^k \leq M_t^k y_t^k, && \forall t = 1, \ldots, T, k = 1, \ldots, K \\
& s_0^k = s_T^k = 0, \\
& x_t^k, s_t^k \in \mathbb{N}_0, y_t^k \in \{0, 1\}, && \forall t = 1, \ldots, T, k = 1, \ldots, k
\end{aligned}$$

Derfor vil vi i næste afsnit udvide multi-item ULS modellen til at tage højde for, at produktionen af forskellige produkter optager delte fællesressourcer.

### 5.3.1 Delte fællesressourcer

For at håndtere delte fællesressourcer, skal en mængde af sådanne ressourcer introduceres. Lad der derfor være givet $I$ ressourcer, som skal deles mellem produkterne. I princippet opstiller vi de følgende begrænsninger således, at alle produkter deles om hver ressource, men vi skal også se, at vi let tager højde for, at kun nogle produkter bruger af en specifik ressource.

For hver af de $I$ ressourcer og $T$ tidsperioder introducerer vi en parameter $L_t^i$ som angiver kapaciteten af ressource $i$ i periode $t$. Ligeledes benytter vi parametrene $\alpha^{ik}$, $\beta^{ik}$ og $\gamma^{ik}$, som henholdsvis angiver ressourceforbruget af ressource $i$ ved produktion af produkt $k$, ved opstart af en produktion af produkt $k$ samt ved lagerførelse af produkt $k$. Vi får dermed begrænsningerne

$$\sum_{k=1}^{K} \left( \alpha^{ik} x_t^k + \beta^{ik} y_t^k + \gamma^{ik} s_t^k \right) \leq L_t^i, \quad \forall t = 1, \ldots, T, \ i = 1, \ldots, I \tag{5.2}$$

Det kan måske virke unaturligt, at både produktionsvariabler, opstartsvariabler og lager-variabler forbruger af den samme ressource - hvilket det også er. Men hvis man for eksempel ønsker at modellere, at lagerets kapacitet er en ressource, som de lagerførte produkter deler, så kan det gøres ved at sætte $\alpha^{ik} = \beta^{ik} = 0$ for denne ressource og $\gamma^{ik} = 1$. Samtidig sættes $L_t^i$ lig med lagerets kapacitet, lad os sige $L_t^{lager}$, i periode $t$, hvorved man opnår lagerbegrænsningen

$$\sum_{k=1}^{K} s_t^k \leq L_t^{lager}, \quad \forall t = 1, \ldots, T$$

I det følgende eksempel skal vi se på, hvorledes det kan ændre en produktionsplan at dis-aggregere produktionen af en produktgruppe, ud på de enkelte produkter:

*Eksempel* 5.2 (Forsættelse af Eksempel 5.1). Tjase sælger naturligvis ikke blot "mælk", men i stedet en række forskellige versioner af mælk. I dette eksempel splittes produktet mælk i to underkategorier, nemlig letmælk og skummetmælk. Den totale mængde, der skal produceres i hver af de 10 uger er den samme som i Eksempel 5.1, men denne mængde er delt ud på de to produkter. Prisen for at producere en liter letmælk og en liter skummetmælk er ikke den samme, men gennemsnitsproduktionsprisen er lig prisen fra Eksempel 5.1. Det samme gør sig gældende for opstartsprisen. Lageromkostningerne forbliver uændret i forhold til det tidligere eksempel. Alt dette er opsummeret i følgende tabel.

| | Uge 1 | Uge 2 | Uge 3 | Uge 4 | Uge 5 | Uge 6 | Uge 7 | Uge 8 | Uge 9 | Uge 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Eftersp. total | 100.000 | 100.301 | 100.194 | 100.597 | 100.457 | 100.154 | 99.699 | 99.538 | 99.553 | 99.704 |
| Eftersp. letmælk | 51.989 | 58.678 | 69.351 | 50.036 | 38.533 | 67.617 | 62.777 | 38.909 | 41.393 | 31.850 |
| Eftersp. sk.mælk | 48.011 | 41.623 | 30.843 | 50.561 | 61.924 | 32.537 | 36.922 | 60.629 | 58.160 | 67.854 |
| Prod.pris letmælk kr/L | 2,33 | 4,12 | 4,68 | 4,46 | 3,78 | 3,68 | 5,75 | 5,23 | 2,60 | 3,27 |
| Prod.pris sk.mælk kr/L | 5,67 | 3,92 | 3,18 | 3,51 | 4,13 | 4,19 | 2,05 | 2,49 | 4,93 | 4,39 |
| Opstartspris letmælk i kr | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 | 45.000 |
| Opstartspris sk.mælk i kr | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 | 55.000 |
| Lageromkostning kr/L | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,6 | 0,6 | 0,6 | 0,6 |

Vi vil som før antage, at der maksimalt kan produceres 125.000 liter mælk om ugen, og at lagerniveauet ikke kan overstige 100.000 liter mælk. Det vil sige, at den *samlede* produktion og lagerets *samlede* kapacitet altså er delte ressourcer mellem de to typer af produkter, der produceres. Det efterlades til læseren at implementere en løsning til ovenstående eksempel (se **Opgave 5.5**).

Selvom vi før brugte gennemsnitsprisen for de to produkter, og selvom vi stadig skal producere den samme totale mængde og har den samme totale efterspørgsel i hver uge, ændres løsningen radikalt, når der skelnes mellem de to produkter (sammenlign med Figur 5.2).

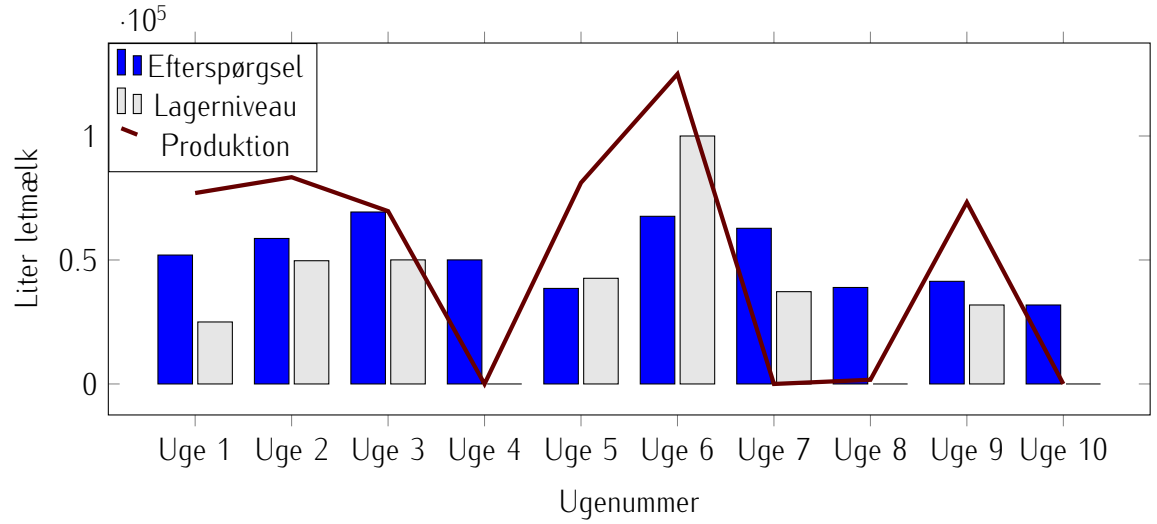
---

**Opgave 5.5:** Implementer en Python model for problemstillingen beskrevet i Eksempel 5.2.

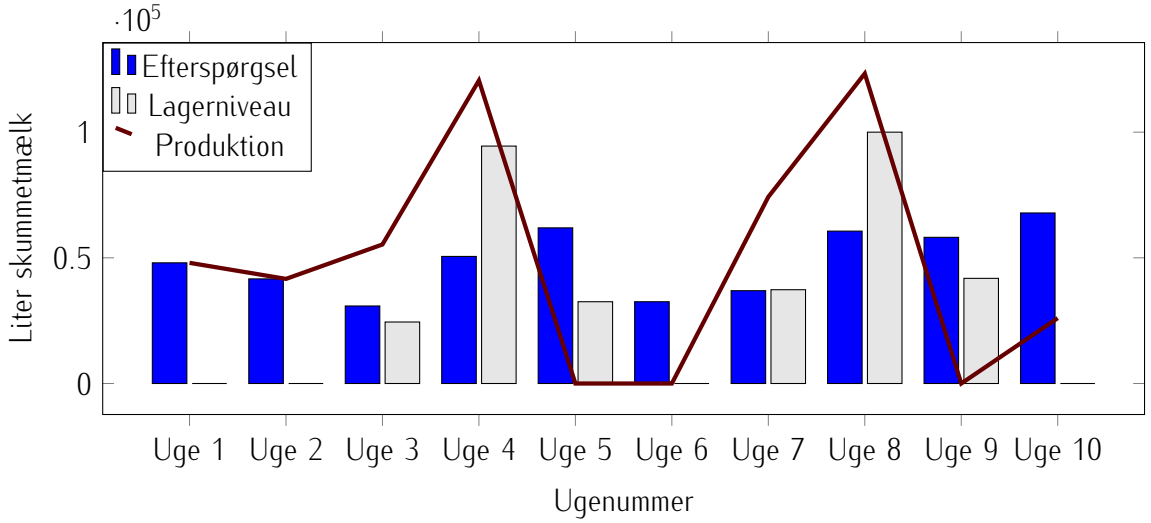
**Opgave 5.6:** Antag, at $\tau^k$ angiver tiden, det tager at producere hver enhed af produkt $k$ og antag videre, at det tager $\psi_k$ minutter at opstarte produktionen af et parti af produkt $k$. Antag yderligere, at der er $H_t$ timer til rådighed til produktion i periode $t$. Opstil, med udgangspunkt i begrænsningerne (5.2), en mængde af begrænsninger, som sørger for, at der ikke bruges flere timer, end der er til rådighed i hver periode.

**Opgave 5.7:** I mange praktiske problemstillinger giver det ikke mening at starte produktionen af et parti af et produkt med mindre partiet har en vis størrelse. Antag, at hvis et parti af produkt $k$ skal produceres i periode $t$, så skal der mindst produceres $m_t^k$ enheder af gangen. Opstil lineære begrænsninger, som sørger for, at partierne ikke tager mindre størrelse end $m_t^k$ for hvert produkt og i hver periode.

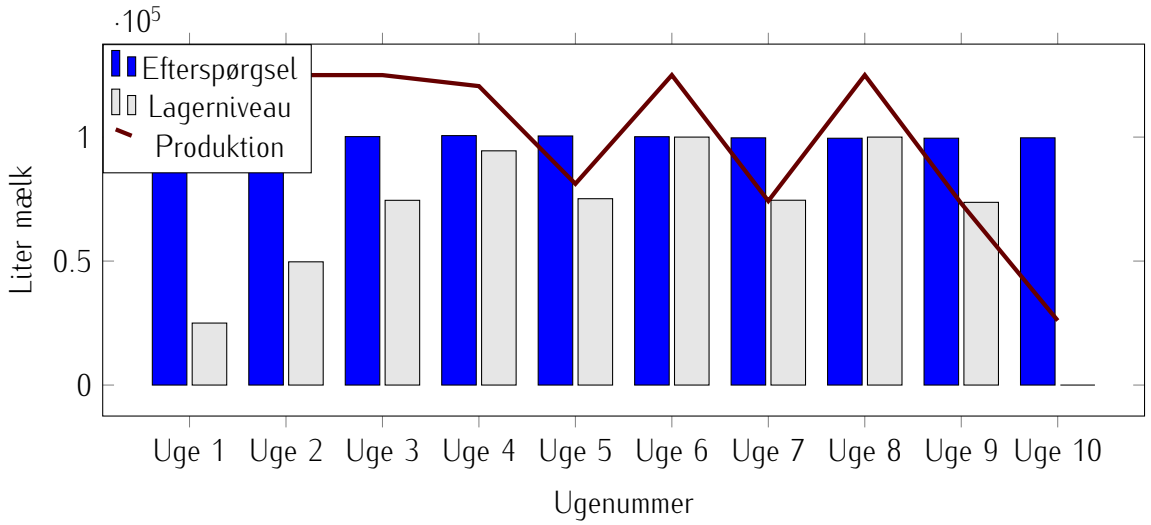
**Opgave 5.8:** I forlængelse af forrige opgave, skal vi nu se på, hvordan vi kan udvide modellerne således, at man ikke kan producere et hvilket som helst antal




(a) Overblik over produktionen af letmælk over de 10 uger.




(b) Overblik over produktionen af skummetmælk over de 10 uger.




(c) Overblik over den totale produktionen af letmælk og skummetmælk over de 10 uger.

**Figur 5.3: Visuel fremstilling af produktionsplanerne for letmælk, skummetmælk og den samlede produktion i en optimal løsning til Eksempel 5.2.**

af produkt $k$, men i stedet kan man producere hele multiplum af en given partistørrelse. Dette sker for eksempel ofte, hvis en maskine producerer 100 styk af et produkt ad gangen. Hvis det er tilfældet, kan vi ikke producere 1, 2, eller 77 enheder, men derimod 0, 100, 200, 300 og så videre. Antag, at partistørrelsen for produkt $k$ er $\psi_k$ og ændr så objektfunktionen og (5.1) begrænsningerne således, at disse tager højde for partistørrelserne. *Hint: redefiner $x_t^k$ variablerne til at angive hvor mange gange, der skal produceres $\psi_k$ enheder af produkt $k$ i periode $t$.*

### 5.3.2 MPS med valg af maskiner

Det ses ofte, at en produktionsvirksomhed har mere end én maskine, som kan producere et produkt, og endda, at en maskine kan indstilles til at producere mere end en type af et produkt. Vi vil i dette afsnit udvide MPS modellen til at tage højde for, at man i planen kan vælge mellem flere maskiner til produktionen af hvert produkt, og at maskinerne potentielt kan sættes op til at producere mere end en produktkategori.

Vi vil her introducere $J$ maskiner, hvorpå de $K$ produkter kan produceres. Vi vil til at starte med antage, at alle produkter kan produceres på alle maskiner, og så argumentere for siden hen, hvorledes denne antagelse kan lempes, så maskinerne hver kan håndtere en delmængde af produkterne. Vi vil introducere en parameter $\tau_t^{kj}$, som angiver ressourceforbruget ved at producere én enhed af produkt $k$ i periode $t$ på maskine $j$, og vi vil introducere parameteren $\rho_t^{kj}$, som angiver ressourceforbruget ved opstart af en produktion af produkt $k$ på maskine $j$ i periode $t$. Endvidere vil vi lade $C_t^j$ være maskine $j$'s ressourcekapacitet i periode $t$. Vi kan, for intuitionens skyld, tænke på $\tau_t^{kj}$ som den tid det tager at producere én enhed af produkt $k$ på maskine $j$ i periode $t$, $\rho_t^{kj}$ som tiden det tager at sætte maskine $j$ op til at producere produkt $k$ i periode $t$, og slutteligt kan vi tænke på $C_j^t$, som tiden der er til rådighed på maskine $j$ i periode $t$. Udover disse nye parametre, udvides de eksisterende variabler og parametre med endnu et indeks, så de er defineret som følger

$p_t^{kj}$: Enhedsomkostninger ved produktion af produkt $k$ på maskine $j$ i periode $t$.

$q_t^{kj}$: Fast omkostning ved produktion af produkt $k$ på maskine $j$ i periode $t$.

$h_t^k$: Enhedsomkostning ved lagerførelse af produkt $k$ i periode $t$.

$x_t^{kj}$: Antallet af enheder af produkt $k$, der produceres på maskine $j$ i periode $t$

$y_t^{kj}$: Binær variabler, som angiver om produkt $k$ produceres på maskine $j$ i periode $t$

$s_t^k$: Antallet af enheder af produkt $k$, som er på lager ultimo periode $t$.

$M_t^{kj}$: Øvre grænse for produktionen af produkt $k$ på maskine $j$ i periode $t$

På den måde kan vi opnå følgende MPS formulering

$$\min \sum_{j=1}^{J}\sum_{k=1}^{K}\sum_{t=1}^{T}\left(p_t^{kj}x_t^{kj}+q_t^{kj}y_t^{kj}\right)+\sum_{k=1}^{K}\sum_{t=1}^{T}h_t^k s_t^k$$

$$\text{s.t.: } s_{t-1}^k+\sum_{j=1}^{J}x_t^{kj}=d_t^k+s_t^k,\quad \forall t=1,\ldots,T,k=1,\ldots,K$$

$$x_t^{kj}\le M_t^{kj}y_t^{kj},\quad \forall t=1,\ldots,T,k=1,\ldots,K,j=1,\ldots,J$$

$$\sum_{k=1}^{K}\tau_t^{kj}x_t^{kj}+\sum_{k=1}^{K}\rho_t^{kj}y_t^{kj}\le C_t^j,\quad \forall t=1,\ldots,T,j=1,\ldots,J$$

$$\sum_{j=1}^{J}\sum_{k=1}^{K}\left(\alpha^{ik}x_t^{kj}+\beta^{ik}y_t^{kj}\right)+\sum_{k=1}^{K}\gamma^{ik}s_t^k\le L_t^i,\quad \forall t=1,\ldots,T,\ i=1,\ldots,I$$

$$s_0^k=s_T^k=0,\quad \forall k=1,\ldots,K$$

$$x_t^{kj},s_t^k\in\mathbb{N}_0,y_t^{kj}\in\{0,1\},\quad \forall t=1,\ldots,T,k=1,\ldots,K,j=1,\ldots,J$$

Objektfunktionen er her udvidet med en sum over maskinerne for produktionsdelen af omkostningerne. Flow-begrænsningerne har fået en sum over maskinerne over alle $x_t^{kj}$ variablerne, og skal nu læses som følger (fra venstre mod højre): Lagerniveauet primo periode $t$ ($s_{t-1}^k$) af produkt $k$ plus den samlede produktion fra alle maskinerne af produkt $k$ i periode $t$ ($\sum_{j=1}^{J}x_t^{kj}$), skal være lig med efterspørgslen for produkt $k$ i periode $t$ ($d_t^k$) plus det, som føres på lager i periode $t$ af produkt $k$ ($s_t^k$).

Indikator-begrænsningerne, $x_t^{kj}\le M_t^{kj}y_t^{kj}$, sørger igen for, at hvis der produceres produkt $k$ på maskine $j$ i periode $t$ ($x_t^{kj}>0$), så antager indikatorvariablerne $y_t^{kj}$ værdien 1.

De nye begrænsninger, $\sum_{k=1}^{k} \tau_t^{kj} x_t^{kj} + \sum_{k=1}^{K} \rho_t^{kj} y_t^{kj} \leq C_t^j$, angiver, at den samlede "tid" der bruges på maskine $j$ i periode $t$ til produktion ($\sum_{k=1}^{k} \tau_t^{kj} x_t^{kj}$) plus den "tid", der bruges til opstart ($\sum_{k=1}^{K} \rho_t^{kj} y_t^{kj}$), ikke må overstige den "tid", som er til rådighed ($C_t^k$). Læg her mærke til, at *tid* er sat i citationstegn. Dette skyldes, at der kunne være tale om en hvilken som helst ressource, som knytter sig til en maskine. Det kan dog være nærliggende at tænke på disse ressourcer som tid.

Slutteligt er begrænsningerne relateret til de delte ressourcer, givet ved

$$\sum_{j=1}^{J} \sum_{k=1}^{K} \left( \alpha^{ik} x_t^{kj} + \beta^{ik} y_t^{kj} \right) + \sum_{k=1}^{K} \gamma^{ik} s_t^k \leq L_t^i$$

blevet udvidet, så den del af begrænsningerne, som relaterer sig til produktionen, nu sørger for, at den delte ressource $i$ i periode $t$ forbruges på alle maskinerne. Man kan her tænke på den delte ressource som en komponent, der skal bruges til produktion af produkterne. Uanset om produkterne laves på den ene eller den anden maskine, skal denne komponent (delt ressource) forbruges.

Denne model kan også bruges, hvis produktionsvirksomheden har separate produktionsfaciliteter, da vi med en løsning til modellen ved præcis hvilke maskiner, der skal bruges til at producere hvilke produkter i hvilke perioder. Dermed haves en produktionsplan, som koordinerer på tværs af produktionsfaciliteterne. Man skal dog være OBS på, at der i lager-variabler ikke er nogen indikation af *hvor* produktionen har fundet sted, hvorfor dette skal overvejes, såfremt det er væsentligt for den pågældende case, man arbejder med.

Det blev indledningsvist i dette afsnit antaget, at alle produkter kunne laves på alle maskiner. Dette er ofte ikke muligt i virkeligheden, hvis der for eksempel er stor forskel på de $K$ produkter der produceres. I sådanne tilfælde vil man, for hvert produkt $k$, naturligvis have en liste af maskiner, som kan producere det pågældende produkt. Lader man nu $J(k) \subseteq \{1, \ldots, J\}$ være en delmængde af alle maskinerne, som kan varetage produktionen af produkt $k$, kan man blot tilføre følgende begrænsninger

$$y_t^{kj} = 0, \qquad \forall k = 1, \ldots, K, \ j \notin J(k), t = 1, \ldots, T$$

til modellen, hvorved produktionen af produkt $k$ *ikke* kan startes på maskinerne, som ikke er i mængden $J(k)$. Dermed kan produktionen af produkt $k$ kun startes på de maskiner, som rent faktisk kan håndtere dette produkt.

## 5.4 Material Requirement Planning udvidelse

I kurser omkring Operations Research og Operations Management beskæftiger man sig ofte med den del af produktionsplanlægningen som kaldes Material Requirement Planning, eller blot MRP. Formålet med MRP er (mindst) trefold:

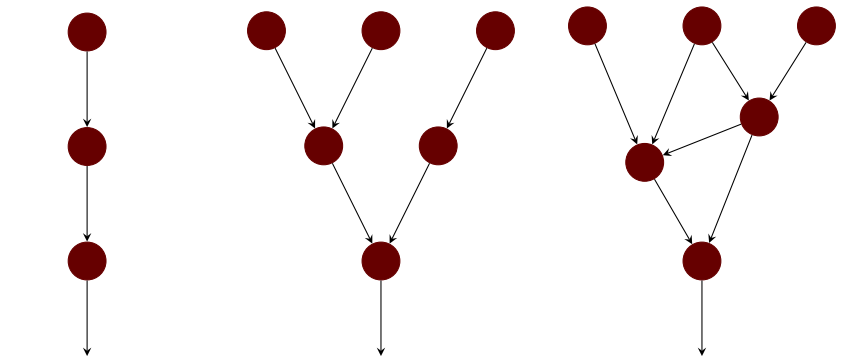
1. MRP skal sørge for, at råmaterialer er til rådighed for produktionen, og for at produkter er klar til leverance til kunderne på de aftalte tidspunkter.
2. MRP skal hjælpe til at have så lave lager-niveauer som muligt af både færdige produkter og materialer, som bruges i produktionen.
3. MRP skal hjælpe med at planlægge produktionen, leveringsplanerne og indkøbsaktiviteterne.

Når man beskæftiger sig med MRP, bliver man nødt til at skelne mellem to slags efterspørgsler: den uafhængige og den afhængige efterspørgsel. Den uafhængige efterspørgsel, er den vi indtil nu har betegnet $d_t$ eller $d_t^k$, altså efterspørgslen på det færdige produkt, som vi ikke selv kan styre, da denne er eksogent givet. Den afhængige efterspørgsel er derimod efterspørgslen, som genereres internt gennem produktionen af de færdige produkter. Denne efterspørgsel kan altså delvist kontrolleres ved at skrue op og ned for produktionen. Den interne afhængighed mellem produkterne bliver ofte beskrevet ved hjælp af en såkaldt *bill of materials* (BOM) og produktstrukturen bliver oftest klassificeret som enten *serie*- (series), *samlings*- (assembly) eller *general* (general) struktur og kan illustreres som diagrammerne vist i Figur 5.4.

Vi vil i det følgende, for overskuelighedens skyld, undlade situationen hvor produkter kan produceres på flere forskellige maskiner. Det vil sige, at vi i dette afsnit dropper $j$-indekset på produktions- og opstartsvariablerne.

For hvert produkt $k$ introduceres en mængde $D(k)$, som angiver de direkte efterfølgende produkter i produktstrukturen - det vil sige, at $D(k)$ indeholder alle de produkter, som forbruger af produkt $k$ når de produceres. Man bør notere sig, at for seriestrukturer og for samlingsstrukturer, består $D(k)$ kun af et enkelt produkt og for færdige produkter/endelige produkter haves, at $D(k) = \emptyset$ - det vil sige, at ingen andre produkter forbruger af de endelige produkter.

For hvert produkt $l \in D(k)$ introduceres en parameter $r^{lk}$, som angiver, hvor meget af produkt $k$ en enhed af produkt $l$ skal bruge for at blive produceret. Det



(a) Seriestruktur (b) Samlingsstruktur (c) General struktur

**Figur 5.4: Typer af produktstrukturer i MRP modeller. Hver knude i graferne repræsenterer et produkt, og en pil indikerer at den knude som pile slutter i, bruger noget af det produkt, som er repræsenteret ved knuden hvor pilen starter.**

vil sige, at ved hjælp af disse $r$-parametre, samt produktionsvariablernes værdi ($x_t^k$), kan den afhængige efterspørgsel af et produkt bestemmes. Den uafhængige efterspørgsel er stadig givet ved $d_t^k$.

For at kunne tage højde for, at nogle af de komponenter/produkter, der skal bruges i produktionen kommer fra eksterne leverandører, vil vi også nu inkludere *lead time* på produkterne. Denne lead time for produkt $k$ (som kan være leveret udefra eller blot er produceret internt med en produktionstid større end én periode) er givet ved $\psi^k \in \mathbb{N}_0$ for alle produkter $k$. På den måde vil vi lade $x_t^k$ angive størrelsen af et produktionsparti eller en ordre af produkt $k$, som er afgivet i periode $t$ med levering i perdiode $t + \psi^k$.

For at formulere en model, som kan håndtere en general produktstruktur (og dermed også serie- og samlingsstruktur), tages udgangspunkt i MPS modellen

med delte ressource (men altså som nævnt uden forskellige maskiner) givet ved

$$\min \sum_{k=1}^{K}\sum_{t=1}^{T}\left(p_t^k x_t^k + q_t^k y_t^k + h_t^k s_t^k\right)$$

s.t.: korrekt flow a varer

$$x_t^k \leq M_t^k y_t^k, \qquad \forall t = 1,\ldots,T \;\; k = 1,\ldots,K$$

$$\sum_{k=1}^{K}\left(\alpha^{ik} x_t^k + \beta^{ik} y_t^k + \gamma^{ik} s_t^k\right) \leq L_t^i, \qquad \forall t = 1,\ldots,T, \;\; i = 1,\ldots,I$$

$$s_0^k = s_T^k = 0, \qquad \forall k = 1,\ldots,K$$

$$x_t^k, s_t^k \in \mathbb{N}_0, y_t^k \in \{0,1\}, \qquad \forall t = 1,\ldots,T \;\; k = 1,\ldots,K$$

Den opmærksomme læser vil notere sig, at den første mængde af begrænsninger er relativt vagt beskrevet. Derfor vil det nu blive gennemgået, hvorledes de såkaldt flow-begrænsninger skal ændres for at tage højde for lead times. Først tydeliggøres den ønskede sammenhæng i ord:

det, som er til rådighed af produkt $k$ til tid $t$
$=$
det, som efterspørges af produkt $k$ til tid $t$
$+$
det, som føres på lager af produkt $k$ til tid $t$

Først opskrives højresiden i denne relation matematisk ved at notere, at hvad der er til rådighed af produkt $k$ til tid $t$, er det som ligger på lager ($s_{t-1}^k$) når periode $t$ starter plus den mængde, som er færdigt produceret/ankommer til tid $t$. Denne sidste del er netop det som blev bestilt/produceret til tid $t - \psi^k$ ($x_{t-\psi^k}^k$), idet produkt $k$ har en lead time på $\psi^k$ perioder. Dermed er mængden af produkt $k$, som er til rådighed i periode $t$, givet ved $s_{t-1}^k + x_{t-\psi^k}^k$.

Som det næste skal venstresiden i flow-begrænsningerne modelleres. Her tages udgangspunkt i mængden af produkt $k$, som føres på lager i periode $t$: $s_t^k$. Herefter tages den uafhængige del af efterspørgslen, som er givet ved $d_t^k$. Slutteligt bestemmes den afhængige del af efterspørgslen på produkt $k$ i periode $t$, som kommer fra andre produkter, der benytter produkt $k$. Husk, at $D(k)$ netop består af de produkter, som bruger af produkt $k$, og at $r^{lk}$ angiver, hvor mange enheder af produkt $k$ én enhed af produkt $l$ skal bruge for at blive produceret.

Dermed vides det, at for alle produkter $l \in D(k)$ efterspørges der $r^{lk}x_t^l$ enheder af produkt $k$ i periode $t$. Summes over alle produkter, som benytter produkt $k$ i produktionen, opnås den samlede afhængige efterspørgsel

$$\sum_{l \in D(k)} r^{lk} x_t^l.$$

Sammenholdes alt dette, fås flow-begrænsningerne givet ved

$$\underbrace{s_{t-1}^k}_{\text{primolager}} + \underbrace{x_{t-\psi^k}^k}_{\text{til rådighed}} = \underbrace{\left[ d_t^k + \sum_{l \in D(k)} r^{lk} x_t^l \right]}_{\text{efterspørgsel}} + \underbrace{s_t^k}_{\text{ultimolager}}, \qquad \forall t = 1, \ldots, T, k = 1, \ldots, K$$

I denne MRP model, hvor lead times er blevet inddraget, skal man være opmærksom på, at lead time for produkt $k$ er lig med $\psi^k$, og dermed er uafhængig af hvilken periode, der produceres/bestilles i. Dette er nok desværre ikke helt realistisk, da nogle produkter har længere lead time i nogle perioder og kortere i andre. Dette er desværre ikke helt trivielt at inkorporere, da man i tilfældet hvor lead time er aftagende hen over perioden, skal holde styr på, at der i én periode, kan ankomme produkter, som er bestilt fra *flere* tidligere perioder. Hvis man for eksempel befinder sig i periode 4, og det givne produkt havde lead time 2 i periode 2 og lead time 1 i periode 3, kan man i periode 4 modtage produkter bestilt/produceret i både periode 2 og 3. Dette tilføjer et ekstra lag af kompleksitet, som ikke behandles i videre i denne bog.

### 5.4.1 Indkøbs- og produktions-lead time

I udviklingen af MRP modellen ovenfor, indførtes en lead time på produkterne. Denne lead time kunne enten knytte sig til produktionstiden for et produkt, eller til leveringstiden af et produkt. Således ville en ordre på/produktion af $x_t^k$ enheder af produkt $k$ i periode $t$ ankomme/være færdigproduceret i periode $t + \psi^k$. Dette er en yderst relevant og realistisk tilføjelse, da produktion og indkøb ofte ikke kan ske øjeblikkeligt. Det vil sige, for at skabe realistiske produktionsplaner, bør dette element indgå for alle komponenter i BOM'en. Den lead time vi benytter i denne model repræsenterer den totale tid, der skal til for at færdiggøre en ordre (indkøb eller produktion) inklusiv forberedelse, administration, ventetid, produktion, kvalitetskontrol, test og levering, og bliver målt i hele tidsperioder.

Lead times i denne model er som nævnt konstante over tid, uafhængige af ordrestørrelser, og så er de *input* til modellen. I optimeringsmodellen angiver $\psi^k$ en

*minimums* lead time for at producere/indkøbe et parti af produkt $k$. Med minimum lead time menes, at der ikke er inkluderet nogle "kø tider", hvor partier venter på, at en maskine eller en ressource bliver ledig. Dette er garanteret på grund af den eksplicitte kapacitetsbegrænsning, som er inkluderet på delte ressourcer (eller maskiner, hvis man tager denne begrænsning med fra MPS modellen), og som en brugbar løsning naturligvis skal overholde. Dermed er der en garanti for, at der er nok kapacitet til rådighed til at producere hvert planlagte parti uden forsinkelser, hvormed det ikke er nødvendigt med *buffer tider* tillagt lead time.

Naturligvis skal man i en praktisk anvendelse være meget opmærksom på, om lead times rent faktisk er uafhængige af ordrestørrelserne. Hvis lead times rent faktisk bliver influeret af ordrestørrelser (altså værdien af $x_t^k$), skal modellen tilpasses således, at man tager højde for dette, da det vil influere på, hvad der i sidste ende vil være optimale produktionskvantiteter. Ligeledes må man spørge sig selv, om lead times i virkeligheden er input til modellen, eller om det er noget vi har kontrol over ved for eksempel at kunne forhandle kortere/længere lead times, ved at betale en højere/lavere pris for nogle af komponenterne. Såfremt dette er tilfældet, bliver lead time pludselig en beslutningsvariabel, som skal indgå i modellen på lige fod med $x$, $y$, og $s$-variablerne.

Desuden er det utroligt vigtigt at estimere sine lead times korrekt, da man riskerer en *dominoeffekt* af forsinkelser igennem en produktionsplan, hvis man har haft for optimistiske forventninger til produkternes lead times. Det er derfor ualmindeligt praktisk (og nødvendigt), at man grundigt validerer sine løsninger gennem for eksempel simulationsstudier, eller blot ved at undersøge hvor stor risikoen er for, at planen bliver ubrugbar, hvis nogle af produkternes lead times forlænges i nogle af planlægningshorisontens perioder.

# 6 Skemaplanlægning

## 6.1 Hvad er skemaplanlægning?

I mange af de problemstillinger man møder som præskriptiv analytiker kræves det, at man på den ene eller anden måde laver en *sekvens* af beslutninger for at løse det forestående problem. *Skemaplanlægning* (*scheduling*, på engelsk) er nærmest allestedsnærværende og optræder inden for planlægning i uddannelsessektoren, inden for transportplanlægning, underholdning og så videre. Selv hvis man gør sig umage, kan det være svært at finde et område, hvor skemalægning i den ene eller anden form *ikke* optræder. Men på trods af, at skemaplanlægning er (eller burde) en så central del af mange problemstillinger, kan det i den virkelige verden være overordentligt svært at lave gennemførlige og attraktive løsninger, selv for erfarne skemalæggere.

Skemaplanlægningsproblemer opstår i alle typer af systemer da det stort set altid er nødvendigt at organisere og fordele arbejdsopgaver mellem enheder. Disse enheder kan både være maskiner, computere og mennesker. For at gøre det klart hvad vi vil betragte som skemaplanlægning bruger vi følgende arbejdsdefinition:

> Skemaplanlægning omhandler allokeringen af begrænsede ressourcer til opgaver over tid. Det er en beslutningsprocess, som har som mål at optimere en eller flere objektfunktioner.

Med denne definition for øje kan vi altså, i en bred forstand, sige, at en skemaplan vil indeholde al nødvendig information for at udføre en process på optimal vis (i forhold til den valgte model, og det data, som er til rådighed). Som eksempler på et skemaplanlægningsproblem, kan man betragte en produktionsplan som består i at allokere ressourcer (materialer, arbejdskraft, udstyr osv.) til opgaver over tid givet nogle fysiske og logiske begrænsninger. Et skemaplanlægningsproblem kunne også være udformningen af en vagtplan som overholder ansættelsesvilkår, behov for personale og personalets præferencer.

Traditionelt set er skemaer som regel lavet i hånden af en til flere personer, som har stor viden omkring en virksomheds problemstillinger. Sådanne håndlavede skemaer er ofte relativt gode, fordi skemalæggerne netop har stor erfaring med at lave skemaerne og fordi skemaerne ofte ligner hinanden fra periode til periode: det vil sige, at meget kan genbruges og små ændringer kan indføres i eksisterende skemaer for at få skemaerne til at passe til nye og ændrede forhold. Selvom de resulterende skemaer ofte er gode på flere parametre, er det let at forestille sig, at en optimal løsning sjældent er fundet. Samtidig er det ofte ekstremt krævende at lægge sådanne skemaer: Vagtplanlægning for sygeplejersker kan ofte tage flere dage at lægge i hånden og ofte er sådanne skemaer lagt af en sygeplejerske som således ikke kan varetage sine kerneområder imens. Derfor er der ofte potentiale til både at forbedre skemaerne og til at fritage skemalæggerne for det arbejdstunge hverv såfremt man benytter optimering til at producere skemaer.

Litteraturen indenfor skemalægning nævner ofte følgende områder som nogle af kerneområder inden for skemalægningen (her med de engelske betegnelser for at det er nemmere at finde litteratur om emnerne)

- Job shop scheduling
- Multiprocessor scheduling
- Multitask scheduling
- Parallel Machine scheduling
- Group Job scheduling
- Course scheduling
- Resource constrained project scheduling
- Dynamic task scheduling

I det følgende vil vi give en række eksempler på skemalægnings-problemer hvor vi tager udgangs punkt i en række "jobs"/arbejdsopgaver, som skal varetages af en til flere maskiner. Vi vil starte med de simpleste modeller og så arbejde os frem til mere avancerede scheduling modeller. Fælles for de modeller der her præsenteres er, at de falde ind under det emne, som kaldes *job shop scheduling*.

Man kan med foredel studere dette kapitel efter at have læst kapitel 7 omkring ruteplanlægning. Såfremt man tænker et job som en kunde og en maskine

som et køretøj, vil man straks se, at modellerne for job shop schedulig og for ruteplanlægning er ækvivalente.

## 6.2 $n$ jobs på én maskine

Antag at vi har en maskine som skal udføre en operation på en række "jobs". Vi vil lade vores jobs være nummereret fra 1 til $n$. Vi skal nu finde en sekvens af de $n$ jobs som gør, at den samlede tid brugt på at processere de $n$ jobs minimeres. For hvert par af jobs $i$ og $j$ er der en omstillingstid sådan at det tager $t_{ij}$ minutter at konfigurere maskinen til at udføre job $j$ hvis det forrige job var job $i$. Endvidere tager det $\tau_i$ minutter at processere job $i$. Der vil også være en opstartstid på $b_i$ minutter hvis job $i$ er det første job og en nedlukningstid på $\beta_i$ minutter hvis job $i$ er det sidste job. Målet er nu at lave en sekvens af jobbene, som minimerer den samlede tid der bruges på at behandle de $n$ jobs.

Vi skal her bruge tre mængder af binære variabler defineret som følger:

$$x_{ij} = \begin{cases} 1, & \text{hvis job } i \text{ behandles lige før job } j \\ 0, & \text{ellers} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{hvis job } i \text{ er det første job der skal behandles} \\ 0, & \text{ellers} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{hvis job } i \text{ er det sidste job der skal behandles} \\ 0, & \text{ellers} \end{cases}$$

Vores objektfunktion skal nu bestå af en del som kommer fra opstarten af første job, tiden der går med at omstille maskinen og slutteligt tiden, som kommer fra at lukke maskinen ned igen. Læg mærke til, at selve den tid jobbene bruger på at blive behandlet er irrelevant, da dette er en fast tid uanset hvorledes vi danner sekvensen af jobs: hvert job skal behandles, så vi ved, at den samlede tid altid vil indeholde $\sum_{i=1}^{n} \tau_i$ minutter uanset hvorledes vi danner sekvensen. Derfor ender objektfunktionen med at blive

$$\min \sum_{i=1}^{n} b_i y_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} t_{ij} x_{ij} + \sum_{i=1}^{n} \beta_i z_i$$

Det næste vi skal have styr på er begrænsningerne. For hvert job $j$ ved vi, at der skal komme et andet job lige før med mindre job $j$ er første job. Det vil altså

sige at vi får begrænsningerne

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} + y_j = 1, \quad \forall j = 1, \ldots, n$$

Med denne begrænsning har vi, at hvis job $j$ *ikke* er det første job ($y_j = 0$) så har vi, at $\sum_{i=1}^{n} x_{ij} = 1$ hvilket præcis siger, at vi skal vælge ét job $i$ til at komme før job $j$. På samme måde kan vi også sørge for, at der kommer præcis ét job efter job $i$ med mindre job $i$ er det sidste job

$$\sum_{\substack{j=1 \\ i \neq j}}^{n} x_{ij} + z_i = 1, \quad \forall i = 1, \ldots, n$$

Endvidere skal vi sørge for, at præcis et job er det første og at præcis et job er det sidste:

$$\sum_{i=1}^{n} y_i = 1,$$
$$\sum_{i=1}^{n} z_i = 1.$$

Givet, at der er mere end ét job der der skal planlægges (ellers er det et trivielt problem), kan vi også sørge for, at et job ikke både er først og sidst, ved at indføre begrænsningerne

$$y_i + z_i \leq 1, \quad \forall i = 1, \ldots, n$$

som sørger for, at maksimalt en af de to muligheder "job $i$ er først" og "jobs $i$ er sidst" kan indfinde sig. Man kunne nu fristes til at tro, at modellen er færdig: hvert job er nu enten først, sidst, eller mellem to jobs. Men overvej nu følgende scenarie: antag at vi har 4 jobs som skal planlægges. Følgende "løsning" opfylder alle begrænsningerne

| | | | $x$ | | | |
|---|---|---|---|---|---|---|
| Job | $y$ | $z$ | 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 |

men er job åbenlyst ikke en løsning til vores problem. Problemet er, at jobbene så at sige ikke er sammensat i en kæde af på-hinanden-følgende jobs. Derfor skal vi indføre endnu en mængde variabler:

$$u_i = t, \text{ hvis jobs } i \text{ er det } t\text{'te job i planen.}$$

Det vil altså sige, at $u_i$ angiver på hvilken plads i rækken af jobs, job $i$ befinder sig på i en optimal løsning. Det første man kan lægge mærke til, er at vi har nogle åbenlyse øvre og nedre grænser for $u_i$:

$$1 \leq u_i \leq n, \quad \forall i = 1, \ldots, n \tag{6.1}$$

Det sidste der mangler for at vore model er valid er at sørge for, at $u_i$ rent faktisk får den rigtige værdi. Vi kan først lægge mærke til, at hvis $y_i = 1$ så er job $i$ det første job, og derfor må $u_i = 1$. Det kan vi modellere på følgende måde

$$u_i \leq 1 + (n-1)(1-y_i), \quad \forall i = 1, \ldots, n$$

For at forstå hvorfor denne begrænsning virker, deles situationen op i to scenarier:

$y_i = 1$: I dette tilfælde reduceres uligheden til $u_i \leq 1 + (n-1)(1-1) = 1 + (n-1)0 = 1$. Da vi fra (6.1) har at $u_i \geq 1$ og nu også har at $u_i \leq 1$ medfører dette, at $u_i = 1$. Det var præcis det vi ønskede.

$y_i = 0$: Hvis $y_i = 0$ reduceres uligheden til $u_i \leq 1 + (n-1)(1-0) = 1 + (n-1)1 = 1 + n - 1 = n$. Dette er altid sandt, da der kun er $n$ jobs, og $u_i$ vil derfor altid være mindre end eller lig med $n$.

På tilsvarende måde, får vi også at $u_i = n$ hvis $z_i = 1$ og at dette kan modelleres som følger

$$u_i \geq n - (n-1)(1-z_i), \quad \forall i = 1, \ldots, n.$$

Slutteligt skal vi sørge for, at hvis $x_{ij} = 1$ (altså job $i$ kommer lige før job $j$) så skal $u_i + 1 \leq u_j$. Dette kan modelleres som følger

$$u_i - u_j + n x_{ij} \leq n - 1, \quad \forall i, j = 1, \ldots, n : i \neq j$$

For at blive overbevist om rigtigheden af disse uligheder, deles situationen igen op i to:

$x_{ij} = 1$: I dette tilfælde reduceres ulighederne til $u_i - u_j + n \leq n - 1$ hvilket kan omskrives til $u_i - u_j \leq -1$, hvilket igen er ækvivalent med $u_i + 1 \leq u_j$. Altså, sagt med menneskeord: hvis job $i$ kommer lige før job $j$, så skal job $j$'s placering være mindst én større end job $i$'s.

$x_{ij} = 0$: I dette tilfælde fås $u_i - u_j \leq n - 1$ hvilket altid er rigtigt, thi det største venstreside ($u_i - u_j$) kan blive er når $u_i$ er så stor som muligt og $u_j$ er så lille som mulig. Den øvre grænse for $u_i$ er $n$ og den nedre grænse for $u_j$ er 1.

Slutteligt skal vi sørge for, at vores model ikke forsøger at skifte fra job $i$ til sig selv, da dette ikke giver mening. Dette gøres let ved en af følgende to måder:

1. Vi kan specificere, at $x_{ii} = 0$ for alle $i = 1, \ldots, n$.
2. Vi kan sætte $t_{ii} = \infty$ for alle $i = 1, \ldots, n$.

Vi vil i det følgende blot antage, at vi har sat $t_{ii} = \infty$ da det har samme effekt som at kræve at $x_{ii} = 0$. Det vil altså sige, at sekvenseringsproblemet for $n$ jobs

på én maskine kan opskrives som følger

$$\min \sum_{i=1}^{n} b_i y_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} t_{ij} x_{ij} + \sum_{i=1}^{n} \beta_i z_i \tag{6.2}$$

$$\text{s.t.:} \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} + y_j = 1, \qquad \forall j = 1, \ldots, n \tag{6.3}$$

$$\sum_{\substack{j=1 \\ i \neq j}}^{n} x_{ij} + z_i = 1, \qquad \forall i = 1, \ldots, n \tag{6.4}$$

$$\sum_{i=1}^{n} y_i = 1, \tag{6.5}$$

$$\sum_{i=1}^{n} z_i = 1. \tag{6.6}$$

$$y_i + z_i \leq 1, \qquad \forall i = 1, \ldots, n \tag{6.7}$$

$$1 \leq u_i \leq n, \qquad \forall i = 1, \ldots, n \tag{6.8}$$

$$u_i \leq 1 + (n-1)(1 - y_i), \qquad \forall i = 1, \ldots, n \tag{6.9}$$

$$u_i \geq n - (n-1)(1 - z_i), \qquad \forall i = 1, \ldots, n \tag{6.10}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \qquad \forall i, j = 1, \ldots, n : i \neq j \tag{6.11}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j = 1, \ldots, n \tag{6.12}$$

$$y_i, z_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, n \tag{6.13}$$

Denne model er i udgangspunkt ret stor og det kræver lidt arbejde at implementere den.

### 6.2.1 IP model med "dummy" job

Et modelleringstrick man ofte ser brugt inden for blandet heltalsprogrammering er at indføre såkaldte *dummy* objekter til modellen, som i deres natur er kunstige, men som kan hjælpe med at lave en mere strømlinet model. Det følgende illustrerer et sådant tilfælde!

Der indføres nu et *dummy job* som får nummer 0. Det vil altså sige, at der nu er $n + 1$ jobs: de rigtige $n$ jobs som er nummereret fra 1 til $n$ og så dummy jobbet som er job 0. Det antages nu, at tiden det tager at skifte *fra job 0* til et

hvert andet job $i$ er givet ved $t_{0i} = b_i$ og at tiden det tager at skifte *til job 0* fra ethvert andet job er givet ved $t_{i0} = \beta_i$. Man kan nu ved at kræve, at man skal starte og slutte med job 0 sørge for, at der bliver "betalt" for opstartstiden og nedlukningstiden fordi der skal skiftes fra og til job 0. På den måde vides det, at job 0 skal være både første og sidste job og man behøver derfor ikke variablerne $y_i$ og $z_i$. Derimod skal der nu findes en sekvens for jobbene 1 til $n$ mellem job 0 og job 0, så at sige. Man kan relativt let modificere IP modellen fra før, så den nu passer til den "nye" model med et dummy job:

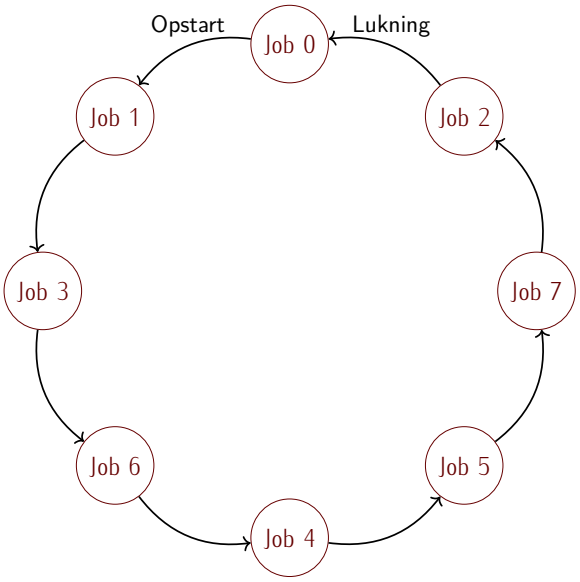
$$
\begin{aligned}
\min \quad & \sum_{i=0}^{n} \sum_{\substack{j=0 \\ i \neq j}}^{n} t_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{\substack{j=i \\ i \neq j}}^{n} x_{ij} = 1, && \forall j = 0, \ldots, n \\
& \sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} = 1, && \forall i = 0, \ldots, n \\
& 1 \leq u_i \leq n, && \forall i = 1, \ldots, n \\
& u_i - u_j + n x_{ij} \leq n - 1, && \forall i, j = 1, \ldots, n : i \neq j \\
& x_{ij} \in \{0, 1\}, && \forall i, j = 1, \ldots, n
\end{aligned}
$$

På denne måde opnås en model som er lidt lettere at overskue, da der ikke er så mange begrænsninger. Med denne model, kan en løsning til sekvenseringsproblemet repræsenteres som en *cirkel* af jobs, der starter og slutter i job 0 og som har en pil ind i og en pil ud af hvert job. For et eksempel med 7 jobs og én maskine kunne en løsning se ud som i Figur 6.1.

---

**Opgave 6.1:** Bevis, at $u_i \geq n - (n-1)(1 - z_i)$ sammen med $u_i \leq n$ sørger for, at hvis job $i$ er det sidste job, så er $u_i = n$.

**Opgave 6.2:** Betragt følgende data til et sekvenseringsproblem for 10 jobs på en maskine:




Figur 6.1: Illustration af en løsning når modellen benytter et dummy job.

| | | | Omstillingstid | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Opstartstid | Nedluk.tid | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 23 | 5 | – | 11 | 18 | 5 | 5 | 10 | 6 | 15 | 21 | 14 |
| 2 | 16 | 16 | 9 | – | 21 | 24 | 5 | 14 | 14 | 21 | 7 | 20 |
| 3 | 11 | 6 | 23 | 21 | – | 9 | 17 | 17 | 22 | 22 | 9 | 16 |
| 4 | 10 | 25 | 5 | 24 | 10 | – | 13 | 25 | 16 | 10 | 14 | 23 |
| 5 | 23 | 6 | 10 | 6 | 5 | 12 | – | 21 | 19 | 18 | 23 | 9 |
| 6 | 15 | 15 | 11 | 8 | 16 | 14 | 17 | – | 9 | 18 | 14 | 15 |
| 7 | 17 | 16 | 22 | 22 | 19 | 14 | 16 | 24 | – | 25 | 18 | 15 |
| 8 | 24 | 25 | 23 | 11 | 24 | 24 | 8 | 9 | 19 | – | 19 | 19 |
| 9 | 12 | 24 | 5 | 11 | 21 | 24 | 15 | 14 | 13 | 9 | – | 9 |
| 10 | 21 | 12 | 22 | 8 | 8 | 21 | 18 | 5 | 12 | 5 | 10 | – |

Find en optimal sekvensering af de 10 jobs ved hjælp af modellen (6.2)–(6.13).

**Opgave 6.3:** Implementer IP modellem med dummy job for problemet i **Opgave 6.2**.

## 6.3 *n* jobs på *m* identiske maskiner

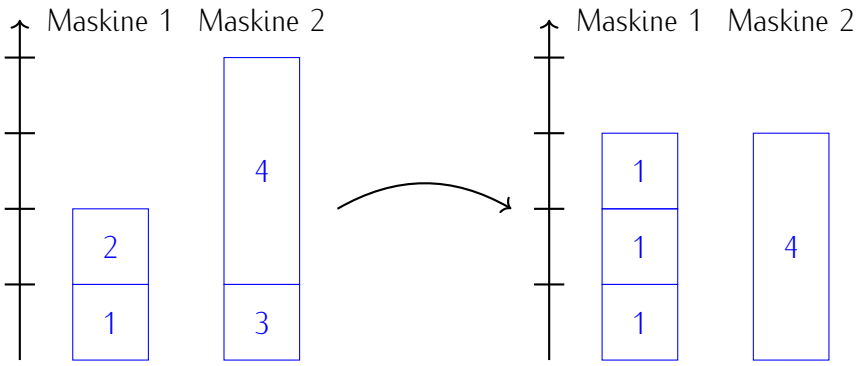
I forrige afsnit blev der argumenteret for, at den tid hvert job tog at processere var irrelevant for optimeringsproblemet, da det alligevel var konstant: Alle jobs

skulle igennem maskinen, så man ville altid komme til at bruge den tid som jobbene samlet krævede, uanset hvilken rækkefølge de blev processeret i. Hvis der er flere maskiner til rådighed, som alle kan klare jobbene, så er dette ikke længere nødvendigvis sandt. Dette skyldes, at man nu har mindst to forskellige objektfunktioner man kan optimere på, nemlig

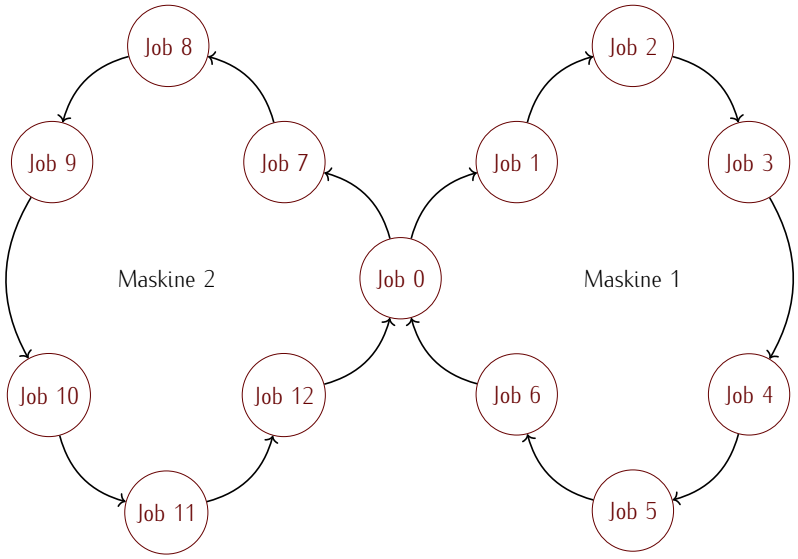
1. Minimering af den samlede tid der bruges på at processere de $n$ jobs på de $m$ maskiner.
2. Minimering af det tidspunkt hvor det sidste job er færdigt.

For at sammenligne dette med lokationsmodeller, så er **punkt 1** en model der fokuserer på effektivitet lige som $p$-median problemet, hvorimod **punkt 2** fokuserer på retfærdighed/flaskehalse ligesom $p$-center modellen. Vi vil i det følgende antage, at der er $m \geq 2$ *identiske maskiner* tilrådighed. Med identiske menes, at det tager $\tau_i$ minutter at processere job $i$ uanset hvilken maskine der produceres på og at det tager $t_{ij}$ at skifte fra job $i$ til job $j$ uanset hvilken maskine skiftet sker på. Som før antages det igen, at opstart af en maskine hvor først job er job $i$ tager $b_i$ minutter, mens nedlukning tager $\beta_i$ minutter hvis job $i$ er sidste job maskinen skal processere. Dette er også uafhængigt af hvilken maskine jobsne bliver produceret på.

*Eksempel* 6.1. For at illustrere at der er forskel på de to typer af objektfunktioner beskrevet oven for, ser vi her på et lille (tænkt) eksempel hvor det antages at der er 4 jobs som skal processeres på to maskiner. Vi vil antage, at omstillingstiden, opstartstiden samt nedlukningstiden alle er lig 0. Altså, $t_{ij} = b_i = \beta_i = 0$. Vi antager videre, at $\tau_1 = \tau_2 = \tau_3 = 1$ mens $\tau_4 = 3$. Hvis vi minimerer den samlede tid, som skal bruges på at processere de 4 jobs, er den optimale objektfunktionsværdi trivielt lig 6: $\tau_1 + \tau_2 + \tau_3 + \tau_4 = 6$. Og det er den for alle de mulige måder vi kan processere på. Det vil sige, at alle løsninger er optimale. Lad os derfor betragte den løsning som siger at job 1 og job 2 skal produceres på maskine 1 mens job 3 og job 4 skal produceres på maskine 2. Rækkefølgerne er job 1→job 2 og job 3→job 4. Dermed er det sidste job som bliver færdigt job 4, efter 4 tidsenheder: $\tau_3 + \tau_4 = 1 + 3 = 4$. Men hvis vi i stedet havde valgt at producere job 1, 2 og 3 på maskine 1 og job 4 på maskine 2, var sidste job færdigt efter 3 tidsenheder. De to løsninger er illustreret i de lodrette Gant chartsi Figur 6.2.




Figur 6.2: Gant charts for fire jobs på to maskiner.




Figur 6.3: Illustration af 12 jobs på to identiske maskiner.

### 6.3.1 Minimering af den samlede tid

Resten af dette dette kapitel vil benytte sig af "dummy" tricket til at modellere situationen. For at give lidt intuition, kan Figur 6.3 nu betragtes. I denne figur ses det, at problemet kan modelleres på samme måde som før hvor der igen blot er tilføjet et dummy job. Nu skal dummy jobbet så at sige blot serviceres to gange antallet af maskiner gange. Dermed kan næsten hele modellen fra tidligere

genbruges hvorved problemet kan formuleres som følger

$$\min \sum_{i=0}^{n} \sum_{\substack{j=0 \\ i \neq j}}^{n} \tilde{t}_{ij} x_{ij} \tag{6.14}$$

$$\text{s.t.:} \sum_{j=1}^{n} x_{0j} = m, \tag{6.15}$$

$$\sum_{i=1}^{n} x_{i0} = m, \tag{6.16}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j = 1, \ldots, n \tag{6.17}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall i = 1, \ldots, n \tag{6.18}$$

$$1 \leq u_i \leq n, \qquad \forall i = 1, \ldots, n \tag{6.19}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \qquad \forall i, j = 1, \ldots, n : i \neq j \tag{6.20}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j = 1, \ldots, n \tag{6.21}$$

Her er tidsparameteren $t_{ij}$ ændret til $\tilde{t}_{ij} := t_{ij} + \tau_i$ for alle $i, j = 1, \ldots, n$. Endvidere er $\tilde{t}_{0j} = b_j$ og $\tilde{t}_{i0} = \tau_i + \beta_i$. I selve modellen er det eneste der er ændret, at for job 0 skal der nu være $m$ jobs som følger umiddelbart efter (6.15) og lige så mange, som kommer før (6.16).

### 6.3.2 Minimering af seneste færdige job

Dette tilfælde er i udgangspunkt noget mere interessant idet modellen kommer til at afvige mere fra den situation hvor der kun er én maskine. Dette skyldes, at der nu skal holdes styr på det tidspunkt hvor den sidste maskine bliver færdig.

Vi vil fortsætte med (næsten) samme notation som i forrige afsnit men indføre nye variabler $y_{ij}$ som får følgende betydning

$$y_{ij} = \begin{cases} 1, & \text{hvis job } i \text{ kommer før job } j \textit{ på den maskine som slutter senest} \\ 0, & \text{ellers} \end{cases}$$

Grunden til, at vi skrev *næsten* samme notation er, at $x_{ij}$-variablerne omdøbes til

$$x_{ij} = \begin{cases} 1, & \text{hvis job } i \text{ kommer før job } j \text{ på en maskine, som } \textit{ikke} \text{ slutter senest} \\ 0, & \text{ellers} \end{cases}$$

Endvidere skal vi indføre en mængde af variabler som holder styr på hvornår jobbene starter deres processer:

$$f_{ij} = \begin{cases} \text{tidspunkt hvor job } j \text{ starter}, & \text{hvis } j \neq 0 \text{ og } x_{ij} + y_{ij} = 1 \\ \text{tidspunkt hvor maskine som behandler job } i \text{ slutter}, & \text{hvis } j = 0 \text{ og } x_{ij} + y_{ij} = 1 \\ 0, & \text{ellers} \end{cases}$$

og slutteligt skal vi have en variabel $F$ som er lig med det seneste tidspunkt en maskine er færdig. Vi definerer ovenstående indicerede variabler for alle jobs $i$ og $j$ *inklusiv dummy jobbet*. Det vil sige, at variablerne $y_{ij}$, $x_{ij}$ og $f_{ij}$ er defineret for alle $i, j \in \{0, 1, \ldots, n\}$.

Vi kan nu notere os, at vi skal have præcis ét job til at starte på den maskine som slutter senest, hvilket vil sige, at der skal være præcis et job, som følger efter job 0 på den maskine som slutter senest:

$$\sum_{j=1}^{n} y_{0j} = 1.$$

Samtidig skal vi start jobs på de resterende $m - 1$ maskiner hvorfor vi får

$$\sum_{j=1}^{n} x_{0j} = m - 1.$$

Nu skal vi sørge for, at hvis job $i$ bliver behandlet på den maskine som bliver sidst færdig, så betyder det, at der kommer et job før og et job efter job $i$ (det kan være job 0):

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ji}, \quad \forall i = 1, \ldots, n$$

Ligeledes skal der gælde for alle jobs $i$, at hvis de skal processeres på en maskine, som ikke bliver sidst færdig, så skal der være et job før og et job efter (dette kan igen potentielt være job 0):

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ji}, \quad \forall i = 1, \ldots, n$$

Vi mangler nu at sørge for, at hvert job rent faktisk behandles på en maskine. Det vil sige, at der skal være præcis et job som kommer efter job $i$ på enten den maskine som bliver sidst færdig eller på en af de andre. Dette kan formuleres som

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} + \sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ij} = 1, \quad \forall i = 1, \ldots, n$$

Det, der nu mangler i modellen, er, at holde styr på hvornår de forskellig jobs starter. Først kan vi notere os, at job j bør starte til tid $b_j$, hvis det er det første job efter dummy jobbet. Derfor har vi

$$f_{0j} = b_j(x_{0j} + y_{0j}), \quad \forall j = 1, \ldots, n$$

Det sørger naturligvis ikke for, at hvert jobs startid er under kontrol. Vi kan dog her notere os, at der for hvert job $i$ er præcis en af $f_{ij}$ variablerne der er forskellig fra 0, da $f_{ij}$ kun er større end 0 hvis $x_{ij} + y_{ij} = 1$ hvilket sker for præcis et job (der følger kun et job efter job $i$). Vi kan nu regne os frem til, at tiden hvor job $i$ starter må være lig med den tid hvor det forrige job startede + den tid det tog at producere det forrige job + omstillingstiden. Dette kan skrives som følger:

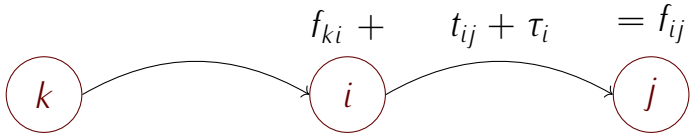
$$\sum_{\substack{j=0 \\ i \neq j}}^{n} f_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} f_{ji} + \sum_{\substack{j=0 \\ i \neq j}}^{n} \tilde{t}_{ij}(x_{ij} + y_{ij}), \quad \forall i = 1, \ldots, n$$

Husk, at $\tilde{t}_{ij} := t_{ij} + \tau_i$ for alle $i, j = 1, \ldots, n$, $\tilde{t}_{0j} := b_j$ og $\tilde{t}_{i0} = \tau_i + \beta_i$.

På venstre siden af lighedstegnet er der kun én variabel i summen, der vil være forskellig fra nul og det vil netop være for det job $j$ som kommer efter job $i$. Altså står der på venstre siden, at starttiden for det job $k$, som kommer efter job $i$ er lig med højresiden. Dissekeres nu højresiden, er det også sandt, at der kun er én $f_{ji}$ variabel i summen, som er forskellig fra nul og det vil netop være for det job $k$, som kommer før job $i$. Altså, er denne del af summen lig med starttiden for job $i$. Den sidste sum på højresiden vil også have præcis ét led hvor $x_{ij} + y_{ij} = 1$, nemlig for det job $j$ som kommer efter job $i$. Resten af leddene vil have $x_{ij} + y_{ij} = 0$. Derfor står der i ligheden

> Starttiden for det job, som kommer efter job $i$ "=" starttiden for job i + omstillingstiden mellem job $i$ og det som kommer efter ($t_{ij}$) + procestiden for job $i$ ($\tau_i$).

Situation kan illustreres med følgende diagram:




Vi er nu næsten i mål og mangler blot at sørge for, at $f_{ij} = 0$ hvis $x_{ij} + y_{ij} = 0$ og at relatere $F$ til de resterende variabler. Det første kan snildt overkommes med en big-$M$ begrænsning af formen:

$$f_{ij} \leq M(x_{ij} + y_{ij}), \quad \forall i, j = 1, \ldots, n$$

mens den sidste del kan ordnes ved at notere sig, at $f_{i0}$ er sluttiden på en maskine såfremt job $i$ er sidste job på en maskine og ellers er $f_{i0} = 0$ (hvorfor?). Dermed kan vi blot introducere begrænsningerne:

$$f_{i0} \leq F, \quad \forall i = 1, \ldots, n$$

Endvidere, kan vi også huske at $y_{ij}$ variablerne netop gav process-rækkefølgen på den maskine, som bliver senest færdig, hvorfor $F$ er lig med den samlede tid brugt på denne maskine. Dermed får vi $F = \sum_{i=0}^{n} \sum_{j=0}^{n} \tilde{t}_{ij} y_{ij}$ og det samlede

program falder således let ud som

$$\min F \tag{6.22}$$

$$\text{s.t.: } F = \sum_{i=0}^{n} \sum_{\substack{j=0 \\ i \neq j}}^{n} \tilde{t}_{ij} y_{ij} \tag{6.23}$$

$$\sum_{j=1}^{n} y_{0j} = 1, \tag{6.24}$$

$$\sum_{j=1}^{n} x_{0j} = m - 1, \tag{6.25}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ji}, \qquad \forall i = 1, \ldots, n \tag{6.26}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ji}, \qquad \forall i = 1, \ldots, n \tag{6.27}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} + \sum_{\substack{j=0 \\ i \neq j}}^{n} y_{ij} = 1, \qquad \forall i = 1, \ldots, n \tag{6.28}$$

$$f_{0j} = b_j(x_{0j} + y_{0j}), \qquad \forall j = 1, \ldots, n \tag{6.29}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} f_{ij} = \sum_{\substack{j=0 \\ i \neq j}}^{n} f_{ji} + \sum_{\substack{j=0 \\ i \neq j}}^{n} \tilde{t}_{ij}(x_{ij} + y_{ij}), \qquad \forall i = 1, \ldots, n \tag{6.30}$$

$$f_{ij} \leq M(x_{ij} + y_{ij}), \qquad \forall i, j = 1, \ldots, n \tag{6.31}$$

$$f_{i0} \leq F, \qquad \forall i = 1, \ldots, n \tag{6.32}$$

$$x_{ij}, y_{ij} \in \{0, 1\}, \qquad \forall i, j = 0, \ldots, n \tag{6.33}$$

$$f_{ij} \geq 0, \qquad \forall i, j = 0, \ldots, n \tag{6.34}$$

Som det ses stiger kompleksiteten af modellen dramatisk når man ønsker at minimere tiden hvorved den sidste maskine bliver færdig. Dette skyldes, i nogen grad, at objektfunktionen i princippet er en ikke-lineær funktion (maksimum af en endelig mængde lineære funktioner) som er linariseret. Dertil kommer, at kun en lille del af modellens variabler indgår i objektfunktionen hvilket leder til en

relativet dårlig nedre grænse fra den *lineære relaksering*, hvorved branch and bound træet potentielt bliver meget stort.

**Opgave 6.4:** Bevis, at begrænsningerne (6.30) sammen med (6.31) sørger for at begrænsninger af type (6.20) som bruger $u_i$ variabler er overflødige.

**Opgave 6.5:** Betragt følgende data til et sekvenseringsproblem for 10 jobs på flere maskiner:

| | | | | Omstillingstid | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\tau$ | Opstartstid | Nedluk.tid | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 6 | 23 | 5 | – | 11 | 18 | 5 | 5 | 10 | 6 | 15 | 21 | 14 |
| 2 | 9 | 16 | 16 | 9 | – | 21 | 24 | 5 | 14 | 14 | 21 | 7 | 20 |
| 3 | 12 | 11 | 6 | 23 | 21 | – | 9 | 17 | 17 | 22 | 22 | 9 | 16 |
| 4 | 11 | 10 | 25 | 5 | 24 | 10 | – | 13 | 25 | 16 | 10 | 14 | 23 |
| 5 | 10 | 23 | 6 | 10 | 6 | 5 | 12 | – | 21 | 19 | 18 | 23 | 9 |
| 6 | 5 | 15 | 15 | 11 | 8 | 16 | 14 | 17 | – | 9 | 18 | 14 | 15 |
| 7 | 5 | 17 | 16 | 22 | 22 | 19 | 14 | 16 | 24 | – | 25 | 18 | 15 |
| 8 | 14 | 24 | 25 | 23 | 11 | 24 | 24 | 8 | 9 | 19 | – | 19 | 19 |
| 9 | 10 | 12 | 24 | 5 | 11 | 21 | 24 | 15 | 14 | 13 | 9 | – | 9 |
| 10 | 9 | 21 | 12 | 22 | 8 | 8 | 21 | 18 | 5 | 12 | 5 | 10 | – |

Find en optimal sekvensering af de 10 jobs ved hjælp af modellen (6.14)–(6.21) for $m = 1, 2, 3, 4, 5$.

**Opgave 6.6:** Find en optimal løsning til modellen (6.22)–(6.34) for datasættet fra **Opgave 6.5** og for $m = 1, 2, 3, 4, 5$.

**Opgave 6.7:** Reflekter over sammenhængen mellem løsningsværdien og $m$ for modellen givet ved (6.14)–(6.21) og modellen givet ved (6.22)–(6.34).

# 7 Ruteplanlægning

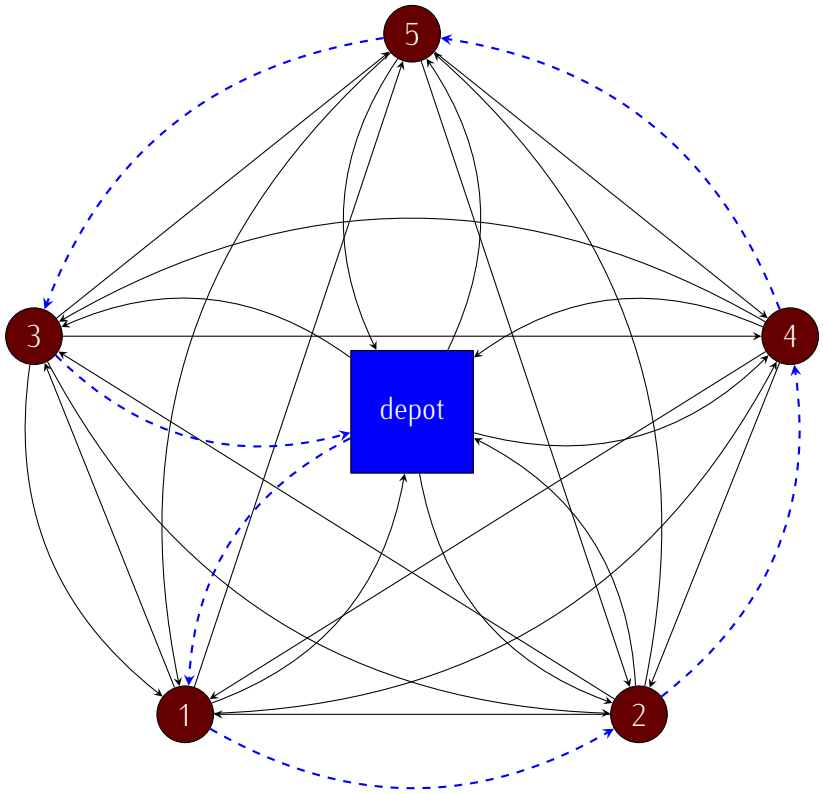
## 7.1 Hvad er ruteplanlægning?

Det, at planlægge optimale ruter, er et af de mest studerede områder inden for operationsanalysen og dette med god grund: ruteplanlægningsproblemer dukker op mange steder, og ikke kun hvor man benytter sig af fysiske køretøjer/fartøjer.

Det hjælper dog ofte at forestille sig en til flere køretøjer, som skal flytte sig rundt mellem en række lokationer, hvor rækkefølgen hvori lokationerne besøges, har stor betydning for, hvor god en løsning man finder er. Det nok mest klassiske ruteplanlægningsproblem er det såkaldte traveling salesperson problem (TSP), og dette kapitel vil derfor tage sit udgangspunkt heri og lige så stille udbygge prDet nok mest klassiske ruteplanlægningsproblem er det såkaldte traveling salesperson problemoblemet derfra.

Inden udviklingen af formuleringer påbegyndes, vil dele af den nødvendige notation, der bruges i de følgende afsnit, blive introduceret. Først og fremmest vil hvad, der følger, benytte sig af et antal kunder, som alle skal besøges af de køretøjerne. Det antages, at dette antal er givet ved $n$. Derudover vil det også blive antaget, at alle køretøjer starter i et centralt depot og at de alle, hvis ikke andet antages, skal returnere til depotet igen efter de har afsluttet deres tur.

Ved at nummerere alle kunderne fra 1 til $n$ og lade depotet være nummereret 0, kan vi bygge en *komplet graf* $G$ givet ved $G = (V, A)$ hvor mængden af knuder $V$ er givet ved $V := \{0, 1, 2, \ldots, n\}$ og mængden $A$ af kanter er givet ved $A := \{(i, j) : i, j \in V\}$. Vi kan tænke på den komplette graf $G$ som grafen i Figur 7.1. Vi vil til hver kant $(i, j) \in A$ tildele en *omkostning* som angiver hvor meget det koster at rejse via kanten $(i, j)$. Vi vil lade den ikke-negative parameter $c_{ij}$ angive denne omkostning. Slutteligt indfører vi en mængde af binære variabler




**Figur 7.1: Illustration af den komplette graf bestående af seks knuder (5 knuder som repræsenterer kunder og én knude som repræsenterer et depot). Grafen er komplet fordi alle knuder er forbundet med en kant til enhver anden knude.**

$x_{ij}$ som er defineret ved

$$x_{ij} = \begin{cases} 1, & \text{hvis kanten } (i,j) \text{ benyttes i en løsning} \\ 0, & \text{ellers} \end{cases}.$$

Når der i definitionen står "benyttes" betyder det, at et køretøj kører direkte fra knude $i$ til knude $j$. Med denne terminologi på plads er vi klar til at definere traveling salesperson problemet.

## 7.2 Traveling salesperson problemet

I al sin enkelhed kan et TSP beskrives som følger: Givet en graf $G = (V, A)$ og afstande/omkostninger $c_{ij}$ for hver $(i,j) \in A$, bestem en kortest mulig hamiltonsk cykel i grafen $G$ (en hamiltonsk cykel er en sammenhængende tur gennem grafen som besøger alle knuder præcis én gang). Det vil altså sige, at man for at løse TSP skal vælge nogle kanter i $A$, således at hver knude besøges præcis én, gang og således at den samlede turs længde er kortest mulig.

Traveling Salesperson problemet er en efterhånden gammel problemstilling, som er studeret igennem mange år, og blev oprindeligt defineret af den østrigske matematiker Karl Menger tilbage i starten af 1930'erne, hvor han til et matematisk kollokvium i Wien opstillede følgende problemstilling

> Vi vil ved *budbringer-problemet* betegne opgaven at finde, for et endeligt antal punkter hvis indbyrdes afstande er kendte, den kortest mulige sti, som forbinder disse punkter. [...]. Reglen hvor man altid går fra startpunktet videre til dets nærmeste punkt, etc.., resulterer ikke generelt set i en korteste sti.

I grafen, som er illustreret i Figur 7.1, er der med blå stiplede pile indtegnet en hamiltonsk cykel, som ikke nødvendigvis er kortest mulig.

I 1954 publicerede Dantzig m.fl. hvad der må antages at være den første MILP formulering af problemet som her skal gengives

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{7.1}$$

$$\text{s.t.:} \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, \qquad \forall j \in V \tag{7.2}$$

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1, \qquad \forall i \in V \tag{7.3}$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1, \qquad \forall S \subset V, 2 \leq |S| \leq \lceil \frac{|V|}{2} \rceil \tag{7.4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in V \tag{7.5}$$

Denne formulering er ofte omtalt som DFJ formuleringen af TSP.

Lad os gennemgå formuleringen linje for linje: Objektfunktionen, (7.1), minimerer simpelthen summen af omkostninger som indtræffer når vi vælger kanter ud. Omkostningen $c_{ij}$ lægges til den samlede omkostning hvis, og kun hvis, den tilsvarende binære variabel $x_{ij} = 1$, og ellers bidrager kanten $(i, j)$ ikke til den samlede omkostning. Begrænsningerne (7.2) sørger for, at der vælges præcis én kant, som har knude $j$ som slutpunkt, og dette skal gælde for alle $j \in V$. Dermed vides det, at et køretøj/salgspersonen kommer *til* knude $j$ præcis én

gang. Ligeledes skal vi sørge for at salgspersonen forlader alle knuder præcis én gang, hvilket begrænsningerne (7.3) sørger for. Begrænsningerne (7.4) er såkaldte subtur-elimineringsbegrænsninger. Det vil sige, at disse begrænsninger sørger for, at den endelige samling af kanter der udvælges er en sammenhængende delgraf. Problemet opstår altså idet, at begrænsningerne (7.2) og (7.3) kun søger for at hver knude besøges, men ikke at den tur der bruges er sammenhængende. Subtur elimineringsbegrænsningerne sørger derimod for, at for en hver delmængde af knuder $S$ som udvælges, så skal der være mindst én kant, som fører fra knuderne i $S$ til de resterende knuder, som ikke er i $S$.

Denne formulering, selvom den har en del år på bagen efter hånden, er en af de stærkeste (mht. lineær relaksering) og mest effektive, såfremt man genererer begrænsningerne (7.4) dynamisk. Grunden til at disse skal genereres dynamisk for at formuleringen virker i praksis er, at antallet af begrænsninger som ligger "gemt" i $\forall S \in V$ er eksponentielt voksende med antallet af knuder i grafen $G$ (der er $\mathcal{O}(2^{|V|})$ delmængder af $V$ hvis størrelse ikke overstiger $\lceil \frac{n}{2} \rceil$). Det betyder i praksis, at hvis man forsøger at loaded alle disse begrænsninger ind i sin computer fra start af, løber man simpelthen tør for hukommelse.

Desværre forholder det sig også sådan, at det bestemt ikke er trivielt at implementere software som kan løse TSP med DFJ formuleringen på en effektiv måde. Et af de mest effektive stykker software som kan bruges til at løse TSP'er er det såkaldte CONCORDE bibliotek som er tilgængeligt gratis for akademisk brug. Det er lykkedes forskerne bag CONCORDE at løse TSP'er med helt op til 85.900 knuder.

Selvom det er særdeles komplekst at benytte DFJ formuleringen på en effektiv måde, betyder det ikke, at det er den eneste farbare vej for at løse TSP'er af det man kunne kalde "rimelig størrelse". Det skal vi undersøge nærmere i næste afsnit.

### 7.2.1 MTZ formuleringen af subtur elimination-begrænsningerne for TSP

Som det blev antyder ovenfor, så findes der også andre måder at undgå subture i sine TSP formuleringer. Miller m.fl. (1960) forslog en formulering, der eksplicit benytter sig af, at man kan betragte en løsning af en TSP som en ordning af knuderne i den rækkefølge, de skal besøges. Betragt for eksempel Eksempel 7.1.

*Eksempel* 7.1. Lad der være givet et depot som får nummeret 0 og lad der yderligere være fem kunder som bliver nummereret fra 1 til 5. En mulig tur ville være at

starte i depotet (knude 0) for så at køre til knude 1, til knude 2, til knude 4, til knude 5, til knude 3 og så tilbage til depotet (som illustreret med blå stiplede linjer i Figur 7.1). Skulle den løsning oversættes til $x_{ij}$-variablerne alene, ville det være som i Tabel 7.1.

**Tabel 7.1: Løsning til Figur 7.1 gengivet ved hjælp af $x_{ij}$-variabler**

| $x_{ij}$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 |

Men man kunne også tildele hver knude en variabel, $u_i$ for hvert $i \in V \setminus \{0\}$, og så lade denne angive hvilket nummer i rækkefølgen knude $i$ kommer på turen. Det vil sige, i dette tilfælde, at $u_1 = 1$, $u_2 = 2$, $u_3 = 5$, $u_4 = 3$ og $u_5 = 4$. Det vil sige, at knude 1 er nummer 1 på turen, knude 2 er nummer to på turen, knude 4 er nummer 3 på turen, knude 5 er nummer 4 på turen og slutteligt er knude 3 nummer 5 på turen.

For at formulere de såkaldte MTZ begrænsninger for at eliminere subture, skal vi som i Eksempel 7.1 benytte nogle nye variabler $u_i$ for hver kunde-knude. Disse variabler angiver ordningen af kunderne på turen. De første begrænsninger der skal bruges, og som kommer meget naturligt, er øvre og nedre grænser på $u_i$ variablerne. For det første, kan vi deducere at der ikke er nogen kunder der kan komme før den første der besøges hvorfor $u_i \geq 1$ for alle $i \in V \setminus \{0\}$. Derudover har vi også, at der ikke er nogen der kommer efter den sidste. Den sidste kunde er den som kommer på den $n$'te plads i rækkefølgen hvorfor $u_i \leq n$. Dermed haves den første mængde af de såkaldte MTZ begrænsninger

$$1 \leq u_i \leq n, \quad \forall i \in V \setminus \{0\}. \tag{7.6}$$

Det, der mangler, for, at MTZ begrænsningerne har deres tilsigtede virkning er, at hvis knude $i$ besøges før knude $j$, så skal $u_i + 1 \leq u_j$ (hvorfor?). Miller m.fl. formulerede denne betingelse på følgende snedige vis:

$$u_i - u_j + nx_{ij} \leq n - 1, \quad \forall i, j \in V \setminus \{0\} \tag{7.7}$$

For at indse, at disse begrænsninger sørger for at $u_i + 1 \leq u_j$ hvis $i$ kommer lige før $j$ på turen, og at der ellers ikke er nogen relation mellem $u_i$ og $u_j$, skal vi her betragte to tilfælde

*$i$ besøges lige før $j$*: Lad to knuder $i$ og $j$ være givet. Hvis knude $i$ besøges lige før knude $j$, betyder det at $x_{ij} = 1$. Dermed reducerer ulighederne (7.7) på følgende vis

$$\begin{aligned} & u_i - u_j + n x_{ij} \leq n - 1 \\ & \Leftrightarrow \\ & u_i - u_j + n \leq n - 1 && (x_{ij} = 1 \text{ indsat}) \\ & \Leftrightarrow \\ & u_i + 1 \leq u_j && (u_j \text{ lagt til og } (n-1) \text{ trukket fra}) \end{aligned}$$

Hvilket var præcis hvad vi ønskede.

*$i$ besøges ikke lige før $j$*: Lad to knuder $i$ og $j$ være givet. Hvis $i$ ikke kommer lige før knude $j$ så haves at $x_{ij} = 0$ hvilket reducerer (7.7) på følgende vis

$$\begin{aligned} & u_i - u_j + n x_{ij} \leq n - 1 \\ & \Leftrightarrow \\ & u_i - u_j \leq n - 1 && (x_{ij} = 0 \text{ indsat}) \end{aligned}$$

For at indse, at denne begrænsning ikke har nogen effekt, skal man overbevise sig selv selv om, at $u_i - u_j$ aldrig kan blive større end $n - 1$ da det vil betyde, at begrænsningerne (7.7) blot er overflødige når $x_{ij} = 0$. For at indse dette, skal vi undersøge hvornår $u_i - u_j$ er størst mulig, hvilket er når $u_i$ er så stor som mulig, og $u_j$ er så lille som mulig da det gør differencen størst. For at finde en øvre grænse for $u_i$ og en nedre grænse for $u_j$ kigger vi på begrænsningerne (7.6), hvorfra vi får at $u_i \leq n$ og $u_j \geq 1$. Dermed ved vi, at $u_i - u_j \leq n - 1$ for alle brugbare løsninger.

Dermed er (7.7) valide både når $x_{ij} = 1$ og $x_{ij} = 0$. Man bør også notere sig, at begrænsningerne også fjerner den simple subtur $x_{ii} = 1$: Hvis $i = j$, haves at ulighederne reducerer til

$$u_i - u_i + n x_{ii} \leq n - 1 \Leftrightarrow n x_{ii} \leq n - 1$$

Deles nu med $n$ på begge sider af ulighedstegnet fås $x_{ii} \leq \frac{n-1}{n} < 1$, hvorved den eneste værdi $x_{ii}$ kan antage er nul. Denne subtur er dog ikke en reel

mulighed, såfremt man husker, at summerne på venstresiden i ind- og udgrads-begrænsningerne for hver knude ikke indeholder diagonal-variablerne.

Det vil altså sige, at vi kan opskrive et TSP vha. det man ofte kalder MTZ formuleringen på følgende vis:

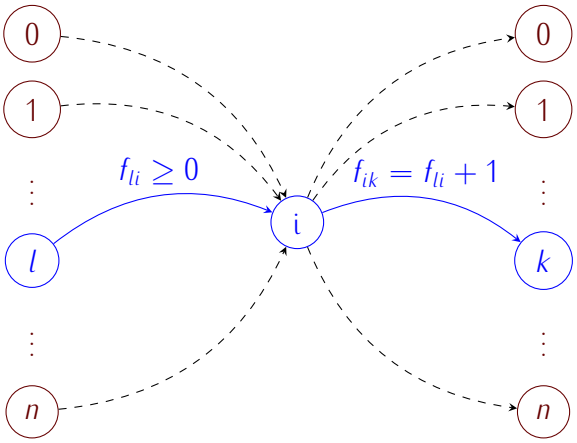
$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} & & \\
\text{s.t.:} \quad & \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, & & \forall j \in V \\
& \sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1, & & \forall i \in V \\
& 1 \leq u_i \leq n, & & \forall i \in V \setminus \{0\} \\
& u_i - u_j + n x_{ij} \leq n - 1, & & \forall i, j \in V \setminus \{0\} \\
& x_{ij} \in \{0, 1\}, & & \forall i, j \in V
\end{aligned}
$$

Denne formulering har signifikant færre begrænsninger end DFJ formuleringen, (7.1)–(7.5), og er dermed noget nemmere at arbejde med. Desværre kommer de færre begrænsninger med den omkostning, at den lineære relaksering er meget svagere end den er for DFJ formuleringen, hvilket også betyder, at man kan løse meget mindre problemer med MTZ formuleringen end med DFJ formuleringen.

### 7.2.2 Gavish og Graves formulering af TSP

Vi har indtil nu set på DFJ formuleringen, som var meget stærk, men til gengæld ikke så håndterbar og MTZ-formuleringen, som var meget håndterbar men ikke så stærk. I dette afsnit vil vi kigge på en model som lægger sig ind imellem de to formuleringer og som i praksis ofte virker rigtig godt. Denne model blev foreslået af Gavish og Graves (1978) og benytter en såkaldt network-flow formulering til at udelukke subture. Denne nye model benytter ikke $u_i$-variablerne som i MTZ-formuleringen, men derimod en disaggregeret version af disse, som er defineret som følger: hvis køretøjet kører direkte fra $i$ til $j$, angiver variablen $f_{ij}$ hvilken plads på turen knude $i$ er, ellers er $f_{ij} = 0$. Det vil sige, at hvis $x_{ij} = 1$, så haves følgende relation mellem $u_i$-variablen fra MTZ-formuleringen, og så de nye variabler

$$u_i = \sum_{j \in V} f_{ij}, \quad \forall i \in V \setminus \{0\}.$$




**Figur 7.2: Illustration af begrænsningerne (7.8) fra Gavish og Graves' formulering af TSP.**

Den første begrænsning vi kan sørge for er, at hvis $x_{ij} \neq 1$ så er $f_{ij} = 0$. Dette er den simple *big-M* begrænsning $f_{ij} \leq M x_{ij}$. Her står vi så med den sædvanlige udfordring med at finde den rette værdi til $M$, men i dette tilfælde er det ikke så svært, for vi ved præcis hvad den største, værdi $f_{ij}$ kan antage, er: $n$. Derfor kan vi opskrive de variable øvre grænser for $f_{ij}$ som $f_{ij} \leq n x_{ij}$ for alle $i, j \in V$.

Vi skal nu, ligesom med $u_i$-variablerne, sørge for, at $f_{ij}$ akkumulerer antallet af kunder, der bliver besøgt på turen, så den rigtige plads bliver tildelt den rigtige knude. Dette gøres ved følgende mængde af begrænsninger

$$\sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji} + 1, \quad \forall i \in V \setminus \{0\} \tag{7.8}$$

På venstresiden af lighedstegnet har vi summen over alle $f_{ij}$ variablerne, og vi ved, at der præcis er én af disse som ikke er lig nul for alle kunder $i$; netop for den kunde, som efterfølger $i$ på turen. Denne ene variabel, som ikke er lig nul, skal så være lig med summen af alle $f_{ji}$-variabler $+1$. Der er også kun én $f_{ji}$ variabel som ikke er lig 0 for den pågældende knude $i$. For at illustrere tankegangen bag begrænsningerne, kan Figur 7.2 betragtes. Her er ideen illustreret hvor knude $i$ holdes fast og hvor vi på venstreside i figuren har de knude hvorfra køretøjet kan komme og på højresiden haves de knuder hvortil køretøjet kan køre. I dette tilfælde kommer køretøjet fra knude $l$ og fortsætter gennem knude $i$ til knude $k$. Dermed ved vi, at knude $i$'s plads på turen ($f_{ik}$) skal være én større end knude $l$'s ($f_{li}$).

Man kan også betragte begrænsningerne (7.8) som flow-conservation begrænsninger i et network-flow problem, hvor hver kunde har et positivt supply på én

enhed. Derfor bliver Gavish og Graves's formulering også omtalt som en "one commodity flow" formulering.

Den samlede formulering foreslået af Gavish og Graves (1978) kan således opsummeres som

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} & & \\
\text{s.t.:} \quad & \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, & & \forall j \in V \\
& \sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1, & & \forall i \in V \\
& \sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji} + 1, & & \forall i \in V \setminus \{0\} \\
& f_{ij} \leq n x_{ij}, & & \forall i, j \in V \\
& x_{ij} \in \{0, 1\}, & & \forall i, j \in V
\end{aligned}
\tag{7.9}
$$

Denne formulering kan styrkes ved at tilføje følgende brugbare uligheder:

$$
f_{ij} \geq \min\{1, i\} x_{ij}, \quad \forall i, j \in V \tag{7.10}
$$

Det er overladt til læseren at argumentere for, at disse uligheder er brugbare (se **Opgave 7.4**).

---

**Opgave 7.1:** Vis at MTZ-begrænsningerne udelukker subture.

**Opgave 7.2:** Vis at Gavish og Graves (1978)'s begrænsninger udelukker subture.

**Opgave 7.3:** I denne opgave skal det vises, at Gavish og Graves (1978)'s model er stærkere end MTZ-formuleringen.

1. Vis, at hvis man har en LP-løsning til Gavish og Graves formulering, kan man konstruere en løsning til MTZ-formuleringen.
2. Argumenter for, at punkt 1. medfører at LP relakseringen af Gavish og Graves (1978) formulering er stærkere end MTZ-formuleringen.

**Opgave 7.4:** Argumenter for, at ulighederne (7.10) er brugbare i den forstand, at de er opfyldt for enhver brugbar løsning til TSP.

**Opgave 7.5:** Betragt en rejsende, som ønsker at rejse ud fra sit hotel for så at besøge 10 seværdigheder på en sådan måde, at hun tager den korteste tur rundt blandt de 10 seværdigheder. Seværdighederne samt hotellet har $(x, y)$-koordinaterne givet ved

| | | Seværdigheder | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | hotel | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| x-koordinat | 5 | 4 | 6 | 1 | 6 | 2 | 9 | 0 | 10 | 5 | 7 |
| y-koordinat | 5 | 4 | 6 | 5 | 6 | 3 | 9 | 9 | 6 | 8 | 2 |

Implementer den rejsendes ruteplanlægningsproblem vha. af både MTZ og Gavish og Graves formuleringerne i Pyomo og løs disse modeller.

## 7.3 Multiple-TSP: mere end et køretøj

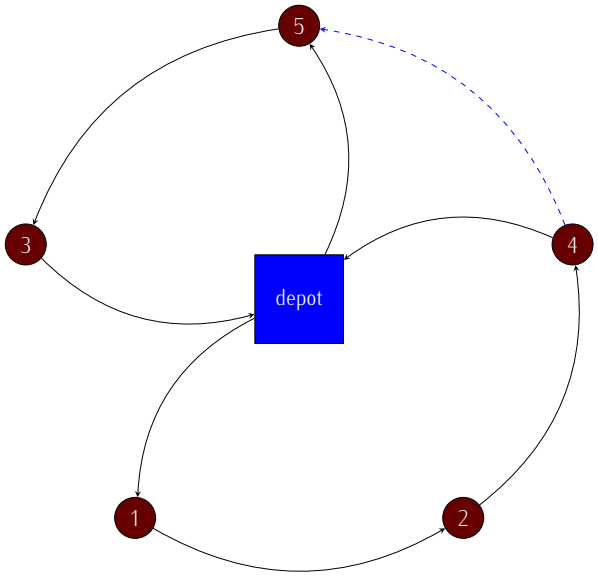
I mange tilfælde er det ikke muligt for et køretøj at servicere alle kunderne på grund af, for eksempel, tidsbegrænsninger eller begrænsninger på køretøjets kapacitet. Derfor er det naturligt at udvide TSP til det man kalder *multiple traveling salesperson problemet* eller *m-TSP*, hvor der nu er $m \geq 2$ køretøjer, som hver starter og ender deres tour ved depotet og samlet set skal besøge alle kunde knuderne præcis én gang, sådan at den totale omkostning ved de $m$ ture minimeres.

Vi vil generelt set ikke antage noget om afstands/omkostningsmatricen $(c_{ij})_{i,j \in V}$, ud over at denne har ikke-negative indgange. Der er dog en naturlig yderligere antagelse som ofte bliver gjort i litteraturen, nemlig at afstandene overholder *trekants-uligheden*:

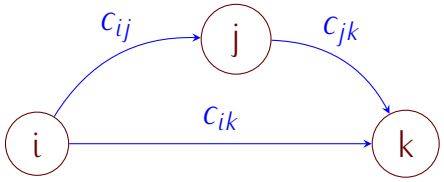
**Definition 7.1. Trekants-uligheden** Man siger, at en afstandsmatrix overholder trekants-uligheden hvis der for alle tripler af knuder $i, j, k \in V$ gælder, at det aldrig er længere at køre fra $i$ direkte til $k$ end det er at køre fra $i$ til $k$ via $j$. Matematisk kan dette opskrives som

$$c_{ik} \leq c_{ij} + c_{jk}, \quad \forall i, j, k \in V, i \neq j, i \neq k, j \neq k$$

Dette kan illustreres ved




Figur 7.3: Illustration af Eksempel 7.2.




Hvis trekants-uligheden er opfyldt for afstandsmatricen kan det fortolkes således, at det ikke er muligt at skyde genvej via en knude uden at øge turens længde. Denne egenskab leder til følgende observation

**Observation 7.1.** Antag at $(c_{ij})_{i,j\in V}$ overholder trekants-uligheden. Da gælder der, at den samlede længde af turene i en optimal løsning til $m$-TSP, er længere end en optimal turs længde til TSP. Mere præcist, hvis vi lader den optimale løsningsværdi til $m$-TSP og TSP være givet ved $Z^m_{TSP}$ for $m = 1, 2, 3 \ldots, \ldots, n$, da gælder

$$Z^1_{TSP} \leq Z^2_{TSP} \leq Z^3_{TSP} \leq \cdots \leq Z^n_{TSP}$$

For at overbevise sig om, at denne observation er valid, kan man betragte følgende lille eksempel

*Eksempel 7.2.* Betragt et 2-TSP problem med et depot og 5 kunder, som illustreret i Figur 7.3. Her er knuderne 1, 2 og 4 serviceret af en bil, mens 3 og 5 serviceres af en anden. Da vi her antager, at afstandene overholder trekants-uligheden kan vi spare noget af turlængden ved at køre direkte fra kunde 4 til 5 da $c_{4,5} \leq c_{4,0} + c_{0,5}$ per antagelse. Denne observation kan gentages, hvis der er tale om flere end to biler indtil der kun er én tur i spil. Man skal her notere sig, at dette argument

er betinget af, at trekants-uligheden er overholdt for de kanter der indsættes og fjernes!

Vi har nu betragtet det scenarie hvor trekants-uligheden holder for afstands-matricen. Det er dog ikke en nødvendig antagelse for det følgende, og derfor bør læseren tænke i baner, hvor trekants-uligheden ikke nødvendigvis er overholdt. Fremadrettet vil vi antage, at hver salesperson kan håndtere $S$ kunder på sin tur hvor $S < n$ og $S \geq \lceil \frac{n}{m} \rceil$ (Se **Opgave 7.6**). Der skal nu meget få ændringer til i forhold til modellerne, for TSP for at inkorporere muligheden for at benytte flere sælgere. Objektfunktionen ændrer sig således ikke i forhold til TSP-modellerne og vil fortsat være givet ved

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}.$$

Begrænsningerne, som definerer ind- og ud-graden af knuderne i $G$, skal nu ændres en smule. For kundeknuderne sker der ingen ændringer, idet hver kunde stadig skal besøges og forlades præcis én gang hver:

$$\sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, \qquad \forall j \in V \setminus \{0\}$$

$$\sum_{\substack{j \in V \\ i \neq j}} x_{ij} = 1, \qquad \forall i \in V \setminus \{0\}$$

Der sker dog en ændring i forhold til ind-ud graden for depotet, da der nu skal udgå og indgå $m$ biler. Det vil sige, at der skal vælges præcis $m$ udgående og $m$ indgående kanter til depotet, eller knude 0:

$$\sum_{j \in V \setminus \{V\}} x_{0j} = m, \tag{7.11}$$

$$\sum_{i \in V \setminus \{V\}} x_{i0} = m. \tag{7.12}$$

Det vil sige, at $m$-TSP kan formuleres som

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, && \forall j \in V \setminus \{0\} \\
& \sum_{\substack{j \in V \\ i \neq j}} x_{ij} = 1, && \forall i \in V \setminus \{0\} \\
& \sum_{j \in V \setminus \{0\}} x_{0j} = m, \\
& \sum_{i \in V \setminus \{0\}} x_{i0} = m. \\
& \text{Ingen subture} && (7.13) \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in V
\end{aligned}
$$

Det første, der burde springe læseren i øjnene, er naturligvis (7.13), som i sagens natur ikke er særligt operationaliserbar. Som med TSP deler vi her subturelimineringsbegrænsningerne op i en MTZ baseret og en Gavish og Graves baseret version.

### 7.3.1 MTZ baseret subtur eliminering for $m$-TSP

Som med TSP bruges her $u_i$ variabler for hver kunde $i \in V \setminus \{0\}$, som her angiver placeringen af kunde $i$ *på den tur hvorpå kunde $i$ serviceres*. Dermed kan vi også hurtigt fastsætte øvre og nedre grænser for $u_i$'s værdi:

$$1 \leq u_i \leq S, \quad \forall i \in V \setminus \{0\}$$

da der maksimalt kan være $S$ kunder på en tur, og da en kunde ikke kan besøges før den første kunde på turen er besøgt.

Ligeledes er begrænsningerne (7.7) relativt lette at tilpasse, da vi blot skal ændre $n$ til $S$:

$$u_i - u_j + S x_{ij} \leq S - 1, \quad \forall i, j \in V \setminus \{0\} : i \neq j \qquad (7.14)$$

Dermed er en MTZ-baseret formulering af $m$-TSP givet ved

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, && \forall j \in V \setminus \{0\} \\
& \sum_{\substack{j \in V \\ i \neq j}} x_{ij} = 1, && \forall i \in V \setminus \{0\} \\
& \sum_{j \in V \setminus \{0\}} x_{0j} = m, \\
& \sum_{i \in V \setminus \{0\}} x_{i0} = m. \\
& 1 \leq u_i \leq S, && \forall i \in V \setminus \{0\} \\
& u_i - u_j + S x_{ij} \leq S - 1, && \forall i, j \in V \setminus \{0\} : i \neq j \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in V
\end{aligned}
$$

### 7.3.2 One-commodity flow baseret subtur eliminering for $m$-TSP

Vi kan naturligvis også formulere $m$-TSP vha. af de stærkere one-commodity flow baserede subtur elimineringsbegrænsninger, som det var tilfældet med TSP. Her er det særligt nemt, da den eneste ændring ligger i en tilpasning af (7.9), så det maksimalt er muligt at have $S$ kunder på en tur. Dette gøres naturligvis ved at stramme den øvre grænse på $f_{ij}$ til

$$f_{ij} \leq S x_{ij}, \forall i, j \in V$$

Dermed bliver one-commodity flow formulering af $m$-TSP som følger:

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} && \\
\text{s.t.:} \quad & \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, && \forall j \in V \setminus \{0\} \\
& \sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1, && \forall i \in V \setminus \{0\} \\
& \sum_{j \in V \setminus \{0\}} x_{0j} = m, && \\
& \sum_{i \in V \setminus \{0\}} x_{i0} = m. && \\
& \sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji} + 1, && \forall i \in V \setminus \{0\} \\
& f_{ij} \leq S x_{ij}, && \forall i, j \in V \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in V
\end{aligned}
$$

---

**Opgave 7.6:** Argumenter for, at $S$ skal overholde $S \geq \lceil \frac{n}{m} \rceil$ for $m$-TSP.

**Opgave 7.7:** Argumenter for, at begrænsningerne (7.14) kan styrkes til

$$u_i - u_j + S x_{ij} + (S-2) x_{ji} \leq S - 1, \quad \forall i, j \in V \setminus \{0\}$$

*Hint: Tjek alle brugbare kombinationer af værdier for $x_{ij}$ og $x_{ji}$ og husk at hvis $x_{ij} = 1 \Rightarrow x_{ji} = 0$ og vice versa.*

**Opgave 7.8:** Løs et $m$-TSP med punkterne givet i **Opgave 7.5** og med to køretøjer, som hver maksimalt kan servicere fem kunder. Brug både den MTZ baserede og den one-commodity flow baserede model.

---

## 7.4 Kapacitetsbegrænsninger: CVRP

I mange praktiske problemstillinger er det ikke antallet af kunder, som sætter en grænse for hvor mange kunder en chauffør kan nå at servicere. Derimod

er køretøjets kapacitet oftere en begrænsende faktor. I dette afsnit vil $m$-TSP blive udvidet til at inkludere kapacitetsbegrænsninger, hvorved det man kalder et "kapacitetsbegrænset ruteplanlægningsproblem" (CVRP) bliver formuleret. En gennemgående antagelse er, at hver kundes efterspørgsel kan måles i én fælles enhed som også måler køretøjets kapacitetsbegrænsning. Denne enhed kunne for eksempel være antal paller, antal lad-meter eller antal tons. Vi vil antage, at hver kunde $i \in V \setminus \{0\}$ har en efterspørgsel på $q_i > 0$ enheder. Af ren modellerings-bekvemmelighed, antager vi at depotet har en efterspørgsel på nul enheder, hvorved $q_0 = 0$. Samtidig antager vi, at der er $m \geq 1$ lastbiler til rådighed, der hver har en kapacitet på $Q_k > 0$ enheder. I det første delafsnit vil vi betragte den situation hvor flåden af køretøjer er homogen i den forstand, at alle biler er ens, således at de har den samme kapacitet. I det følgende afsnit vil vi kort introducere konceptet omkring heterogene flåder, men vi vil ikke etablere en fuld model for dette problem.

### 7.4.1 Det homogene CVRP

I tilfældet hvor alle køretøjer er ens og har den samme kapacitet, vil vi blot omtale kapaciteten som $Q$ (uden fodtegn), altså $Q_k := Q$ for alle $k = 1, \ldots, m$. Da køretøjerne er ens er det, som med $m$-TSP, ikke nødvendigt at holde styr på præcis hvilket køretøjer, der kører hvor. I stedet skal vi blot sørge for, at hver kunde bliver besøgt af præcis ét køretøj og at kapacitetsbegrænsningen bliver overholdt for hvert køretøj. Dermed kan et homogent CVRP modelleres meget lig $m$-TSP da

det eneste, der mangler, er, at kapacitetsbegrænsningerne overholdes:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{7.15}$$

$$\text{s.t.:} \sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1, \qquad \forall j \in V \setminus \{0\} \tag{7.16}$$

$$\sum_{\substack{j \in V \\ i \neq j}} x_{ij} = 1, \qquad \forall i \in V \setminus \{0\} \tag{7.17}$$

$$\sum_{j \in V \setminus \{0\}} x_{0j} = m, \tag{7.18}$$

$$\sum_{i \in V \setminus \{0\}} x_{i0} = m. \tag{7.19}$$

$$\text{Ingen subture} \tag{7.20}$$

$$\text{Kapaciteten skal respekteres} \tag{7.21}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in V \tag{7.22}$$

Ligesom med $m$-TSP har vi altså, at de samlede omkostninger skal minimeres, (7.15), under bibetingelse af, at hver kunde skal besøges og forlades præcis en gang hver, (7.16) og (7.17), at $m$ biler skal køre ud fra og hjem til depotet, (7.18) og (7.19), at der ikke må være subtur, (7.20), og at kapacitetsbegrænsningen på hvert køretøj skal overholdes, (7.21). Det viser sig, at begrænsningerne (7.20) og (7.21) kan slås sammen i den forstand, at vi kan formulere en række begrænsninger som både sørger for at udelukke muligheden for subture *og* sørger for at bilernes kapacitet overholdes. Endnu en gang kan modellerne deles op i en MTZ-baseret og en one-commodity flow baseret formulering inspireret af Gavish og Graves (1978).

### 7.4.2 En MTZ-basret formulering af kapacitetsbegrænsninger

I denne formulering skal vi introducere en variabel for hver kunde $i$, som angiver hvor meget lastbilen *højest* har leveret på sin tur fra depotet, frem til den forlader ved kunde $i$. Vi vil kalde denne variabel $v_i$, og den skal altså være en øvre grænse for, hvor meget bilen, der servicerer knude $i$, har leveret når den har serviceret kunde $i$. Vi kan så notere os, at lige som med MTZ formuleringen for $m$-TSP kan vi let opstille øvre og nedre grænser for hver $v_i$:

$$q_i \leq v_i \leq Q, \quad \forall i \in V \setminus \{0\} \tag{7.23}$$

Dertil kommer behovet for at sørge for, at $u_i$ variablerne, så at sige, akkumulerer den rigtige mængde leveret, hvis der rejses direkte fra $i$ til $j$. Det vil altså sige, at der skal opstille begrænsninger der tvinger $u_j$ til at stige med mindst $q_j$ enheder i forhold til $u_i$ hvis der køres direkte fra $i$ til $j$, altså skal der gælde

$$x_{ij} = 1 \Rightarrow v_i + q_j \leq v_j, \quad \forall i, j \in V \setminus \{0\} : i \neq j$$

Dette kan modelleres meget lig subtur-elimineringsbegrænsningerne, (7.14), for $m$-TSP problemet:

$$v_i - v_j + Qx_{ij} \leq Q - q_j, \quad \forall i, j \in V \setminus \{0\} : i \neq j \tag{7.24}$$

Det er nu op til læseren at tjekke de to tilfælde $x_{ij} = 0$ og $x_{ij} = 1$ og overbevise sig om, at disse begrænsninger er valide, fjerner subture og håndhæver bilernes kapacitet. Dermed haves, at hvis (7.20) og (7.21) erstattes af (7.23) og (7.24) opnås en korrekt formulering af det kapacitetsbegrænsede ruteplanlægnings-problem.

### 7.4.3 En one-commodity flow-basret formulering af kapacitetsbegrænsninger

Som med $m$-TSP kan CVRP også formuleres ved hjælp af en underliggende network flow struktur. Vi skal nu omdefinere flow variablerne således at $f_{ij}$ er defineret som følger: hvis $x_{ij} = 1$ så er $f_{ij}$ lig med mængden af varer leveret på turen som servicerer knude $i$ *når bilen forlader i*. Ellers er $f_{ij} = 0$. Dermed kan vi let udlede nedre og øvre grænser for $f_{ij}$-variabler. Først og fremmest haves at $f_{ij} \leq (Q - q_j)x_{ij}$. For at indse, at dette er korrekt skal man notere sig, at hvis $x_{ij} = 0$ haves at $f_{ij} = 0$ som ønsket. På den anden side, hvis $x_{ij} = 1$ haves at når bilen forlader knude $i$ (på vej mod knude $j$), har den ikke leveret mere end $Q - q_j$. Grunden til at vi kan trække $q_j$ fra er, at der skal være plads på bilen til at servicere knude $j$ efter knude $i$, da vi har at $x_{ij} = 1$.

Ligeledes udleder vi snildt en nedre grænse for $f_{ij}$ på følgende vis: hvis $x_{ij} = 0$ skal $f_{ij} = 0$ hvorfor den nedre grænse skal være 0. På den anden side, hvis $x_{ij} = 1$ ved vi, at der som minimum er leveret $q_i$ enheder når bilen forlader knude $i$ hvorfor $f_{ij} \geq q_i x_{ij}$ for alle $i, j \in V$ (Man skal notere sig, at dette virker *for alle* $i, j \in V$ fordi vi har antaget at $q_0 = 0$).

Når disse øvre og nedre grænser er på plads for $f_{ij}$-variablerne, kan vi sørge for, at akkumuleringen af varer leveret over en rute sker korrekt vha. følgende *flow conservation* begrænsninger:

$$\sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji} + q_i, \quad \forall i \in V \setminus \{0\} \tag{7.25}$$

Venstresiden i (7.25) måler hvor meget bilen som servicerer knude $i$ har leveret når den forlader knude $i$ ($\sum_{j \in V} f_{ij}$). Dette skal være lig med højresiden, der måler det, som bilen, der besøger knude $i$, har leveret *inden* den ankommer til knude $i$ ($\sum_{j \in V} f_{ji}$), tillagt det, som leveres ved knude $i$ ($q_i$). Dermed kan et CVRP også formuleres som (7.15)–(7.22) med (7.20) og (7.21) erstattet af

$$\begin{aligned} f_{ij} &\geq q_i x_{ij}, && \forall i, j \in V \\ f_{ij} &\leq (Q - q_j) x_{ij}, && \forall i, j \in V \\ \sum_{j \in V} f_{ij} &= \sum_{j \in V} f_{ji} + q_i, && \forall i \in V \setminus \{0\} \end{aligned} \tag{7.26}$$

---

**Opgave 7.9:** Formuler den matematiske sammenhængen mellem $v_i$-variabler i MTZ formuleringen af CVRP og $f_{ij}$ fra network flow formuleringen af CVRP.

**Opgave 7.10:** Lad $q^{min} = \min_{i \in V \setminus \{0\}} q_i$ være den mindste efterspørgsel blandt alle kunderne. Argumenter for, at følgende styrkelse af den nedre grænse på $f_{ij}$ variablerne i network flow formuleringen er valid:

$$f_{ij} \geq (q_i + q^{min}) x_{ij} - q^{min} x_{0i}, \quad \forall i, j \in V$$

**Opgave 7.11:** Lad $q^{min} = \min_{i \in V \setminus \{0\}} q_i$ være den mindste efterspørgsel blandt alle kunderne. Argumenter for, at følgende udvidede øvre grænser på $f_{ij}$ variablerne i network flow formuleringen er valide uligheder for formuleringen:

$$f_{ij} \leq (Q - q_j - q^{min}) x_{ij} + q^{min} x_{j0}, \quad \forall i, j \in V \tag{7.27}$$

**Opgave 7.12:** Betragt ulighederne givet i (7.27). Argumenter for, at disse ikke er tilstrækkelige til at håndhæve at $x_{ij} = 0 \Rightarrow f_{ij} = 0$.

---

### 7.4.4 Det heterogene CVRP

Det er naturligvis også muligt, at modellere den situation hvor hvert type fartøj, $k = 1, \ldots, K$, har forskellige omkostninger $c_{ij}^k$ og forskellige kapaciteter $Q^k$. Når omkostningerne og kapaciteterne afhænger af fartøjet, er det pludseligt nødvendigt at holde styr på, *hvilken type* fartøj der bevæger sig mellem de forskellige

efterspørgselspunkter. I sådanne tilfælde er det naturligt at udvide modellen med binære variabler med tre index, som defineres som følger

$$x_{ij}^{k} = \begin{cases} 1, & \text{Hvis et fartøj af type } k \text{ rejser direkte fra knude } i \text{ til knude } j \\ 0, & \text{ellers} \end{cases}$$

Det vil sige, at vi er gået fra en to-index formulering til en tre-index formulering, hvilket også betyder at modellen bliver meget større, da antallet af binære kant-variabler går fra $(n+1)^2$ til $(n+1)^2 \times K$ - altså, der kommer $K$ gange så mange variabler i modellen. Det er i praksis ikke muligt at løse særligt store instanser af det heterogene CVRP ved hjælp af en matematisk model uden at tilføje et lag af kompliceret algoritmik, som udnytter problemets struktur til at løse en række delproblemer. Derfor vil vi ikke beskæftige os yderligere med denne problemklasse i denne bog. For den interesserede læser henvises til Koç m.fl. (2016), hvor de sidste 30 års forskning inden for heterogene ruteplanlægningsproblemer gennemgås.

## 7.5 Yderligere udvidelser af ruteplanlægningsproblemer

I dette afsnit vil fokus være på udvidelser af CVRP og målet er at give læseren en indikation på hvorledes et standard problem fra litteraturen, her CVRP, kan udvides til at inkludere flere elementer fra virkeligheden.

### 7.5.1 Tidsvinduer

I mange praktiske problemstillinger inden for ruteplanlægning skal en vare leveres til en kunde inden for et på forhånd fastsat *tidsvindue*. Med tidsvindue menes en tidsperiode indenfor hvilken varen skal leveres. Det kunne eksempelvis være, at en kunde kræver at levering af varer foregår mellem klokken 8 og 10 om morgenen, da man først møder ind klokken 8 og skal bruge varerne i produktionen klokken 10. For at kunne snakke tidsvinduer, vil vi indføre et tidsperspektiv til CVRP modellen, som ellers før udelukkende beskæftigede sig med kvantiteter af en efterspurgt vare og omkostninger ved at traversere knuderne i en graf. Derfor vil vi introducere en "køretidsmatrix" $(t_{ij})_{i,j=1}^{n}$, som angiver hvor lang tid det tager at køre direkte fra knude $i \in V$ til knude $j \in V$. Vi vil naturligt antage, at $t_{ij} \geq 0$. Derudover, vil vi også benytte en parameter $s_i > 0$ for hver efterspørgselsknude som angiver denne kundes *servicetid*. Slutteligt vil vi introducere tidsvinduer for hver knude i grafen givet ved $[a_i, b_i]$, hvor $a_i$ er det tidligste tidspunkt vi kan starte service hos knude $i \in V$ og $b_i$ er det seneste tidspunkt vi kan starte service hos knude $i$. Ofte

sætter man $a_0 = 0$ og $b_0$ lig med den fulde arbejdsdags længde, hvilket vil sige, at man kan forlade depotet når tidsperioden begynder (man siger ofte "til tid 0") og man skal returnere til depotet inden arbejdsdagen er slut. Læg her mærke til, at vi definerer tidsvinduet som "det tidligste/senest tidspunkt service kan *starte*". Det vil sige, at hvis en kunde har et tidsvindue givet ved $[8.00, 10.00]$ og har en servicetid på 15 minutter, så er det fuldt i orden at starte service klokken 9.55 selvom servicen således ikke er færdig klokken 10.00. Hvis man ønsker at service skal være færdiggjort på et bestemt tidspunkt, skal tidsvinduet naturligvis justeres derefter. Man bør også notere sig, at det er tilladt at vente på, at et tidsvindue åbner i denne model. Så principielt må en chauffør gerne køre hen til en kunde og vente på, at denne kundes tidsvindue åbner, for så at påbegynde service. Vi vil i dette afsnit bygge videre på CVRP modellen givet ved (7.15)–(7.22) hvor kapacitetsbegrænsninger og subtur-eliminering kan formuleres, som man ønsker det.

For at modellere aspektet omkring tidsvinduer, vil vi introducere en ny variabel $\tau_i$ for hver knude $i \in V \setminus \{0\}$, der angiver det tidspunkt hvor service starter hos knude $i$. Da skal der naturligvis gælder at

$$a_i \leq \tau_i \leq b_i, \qquad \forall i \in V \setminus \{0\}$$

Endvidere, skal der for depotet gælde, at nå vi servicerer kunde $i$ og kører tilbage til depotet, så skal det være muligt at nå tilbage før arbejdsdagen slutter (inden tid $b_0$). Det kan modelleres som følger

$$\tau_i + (s_i + t_{i0})x_{i0} \leq b_0, \qquad \forall i \in V \setminus \{0\}$$

Det vil altså sige, at hvis en bil kører fra kunde $i$ tilbage til depotet ($x_{i0} = 1$) så skal det tidspunkt hvor service starter ved kunde $i$ tillagt servicetiden ved denne kunde plus tiden det tager at køre tilbage til depotet være mindre end det seneste en bil kan returnere til depotet. Hvis i stedet $x_{i0} = 0$ vides det blot at $\tau_i \leq b_0$ da alle servicetider og rejsetider er antaget ikke-negative.

Det sidste vi mangler at modellere er akkumulationen af tid over ruterne bilerne kører. Disse begrænsninger antager en kendt form som minder om MTZ-begrænsningerne som blev brugt til både at modellere subtur-eliminering og kapacitetsbegrænsninger. Det vi ønsker at modellere er følgende implikation

$$x_{ij} = 1 \Rightarrow \tau_i + s_i + t_{ij} \leq \tau_j, \quad \forall i, j \in V \setminus \{0\}$$

Beskrevet i ord vil det sige, at vi ønsker at modellere, at hvis der køres direkte fra kunde $i$ til kunde $j$ kan der tidligst startes service hos kunde $j$ efter service er

startet hos kunde $i$ ($\tau_i$) plus service tid hos kunde $i$ og rejsetid mellem knuderne $i$ og $j$. Vi kan modellere dette med en big-$M$ begrænsning på følgende vis

$$\tau_i + s_i + t_{ij} \leq \tau_j + M_{ij}(1 - x_{ij}), \quad \forall i, j \in V \setminus \{0\} \tag{7.28}$$

Her er $M_{ij} > 0$ en tilpas stor konstant, som er så lille som muligt. For at se, at logikken holder, kan vi betragte de to mulige værdier $x_{ij}$ kan tage:

$x_{ij} = 0$: I dette tilfælde reduceres ulighederne (7.28) til

$$\tau_i + s_i + t_{ij} \leq \tau_j + M_{ij}, \quad \forall i, j \in V \setminus \{0\}$$

Da $M_{ij}$ vælges tilpas stor, betyder det, at venstresiden aldrig bliver større end $\tau_j + M_{ij}$ hvorfor ulighederne bliver trivielt opfyldte for $x_{ij} = 0$ og derfor overflødige som ønsket.

$\mathbf{x_{ij}} = 1$: Når der køres direkte fra $i$ til $j$ reduceres ulighederne (7.28) til

$$\tau_i + s_i + t_{ij} \leq \tau_j, \quad \forall i, j \in V \setminus \{0\}$$

hvorved vi opnår den ønskede implikation $x_{ij} = 1 \Rightarrow \tau_i + s_i + t_{ij} \leq \tau_j$.

Det næste skridt er naturligt nok at bestemme en tilstrækkeligt stor, men så lille som mulig, værdi af $M_{ij}$. Da $M_{ij}$ slet ikke optræder i tilfældet hvor $x_{ij} = 1$ kan vi altså blot koncentrere indsatsen omkring tilfældet hvor $x_{ij} = 0$ hvor ulighederne (7.28) reducerede til

$$\begin{aligned} &\tau_i + s_i + t_{ij} \leq \tau_j + M_{ij} \\ &\Updownarrow \\ &\tau_i - \tau_j \leq M_{ij} - s_i - t_{ij} \end{aligned}$$

Her er uligheden, som følger efter bi-implikationen blot opnået ved at flytte rundt på de forskellige led, således at variabler står på venstresiden og parametre på højresiden. Vi skal nu sørge for, at $M_{ij}$ er stor nok til, at højresiden samlet set er så stor, at $\tau_i - \tau_j$ ikke bliver påvirket af ulighederne - eller at ulighederne bliver overflødige. Vi undersøger derfor hvilken værdi $\tau_i - \tau_j$ maksimalt kan antage og så sørger vi for, at $M_{ij} - s_i - t_{ij}$ er større end eller lig med denne maksimale værdi. Da $\tau_i$ og $\tau_j$ begge er ikke-negative antager $\tau_i - \tau_j$ sin største værdi når $\tau_i$ er størst mulig og $\tau_j$ er mindst mulig. Altså antages maksimumsværdien af $\tau_i - \tau_j$ når $\tau_i$ antager værdien af sin øvre grænse og $\tau_j$ antager værdien af sin nedre

grænse. Det vil sige sige, at vi ved, at $\tau_i - \tau_j \leq b_i - a_j$. Derfor ved vi, at $M_{ij}$ skal vælges således at $b_i - a_j \leq M_{ij} - s_i - t_{ij}$. Isoleres $M_{ij}$ nu på højresiden fås at for at $M_{ij}$ er tilstrækkeligt stor skal

$$b_i - a_j + s_i + t_{ij} \leq M_{ij}, \quad \forall i, j \in V \setminus \{0\}$$

Dermed er $M_{ij} = b_i - a_j + s_i + t_{ij}$ netop tilstrækkeligt stor og så lille som muligt, for at logikken i (7.28) opretholdes.

Med denne nye viden om størrelsen på $M_{ij}$ kan vi opskrive ulighederne som leder til den korrekte akkumulering af tid over ruter

$$\begin{aligned}
&\tau_i + s_i + t_{ij} \leq \tau_j + M_{ij}(1 - x_{ij}), \quad \forall i, j \in V \setminus \{0\} \\
&\Updownarrow \text{(Gang } M_{ij} \text{ ind i parentesen)} \\
&\tau_i + s_i + t_{ij} \leq \tau_j + M_{ij} - M_{ij}x_{ij}, \quad \forall i, j \in V \setminus \{0\} \\
&\Updownarrow \text{(Flyt led med variabler til venstresiden)} \\
&\tau_i - \tau_j + s_i + t_{ij} + M_{ij}x_{ij} \leq M_{ij}, \quad \forall i\, j \in V \setminus \{0\} \\
&\Updownarrow \text{(Indsæt definition af } M_{ij}\text{)} \\
&\tau_i - \tau_j + s_i + t_{ij} + (b_i - a_j + s_i + t_{ij})x_{ij} \leq b_i - a_j + s_i + t_{ij}, \quad \forall i, j \in V \setminus \{0\} \\
&\Updownarrow \text{(Lad } s_i \text{ og } t_{ij} \text{ gå ud med hinanden)} \\
&\tau_i - \tau_j + (b_i - a_j + s_i + t_{ij})x_{ij} \leq b_i - a_j, \quad \forall i, j \in V \setminus \{0\}
\end{aligned}$$

Den sidste lille detalje der mangler er, at sørge for, at hvis en kunde $i$ er den første på en rute ($x_{0i} = 1$), så kan dennes starttid ikke bliver mindre end den tid det tager at køre fra depotet til kunde $i$. Det vil sige, at vi for alle kunder $i \in V \setminus \{0\}$ har

$$\tau_i \geq t_{0i}x_{0i}.$$

Med disse uligheder på plads, kan vi opskrive CVRP med tidsvinduer (ofte forkortet

TW-CVRP eller CVRP-TW) som følgende MILP

$$
\begin{aligned}
\min\ & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\text{s.t.: } & (7.16) - (7.21) \\
& a_i \leq \tau_i \leq b_i, && \forall i \in V \setminus \{0\} \\
& \tau_i + (s_i + t_{i0}) x_{i0} \leq b_0, && \forall i \in V \setminus \{0\} \\
& \tau_i - \tau_j + (b_i - a_j + s_i + t_{ij}) x_{ij} \leq b_i - a_j, && \forall i, j \in V \setminus \{0\} \\
& \tau_i \geq t_{0i} x_{0i}, && \forall i \in V \setminus \{0\} \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in V
\end{aligned}
$$

Da der i dette problem er indført tidsvinduer er der mulighed for at lave en forholdsvis aggressiv reduktion i antallet af $x_{ij}$-variabler, som kan bruges i en brugbar løsning og dermed hjælpe solveren med at fjerne variabler, før man kalder den. For at se én mulig reduktionsregel læg da mærke til, at hvis vi starter service hos kunde $i$ så tidligt som muligt og stadig ikke kan nå at servicere kunde $i$, køre til kunde $j$ og starte service der inde tidsvinduet lukker, kan vi effektivt fjerne muligheden for at køre direkte fra $i$ til $j$. Mere formelt, lad $i \in V \setminus \{0\}$ og $j \in V$ være to knuder hvor $i \neq j$, da haves følgende implikation

$$a_i + s_i + t_{ij} > b_j \ \Rightarrow\ x_{ij} = 0$$

På den måde kan man gennemløbe alle par af knuder i mængden af kunder og tjekke om betingelsen til venstre for implikationen og såfremt det er tilfældet kan variablen $x_{ij}$ sættes til at tage værdien 0 i alle brugbare løsninger.

### 7.5.2 Prize collection

I dette afsnit tager vi igen udgangspunkt i formuleringen af CVRP givet i (7.15)–(7.22) og udvider denne med muligheden for "prize collection". Indtil nu har vi antages at *alle* kunder skulle besøges af præcis et køretøj, men ofte vil man have muligheden for at fravælge at servicere nogle kunder såfremt det ikke er rentabelt. Derfor vil vi her indføre muligheden for at vælge kunder til og fra baseret på om det er rentabelt at servicere dem eller ej. For at kunne tale om retabilitet vil vi, for hver kunde $i \in V \setminus \{0\}$ indføre en "prize" $p_i > 0$ som opnås såfremt en kunde serviceres og ellers opnås ingen indtægt fra denne kunde. For at kunne vælge kunder til og fra, indføres en ny binær variabel for hver kunde $i \in V \setminus \{0\}$

defineret ved

$$y_i = \begin{cases} 1, & \text{hvis kunde } i \text{ serviceres} \\ 0, & \text{ellers} \end{cases}$$

Med de nye parametre og variabler kan vi opstille en objektfunktion som maksimerer profitten opnået ved ruteplanen givet ved

$$\max \sum_{i \in V \setminus \{0\}} p_i y_i - \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

Denne objektfunktion maksimerer den opnåede indtægt minus de omkostninger der er forbundet med at servicere kunderne.

Vi bliver også nødt til at ændre begrænsningerne (7.16) og (7.17) da vi nu ikke nødvendigvis behøver at have at en bil køre til og fra hver kunde. Dette skal kun gælde for de kunder, som bliver valgt til at blive serviceret. Derfor opnås nye begrænsninger givet ved

$$\sum_{i \in V} x_{ij} = y_j, \quad \forall j \in V \setminus \{0\}$$
$$\sum_{j \in V} x_{ij} = y_i, \quad \forall i \in V \setminus \{0\}$$

hvor den første mængde af begrænsninger sørger for, at hvis kunde $j$ vælges ($y_j = 1$) så skal der ankomme præcis en bil til kunde $j$ og den anden mængde af begrænsninger sørger for, at hvis kunde $i$ vælges, så skal der og være præcis én bil der forlader kunde $i$.

I og med, at vi nu potentielt har for mange biler til at det giver mening at bruge dem alle (da vi potentielt vælger at servicere en lille delmængde af kunderne), er det naturligt at ændre begrænsningerne som sørger for, at der bliver sendt præcis $m$ biler ud og ind ad depotet (begrænsningerne (7.18) og (7.19)). Disse bør nu i stedet sørge for, at der bliver sendt *maksimalt* $m$ biler ind og ud af depotet:

$$\sum_{j \in V \setminus \{0\}} x_{0j} \leq m,$$
$$\sum_{i \in V \setminus \{0\}} x_{i0} \leq m,$$

Med disse ændringer kan et *prize collecting CVRP* opskrives som følgende MILP

$$
\begin{aligned}
\max \quad & \sum_{i \in V \setminus \{0\}} p_i y_i - \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{i \in V} x_{ij} = y_j, && \forall j \in V \setminus \{0\} \\
& \sum_{j \in V} x_{ij} = y_i, && \forall i \in V \setminus \{0\} \\
& \sum_{j \in V \setminus \{0\}} x_{0j} \leq m, \\
& \sum_{i \in V \setminus \{0\}} x_{i0} \leq m, \\
& \text{Ingen subture} \\
& \text{Kapaciteten skal respekteres} \\
& y_i \in \{0, 1\}, && \forall i \in V \setminus \{0\} \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in V
\end{aligned}
$$

Subtur elimineringsbegrænsningerne og kapacitetsbegrænsningerne i denne prize-collecting version skal man være lidt forsigtige med. Man kan bruge de MTZ baserede begrænsninger direkte som de er, men ønsker man at benytte en one-commodity flow formulering som den angivet i (7.26) skal disse modificeres som følger

$$\sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji} + q_i y_i, \quad \forall i \in V \setminus \{0\}$$

Dette skyldes, at der ikke skal akkumuleres efterspørgsel for kunder, som ikke skal serviceres ($y_i = 0$).

---

**Opgave 7.13:** Omskriv Prize collecting CVRP til kun at bruge $x_{ij}$ variabler. Det vil sige, opskriv problemet uden brug af $y_i$ variabler. *Hint: En mængde af de angivne begrænsninger definerer $y_i$ variablernes værdi direkte ud fra $x_{ij}$ variablernes værdi.*

**Opgave 7.14:** Der tales i mange praktiske applikationer af ruteplanlægning om *rute balancering*. Det vil sige, at man vil sprede arbejdsbyrden nogenlunde ligeligt ud over alle ruter. I denne opgave vil arbejdsbyrden defineres som den tid en rute tager at gennemføre. Målet er altså at sørge for, at alle ruter tager nogenlunde lige

lang tid og én måde at gøre dette på er minimere tiden det tager at gennemføre den rute der tager længst tid. Lad $T$ være en variabel, som angiver tiden det tager at gennemføre den rute der tager længst tid. Benyt formuleringen af CVRP med tidsvinduer samt variablen $T$ til at minimere tiden, det tager at gennemføre den længste rute.

**Opgave 7.15:** I forbindelse med formuleringen af CVRP med tidsvinduer var det en antagelse, at hver kunde *skulle* serviceres inden for tidsvinduerne. I realiteten er det ofte *muligt* at ankomme for sent ($\tau_i > b_i$) men ofte er dette forbundet med en form for bødestraf, som skal betales til kunden. Dette kan modelleres ved at tilføje en binære variabel $z_i$, som tager værdien 1 hvis og kun hvis en bil ankommer for sent hos kunde $i \in V \setminus \{0\}$. Der skal yderligere indføres en straf $\beta_i$ for at komme for sent og en tilstrækkeligt stor, men så lille som mulig, parameter $M$. Da kan objektfunktionen ændrestil

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + \sum_{i \in V \setminus \{0\}} \beta_i z_i$$

og tidsvinduets afslutning kan ændres til

$$\tau_i \leq b_i + M z_i, \quad \forall i \in V \setminus \{0\}$$

Der er dog ét problem med denne formulering: bøden $\beta_i$, der betales for at komme for sent, afhænger ikke af *hvor meget* man kommer for sent. Det vil sige, at det er lige slemt at komme 5 minutter og 2 timer for sent. Foreslå derfor en formulering, hvor det koster $\beta_i^1$ at komme mellem 1 og 30 minutter for sent, $\beta_i^2$ at komme mellem 31 og 45 minutter for sent og $\beta_i^3$ at komme mere end 45 minutter for sent. Det kan antages at $0 < \beta_i^1 < \beta_i^2 < \beta_i^3$.

**Opgave 7.16:** Antag samme situation som i **Opgave 7.15**, men antag i stedet, at det koster $\beta_i$ per tidsenhed man kommer for sent. Modeller denne situation.

## 7.6 En "unified" tilgang til ruteplanlægning

I mange tilfælde kan det være overordentligt svært at løse ruteplanlægningsproblemer af de typer som er listet ovenfor. Dette skyldes i store træk, at for at opnå en *tight* formulering med en stærk nedre grænse, kræves *eksponentielt* mange begrænsninger. En måde man kan overkomme dette problem på er ved at benytte

en model som har eksponentielt mange variabler i stedet. I det følgende vil vi udvikle en sådan en model, som i princippet indeholder alle de ovenfor beskrevne modeller i én model.

### 7.6.1 En rute-udvælgningsmodel

I det følgende vil vi arbejde med antagelsen om, at vi er blevet givet et ruteplanlægningsproblem og at vi skal forsøge at løse dette. Vi vil ikke lave andre antagelser i dette afsnit, end at alle kunder skal besøges præcis én gang.

For at udvikle denne model vil vi antage, at vi *a priori* har genereret *alle* brugbare ruter. Det vil i de problemstiller vi har betragtet ovenfor betyde, at vi har genereret alle ruter, som starter og slutter i depotet, besøger hver kunde maksimalt én gang og som derudover respekterer eventuelle tidsvinduer og kapacitetsbegrænsninger. Vi vil lade $\mathcal{R}$ være mængden af alle brugbare ruter til vores ruteplanlægningsproblem og vi vil lade en *specifik* rute være givet ved $r \in \mathcal{R}$. Mængden $\mathcal{R}$ er naturligvis enorm og kan indeholde millioner af ruter (hvoraf mange af dem er meget dårlige!). Ydermere, nu da vi har antaget, at vi har en fuldkommen liste af alle brugbare ruter, vil vi også antage, at vi kender hver rutes længde/omkostning. Lad dermed $c_r$ være rute $r$'s omkostning.

Vi kan endvidere betragte hver rute, $r \in \mathcal{R}$, og bestemme om kunde $i \in V \backslash \{0\}$ besøges på ruten $r$ eller ej. Vi vil for at holde styr på hvilke kunder der bliver besøgt på hvilke ruter benytte en binær *parameter* $a_{ir}$ givet ved

$$a_{ir} = \begin{cases} 1, & \text{hvis kune } i \text{ besøges på rute } r \\ 0, & \text{ellers} \end{cases}$$

Med $\mathcal{R}$, $c_r$ og $a_{ir}$ defineret, har vi alt det data, der skal bruges til denne model. Det næste, der skal til, er en definition af variabler. I denne model skal vi udvælge hvilke ruter fra mængden $\mathcal{R}$ af alle ruter vi vil benytte i en endelig løsning. Derfor benytter vi en mængde af binære variabler definet som følger

$$x_r = \begin{cases} 1, & \text{hvis rute } r \text{ vælges til den endelige løsning} \\ 0, & \text{ellers} \end{cases}$$

Dermed har vi én variabel for hver brugbar rute til vores problem og dermed potentielt rigtig, rigtig mange variabler. Det er relativt let, at modellere en objektfunktion baseret på det data og de variabler vi har til rådighed, da vi blot vil

minimerer summen af omkostningerne forbundet med de udvalgte ruter:

$$\min \sum_{r \in \mathcal{R}} c_r x_r$$

Det næste vi skal sørge for er, at hver kunde bliver besøgt på præcis én rute. Dette gøres ved, for hver kunde $i \in V \setminus \{0\}$, at summe over alle de ruter, som besøger kunde $i$ og så kræve at denne sum er lig med 1. Det vil sige, at vi skal have følgende mængde af begrænsninger

$$\sum_{r \in \mathcal{R}} a_{ir} x_r = 1, \quad \forall i \in V \setminus \{0\}$$

Hvis der yderligere er en grænse for hvor mange ruter/biler der kan benyttes i en løsning, kan vi tilføje en kardinalitetsbegrænsing, som udfylder samme rolle som (7.12) og (7.11), givet ved

$$\sum_{r \in \mathcal{R}} x_r \leq m.$$

Her er $m$ det maksimale antal ruter man kan tillade i en løsning. Dermed opnås en særdeles kompakt model givet ved

$$\min \sum_{r \in \mathcal{R}} c_r x_r \tag{7.29}$$

$$\text{s.t.:} \sum_{r \in \mathcal{R}} a_{ir} x_r = 1, \qquad \forall i \in V \setminus \{0\}$$

$$\sum_{r \in \mathcal{R}} x_r \leq m$$

$$x_r \in \{0, 1\}, \qquad \forall r \in \mathcal{R} \tag{7.30}$$

### 7.6.2 Generering af mængden $\mathcal{R}$

Det er desværre ikke muligt at generere hele mængden $\mathcal{R}$, da der kan være et astronomisk stort antal brugbare ruter til et ruteplanlægningsproblem. Desuden er der en stor del af mængden af brugbare løsninger, som på ingen måde skal indgå i en optimal ruteplan, da de potentielt er meget lange uden at servicerer ret mange kunder eller kun besøger et lille antal kunder. Derfor er det nødvendigt, og smart, kun at generere en lille delmængden af de brugbare ruter, som er af høj kvalitet og så vælge mellem disse. Der er mange måder at generere ruter til en formulering, som blev præsenteret i forrige afsnit og vi vil her nævne to som er særligt populære:

1. Man kan bruge en heuristik, som man har en forventning om er i stand til at producere ruter af høj kvalitet, til at generere et stort antal ruter (potentielt tusindvis). Man vil ofte bruge en heuristik, som har en grad af *stokastik* indbygget i sin søgestrategi sådan, at man kan kalde den flere gang og få forskellige løsninger ud på den anden side. På den måde kan man genererer et stort antal ruter og så bruge formuleringen (7.29)–(7.30) til at "rydde op" i de genererede ruter ved at udvælge en bedste kombination af de genererede ruter.

2. En anden metode, som har fået navnet *søjle generering* starter med en meget lille delmængde af ruterne, $\hat{\mathcal{R}}$, og så løses den lineære relaksering af programmet

$$\begin{aligned}
\min \quad & \sum_{r \in \hat{\mathcal{R}}} c_r x_r \\
\text{s.t.:} \quad & \sum_{r \in \hat{\mathcal{R}}} a_{ir} x_r = 1, && \forall i \in V \setminus \{0\} && (7.31) \\
& \sum_{r \in \hat{\mathcal{R}}} x_r \leq m && && (7.32) \\
& x_r \in \{0, 1\}, && \forall r \in \hat{\mathcal{R}}
\end{aligned}$$

   Dette program kaldes et *restricted master program* (RMP), da vi restringerer os til kun at betragte en delmængde af ruterne.

   Da vi har løst den lineære relaksering af RMP kan vi bede en solver (CPLEX eller den solver man nu bruger til at løse lineære programmer) om at returnere *skyggepriserne* for begrænsningerne (7.31) og (7.32) hvorved man i prickippet kan udregne de reducerede omkostninger for alle de variablerne som *ikke* er i $\hat{\mathcal{R}}$. Vi kan betragte de variabler som ikke er i $\hat{\mathcal{R}}$ som om de var der, men at de blot antager værdien 0. Dermed kan vi fortolke den reducerede omkostning for en variable som "ændringen i objektfunktionen hvis en variabel som antager værdien 0 i den optimale løsning, tvinges til at antage værdien 1". Dermed vil variabler, som har en negativ reduceret omkostning, være variabler, som er attraktive at inkludere i RMP. Eller mere specifikt for denne problemstilling vil det sige, at hvis en variabel $x_r$ har en negativ reduceret omkostning, så er det attraktivt at tilføje ruten $r$ til mængden $\hat{\mathcal{R}}$.

   Det er dog stadig problematisk, at der potentielt er så mange ruter i $\mathcal{R}$ at vi ikke kan forvente at beregne de reducerede omkostninger for alle

$x_r$ hvor $r \in \mathcal{R} \setminus \hat{\mathcal{R}}$ ekplicit. Derfor må man gøre dette *implicit* ved at løse et optimeringsproblem som finder en variabel med mindste reducerede omkostnig.

Når en eller flere ruter med negative reducerede omkostninger er identificeret, skal disse tilføjes til $\hat{\mathcal{R}}$ og den lineære relaksering af RMP skal så genløses. Når der ikke kan findes flere ruter med negative reducerede omkostninger, er den lineære relaksering af RMP løst til beviselig optimalitet og dermed er den fulde lineære relaksering

$$\begin{aligned}
\min \quad & \sum_{r \in \mathcal{R}} c_r x_r & \\
\text{s.t.:} \quad & \sum_{r \in \mathcal{R}} a_{ir} x_r = 1, & \forall i \in V \setminus \{0\} \\
& \sum_{r \in \mathcal{R}} x_r \leq m & \\
& 0 \leq x_r \leq 1, & \forall r \in \mathcal{R}
\end{aligned}$$

også løst til optimalitet. Dog skal man være opmærksom på, at blot fordi den lineære relaksering er løst til optimalitet, betyder det ikke, at man dermed er sikker på, at man har genereret en mængde $\hat{\mathcal{R}}$, som indeholder alle ruterne som vil indgå i en optimal løsning til ruteplanlægningsproblemet. Men, man vil meget ofte have genereret en mængde $\hat{\mathcal{R}}$ hvor løsningen til

$$\begin{aligned}
\min \quad & \sum_{r \in \hat{\mathcal{R}}} c_r x_r & \\
\text{s.t.:} \quad & \sum_{r \in \hat{\mathcal{R}}} a_{ir} x_r = 1, & \forall i \in V \setminus \{0\} \\
& \sum_{r \in \hat{\mathcal{R}}} x_r \leq m & \\
& x_r \in \{0, 1\}, & \forall r \in \hat{\mathcal{R}}
\end{aligned}$$

har en rimelig kvalitet. Sommetider er man dog i den ærgerlige situation, at det ovenstående program slet ikke har en brugbar løsning, hvorfor man i sådanne tilfælde må ty til andre metoder, som hjælper med at generere yderligere ruter.

# 8 Verifikation, validering og repræsentation af modeller og løsninger

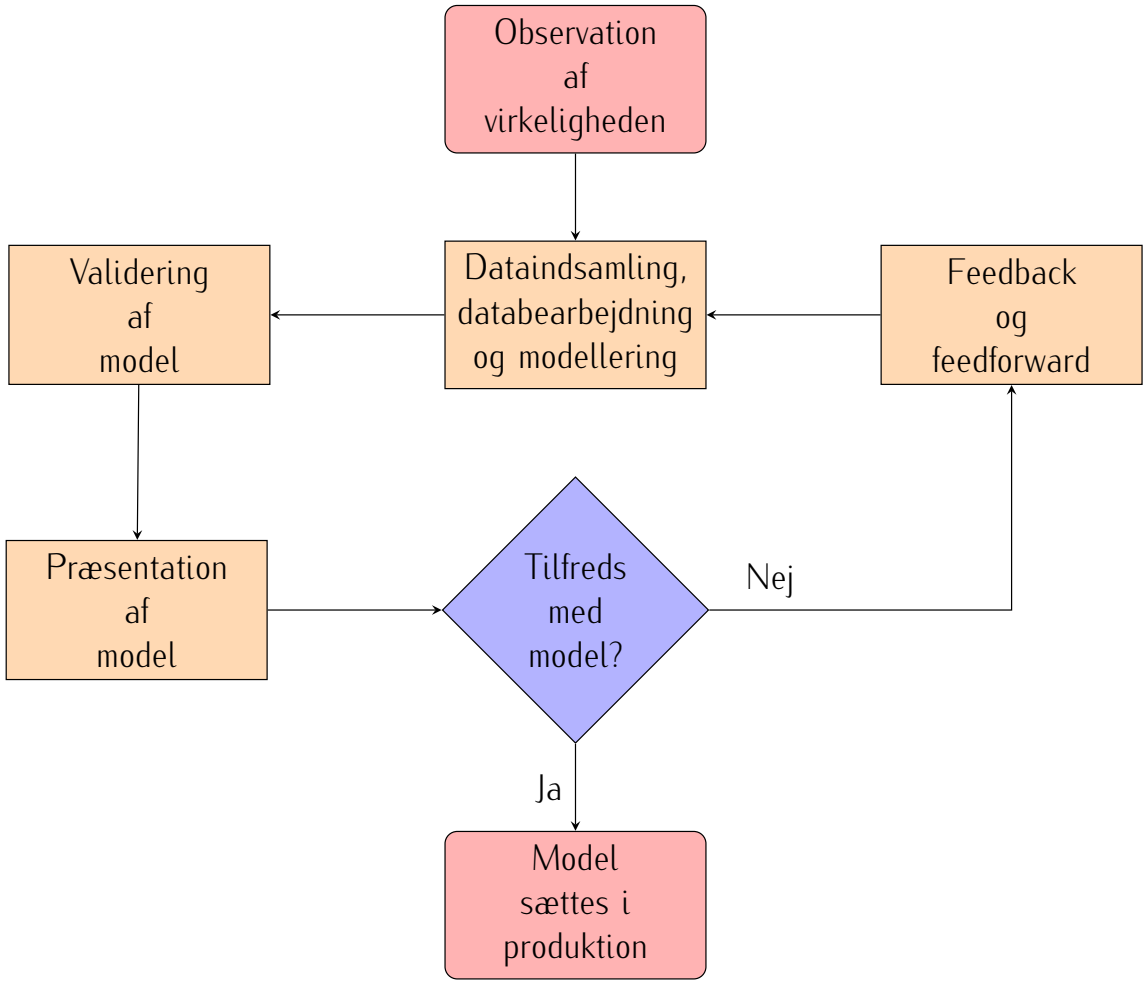
I dette kapitel vil fokus være på validering og verificering af modeller inden for prescriptive analytics. Kapitlet er delvist baseret på Kapitel 6.1 i Williams (2013) (Williams (2013) er for øvrigt en rigtig fin indføring i matematisk modellering, skulle man have lyst til mere end hvad der præsenteres i dette dokument).

## 8.1 Hvorfor verificere, validere, og repræsentere modeller og løsninger

Når man har udviklet en matematisk model, er det ekstremt vigtig at være forsigtig med at stole blindt på outputtet fra sådan en model – i hvert fald ind til modellen og de resulterende løsninger er validere og verificerede. Når en problemstilling er blevet modelleret, er ønsket for de fleste at løse modellen og på den måde opnå indsigt i hvorledes man skal handle, baseret på det inputdata, man har bygget modellen over. Efter en kortere eller længere implementeringsfase opnås en model som *syntaktisk* er korrekt i den forstand, at det software der benyttes til at løse den, er tilfreds med inputtet (dette vil eksempelvis være når CPLEX Studio ikke melder flere fejl). Efter denne fase hvor man har opnået en syntaktisk korrekt model, kan der således opstå flere muligheder:

1. Modellen modellerer i virkeligheden ikke det rigtige problem (fx. nogle begrænsninger mangler).
2. Modellen producerer brugbare løsninger, men de er ikke optimale i forhold til hvad beslutningstager ønsker (potentielt en forkert objektfunktion, eller nogle begrænsninger mangler).

Det vil oftest være sådan, at den første prototype af en model man afleverer til en beslutningstager, ikke stemmer fuldstændig overens med det, som beslutningstageren ønsker sig, hvorfor modellering er en iterativ proces som følger det




Figur 8.1: Illustration af modelleringscyklen.

man vil kalde en modelleringscyklus illustreret i Figur 8.1. Denne illustration er naturligvis ikke en skabelon for hvorledes alle modelleringscykler ser ud, men er dog relativt repræsentativ. Det første man gør i forbindelse med udviklingen af modeller, er at observere det "virkelige" system således, at man opnår en forståelse af den/de problemstillinger beslutningstageren står overfor. Uden denne "hands on" forståelse og fornemmelse af problemstillinger, vil de første mange iterationer af modelleringscyklen være spildt, fordi man som analytiker simpelthen ikke forstår det modellerede system i tilstrækkeligt detaljeret grad.

Når denne første introduktion til det system som ønskes modelleret er overstået, entrerer man selve modelleringscyklen. Det første, der skal ske, er, at man indsamler hvad man mener er den nødvendige data. Denne data bearbejdes således, at kvaliteten er høj nok til, at man vil basere beslutninger på denne bearbejdede data. Ofte er det ikke nok, at indsamle og oprense data, man bliver i de fleste tilfælde nødt til at udarbejde prædiktive forcasting-modeller til at fremskrive data.

Der findes en stor akademisk litteratur omkring forecasting, og det kan ikke understreges nok, hvor vigtigt dette område er for kvaliteten af de løsninger de udviklede ordinerende modeller leverer. Lidt populært har udtrykket "garbage in, garbage out" været brugt for at understrege, at hvis inputdata er af dårlig kvalitet, vil de beregnede løsninger også være det. Emnet omkring forecasting ligger dog uden for formålet med indeværende bo,g og kvaliteten af data vil, som sådan, ikke blive betvivlet i dette kursus.

Efter dataet er bearbejdet og forecasting modellerne er validerede, kan man udvikle sine matematiske optimeringsmodeller. Når man mener, at en model er "færdig", bør modellen og dens output valideres, hvorefter modellen og de beregnede løsninger skal præsenteres for beslutningstager. Hvis beslutningstageren er tilfreds med model og de løsninger modellen producerer, sættes modellen i produktion. Hvis ikke, identificeres, sammen med beslutningstager, hvad der er galt med den nuværende model, det diskuteres hvorledes problemerne kan udbedres, og der lægges en plan for hvorledes modellen opdateres. Herefter er det tilbage til dataindsamling, databearbejdning og modellering, for at få en mere retvisende model af virkeligheden. I dette kapitel vil fokus være på validering af modellen og dennes output.

I valideringsfasen kan tre følgende tilfælde optræde

1. Modellen har ikke en brugbar løsning (*infeasible*)
2. Modellen har en $\pm$ uendelig objektfunktionsværdi (*unbounded*)
3. Modellen kan løses (*solvable*)

I det følgende skal det undersøges hvorledes man kan validere sin model, i tilfælde af at en af ovenstående situationer opstår.

## 8.2 Modellen har ingen brugbar løsning

Et matematisk optimeringsproblem har ingen brugbare løsninger, hvis modellens begrænsninger er selvmodsigende. For at være en anelse mere præcis, vil vi definere et matematisk optimeringsproblem til at være *infeasible* som følger

**Definition 8.1** (Infeasible)**.** Lad i det følgende $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$ hvor $i = 1 \ldots, m_1$ og $h_i : \mathbb{R}^n \to \mathbb{R}$ hvor $i = 1, \ldots, m_2$ være funktioner, og lad $J^{int}$

være en delmængde af $\{1, \ldots, n\}$. Lad et matematisk program være givet ved

$$\begin{aligned} \max\ & f(x) \\ \text{s.t.:}\ & g_i(x) \leq b_i, \forall i = 1, \ldots, m_1 \\ & h_i(x) = d_i, \forall i = 1, \ldots, m_2 \\ & x_j \in \mathbb{N}_0, \forall j \in J^{int} \end{aligned} \tag{8.1}$$

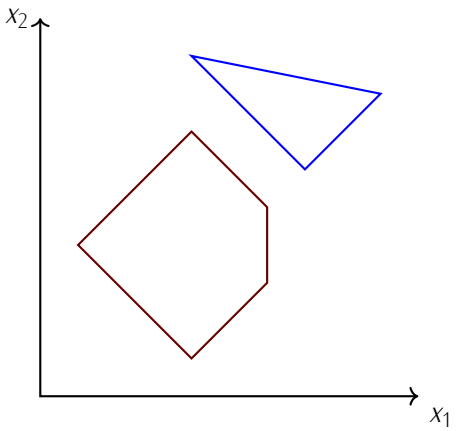
Da siges, at det matematiske optimeringsproblem (8.1) er *infeasible*, hvis der *ikke* eksisterer en vektor $\hat{x} \in \mathbb{R}^n$, som opfylder alle begrænsningerne til problemet. Det vil sige, at der *ikke* eksisterer en vektor $\hat{x} \in \mathbb{R}^n$ således at

$$\begin{aligned} & g_i(\hat{x}) \leq b_i, \forall i = 1, \ldots, m_1 \\ & h_i(\hat{x}) = d_i, \forall i = 1, \ldots, m_2 \\ & \hat{x}_j \in \mathbb{N}_0, \forall j \in J^{int} \end{aligned}$$

Det er velkendt, at de brugbare løsninger til et lineært program definerer et polyeder. I tilfældet hvor en blandet heltalsmodel ingen løsning har, kan begrænsningerne deles op i (mindst) to mængder, som hver danner en polyeder på en sådan måde at disse to polyeder ikke har noget overlap. Tag for eksempel følgende lille lineære program

$$\begin{aligned} \max\ & 2x_1 + 2x_2 \\ \text{s.t.:}\ & 2x_1 + 2x_2 \geq 5, \\ & 2x_1 - 2x_2 \leq 3, \\ & x_1 \leq 3, \\ & 2x_1 + 2x_2 \leq 11, \\ & -2x_1 + 2x_2 \leq 3, \\ & 2x_1 - 2x_2 \geq 1, \\ & 2x_1 + 10x_2 \leq 49 \\ & 2x_1 + 2x_2 \geq 13 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Det brugbare område for dette program kan illustreres som i Figure 8.2 på side 153, hvor de begrænsninger, som ovenfor er rødbrune definere den rødbrune polytop mens de blå begrænsninger definerer den blå polytop. Som det tydeligt ses, har de to polytoper intet overlap, og da vi ved at alle begrænsninger til et lineært program (og et lineært blandet heltalsprogram) alle skal være opfyldt




Figur 8.2: Illustration af et lineært program uden brugbare løsninger.

på samme tid, er det tydeligt, at programmet ikke har nogen brugbare løsninger overhovedet. Det er endvidere relativt let at gennemskue hvad problemet er: Der er to begrænsninger som åbenlyst modsiger hinanden nemlig

$$2x_1 + 2x_2 \leq 11,$$
$$2x_1 + 2x_2 \geq 13$$

Funktionen på venstresiden, kald den $g(x_1, x_2)$, i disse to begrænsninger er den samme. Dog kræver begrænsningerne at $g(x_1, x_2)$ skal være mindre end 11 og *samtidigt* større end 13. Dette kan ikke lade sig gøre. Heldigvis var det nemt at se hvad problemet var her, men i de fleste praktiske tilfælde er sådan en selvmodsigelse noget sværere at identificere.

I langt de fleste tilfælde indikerer en infeasible model, at man har lavet en fejl i sin matematisk model. En mulighed er jo selvfølgelig, at man ønsker at finde en bedste løsning til et problem som simpelthen ikke har en løsning – dette er dog sjældent tilfældet. Ofte modellerer vi problemer, hvor vi *ved*, at der er en brugbar løsning. Det første tjek man bør udføre, er at udelukke, at man blot har lavet en "typo" i sin implementering af modellen, da dette ofte kan rede en infeasible model.

Når typos er udelukket, kan man forsøge at løse sin model. Mange software pakker vil i tilfælde af infeasiblity returnere en løsning, som ikke er brugbare. Vi kan med sådan en ikke-brugbar løsning i hånden gennemgå begrænsningerne, og tjekke hvilke der ikke er overholdt. Det kan ske, at man på den måde kan identificere begrænsninger som er unødigt begrænsende eller simpelthen forkerte. Desværre

vil man dog ofte have, at infeasibility er en anelse mere subtilt at identificere, end blot at se på brudte begrænsninger. Betragt følgende lille eksempel med tre variabler

$$\begin{aligned} x_1 - x_2 &\geq 1 \\ x_2 - x_3 &\geq 1 \\ -x_1 + x_3 &\geq 1 \end{aligned}$$

Hvis en solver returnerer en (infeasible) løsning til ovenstående ligningssystem, som ikke er brugbar, kan man ikke blot sige, at den/de begrænsninger, som ikke er opfyldt er forkert(e). Det er nemlig ikke muligt at opfylde alle ovenstående tre uligheder på samme tid, men man kan opfylde alle kombinationer af de tre, som består af to uligheder. Det vil altså sige, at der *ikke* er én begrænsning, som er *forkert*. Det er den samlede inkompatibilitet der er problemet. Altså, noget mere dybtliggende i selve modellen.

Under antagelse af, at modellen beskriver et system, som man ved har en brugbar løsning, er det ofte muligt at konstruere en sådan (ofte ikke optimal) løsning. Tag for eksempel en situation hvor man modellerer et allerede eksisterende system, da kan man blot benytte den nuværende plan som en mulig brugbare løsning til problemet. Med en sådan beviseligt brugbar løsning for det problem man forsøger at modellere, kan vi substituere denne kendte løsning ind i modellen og på den måde identificere begrænsninger, som er brudte. Da det vides, at den givne løsning er brugbar, ved vi også, at alle de brudte begrænsninger må være modelleret forkert i forholde til den problemstilling vi er ude for at modellere. Disse begrænsninger er tydeligvis for restriktive og skal derfor genovervejes.

## 8.3 Modellen er ubegrænset

Et matematisk optimeringsproblem siges at være ubegrænset, hvis objektfunktionen kan øges uden at ramme en øvre grænse når der maksimerers, eller mindskes uden at nå en nedre grænse når der minimeres. Mere formelt kan ubegrænsethed defineres som følger:

**Definition 8.2** (Ubegrænset)**.** Lad i det følgende $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$ hvor $i = 1 \ldots, m_1$ og $h_i : \mathbb{R}^n \to \mathbb{R}$ hvor $i = 1, \ldots, m_2$ være funktioner, og lad $J^{int}$

være en delmængde af $\{1, \ldots, n\}$. Lad et matematisk program være givet ved

$$\begin{aligned} \max\ & f(x) \\ \text{s.t.:}\ & g_i(x) \leq b_i, \forall i = 1, \ldots, m_1 \\ & h_i(x) = d_i, \forall i = 1, \ldots, m_2 \\ & x_j \in \mathbb{N}_0, \forall j \in J^{int} \end{aligned} \tag{8.2}$$

Da siges det matematiske optimeringsproblem (8.2) at være *ubegrænset*, hvis der for alle $u \in \mathbb{R}$ eksisterer en vektor $\hat{x} \in \mathbb{R}^n$, således at

$$\begin{aligned} & f(\hat{x}) > u \\ & g_i(\hat{x}) \leq b_i, \forall i = 1, \ldots, m_1 \\ & h_i(\hat{x}) = d_i, \forall i = 1, \ldots, m_2 \\ & \hat{x}_j \in \mathbb{N}_0, \forall j \in J^{int} \end{aligned}$$

Som et eksempel på et ubegrænset lineært blandet heltalsprogram, kan man betragte

$$\begin{aligned} \max\ & 2x_1 - x_2 \\ \text{s.t.:}\ & x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \\ & x_1 \in \mathbb{N}_0 \end{aligned}$$

Det ses tydeligt, at $x_1 = x_2$ er en brugbar løsning for alle $x_1 \in \mathbb{N}_0$ med en objektfunktionsværdi givet ved $z(x_1) = 2x_1 - x_1 = x_1$. Lader vi nu $x_1$ gennemløbe de naturlige tal mod uendelig fås $\lim_{x_1 \to} x_1 = \infty$, hvorfor det altså er muligt at opnå en objektfunktionsværdi som er så stor som man ønsker det.

Hvor en korrekt formuleret model sagtens kan være *infeasible* er det noget mindre sandsynligt at en korrekt formuleret model kan være ubegrænset. Desværre kan det, som med identifikationen af en infeasible model, være særdeles svært at finde ud af hvorfor, en model er ubegrænset: modsat tilfældet hvor modellen var infeasible, har man i en ubegrænset model begrænsninger som ikke er restriktive nok, eller som måske helt er fraværende. Som oftest er der nogle begrænsninger som relaterer sig til det fysiske system som mangler, da det sjældent er muligt at forbruge af ressourcer i ubegrænsede mængder (det værende sig tid, råmaterialer, penge eller lignende). Det er med det mest optimeringssoftware muligt at tilgå den løsning, som solveren har fundet, der kan skaleres mod det uendelige (en

såkaldt ekstremretning). Ved inspektion af sådan en løsning, er det ofte muligt at opdage hvor i modellen, der mangler begrænsninger.

Et andet trick man kan bruge er, at definere øvre og nedre grænser for alle sine variabler, som er så tilpas store og små, at hvis en variabel opnår en værdi lig med en af sine grænser, ved man at noget er galt. Tag for eksempel en variabel $f_{ij}$ som måler hvor meget en lastbil transportere fra adresse $i$ til adresse $j$. Da gælder, at $f_{ij}$ altid, i enhver fornuftig løsning, er større end eller lig med nul. Samtidig ved vides det, at mængden, som lastbilen transportere fra $i$ til $j$, aldrig overstiger bilens kapacitet, da chaufføren ellers risikerer at få en stor bøde. Hvis $Q$ betegner lastbilens samlede kapacitet, haves altså, at $0 \leq f_{ij} \leq Q$ i alle brugbare løsninger. Hvis det så kræves, at $-1 \leq f_{ij} \leq Q + 1$ er det åbenbart, at hvis $f_{ij}$ antager værdien $-1$ eller $Q + 1$ i en optimal løsning returneret af solveren, så er der noget galt med enten model eller implementeringen af denne.

## 8.4 Modellen kan løses

Hvis et matematisk optimeringsproblem hverken er infeasible eller ubegrænset, har det mindst en optimal løsning, og vi vil betegne problemet som løsbart (solvable) (dette postulat gælder under tekniske antagelser, som alle er opfyldt for blandede heltalsproblemer). Når en optimal løsning til et matematisk optimeringsproblem er opnået, er det naturligt at ønske at få verificeret om løsningen så også er fornuftig og meningsfuld. Hvis løsningen ikke er meningsfuld eller strider mod nogle åbenlyse restriktioner, må der være noget galt med den matematiske model. Derfor bør den første test af en løsning bestå af sund fornuft. En god skeptisk tilgang til løsningen kan i mange tilfælde blotlægge mangler eller fejl i modellen, som ellers var skjulte inden en løsning blev produceret. Der er, for den uerfarne analytiker, ofte et stort fokus på objektfunktionsværdien: vi kan spare $X$-antal millioner eller tjene $Y$-milliarder. Men en lavpraktisk illustration af *løsningen* og ikke løsningsværdien, er altså særdeles nyttigt i denne og fremtidige tests. Ved at illustrere løsningen i et passende grafisk format, gøres det særdeles meget nemmere at gennemskue om den fundne løsningen nu også har de egenskaber, der forventes af en løsning til den givne problemstilling.

Hvis løsningen stadig virker meningsfuld efter denne første "sund fornuft"-test, er det næste naturlige skridt at sammenligne objektfunktionsværdien af den fundne løsning, med hvad vi ville forvente. Hvis objektfunktionsværdien er langt højere end vi forventede (i tilfælde af maksimering), er der tydelige tegn på, at modellen ikke

er restriktiv nok eller, at nogle begrænsninger helt mangler. På den anden side, hvis objektfunktionsværdien er lavere end forventet, er det et tegn på, at modellens begrænsninger er for restriktive. Som eksempel, husk på skemalægningsproblemet af $n$ jobs på flere maskiner (se afsnit 6.3 på side 107). Hvis løsningen resulterer i et tidsforbrug som er lavere end vi forventede muligt, tyder det på, at der er udeladt nogle begrænsninger. Omvendt, hvis tidsforbruget er højere end for en løsning vi allerede kender (måske den eksisterende plan), så er der tegn på, at modellen er for restriktiv. I det sidste tilfælde, er det naturligt at fiksere modellens variabler til værdier der reproducerer den kendte løsning, for på den måde at identificere de overdrevent restriktive begrænsninger (det vil være de brudte begrænsninger).

I tilfælde af, at løsningen klarer denne sammenligningstest, er det tid til at præsentere løsningen for de, som skal implementere den i praksis. Erfaringen blandt operatører, planlæggere og ledere, vil ofte kunne identificere problemer med løsningen, og udpege præcist hvad, der er galt. Der er ofte tale om mindst én af følgende tre typer af problemer

1. Der er praktiske begrænsninger, som ikke er overholdt.
2. Løsningen er ikke god nok, og der kan identificeres en bedre løsning, som i praksis er brugbar men som ikke er brugbar i modellen.
3. Løsningen er ikke god nok, og der kan identificeres en bedre løsning, som i praksis er brugbar, og som også er brugbar i modellen.

De to første tilfælde er relativt lette at håndtere, da de er inkluderede i diskussionen ovenfor: i tilfælde af 1. skal de begrænsninger, der mangler i modellen tilføjes. I tilfælde 2., skal de begrænsninger, som er brudt for den i praksis brugbare løsning revideres. Det tredje tilfælde er mere interessant: her har de erfarne folk fundet en løsning, som ikke bryder nogle begrænsninger, men som de mener,sine løsninger for en eller anden form for ændring i data for at undersøge er bedre end den løsning, som modellen har bevist er optimal. Der er en tydelig diskrepans her, som skal blotlægges: Objektfunktionen, som er valgt for modellen, afspejler ikke fuldstændigt brugerens præferencer. Dermed skal modellens *objektfunktion* ændres til at være i overensstemmelse med det, som forventes af løsningens brugere.

### 8.4.1 Stresstest af en brugbar løsning

Det er ikke altid nok, at man kan producere en løsning, som er både brugbar og optimal i forhold til hvad beslutningstageren har af kriterier. Dette skyldes, at

man ikke altid, blot ved at betragte en løsning, kan gennemskue om løsningen også vil præstere godt, når den bliver udsat for virkeligheden. Derfor er det ofte en rigtig god idé at stressteste sin løsning, inden den sættes i produktion. På den måde kan man danne sig et overblik over hvordan løsningen præsterer, og måske den vej igennem få et prej om, at løsningen i virkeligheden *ikke* er optimal til den problemstilling man forsøger at løse.

*Eksempel* 8.1. Her vil vi betragte et kapacitetsbegrænset lokationsplanlægningsproblem, som er løst til optimalitet med følgende allokering af kunder til faciliteter

| | | Allokerede kunder | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Åbne faciliteter | Kapacitet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 25 | x | x | x | x | | | | |
| 2 | 25 | | | | | x | x | x | x |
| | Efterspørgsel | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

Det er åbenlyst, at faciliteternes kapaciteter er overholdt, da der er allokeret efterspørgsel svarende til 24 enheder og der er kapacitet til 25 enheder. Betragt nu tilfældet hvor kunderne 1, 2, 3, og 4 har *stokastisk* efterspørgsel, hvilket oftest er tilfældet. Antag, at hver kundes efterspørgsel er en stokastisk variabel, som er et uniformt fordelte heltal mellem $a_i$ og $b_i$, for $i = 1, 2, 3, 4$. Antag yderligere, at grænserne $a_i$ og $b_i$ er givet ved:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $a_i$ | 5 | 4 | 5 | 3 |
| $b_i$ | 7 | 8 | 7 | 9 |

Da haves, at den forventede værdi af efterspørgslerne alle er lig med 6, som i tabellen ovenfor. Men antages det yderligere, at de stokastiske efterspørgsler er uafhængige, vil et simpelt simulationsstudie vise, at man kan forvente at bryde facilitet 1's kapacitet mellem 29–33% af tiden, hvilket må siges, at være sub-optimalt i praksis.

På baggrund af Eksempel 8.1 burde det være tydeligt, at man bør udsætte sine løsninger for en eller anden form for ændring i data, for at undersøge om løsningen er robust i praksis. Der er mindst to typer af robusthed man bør overveje når man vurdere om en løsning er robust:

1. Er løsningen robust i forhold til brugbarhed.
2. Er løsningen robust i forhold til optimalitet.

### Robust i forhold til brugbarhed

Dette er i mange situationer den vigtigste af de to typer robusthed. Når man undersøge om en løsning er robust med hensyn til brugbarhed, undersøger man, om en løsning forbliver brugbar selv efter ændringer i data. Det er også den simpleste af de to at teste, idet man blot skal ændre data og tjekke om den *givne løsning* forbliver brugbar. I Eksempel 8.1 så vi, at vi kunne udføre et simulationsstudie, hvor vi tjekkede om løsningen forblev brugbar når data ændrede sig, og det vist sig, at løsningen *ikke* var robust med hensyn til brugbarhed i over 30% af de realiserede tilfælde.

De mest naturlige måder at teste om en løsning er robust med hensyn til brugbarhed, er ved, at man enten udfører et simulationsstudie, hvor man estimerer sandsynlighedsfordelinger over sit inputdata, eller ved, at man bruger historisk data direkte som forskellige scenarier. På den må kan man undersøge hvor robust en løsning er, når data varierer.

### Robusthed i forhold til optimalitet

Når man undersøger om en løsning er robust i forhold til optimalitet, undersøger man om løsningen forbliver *optimal*, hvis data ændrer sig. Fra teorien om linear programmering vides det, at man kan benytte følsomhedsanalyse til at undersøge om en løsning forbliver optimal efter en ændring af én objektfunktionskoefficient. Der findes også teori, der beskriver om en løsning forbliver optimal, hvis mere end en objektfunktionskoefficient ændres på samme tid (se for eksempel "100% rule" i Balakrishnan m.fl. (2020)). Desværre er der på det tidspunkt, hvor dette kapitel skrives, ikke implementeret følsomhedsanalyse-funktionalitet for kombinatoriske og blandede heltalsoptimeringsproblemer i nogen moderne software pakker (som er forfatteren bekendt), da det kræver, at man genløser sine modeller flere gange for at opnå de "sensitivity ranges", som er kendt fra lineær programmering.

Dermed gælder der, at arbejder man med heltalsoptimering, så bliver man nødt til at *genløse* sine modeller, hvis man vil undersøge, hvordan en optimal løsning ændres, når inputdaten ændres. Hvis den nye løsning, som fremkommer efter ændring af data, er identiske med den oprindelige løsning, vil man sige, at den oprindelige løsning er robust i forhold til optimalitet, af den simple grund, at

den forbliver optimal efter en ændring af data. Det er dog særdeles sjældent, at en løsning forbliver optimal, hvis man ændrer data signifikant hvilket på mange måder er naturligt: en optimal løsning er en fuldstændig optimal udnyttelse af ressourcerne i et system på en sådan måde, at en objektfunktion antager den allerbedste værdi muligt. Det ville på mange måder være overraskende, hvis en så skræddersyet løsning også var optimal i andre situationer. Derfor vil man ofte tillade en vis forringelse af objektfunktionsværdien uden, at man på den måde vil forkaste den oprindelige løsning.

Antag at en optimal løsning til en matematisk model er $\bar{x}$. Antag videre, at der er $K$ datasæt til rådighed, som eksempelvis kan bestå af $K$ tidligere dages data, eller være blevet generet som scenarier i et simulationsstudie. Vi vil så, for hver datasæt genløse modellen hvorved $K$ objektfunktionsværdier, $z_1^*, \ldots, z_K^*$ opnås. Samtidig evalueres den oprindelige løsning $\bar{x}$ i de $K$ scenarier hvorved $K$ formodentligt suboptimale objektfunktionsværdier opnås. Lad disse være betegnet $\bar{z}_1, \ldots, \bar{z}_K$. Vi vil så sige, at løsning er $\varepsilon$-robust med hensyn til optimalitet hvis

$$\frac{|z_k^* - \bar{z}_k|}{|z_k^*|} \leq \varepsilon, \qquad \forall k = 1, \ldots, K$$

Det vil altså sige, at løsningen $\bar{x}$ er inden for $\varepsilon$ procent af en optimal løsning i alle scenarierne, hvorved den så at sige er "robust nok".

I det ovenfor beskrevne har vi ikke taget stilling til, at løsningen $\bar{x}$ nemt kan være *infeasible* til modellen i nogle af scenarierne, hvorved værdien $\bar{z}_k$ ikke er defineret for disse scenarier (dette kan *ikke* ske hvis stokastikken kun optræder i objektfunktionen (hvorfor?)). Hvis dette er tilfældet, er det ofte belejligt at definere værdien som følger: Hvis $\bar{x}$ ikke er brugbar i scenarie $k$, da bestemmes $\bar{z}_k$ ved

$$\bar{z}_k = \begin{cases} \infty, & \text{hvis det er et minimeringsproblem} \\ -\infty, & \text{hvis det er et maksimeringsproblem} \end{cases}$$

På den måde er det tydeligt, at hvis en løsning ikke er robust med hensyn til brugbarhed, er den ikke robust med hensyn til optimalitet. Samtidig kan en løsning nemt være robust med hensyn til brugbarhed, uden at være det med hensyn til robusthed. Dermed haves følgende implikation

Løsning er robust mht. optimalitet $\Rightarrow$ Løsningen er robust mht. brugbarhed

Det blev i foregående underafsnit postuleret, at robusthed i forhold til brugbarhed ofte var vigtigere end robusthed med hensyn til optimalitet. Dette skyldes,

naturligvis, at det i langt de fleste praktiske henseender er vigtigere at have en plan der virker, end en, der nødvendigvis er beviseligt optimal.

Der findes naturligvis også andre måder, at definere robusthed, og faktisk er dette et stort forskningsområde, som på ingen måde er færdigt-udforsket. Endnu en tilgang til håndtering af at data ikke er deterministisk er *stokastisk optimering*, som vi vil behandle i Kapitel 9 på side 163. I stokastisk optimering har man en anden tilgang til robusthed, idet man optimerer den *forventede værdi* af en objektfunktion givet stokastisk data, hvorved man mere proaktivt forholder sig til stokastisiteten i data end dette kapitel har gjort.

# 9 Avancerede emner

I dette kapitel vil vi behandle nogle mere avancerede teknikker til løsning af optimeringsproblem. I den første del af kapitlet behandler vi den situation hvor data ikke er deterministisk men derimod stokastisk. Vi vil her udvikle en metode til at løse det som kaldes et *two stage stochastic* optimeringsproblem, hvor den *forventede* værdi af en objektfunktion optimeres.

Efterfølgende vil vi betragte tilfældet hvor et optimeringsproblem ikke blot har én objektfunktion men derimod to eller flere. Vi vil introducere begrebet *Pareto optimalitet* som en erstatning for optimalitet kendt fra et-kriterie optimering og præsentere en løsningsmetode, som kan finde alle rationelle kompromiser når der er tale om to konfliktende objektfunktioner.

## 9.1 Stokastisk optimering

Indtil nu, har vi studeret det man kalder *deterministiske* modeller. Det vil sige, at vi har betragtet modeller hvor vi har lavet den relativt restriktive antagelse, at alt data er kendt med sikkerhed over hele planlægningsperioden. Dette er naturligvis kun tilfældet, hvis planlægningshorisonten er meget kort, eller hvis data er usædvanligt stabilt. Tag som for eksempel ruteplanlægningsproblemet CVRP. Her har vi tre typer af data: der er en "omkostningsmatrix" $(c_{ij})_{i,j=0}^{n}$, efterspørgsel hos hver kunde $q_i$ samt en kapacitet på bilerne på $Q$.

I de fleste praktiske tilfælde vil man uden videre kunne antage at bilens kapacitet er kendt og at $Q$ dermed er fast og kendt over hele planlægningshorisonten.

Afhængig af horisonten kan $q_i$ være underlagt stokastik eller være deterministisk. Hvis $q_i$ angiver en mængde af en vare, som skal leveres hos kunde $i$, kan den ofte betragtes som deterministisk når man planlægger ruten, idet det er noget man som planlægger har kontrol over. Det vil sige, hvis kunde $i$ har bestilt 20 ton sand, så leverer vi 20 ton sand uafhængigt af om kunden egentlig kun skulle bruge 18. I sådan et tilfælde har vi så at sige uddelegeret usikkerheden til

kunden: kunde $i$ bestiller $q_i$ tons, og vi leverer $q_i$ tons. På den anden side, hvis $q_i$ angiver en mængde, som skal hentes hos kunden uanset hvor meget kunden rent faktisk har (tænk for eksempel på indsamling af tomme flasker hos supermarkeder eller affald hos husholdninger), så er $q_i$ pludseligt en stokastisk variabel, fordi vi ikke kender den præcise mængde, som skal afhentes. Derimod kunne det være, at vi kendte en sandsynlighedsfordeling $\mathcal{P}_i$ sådan at $q_i \sim \mathcal{P}_i$. Vi kan komplicere tingene ved at antage, at $q_i$ og $q_j$ ikke er uafhængige, således at $q_i \sim \mathcal{P}_i(q_j)$ eller endnu mere komplekst at $q_i \sim \mathcal{P}(q_1, q_2, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n)$.

Det er med omkostningsmatricen $(c_{ij})_{i,j=0}^n$ som med efterspørgslen $q_i$ ikke åbenbart, hvorvidt man kan betragte den som deterministisk eller stokastisk. Hvis $(c_{ij})_{i,j=0}^n$ angiver rejse*afstandene* mellem kunderne, kan vi betragte den som deterministisk: afstanden mellem to punkter på jorden ændrer sig ikke hurtigt nok til, at det skal tages med i vores model. På den anden side, hvis $(c_{ij})_{i,j=0}^n$ angiver et mål som er stærkt afhængig af kørsels*tiden* må man i de fleste praktiske problemstillinger skulle betragte $(c_{ij})_{i,j=0}^n$ som værende stokastisk. Det er en ofte brugt antagelse, at $c_{ij}$ er tidsafhængig hvilket vil sige, at tiden det tager at køre fra $i$ til $j$ afhænger ikke kun af start og slutpunktet men også af hvornår man kører på strækningen. På den måde får vi en omkostningskoefficient $c_{ij}(t)$ hvis sandsynlighedsfordeling ikke kun afhænger af hvor man kører men også af til hvilken tid $t$, det vil sige $c_{ij}(t) \sim \mathcal{P}_{ij}(t)$. Dette betyder at $c_{ij}(t)$ er det man kalder en "continuous-time stochastic process".

Når man arbejder med stokastiske optimeringsproblemer skal man bestemmer sig for hvorledes man ønsker at håndtere denne stokastik: Vil man optimere en forventet værdi af en objektfunktion, vil man begrænse overskridelse af begrænsninger, vil man helt undgå overskridelse af begrænsninger osv.. Lad os begtragte

CVRP problemet som beskrevet tidligere:

$$
\begin{aligned}
\min \quad & \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{i=1}^{n} x_{i0} = m, \\
& \sum_{j=1}^{n} x_{0j} = m, \\
& \sum_{i=0}^{n} x_{ij} = 1, && \forall j = 1, \ldots, n \\
& \sum_{j=1}^{n} x_{ij} = 1, && \forall i = 1, \ldots, n \\
& u_i - u_j + Q x_{ij} \leq Q - d_j, && \forall i, j = 1, \ldots, n : i \neq j \\
& q_i \leq u_i \leq Q, && \forall i = 1, \ldots, n \\
& x_{ij} \in \{0, 1\}, && \forall i, j = 0, \ldots, n
\end{aligned}
$$

Lad os nu antage, at $c_{ij}$ i virkeligheden er uafhængige stokastiske variabler, og at $c_{ij}$ *ikke* er tidsafhængig. Vi kan da sige, at vi ønsker at minimere den forventede samlede omkostning ved turene. Dette kan gøres ved at minimere $\mathbb{E}[\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}]$. Hvis vi lader $\bar{c}_{ij} = \mathbb{E}[c_{ij}]$ kan dette opskrives som $\mathbb{E}[\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}] = \sum_{i=0}^{n} \sum_{j=0}^{n} \bar{c}_{ij} x_{ij}$ idet vi har antaget at $c_{ij}$'erne er uafhængige, hvilket vil sige, at hvis kun $c_{ij}$'erne er stokastiske, så kan vi minimere den forventede omkostning ved at løse et *deterministisk* problem. Hvis vi på den anden side ønsker at minimere den forventede omkostning *under bibetingelse af* at sandsynligheden for at omkostningen overstiger en hvis tærskel, fx $\tilde{C}$, er under en hvis tærskel så kan vi inkludere det, der hedder en *probabilistisk begrænsning*, som opskrives som

$$P(\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} > \tilde{C}) \leq \alpha.$$

Denne begrænsning skal læses som: Givet en sandsynlighed $\alpha \in [0, 1]$, begræns da de mulige ruter således, at sandsynligheden for at rutens omkostning overstiger $\tilde{C}$ (det vil sige $P(\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} > \tilde{C})$) er mindre end $\alpha$. Ved at sætte $\alpha = 0$ har vi en hård begrænsning som siger, at omkostningen aldrig må overstige $\tilde{C}$ mens $\alpha = 1$ betyder, at begrænsningen er overflødig.

Havde efterspørgslen været stokastisk kunne vi have gjort noget tilsvarende:

$$P(u_j > Q) \leq \alpha, \quad \forall j = 1, \ldots, n$$

hvor $u_j$ igen er defineret som $u_j = u_i + d_j$ hvis $x_{ij} = 1$. Det vil sige, at vi ønsker, at sandsynligheden for, at vi overstiger bilens kapacitet er mindre end en procentsats, $\alpha$. En komplikation ved de probabilistiske begrænsninger der her er opskrevet er, at sandsynligheden for at omkostningen overstiger $\tilde{C}$/at bilens kapacitet overskrides afhænger af realiseringen af mange stokastiske variabler på samme tid. Derfor er løsningen af stokastiske problemer særdeles meget mere komplicerede end løsningen af deterministiske problemer.

### 9.1.1 Stokastisk optimering med *recourse*

I mange problemstillinger hvor man skal forholde sig til stokastik, er beslutningsprocessen sekventiel i den forstand, at man skal tage en (partiel) beslutning *nu og her* for så at se en realisering af nogle stokastiske parametre, som man så skal reagere på. Denne mulighed for at reagere på realiseringen af nogle stokastiske parametre kaldes *recourse*. Lidt forenklet kan man tænke på recourse på følgende måde:

1. Vælg værdier til nogle variabler $y$ som kontrollerer hvad der sker *i dag*
2. I løbet af en given tidsperiode får vi information om nogle parametre som var stokastiske
3. Herefter foretager vi en recourse handling, $x$, som skal imødegå det som nu måske ikke fungerer mere efter realiseringen af de ukendte parametre.

Denne type recouse kaldes *two-stage recourse* grundet de to stadier i beslutningsprocessen.

I det følgende vil vi opskrive hvad vi præcis mener med et two-stage stokastisk optimeringsproblem med recourse. Vi vil lade $\xi$ være en stokastisk variabel som influerer på både begrænsninger og objektfunktionen. Man kan tænke på $\xi$ som et scenarie: et scenarie er ikke bare en realisering af én stokastisk variabel men en realisering af alle stokastiske variabler på en gang. Fx, lav efterspørgsel hos kunde 1, høj efterspørgsel hos kunde 2, høje brændstofpriser, modvind, lav trafik mængde, osv.. Vi kan ved hjælp af denne abstraktion matematisk, og ret teknisk,

opskrive et generelt two stage stokastisk optimerings problem med recourse som

$$\begin{aligned} \min \ & \sum_{i=1}^{n} f_i y_i + \mathbb{E}_\xi[Q(y,\xi)] \\ \text{s.t.: } & y \in \mathcal{Y} \end{aligned} \tag{9.1}$$

hvor

$$\begin{aligned} Q(y,\xi) = \min \ & \sum_{j=1}^{m} c(\xi)_j x_j \\ \text{s.t.: } & T(\xi)y + W(\xi)x = h(\xi) \\ & x \geq 0 \end{aligned} \tag{9.2}$$

Det første optimeringsproblem, (9.1), minimerer den første periodes *direkte omkostninger* plus de *forventede recourse omkostninger*, her givet ved funktionen $Q(y,\xi)$. Recourse funktionen er her defineret for alle scenarier, $\xi$, og for alle mulige *first stage* beslutninger $y$. Vi skal altså læse $Q(y,\xi)$ som second stage omkostningen hvis vi tager beslutningen $y$ i første stadie *og* vores stokastiske variabler realiseres ved $\xi$. I det første optimerings problem, (9.1), betegner $\mathcal{Y}$ alle de mulige løsninger som vi kan vælge imellem i første stadie. Begrænsningerne $T(\xi)y + W(\xi)x = h(\xi)$ i (9.2) sammenkobler beslutningerne i første og andet stadie således at givet en beslutning $y$, kan vi *ikke* frit vælge en second stage beslutning $x$. Man bør notere sig, at $T$ og $W$ er matricer mens $h$ er en vektor.

### 9.1.2 Løsning af two stage stokastisk optimerings problem med recourse: deterministisk ækvivalent

Det er, indrømmet, ikke særligt nemt at forestille sig hvorledes problemet (9.1) skal løses: det er et optimeringsproblem, som minimerer summen af noget kendt og noget forventet og det forventede kan kun udregnes ved at løse et andet optimeringsproblem. Det virker bøvlet!

Vi vil i dette afsnit antage, at den stokastiske variabel $\xi$ kan realisere sig i $K$ scenarier $\xi_1, \ldots, \xi_K$, som hver har en kendt sandsynlighed for at indtræffe givet ved $p_1, \ldots, p_K$. Det første der skal ske er at udregne omkostnings- og

begrænsningskoefficienterne for hvert scenarie

$$\begin{aligned} c_j^k &:= c_j(\xi_k) \\ T^k &:= T(\xi_k) \\ W^k &:= W(\xi_k) \\ h^k &:= h(\xi_k) \end{aligned}$$

for alle $k = 1, \ldots, K$. Det vil altså sige, at vi kan tænke på $c_j^k$ som værende omkostningskoefficienten for $x_j$ hvis scenarie $k$ indtræffer, $T^k$ er ressourceforbruget af $y$-variablerne i scenarie $k$, $W^k$ er ressourceforbruget af $x$-variablerne i scenarie $k$ og $h^k$ er de tilgængelige ressourcer i scenarie $k$.

Ved hjælp at disse scenarie-indicerede parametre kan vi opskrive det, som man i litteraturen kender som den *deterministiske ækvivalent*:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} f_i y_i + \sum_{k=1}^{K} p_k \left( \sum_{j=1}^{m} c_j^k x_j^k \right) \\ \text{s.t.:} \quad & y \in Y \\ & T^k y + W^k x^k = h^k, && \forall k = 1, \ldots, K \\ & x^k \geq 0, && \forall k = 1, \ldots, K \end{aligned}$$

Her har vi indført et sæt variabler $x^k$ for hver scenarie, som fortæller os hvordan vi skal reagere i scenarie $k$ hvis vi i vores første stadie har truffet beslutningen $y$. Læg særligt mærke til hvorledes vi har omskrevet objektfunktionen således at den forventede værdi af $Q(y, \xi)$ nu skrives som summen af omkostningen forbundet med andet stadie i scenarie $k$ *vægtet* med sandsynligheden for at scenariet indtræffer. Endvidere, haves at begrænsningerne for variablerne $x^k$ stadig siger, at hvis vi har taget beslutningen $y$, så skal responsen $x^k$ stadig opfylde begrænsningerne $T^k y + W^k x^k = h^k$ i alle scenarier.

Vi har altså nu omskrevet det stokastiske problem til et ækvivalent deterministisk problem hvor alle parametre er kendte. Derfor kaldes dette program, den *deterministiske ækvivalent* til det stokastiske problem.

### 9.1.3 Generering af scenarier

Det vil næsten aldrig være praktisk muligt at liste alle scenarier som kan indtræffe. Tag for eksempel CVRP problemet fra før. Hvis vi antager hver $q_i$ kan tage

værdierne $0, \ldots, Q$ med sandsynlighederne $p_i^1, p_i^2, \ldots, p_i^Q$, så kan vi hurtigt regne ud, at der er $(Q+1)^n$ scenarier vi skal håndtere. For et meget lille CVRP med $Q = 10$ og $n = 10$ haves der altså 25.937.424.601 scenarier, som skal håndteres. Hvis efterspørgslerne $q_i$ er kontinuert fordelt, haves pludseligt et uendeligt (og overtælleligt) antal mulige scenarier.

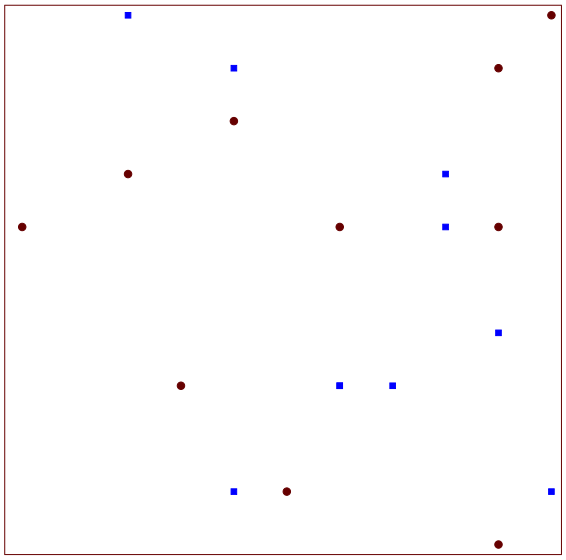
Derfor vil den deterministiske ækvivalent lynhurtigt eksplodere i både antallet af variabler og begrænsninger, hvorved det bliver praktisk umuligt at løse problemet. Hvad man derimod ofte vil benytte er *Monte Carlo simulering* til at genere en række repræsentative scenarier. Vi kan her antage, at vi har generet $\bar{K}$ scenarier vha. Monte Carlo Simulering, hvorved den deterministiske ækvivalent bliver

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} f_i y_i + \sum_{k=1}^{\bar{K}} \frac{1}{\bar{K}} \left( \sum_{j=1}^{m} c_j^k x_j^k \right) \\
\text{s.t.:} \quad & y \in Y \\
& T^k y + W^k x^k = h^k, && \forall k = 1, \ldots, \bar{K} \\
& x^k \geq 0, && \forall k = 1, \ldots, \bar{K}
\end{aligned}
$$

hvor det eneste vi har ændret er, at vi estimerer sandsynligheden for at scenarie $k$ indtræffer til at være $\frac{1}{\bar{K}}$ og at vi altså ikke har alle scenarier i spil.

*Eksempel* 9.1. I dette eksempel skal vi se på et stokastisk facility location problem hvor efterspørgslen hos kunderne er stokastisk mens de faste omkostninger i forbindelse med etableringen af faciliteterne er deterministisk. Læg mærke til, at der i facility location problemer er en naturlig opdeling af beslutningerne: først beslutte vi hvor faciliteterne skal ligger og når de står færdige og efterspørgslen realiseres bestemmer vi hvorledes denne efterspørgsel skal allokeres til de åbne faciliteter. I dette eksempel, vil vi antage at der er 10 kunder og 10 mulige placeringer for faciliteterne. Dette instans er vist i Figur 9.1.

Vi vil antage, at der en omkostning på $c_{ij}$ kroner *per enhed* af produktet som flyttes fra facilitet $i$ til kunde $j$. Samtidig antager vi, at hver kunde har en *stokastisk* efterspørgsel på $d_j(\xi)$ som afhænger af scenariet $\xi$ (vi vil senere angive ssh. fordelinger for $d_j$'s værdier). Det vil altså sige, at hvis vi servicerer al kunde $j$'s efterspørgsel fra facilitet $i$, så koster det $d_j(\xi)c_{ij}$ kroner. Vi vil, som vi plejer, indføre binære lokationsvariabler $y_i$ som er lig 1 hvis og kun hvis en facilitet etableres på lokation $i$ og kontinuerte variabler $x_{ij}$ som angiver andelen af kunde $j$'s efterspørgsel som håndteres fra facilitet $i$. Hvis en facilitet etableres på lokation



Figur 9.1: Illustration af lokationsproblemet. Kunderne er markeret med rødbrune cirkler mens de mulige facilitetsplaceringer er angivet med blå kvadrater.

| | Kunder | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_{ij}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | $s_i$ | $f_i$ |
| 1 | 4 | 4 | 6 | 6 | 3 | 6 | 1 | 6 | 8 | 2 | 50 | 210 |
| 2 | 5 | 8 | 3 | 6 | 7 | 2 | 4 | 4 | 7 | 3 | 50 | 340 |
| 3 | 5 | 8 | 3 | 6 | 7 | 2 | 4 | 4 | 7 | 3 | 25 | 370 |
| 4 | 9 | 9 | 7 | 10 | 8 | 5 | 5 | 1 | 11 | 6 | 35 | 300 |
| 5 | 6 | 8 | 4 | 6 | 6 | 3 | 4 | 4 | 8 | 3 | 45 | 360 |
| 6 | 4 | 4 | 6 | 6 | 2 | 7 | 1 | 7 | 8 | 2 | 30 | 290 |
| 7 | 7 | 11 | 2 | 6 | 9 | 1 | 7 | 5 | 6 | 5 | 35 | 110 |
| 8 | 6 | 6 | 6 | 8 | 5 | 5 | 2 | 4 | 9 | 4 | 30 | 400 |
| 9 | 1 | 6 | 6 | 3 | 5 | 8 | 6 | 10 | 5 | 4 | 50 | 190 |
| 10 | 3 | 8 | 7 | 3 | 7 | 9 | 8 | 12 | 4 | 6 | 40 | 130 |

Tabel 9.1: Det deterministiske data til det stokastiske lokationsproblem

$i$ antages det, at den kan håndtere $s_i$ enheder og at det koster $f_i$ kroner at etablere den. Den deterministiske del af dataet for problemet er givet i Tabel 9.1.

Vi kan således opskrive et two stage stokastisk optimerings problem med recourse, som minimerer den forventede omkostning, på følgende vis

$$\min \sum_{i=1}^{10} f_i y_i + \mathbb{E}[Q(y, \xi)]$$
$$\text{s.t.: } y_i \in \{0, 1\}, \forall i = 1, \ldots, 10$$

hvor second stage problemet er givet ved

$$
\begin{aligned}
Q(y,\xi) = \min \quad & \sum_{i=1}^{10}\sum_{j=1}^{10} d_j(\xi)c_{ij}x_{ij} \\
\text{s.t.:} \quad & \sum_{i=1}^{10} x_{ij} = 1, && \forall j = 1,\ldots,10 \qquad (9.3)\\
& \sum_{j=1}^{10} d_j(\xi)x_{ij} \leq s_i y_i, && \forall i = 1,\ldots,10 \qquad (9.4)\\
& x_{ij} \geq 0, \forall i,j = 1\ldots,10
\end{aligned}
$$

I dette problem, har vi at $\mathcal{Y} = \{0,1\}^{10}$ og at $T(\xi)y + W(\xi)x = h(\xi)$ er givet ved begrænsningerne (9.3) og (9.4). Vi kan lægge mærke til, at i dette eksempel er det egentlig blot efterspørgslen som er stokastisk, men fordi efterspørgslen både indgår i objektfunktionen og i begrænsningerne "spreder" stokastikken sig rundt i næsten hele second stage problemet. Interessant!

Lad os nu antage, at alle kundernes efterspørgsler er normalfordelte, det vil sige $d_j \sim \mathcal{N}(\mu_j, \sigma_j^2)$ med parametre som givet i tabellen

| Kunde | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | 10 | 12 | 17 | 18 | 15 | 18 | 18 | 12 | 15 | 11 |
| $\sigma^2$ | 2 | 5 | 3 | 4 | 4 | 3 | 3 | 3 | 5 | 4 |

Vi vil her antage, for at simplificere tingene lidt, at $d_j(\xi) \geq 0$ og at $d_j(\xi) \in \mathbb{N}$. Vha. ethvert stykke software som er i stand til at generere (pseudo) tilfældige til, kan man nu sample en række scenarier. For dette eksempel er der samplet 5 scenarier givet i Tabel 9.2. Nu da vi har en række scenarier kan vi opskrive den deterministiske ækvivalent til det stokastiske program:

| | Kunder | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Scenarie | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 10 | 13 | 17 | 23 | 15 | 21 | 17 | 11 | 19 | 10 |
| 2 | 9 | 17 | 16 | 21 | 17 | 19 | 12 | 9 | 12 | 4 |
| 3 | 10 | 11 | 18 | 17 | 15 | 17 | 18 | 14 | 12 | 7 |
| 4 | 11 | 8 | 13 | 21 | 8 | 21 | 17 | 13 | 11 | 10 |
| 5 | 7 | 8 | 24 | 25 | 16 | 25 | 20 | 16 | 19 | 5 |

**Tabel 9.2: Efterspørgslen for hver kunde i hvert af de fem scenarier. For eksempel er kunde 3's i fjerde scenarie, $d_3^4$, lig med 13.**

$$\min \sum_{i=1}^{10} f_i y_i + \sum_{k=1}^{5} \frac{1}{5} \left( \sum_{i=1}^{10} \sum_{j=1}^{10} d_j^k c_{ij} x_{ij}^k \right)$$

$$\text{s.t.:} \sum_{i=1}^{10} x_{ij}^k = 1, \qquad \forall j = 1, \ldots, 10, \;\; k = 1, \ldots, 5$$

$$\sum_{j=1}^{10} d_j^k x_{ij}^k \leq s_i y_i, \qquad \forall i = 1, \ldots, 10, \;\; k = 1, \ldots, 5$$

$$x_{ij}^k \geq 0, \qquad \forall i, j = 1, \ldots, 10, \;\; k = 1, \ldots, 5$$

$$y_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, 10$$

Prøv selv at løs ovenstående problem vha. CPLEX.

## 9.2 Multikriterie optimering

Når man som præskriptiv analytiker beskæftiger sig med erhvervsøkonomiske problemstillinger har man et arsenal af modelleringstekniske redskaber som man kan bruge til at konceptualisere en given problemstilling. Man ved at man skal identificere

1. hvilke data der er til rådighed (parametre),
2. hvilke beslutninger der skal tages (beslutningsvariabler),
3. hvorfor én løsning er bedre end en anden (objektfunktion),

4. hvorledes forskellige beslutninger berører hinanden og hvorledes beslutninger påvirker knappe ressourcer (begrænsninger).

I et stokastisk miljø er punkt 1 ikke udelukkende et spørgsmål om at identificere en række konstanter men også et spørgsmål om at identificere *fordelinger* for hvorledes data opfører sig.

En komplikation som man ikke altid er sig bevidst som analytiker er, at der også kan ligge stor usikkerhed i punkt tre, altså definition af en objektfunktion. Hvorledes skal man for eksempel forholde sig til en beslutningstager, som ønsker at minimere udgifterne i forbindelse med en produktionsplan, men også ønsker at minimere det maksimale antal overarbejdstimer i hver produktionsafdeling? Eller en beslutningstager, som ønsker at minimere den samlede tid en flåde af køretøjer skal køre, men samtidig ønsker at minimere den miljømæssige påvirkning. I disse to eksempler, har beslutningstageren to (eller flere) mål som ønskes optimeret samtidig, og i mange tilfælde findes der ikke én løsning, som er optimal for begge mål; de to mål er så at sige i konflikt med hinanden. Man kunne også forestille sig, at beslutningstageren ikke er én person men derimod 2 (eller flere) personer, som har forskelligt-rettede mål, der skal optimeres samtidigt.

For at håndtere disse problemstillinger er man nødsaget til at indføre et nyt optimalitetsbegreb. Når man blot har én objektfunktion, er det lige til at definere optimalitet, da man blot ser på værdien af objektfunktionen: en brugbar løsning er optimal, hvis der *ikke* findes nogle brugbare løsninger med en *strengt mindre* objektfunktionsværdi (i tilfælde af et minimeringsproblem). Men hvorledes afgør man om en løsning til ruteplanlægningsproblemet, som har en total tid på 42 minutter og en miljømæssig påvirkning på 12 ton $CO_2$ er bedre end en anden løsning som tager 45 minutter men som har en miljømæssig påvirkning på 11 tons $CO_2$? Disse to løsninger er så at sige usammenlignelige da vi ikke blot kan sige at

$$\begin{pmatrix} 42 \\ 12 \end{pmatrix} \leq \begin{pmatrix} 45 \\ 11 \end{pmatrix} \qquad \text{eller} \qquad \begin{pmatrix} 42 \\ 12 \end{pmatrix} \geq \begin{pmatrix} 45 \\ 11 \end{pmatrix}$$

Havde vi derimod fundet en løsning med en total tid på 43 minutter og som udledte 13 tons $CO_2$ så kunne vi sige, at vi ville foretrække løsningen givet ved $(42, 12)$ frem for løsningen resulterende i $(43, 13)$ da denne er bedre for begge objektfunktioner. Dette leder til en ny definition af optimalitet for problemer med mere end en objektfunktion

**Definition 9.1** (Pareto optimalitet)**.** Antag at en mængde $\mathcal{X}$ af brugbare løsninger er givet og at objektfunktionerne $f_1(x)$ og $f_2(x)$ *ønskes minimeret på samme tid*. Da siger vi, at en løsning $x^* \in \mathcal{X}$ er *Pareto optimal* hvis der *ikke eksisterer* en anden løsning $\bar{x} \in \mathcal{X}$ således at

1. $f_1(\bar{x}) \leq f_1(x^*)$ *og* $f_2(\bar{x}) \leq f_2(x^*)$
2. $f_1(\bar{x}) < f_1(x^*)$ *eller* $f_2(\bar{x}) < f_2(x^*)$

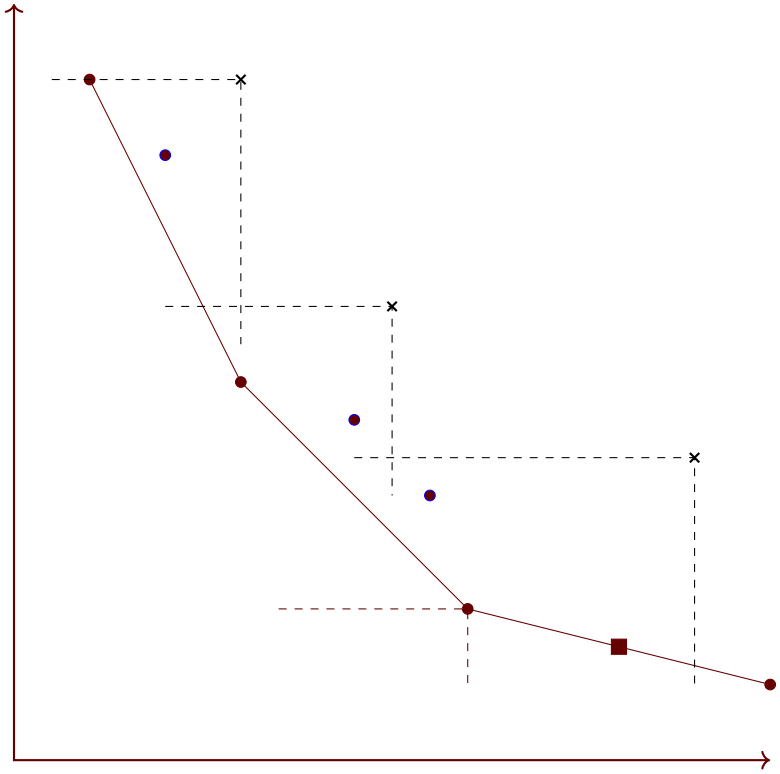
Vi siger, at det punkt i $\mathbb{R}^2$ som en Pareto optimal løsning definerer, det vil sige $y = \begin{pmatrix} f_1(x^*) \\ f_2(x^*) \end{pmatrix}$, er et *ikke-domineret* punkt. En Pareto optimal løsning kaldes også *efficient*.

Intuitivt kan man tænke på en Pareto optimal løsning som en løsning, hvor vi ikke kan forbedre en af objektfunktionerne uden samtidigt at forværre en af de andre. Definition siger da også, at en løsning $x^*$ er Pareto optimal, hvis der ikke findes en anden løsning, som er mindst lige så god for begge objektfunktioner og samtidig strengt bedre for mindst en af dem. Dette betyder også, at man ikke længere taler om én optimal løsning men om en *mængde* af Pareto optimale løsninger. Mængden af Pareto optimale løsninger giver således en samlet oversigt over alle *rationelle kompromiser* mellem objektfunktionerne. Vi vil her definere notation, som oftest bruges inden for litteraturen omkring multi-kriterie optimering

**Definition 9.2.** Mængden af alle Pareto optimale/efficiente løsninger betegnes med mængden $\mathcal{X}_E$ og afbildningen i *kriterierummet* defineres som $\mathcal{Y}_N$. Det vil sige, at $\mathcal{Y}_N = \{\ y \in \mathbb{R}^2\ :\ y = (f_1(x), f_2(x)), x \in \mathcal{X}_E\}$.

Fodtegnet på $\mathcal{X}_E$ står for "*E*fficient" mens fodtegnet på $\mathcal{Y}_N$ står for "*N*on-dominated".

Man kan ved at plotte løsningernes objektfunktions-værdier i et todimensionalt koordinatsystem få et geometrisk indblik i hvad et ikke domineret punkt er (se Figur 9.2). De ikke dominerede punkter er således afbildningen af de løsninger, hvor der ikke ligger andre punkter "sydvest" på i koordinatsystemet. Tager vi for eksempel de løsninger, som er markeret med et kryds og betragter området syvest for disse punkter (i keglen markeret med stiplede linjer) ser vi, at der i alle tilfælde ligger andre punkter i disse *dominans-kegler*. Derfor er disse kryds-markerede punkter *ikke* løsnings-vektorer for Pareto optimale løsninger. Betragter vi i stedet



Figur 9.2: Geometrisk illustration af en mængde af ikke-dominerede punkter (markeret med cirkler) og dominerede punkter (markeret med krydser)

punkterne markeret med cirkler, findes der ikke nogle andre punkter sydvest for disse, og dermed er de ikke-dominerede punkter. Vi kalder også afbildningen af alle de Pareto optimale løsninger for den *efficiente front* og den giver os en trade off kurve for objektfunktionerne: hvor meget skal opgives i forhold til én objektfunktion for at kunne forbedre en anden.

Den efficiente front kan deles op i det man kalder, støttende punkter, ekstremt støttende punkter og ikke-støttende punkter. De støttende punkter, er alle de ikke-dominerede punkter som ligger på *randen* af den efficiente front. I Figur 9.2 er det alle de ikke-dominerede punkter, som ligger på de linjer, som forbinder nogle af de ikke-dominerede punkter (de røde cirkler og den røde firkant). De ekstremt støttende punkter, er de ikke-dominerede støttende punkter, som definerer i hjørnepunkterne på de linjer der forbinder de røde cirkler i Figur 9.2. Dermed er den røde firkant *ikke* ekstremt støttende, men blot støttende. De ikke-støttende ikke-dominerede punkter, er de ikke-dominerede punkter, som ligger nord-øst for linjerne der forbinder de røde cirkler i Figur 9.2. Det vil sige de punkter, som er markeret må blå cirkler.

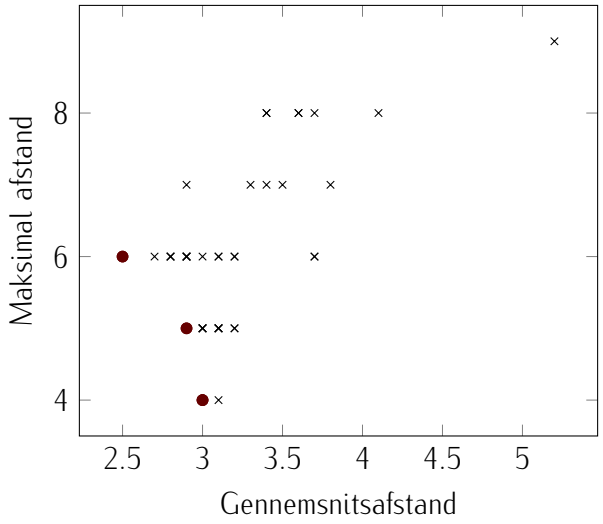
For at få en bedre føling med konceptet Pareto optimalitet, betragtes følgende

eksempel.

*Eksempel* 9.2. Her skal vi betragte et lille lokationsproblem hvor en beslutningstager ønsker at placere 2 faciliteter blandt 10 mulige lokationer for at servicere 10 kunder. Der er ingen kapacitetsbegrænsninger på faciliteterne og ingen faste etableringsomkostninger. Målet for beslutningstageren er at minimere den gennemsnitlige kørselsafstand fra kunderne til faciliteterne og samtidig skal den *maksimale kørselsafstand*, fra en en kunde til dennes nærmeste facilitet, minimeres. Dette er en kombination af de velkendte $p$-median og $p$-center problemer og kan formuleres som et multikriterie (bikriterie i dette tilfælde) problem på følgende vis: Lad $d_{ij}$ være afstanden fra lokation $i$ til kunde $j$. Afstandene er givet ved tabellen

| | kunder | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $d_{ij}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 3 | 5 | 3 | 4 | 8 | 6 | 4 | 2 | 2 | 4 |
| 2 | 1 | 6 | 6 | 1 | 6 | 6 | 5 | 2 | 4 | 1 |
| 3 | 3 | 2 | 8 | 6 | 4 | 2 | 4 | 3 | 7 | 5 |
| 4 | 1 | 4 | 7 | 4 | 4 | 4 | 4 | 1 | 5 | 3 |
| 5 | 0 | 4 | 6 | 3 | 5 | 5 | 4 | 1 | 4 | 2 |
| 6 | 4 | 1 | 7 | 6 | 6 | 2 | 1 | 3 | 6 | 6 |
| 7 | 4 | 8 | 5 | 1 | 8 | 9 | 7 | 4 | 4 | 2 |
| 8 | 3 | 3 | 9 | 5 | 3 | 3 | 4 | 3 | 7 | 4 |
| 9 | 4 | 8 | 5 | 1 | 8 | 9 | 7 | 4 | 4 | 2 |
| 10 | 3 | 4 | 4 | 5 | 8 | 5 | 2 | 2 | 3 | 5 |

Endvidere lader vi $y_i$ være en binær variabel som er lig med 1 hvis og kun hvis en facilitet placeres på lokation $i$. Variablen $x_{ij}$ er ligeledes binær og er lig med 1 hvis og kun hvis facilitet $i$ servicerer kunde $j$. Med disse definitioner kan vi opskrive et bikriterie heltalsproblem, som minimere den gennemsnitlige rejseafstand og den maksimale rejseafstand på følgende vis (se Kapitel 4 om lokationsplanlægning for




**Figur 9.3: Plot af alle 45 løsninger til lokationsproblemet i Eksempel 9.2. Ikke-dominerede løsninger er markeret med cirkler, mens dominerede løsninger er markerede med sorte krydser.**

flere detaljer)

$$
\begin{aligned}
\min\ & \frac{1}{10}\sum_{i=1}^{10}\sum_{j=1}^{10} d_{ij}x_{ij} \\
\min\ & \max_{j=1,\ldots,10}\{d_{ij}x_{ij}\} \\
\text{s.t.:}\ & \sum_{i=1}^{10} x_{ij} = 1, && \forall j = 1,\ldots,10 \\
& x_{ij} \leq y_i, && \forall i,j = 1,\ldots,10 \\
& \sum_{i=1}^{10} y_i = 2, \\
& x_{ij}, y_i \in \{0,1\}, && \forall i,j = 1,\ldots,10
\end{aligned}
$$

Hvis man generer alle 45 ($\binom{10}{2} = 45$) løsninger til lokationsproblemet og plotter deres løsningsværdier i et todimensionalt koordinatsystem med den gennemsnitlige afstand ud af den vandrette akse og den maksimale afstand fra en kunde til dennes facilitet op ad den lodrette akse kan løsningsværdierne visualiseres som i Figur 9.3. For hver af de sorte krydser vil der være en ikke-domineret løsning som dominerer den, og som altså er mindst lige så god for begge objektfunktioner og som er strengt bedre for mindst en af dem. Man vil derfor aldrig præsentere en beslutningstager for løsningerne svarende til de sorte krydser, da der altid er mindst en anden løsning som kan forbedre en af objektfunktionerne uden at forringe en af de andre.

### 9.2.1 Løsning af bikriterie optimeringsproblemer

For bikriterie optimering findes der en særdeles effektiv måde at genere alle ikke-dominerede løsninger, som kaldes $\varepsilon$-metoden. Denne metode tager en af objektfunktionerne og flytter den ned til begrænsningerne med en øvre grænse på $\varepsilon$. Ved at starte med $\varepsilon = \infty$ og så gradvist sænke den, kan man generere alle ikke-dominerede løsninger ved at løse en række et-kriterie problemer. Mere generelt, kan vi løse et bikriterie problem givet ved

$$\begin{aligned} &\min\ f_1(x) \\ &\min\ f_2(x) \\ &\text{s.t.: } x \in \mathcal{X} \end{aligned}$$

hvor $\mathcal{X}$ er en mængde af brugbare løsninger, med $\varepsilon$-metoden på følgende vis

1. Sæt $\varepsilon^1 = \infty$, $\mathcal{X}^* = \emptyset$ og lad en iterationstæller $k = 1$ være givet
2. Løs problemet

$$\begin{aligned} &\min\ f_1(x) \\ &\text{s.t.: } x \in \mathcal{X} \\ &\quad f_2(x) \leq \varepsilon^k \end{aligned}$$

   Hvis problemet ikke er *infeasible*, lad da $x^k$ være en optimal løsning og sæt $\mathcal{X}^* = \mathcal{X}^* \cup \{x^k\}$. Ellers gå til 4.
3. Sæt $\varepsilon^{k+1} = f_2(x^k) - \alpha$ hvor $\alpha > 0$ er en lille tal og sæt $k = k + 1$. Gå til Step 2.
4. Fjern eventuelt dominerede løsning fra $\mathcal{X}^*$ og præsenter de fundne rationelle kompromiser for en beslutningstager.

Hvorfor kan der være punkter, som er domineret i mængden $\mathcal{X}^*$?. Lad os anvende $\varepsilon$-metoden på problemet fra Eksempel 9.2:

*Eksempel* 9.3 (Eksempel 9.2 fortsat). Tag nu bikriterie problemet fra Eksempel 9.2 givet ved

$$
\begin{aligned}
&\min \ \frac{1}{10}\sum_{i=1}^{10}\sum_{j=1}^{10} d_{ij}x_{ij} \\
&\min \ \max_{j=1,\dots,10}\{d_{ij}x_{ij}\} \\
&\text{s.t.:} \ \sum_{i=1}^{10} x_{ij} = 1, && \forall j = 1,\dots,10 \\
&\quad x_{ij} \leq y_i, && \forall i,j = 1,\dots,10 \\
&\quad \sum_{i=1}^{10} y_i = 2, \\
&\quad x_{ij}, y_i \in \{0,1\}, && \forall i,j = 1,\dots,10
\end{aligned}
$$

Hvis vi tager, og flytter den anden objektfunktion ned til begrænsningerne med en øvre grænse på $\varepsilon$ fås programmet

$$
\begin{aligned}
&\min \ \frac{1}{10}\sum_{i=1}^{10}\sum_{j=1}^{10} d_{ij}x_{ij} \\
&\text{s.t.:} \ \sum_{i=1}^{10} x_{ij} = 1, && \forall j = 1,\dots,10 \\
&\quad x_{ij} \leq y_i, && \forall i,j = 1,\dots,10 \\
&\quad \sum_{i=1}^{10} y_i = 2, \\
&\quad \max_{j=1,\dots,10}\{d_{ij}x_{ij}\} \leq \varepsilon \\
&\quad x_{ij}, y_i \in \{0,1\}, && \forall i,j = 1,\dots,10
\end{aligned}
$$

Vi kan så bruge $\varepsilon$-metoden på følgende vis

1. Start med at sætte $\varepsilon = \infty$. På den måde, minimeres den gennemsnitlige afstand uden hensyntagen til den maksimale afstand.
2. Løs problemet hvor $\varepsilon = \infty$ hvorved en løsning givet ved $y_5 = y_6 = 1$ og resten af lokationsvariablerne er lig nul findes. Dette giver en gennemsnitlig afstand på 2.5 og en maksimal afstand på 6.

3. Vi vil nu tvinge modellen til at forbedre objektfunktionen, som måler den maksimale afstand. Dette gør vi ved at sætte $\varepsilon = 6 - \alpha$ hvor $\alpha$ er et lille positivt tal. På den måde kan vi tvinge vores model til nu at minimere den gennemsnitlige afstand *under hensyntagen til* at den maksimale afstand skal forbedres. Da vi her har heltalsdata, kan vi sætte $\alpha = 1$ (hvorfor?).

4. Løs nu modellen med $\varepsilon = 6-1 = 5$ og få løsningen givet ved $y_5 = y_{10} = 1$, som resulterer i en gennemsnitlig afstand på 2.9 og en maksimal afstand på 5.

5. Gentag processen ved at sætte $\varepsilon = 5 - 1 = 4$. Løses problemet fås en løsning givet ved $y_4 = y_{10} = 1$ med en gennemsnitlig afstand på 3 og en maksimal afstand på 4.

6. Endnu engang, sæt $\varepsilon = 4 - 1 = 3$ og løs. Solveren siger nu, at problemet er "infeasible" hvilket betyder, at vi ikke kan finde en løsning, som giver en maksimal afstand på mindre end eller lig med 3. Derfor findes der ikke flere ikke-dominerede løsninger til problemet.

### 9.2.2 Repræsentation af den efficiente front

I mange tilfælde med store problemer vil det være sådan at antallet af Pareto optimale løsninger er voldsomt stort og dermed alt for stort til at bidrage med beslutningsstøtte hvis alle præsenteres for en beslutningstager. Samtidig kan det også blive beregningsmæssigt ubelejligt at generere alle Pareto løsningerne, særligt hvis optimeringsproblemet, som skal løses gentagende gange, er meget tidskrævende. En måde man kan omgå dette på er ved at minimere en *vægtet sum* af objektfunktionerne:

$$\begin{aligned} &\min \ \lambda_1 f_1(x) + \lambda_2 f_2(x) \\ &\text{s.t.: } x \in \mathcal{X}, \end{aligned} \tag{9.5}$$

hvor $\lambda_1, \lambda_2 > 0$. Gøres dette, er man garanteret en Pareto optimal løsning og for konvekse problemer (for eksempel LP'er) kan alle Pareto optimale løsninger findes ved at variere $\lambda$. Der er dog to åbenlyse problemstillinger: Den første er, at ikke alle Pareto optimale løsninger til blandede heltalsprogrammer kan findes vha. den vægtede sum metode (det er kun de løsninger som ligger på den røde linje der forbinder nogle af de ikke-dominerede punkter i Figur 9.2 som kan findes vha. en vægtet sum). Den anden er, at det er svært *a priori* at vide, hvorledes forskellige vægte kan influere på de løsninger man ender med at få ud.

Derfor er det generelt ønskværdigt at kunne genere alle Pareto optimale løsninger til at starte med, for derefter at sortere i dem og udvælge de løsninger, som kunne være af interesse for en beslutningstager. En åbenlys metode ville være at benytte clustering teknikker på de generede ikke-dominerede løsninger. På den måde kan man bruge en clustering metode til at finde repræsentanter blandt de ikke-dominerede punkter som repræsenterer de rationelle kompromiser mellem objektfunktionerne. Repræsentationen kommer naturligvis til at afhænge af hvilket mål man bruger til at måle clustrenes kvaliteter. I Figur 9.4 er den efficiente front for et knapsack problem med 300 variabler illustreret. Den efficiente front består af 157 punkter, som er for mange at belemre en beslutningstager med. I Figur 9.4 er de efficiente løsninger markeret med røde krydser mens de punkter som er udvalgt som repræsentanter er markeret som blå firkanter. Der er brugt to forskellige lokationsbaserede clustering metoder: den $p$-median baserede som minimere den samlede afstand fra punkter til deres repræsentanter og den $p$-center baserede, som minimerer den maksimale afstand fra et punkt til dennes repræsentant.

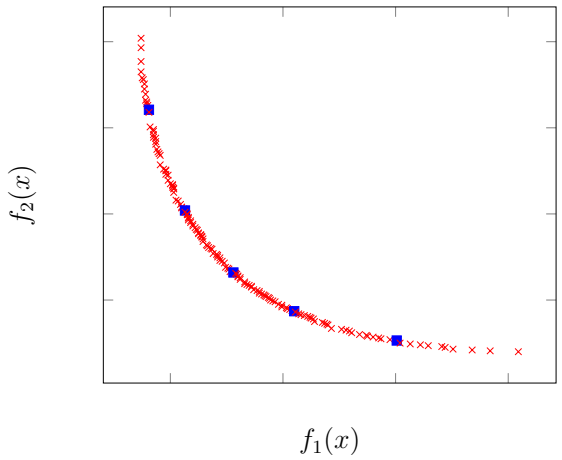
### 9.2.3 Generering af alle ekstremt-støttende punkter for bikriterie optimering

Som nævnt ovenfor, er en mulighed, når man ønsker at generere en repræsentation af Pareto-fronten, at benytte sig af vægtede sum-problemer og så variere vægtene. Der var dog ingen yderligere oplysninger omkring, hvorledes disse vægte skulle bestemmes. I indeværende afsnit, vil vi se en klassisk metode til at bestemme alle ekstremet-støttende punkter til et bikriterie problem – altså et multi-kriterie problem med kun to objektfunktioner. Metoden blev udviklet uafhængigt af Aneja og Nair 1979 and Cohon 1978 og benytter sig at tankegangen fra den *grafiske metode* til lineær programmering, blot i kriterierummet. Metoden er blandt andet kendt under navnet *non-inferior set estimation* algoritmen (NISE-algoritmen).

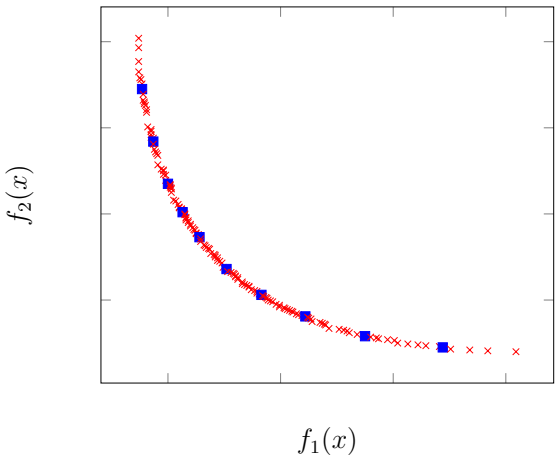
Den basale, underliggende idé bag NISE-algoritmen er at man først bestemmer de to optima for hver objektfunktion, uden at skele "for megetø" til den anden, for så derefter, at "fylde hullerne ud" mellem de efterfølgende generede efficiente løsninger. Dette gøres ved hele tiden at søge efter nye løsninger i en retning vinkelret på linjen mellem to"nabopunkter" (se Figur 9.5).

Selve algoritmen kan opskrives i en iterativ trinvis algoritmen som følger:

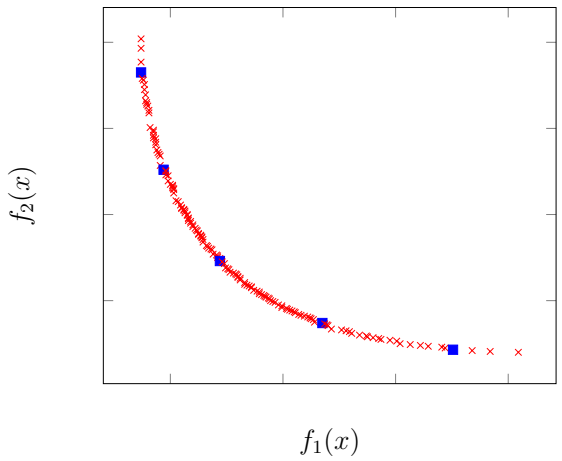
1. Bestem $x^{ul} \in \arg\min\{f_1(x) + \alpha f_2(x) : x \in \mathcal{X}\}$. Her er $\alpha > 0$ en lille positiv konstant. *ul* står for den geografiske placering objektvektoren af denne løsning:*upper left*.



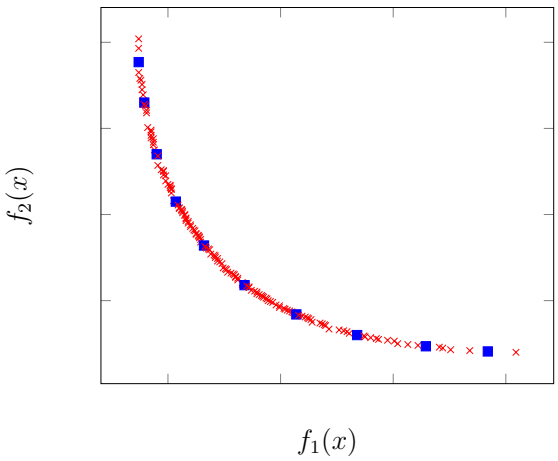

(a) Repræsentanter fundet vha. $p$-median baseret clustering, med $p = 5$.




(b) Repræsentanter fundet vha. $p$-median baseret clustering, med $p = 10$.



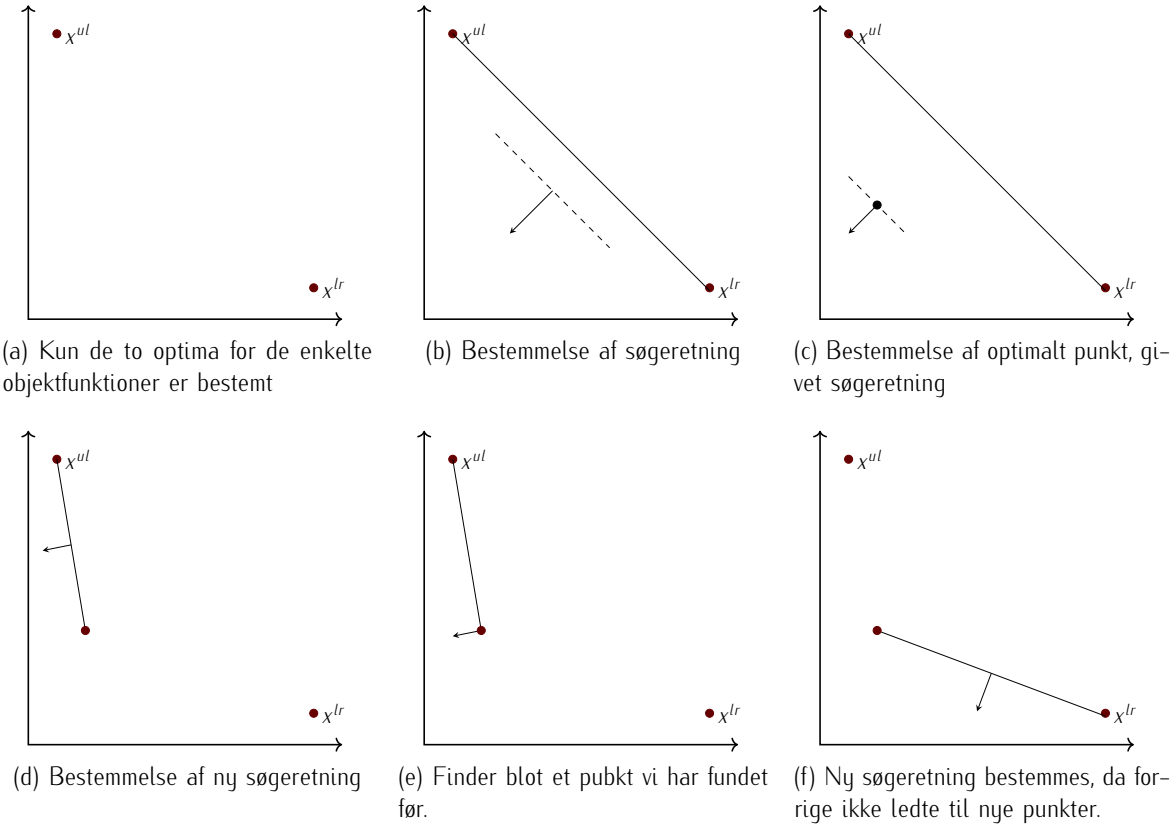

(c) Repræsentanter fundet vha. $p$-center baseret clustering, med $p = 5$.




(d) Repræsentanter fundet vha. $p$-center baseret clustering, med $p = 10$.

**Figur 9.4: Illustration af forskellige clustering algoritmer anvendt på den efficiente front af et bikriterie problem.**

2. Bestem $x^{lr} \in \arg\min\{f_2(x) + \alpha f_1(x) : x \in \mathcal{X}\}$. Her er $\alpha > 0$ en lille positiv konstant. *lr* står for den geografiske placering objektvektoren af denne løsning:*lower right*.
3. Sæt nu $y^{ul} = (f_1(x^{ul}), f_2(x^{ul}))$ og $y^{lr} = (f_1(x^{lr}), f_2(x^{lr}))$
4. Hvis $y^{ul} = y^{lr}$ stoppes algoritmen, da der kun er én Pareto optimal løsning. Ellers fortsættes til næste trin.
5. Sæt $\hat{\mathcal{Y}}_N = \{y^{ul}, y^{lr}\}$, $y^+ = y^{ul}$ og $y^- = y^{lr}$.




(a) Kun de to optima for de enkelte objektfunktioner er bestemt

(b) Bestemmelse af søgeretning

(c) Bestemmelse af optimalt punkt, givet søgeretning

(d) Bestemmelse af ny søgeretning

(e) Finder blot et pubkt vi har fundet før.

(f) Ny søgeretning bestemmes, da forrige ikke ledte til nye punkter.

**Figur 9.5: Illustration af hvordan NISE-algoritmen søger i kriterierummet efter nye uopdagede løsninger.**

6. Tjek om $y^+ = y^{lr}$. I så tilfælde, returner $\hat{\mathcal{Y}}_N$ som en repræsentation af $\mathcal{Y}_N$. Ellers fortsæt til næste trin.
7. Bestem søgeretningen ved at sætte $\lambda_1 = (y_2^+ - y_2^-)/(y_1^- - y_1^+)$ og $\lambda_2 = 1$.
8. Løs nu (9.5) og lad $x^*$ være en optimal løsning og $y^* = (f_1(x^*), f_2(x^*))$.
9. Sæt $f_\lambda = \lambda_1 f_1(x^*) + \lambda_2 f_2(x^*)$. Hvis $f_\lambda < \lambda_1 y_1^+ + \lambda_2 y_2^+$ indsættes $y^*$ mellem $y^+$ og $y^-$ i $\hat{\mathcal{Y}}_N$. Hvis ikke, sættes $y^+ = y^-$.
10. Sæt $y^-$ lig med punktet til højre for $y^+$ i $\hat{\mathcal{Y}}_N$. Gå tilbage til trin 6..

Denne algoritme fortsætter indtil man den har tjekket alle parvise naboløsninger, og konstateret at der ikke ligger nogle nye løsninger imellem disse.

*Eksempel* 9.4. Vi vil her udføre en række iterationer af NISE-algoritmen på følgende lille eksempel: En beslutningstager skal vælge en række aktiver at investere i. Beslutningstageren kan vælge aktiver fra mængden $I$, og for hvert projekt $i \in I$ er

givet en forventet pay-off $\pi_i$ og en "risikoscore" givet ved $r_i$. Desuden har beslutningstageren givet et minimumsniveau for samlet pay-off givet ved $b$. Slutteligt er der også for hvert projekt $i \in I$ givet en "miljø-påvirksningsscore", her symboliseret ved $s_i$. Data for 10 mulige aktiver er givet i følgende tabel:

| | Aktiver | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Risikoscore | 19 | 11 | 17 | 20 | 11 | 22 | 22 | 6 | 6 | 24 |
| Miljø-påvirkningsscore | 7 | 16 | 11 | 9 | 16 | 11 | 6 | 24 | 17 | 9 |
| Pay-off | 13 | 20 | 24 | 16 | 7 | 23 | 22 | 13 | 14 | 24 |
| Minimumsniveau for payoff: 100 | | | | | | | | | | |

Målet for beslutningstageren er at opnå en forståelse af de trade-offs der er mellem samlet pay-off og bæredygtighed. Vi vil med dette formål for øje definere binære binære variabler, $x_i$, til at vælge hvilke aktiver der skal udvælges:

$$x_i = \begin{cases} 1, & \text{hvis aktiv } i \in I \text{ vælges} \\ 0, & \text{ellers} \end{cases}$$

Med denne definition opnår vi med relativ lille indsats frem til følgende bikriterie knapsack problem som beskriver beslutningstagerens problemstilling

$$\begin{aligned} \min \quad & \sum_{i \in I} r_i x_i \\ \min \quad & \sum_{i \in I} s_i x_i \\ \text{s.t.:} \quad & \sum_{i \in I} \pi_i x_i \geq b \\ & x_i \in \{0, 1\}, \qquad \forall i \in I \end{aligned}$$

Med dette i mente kan vi begynde at bruge NISE algoritmen til at bestemme de ekstremt støttende punkter af den efficiente front for det ovenstående bikriterie-problem.

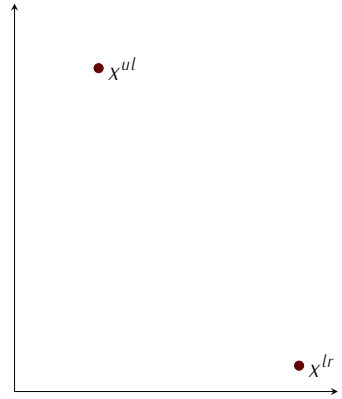
### Iteration 1 af NISE

1. Først bestemmes $x^{ul}$ ved at minimere risikoen. Her sættes $\alpha = 10^{-6}$. Dette leder til, at vi skal vælge aktiverne 2, 3, 5, 7, 8 og 9. Den totale risiko af denne løsning er 73 mens den totale miljøpåvirkning er på 90.
2. Nu bestemmes $x^{lr}$ igen med $\alpha = 10^{-6}$. Således skal aktiverne 1, 3, 6, 7 og 10 vælges, hvilket leder til en total risiko på 104 og en total miljøpåvirkning på 44. Et plot af de to fundne punkter kan ses i Figur 9.6a
3. $y^{ul} = (73, 950)$, $y^{lr} = (104, 44)$
4. Vi kan konkludere at $y^{ul} \neq y^{lr}$ hvorved der er mere end én Pareto optimal løsning, og vi fortsætter derfor NISE algoritmen til næste trin.
5. $\hat{\mathcal{Y}}_N = \{(73, 90), (104, 44)\}$, $y^+ = (73, 90)$ og $y^- = (104, 44)$.
6. Vi ved at $y^+ = (73, 90)$ og $y^{lr} = (104, 44)$ hvorved $y^+ \neq y^{lr}$. Derfor fortsættes til næste trin.
7. Vi bestemmer nu søgeretningen ved at sætte $\lambda_1 = (90 - 44)/(104 - 73) \approx 1,484$ og $\lambda_2 = 1$.
8. Vi løser nu det vægtede sum-problem med ovenstående vægte og får en optimal løsning som består af aktiverne 2, 3, 6, 7 og 9. Objektfunktionsværdien for den vægtede sum er 176,742, den samlede risiko er på 78 mens miljøpåvirkningen er 61. Dermed er $y^* = (78, 61)$.
9. Vi sætter $f_\lambda = 176,742$. Da $f_\lambda < 1,484 \cdot 73 + 1 \cdot 90 = 198,332$ hvorfor vi indsætter $y^*$ i $\hat{\mathcal{Y}}_N$ således at $\hat{\mathcal{Y}}_N = \{(73, 90), (78, 61), (104, 44)\}$.
10. Da $y^+ = (73, 90)$ skal vi nu opdatere $y^-$ til punkt til højre for dette i $\hat{\mathcal{Y}}_N$, hvilket vil sige, at $y^- = (78, 61)$. Dette afslutter iteration nummer 1, og vi skal starte på iteration nummer 2 fra trin 6. Et plot af $\hat{\mathcal{Y}}_N$ kan ses i Figur 9.6b.

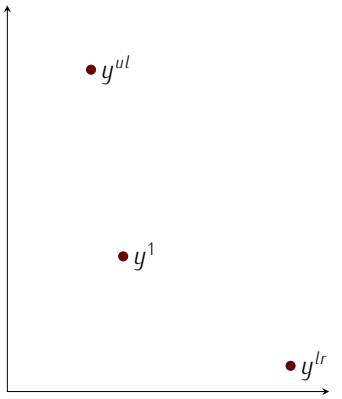
### Iteration 2 af NISE

Vi skal her søge mellem punkterne $y^{ul}$ og $y^1$ i Figur 9.6.

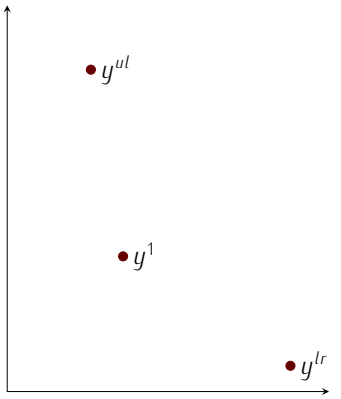
6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (73, 90)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.
7. Vi bestemmer søgeretningen til $\lambda_1 = (90 - 61)/(78 - 73) = 5.8$ og $\lambda_2 = 1$.



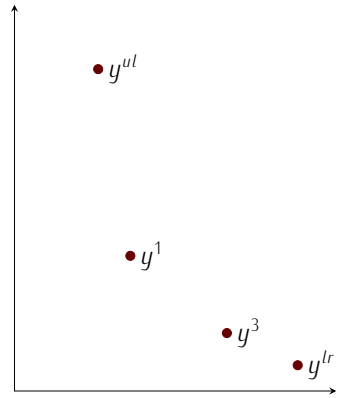

(a) Plot af $\hat{\mathcal{Y}}_N$ efter trin 1 og 2 i første iteration af NISE algoritmen



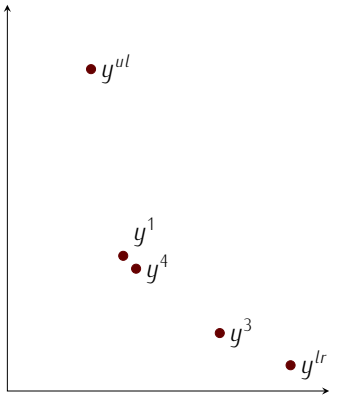

(b) Plot af $\hat{\mathcal{Y}}_N$ efter første iteration af NISE algoritmen



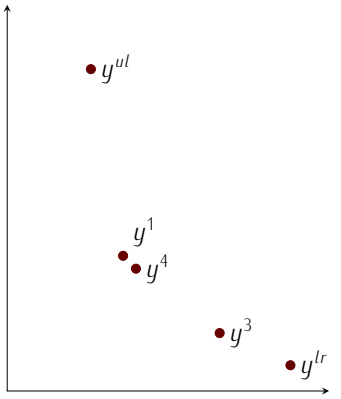

(c) Plot af $\hat{\mathcal{Y}}_N$ efter anden iteration af NISE algoritmen



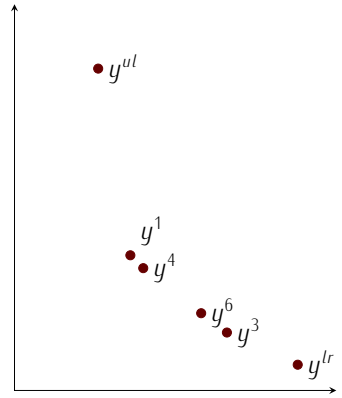

(d) Plot af $\hat{\mathcal{Y}}_N$ efter tredje iteration af NISE algoritmen



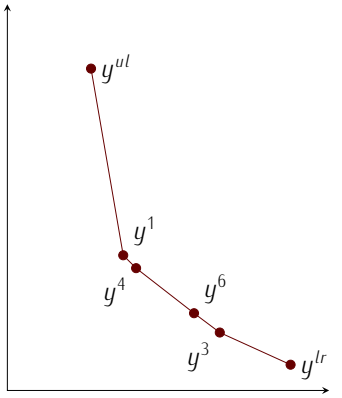

(e) Plot af $\hat{\mathcal{Y}}_N$ efter fjerde iteration af NISE algoritmen



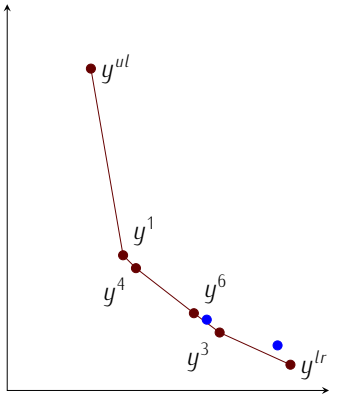

(f) Plot af $\hat{\mathcal{Y}}_N$ efter femte iteration af NISE algoritmen




(g) Plot af $\hat{\mathcal{Y}}_N$ efter sjette iteration af NISE algoritmen




(h) Plot af $\hat{\mathcal{Y}}_N$ efter sidste iteration af NISE algoritmen




(i) Plot af $\mathcal{Y}_N$ inklusiv ikke-støttende Pareto optimale løsninger (markeret med blå).

**Figur 9.6: Overblik over udviklingen i NISE algoritmens output. Punkterne er nummereret efter hvilken iteration de er fundet i.**

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 2, 3, 5, 7, 8 og 9 med en objektfunktionsværdi på 513,4. Løsningen har total risiko på 73 og en miljøpåvirkning på 90. Vi sætter $y^* = (73, 90)$ (læg mærke til, at vi allerede nu ved, at dette ikke var en frugtbar iteration, da vi allerede kender dette punkt).

9. Sæt nu $f_\lambda = 513,4$. Da $f_\lambda = 513,4 \not< 5,8 \cdot 73 + 1 \cot 90 = 513,4$ skal $y^*$ ikke indsættes i $\hat{\mathcal{Y}}_N$. I stedet skal vi nu flytte fokus for vores søgning ved at sætte $y^+ = y^-$. Det vil sige, at nu er $y^+ = (78, 61)$.

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$ hvilket vil sige, at nu skal $y^- = (93, 49)$. Dette afslutter iteration 2 og et plot kan ses i Figur 9.6c.

### Iteration 3 af NISE

Vi skal her søge mellem punkterne $y^1$ og $y^{lr}$ i Figur 9.6.

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (78, 61)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.

7. Vi bestemmer søgeretningen til $\lambda_1 = (61 - 44)/(104 - 78) \approx 0,6538$ og $\lambda_2 = 1$.

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 1, 2, 3, 7 og 10 med en objektfunktionsværdi på 109,808. Løsningen har total risiko på 93 og en miljøpåvirkning på 49. Vi sætter $y^* = (93, 49)$.

9. Sæt nu $f_\lambda = 109,808$. Da $f_\lambda = 109,808 < 0,6538 \cdot 78 + 1 \cot 61 = 111,999$ skal $y^*$ indsættes i $\hat{\mathcal{Y}}_N$ hvorved $\hat{\mathcal{Y}}_N = \{(73, 90), (78, 61), (93, 49), (104, 44)\}$.

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$ hvilket vil sige, at nu skal $y^- = (80, 59)$. Dette afslutter iteration 3 og et plot kan ses i Figur 9.6d.

### Iteration 4 af NISE

Vi skal her søge mellem punkterne $y^1$ og $y^3$ i Figur 9.6.

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (78, 61)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.

7. Vi bestemmer søgeretningen til $\lambda_1 = (61 - 49)/(93 - 78) = 0,8$ og $\lambda_2 = 1$.

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 2, 3, 7, 9 og 10 med en objektfunktionsværdi på 123. Løsningen har total risiko på 80 og en miljøpåvirkning på 59. Vi sætter $y^* = (80, 59)$.

9. Sæt nu $f_\lambda = 123$. Da $f_\lambda = 139t < 0,8 \cdot 78 + 1 \cot 61 = 123,4$ skal $y^*$ indsættes i $\hat{\mathcal{Y}}_N$ hvorved denne bliver $\hat{\mathcal{Y}}_N = \{(73, 90), (78, 61), (80, 59), (93, 49), (104, 44)\}$.

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$ hvilket vil sige, at nu skal $y^- = (80, 59)$. Dette afslutter iteration 4 og et plot kan ses i Figur 9.6e.

### Iteration 5 af NISE

Vi skal her søge mellem punkterne $y^1$ og $y^4$ i Figur 9.6.

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (78, 61)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.

7. Vi bestemmer søgeretningen til $\lambda_1 = (61 - 59)/(80 - 78) = 1$ og $\lambda_2 = 1$.

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 2, 3, 6, 7 og 9 med en objektfunktionsværdi på 139. Løsningen har total risiko på 78 og en miljøpåvirkning på 61. Vi sætter $y^* = (78, 61)$ (læg igen mærke til, at vi allerede nu ved, at dette ikke var en frugtbar iteration, da vi allerede kender dette punkt).

9. Sæt nu $f_\lambda = 139$. Da $f_\lambda = 139 \not< 1 \cdot 78 + 1 \cot 61 = 139$ skal $y^*$ ikke indsættes i $\hat{\mathcal{Y}}_N$. I stedet skal vi sætte $y^+ = y^-$ hvilket vil sige, at $y^+ = (80, 49)$.

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$ hvilket vil sige, at nu skal $y^- = (93, 49)$. Dette afslutter iteration 5 og et plot kan ses i Figur 9.6f.

### Iteration 6 af NISE

Vi skal her søge mellem punkterne $y^4$ og $y^3$ i Figur 9.6.

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (80, 59)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.

7. Vi bestemmer søgeretningen til $\lambda_1 = (59 - 49)/(93 - 80) \approx 0,7692$ og $\lambda_2 = 1$.

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 3, 4, 7, 9 og 10 med en objektfunktionsværdi på 120,462. Løsningen har total risiko på 89 og en miljøpåvirkning på 52. Vi sætter $y^* = (89, 52)$.

9. Sæt nu $f_\lambda = 120,462$. Da $f_\lambda = 120,462 < 0,7692 \cdot 80 + 1 \cot 59 \approx 120,5385$ skal $y^*$ indsættes i $\hat{\mathcal{Y}}_N$ hvorved denne bliver

$$\hat{\mathcal{Y}}_N = \{(73, 90), (78, 61), (80, 59), (89, 52), (93, 49), (104, 44)\}$$

.

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$ hvilket vil sige, at nu skal $y^- = (89, 52)$. Dette afslutter iteration 6 og et plot kan ses i Figur 9.6g.

Det viser sig, at der ikke findes flere ekstremt støttende punkter på denne Pareto front. De næste iteration leder efter nye punkter mellem $y^4$ og $y^6$, så mellem $y^6$ og $y^3$ og slutteligt mellem $y^3$ og $y^{lr}$. Denne sidste iteration skal her gengives for at vise hvorledes NISE algoritmen terminerer

### Sidste iteration af NISE

Vi skal her søge mellem punkterne $y^3$ og $y^{lr}$ i Figur 9.6. Her er der i forrige iteration opdateret så $y^+ = y^3 = (93, 49)$ og $y^- = y^{lr} = (104, 44)$.

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette ikke er tilfældet, da $y^+ = (93, 44)$ og $y^{lr} = (104, 44)$. Derfor fortsætter vi til trin 7.

7. Vi bestemmer søgeretningen til $\lambda_1 = (49 - 44)/(104 - 93) \approx 0,4546$ og $\lambda_2 = 1$.

8. Vi løser det vægtede sum-problem og får en løsning som består af aktiverne 1, 3, 6, 7 og 10 med en objektfunktionsværdi på 91,273. Løsningen har total risiko på 104 og en miljøpåvirkning på 44. Vi sætter $y^* = (104, 44)$.

9. Sæt nu $f_\lambda = 91,273$. Da $f_\lambda = 120,462 \not< 0,4546 \cdot 93 + 1 \cot 49 \approx 91,273$ skal $y^*$ ikke indsættes i $\hat{\mathcal{Y}}_N$. I stedet sættes $y^+ = y^- = y^{lr} = (104, 44)$

10. Vi skal også opdatere $y^-$ til punktet til højre for $y^+$. Dette kan vi dog ikke, da der ikke er noget til højre for $y^{lr}$. Vi går nu til trin 6 igen

6. Vi tjekker om $y^+ = y^{lr}$ og konstaterer, at dette faktisk tilfældet, da $y^+ = (104, 44)$ og $y^{lr} = (104, 44)$. Derfor terminerer algoritmen og vi returnerer løsningerne $\hat{\mathcal{Y}}_N = \{(73, 90), (78, 61), (80, 59), (89, 52), (93, 49), (104, 44)\}$

I dette lille eksempel, er der ikke vundet enormt meget ved kun at finde de ekstremt støttende punkter, da der faktisk kun er 2 ikke-støttende Pareto optimale løsninger til det lille eksempel. Dog vil det generelt set kunne forventes, at mængden af ekstremt støttende Pareto optimale løsninger udgør en relativt lille del af den samlede mængde af Pareto optimale løsninger, hvorved man opnår opnår en både lille og *spredt* repræsentation af den fulde efficiente front ved at benytte sig af NISE-algoritmen.

# Appendices

# A Forudsætninger og matematisk notation

Formålet med dette kapitel er at føre læseren frem til et punkt, hvor de resterende kapitler kan læses uden, at den matematiske symbolik står i vejen for forståelsen. Er man vant til at læse og studeren matematik og kender til logiske kvantorer som for eksempel $\vee$, $\wedge$, $\exists$, $\forall$ og $\Rightarrow$ kan man springe sektion A.1 over. Er man tillige velfunderet i brugen af $\sum$ (sumtegn) til repræsentation af summer, kan sektion A.2 tillige udelades. Slutteligt vil nogle få resultater fra lineær og lineær blandet heltalsoptimering bliver gennemgået overfladiske. Er læseren bekendt med teorien omkring disse emner, kan hele Bilag A udelades og man kan starte direkte med kapitel 1.

Igennem denne bog vil en række matematiske symboler blive benyttet, når det skaber en mere præcis formulering af et givent udsagn. Det er vigtigt at holde sig for øje, at matematisk notation kun skal benyttes, når denne gør det mere klart, hvad en given sætning betyder. Det vil sige, at hvis det er muligt at give en tilstrækkeligt præcis verbal beskrivelse af et givent udsagn, så er det ofte at foretrække frem for en matematisk beskrivelse. Nogle gange er det dog nødvendigt med matematikken.

## A.1 Matematiske symboler og deres definitioner

Dette afsnit indeholder en liste (se nedenfor) med matematiske og logiske symboler samt en definition af disse. Dette er for at yde en ekstra service for læseren, som kunne finde en sådan gennemgang nyttig for at læse resten af denne bog.

| Symbol | Forklaring |
|---|---|
| $\{\}$ | Notation for en mængde. De "krøllede" parenteser bruges til at notere en mængde. En mængde er en samling af *forskellige* objekter. En mængde kan for eksempel være tallene fra 1 til 4 ($\{1, 2, 3, 4\}$) eller en samling af søde dyr ($\{\text{hund}, \text{koala}, \text{dovendyr}\}$). Man kan også mere abstrakt definere en mængde ved hjælp af følgende notation $\{i : P(i)\}$, som skal læses som "mængden af alle elementer $i$ *hvorom der gælder*, at *udsagnet* $P(i)$ er sandt". Det kunne for eksempel (for $a < b$) være $\{x : a \leq x \leq b\}$, som her angiver alle tal $x$ mellem $a$ og $b$, som altså er lig med intervallet mellem $a$ og $b$. Nogle gange specificerer man hvilke typer af elementer man tænker på, på venstres side af kolon i mængden. Hvis man for eksempel er på udkig efter alle *heltal* som ligger mellem $a$ og $b$ kan dette skrives som $\{x \in \mathbb{Z} : a \leq x \leq b\}$ (se definition af $\mathbb{Z}$ længere nede i denne liste).<br>To mængder er lig med hinanden, hvis de indeholder de samme elementer ($\{1, 2, 3\} = \{2, 3, 1\}$) og den *tomme mængde* skrives som $\emptyset = \{\}$. |
| $\in$ | Dette symbol betyder "element i". Vi ved for eksempel, at $2 \in \{x : 0 \leq x \leq 4\}$. Altså, tallet 2 er et element i mængden af alle tal, som ligger mellem 0 og 4. Man benytter $\notin$-symbolet til at indikere, at et element *ikke* er et element i en mængde: $4 \notin \{1, 2, 3\}$. |
| $\subseteq$ | Symbol som angiver at én mængde er en *delmængde* af en anden. For eksempel haves at $\{1, 2\} \subseteq \{1, 2, 3, 4\}$ idet hvert element i mængden til venstre for $\subseteq$ også er et element i mængden til højre for symbolet. Man kan bruge symbolet $\not\subseteq$ til at indikere, at én mængde ikke er indeholdt i en anden: $\{0, 1, 2\} \not\subseteq \{1, 2, 3, 4\}$. |
| $\lvert \cdot \rvert$ | Antallet af elementer i en mængde kaldes *kardinaliteten*. For en mængde $S$ udtrykkes antallet af elementer i $S$ som $\lvert S \rvert$. For $S = \{1, 5, 8\}$ haves at $\lvert S \rvert = 3$. Man har naturligvis at $\lvert \emptyset \rvert = 0$.<br>Man benytter også denne notation til at angive den *absolutte værdi* af et tal. Altså haves, at $\lvert -2 \rvert = 2$, $\lvert -42 \rvert = 42$ og $\lvert 7 \rvert = 7$. |
| $\mathbb{N}$ | Mængden af de naturlige tal: $\mathbb{N} = \{1, 2, 3, 4, \dots\}$. Hvis tallet 0 skal inkluderes, skrives ofte $\mathbb{N}_0$, som altså så skal forstås som mængden $\{0, 1, 2, 3, \dots\}$. |
| $\mathbb{Z}$ | Mængden af alle heltal: $\mathbb{Z} = \{\dots, -3, -2 - 1, 0, 1, 2, 3, \dots\}$. Det ses straks, at $\mathbb{N} \subseteq \mathbb{Z}$. |

$\mathbb{Q}$ Mængden ad rationelle tal. Det vil sige alle de tal, der kan skrives som et ratio mellem to heltal: $\mathbb{Q} = \{r : r = \frac{p}{q}, p, q \in \mathbb{Z}\}$. Det ses igen umiddelbart at $\mathbb{Z} \subseteq \mathbb{Q}$.

$\mathbb{R}$ Mængden af de reelle tal. De reelle tal udvider de rationelle tal med de såkaldt irrationelle tal som for eksempel $\pi$ og $\sqrt{2}$. De reelle tal kan repræsenteres som en *tallinje* fra (ikke-inklusiv) $-\infty$ til $\infty$ (ikke-inklusiv).

$\sum$ Symbol for summer. For eksempel kan summen $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$ skrives mere kompakt som $\sum_{i=1}^{10} i$.

$\forall$ Logisk kvantor, som skal læses som "for alle". Man bruger ofte dette symbol inden for prescriptive analytics i forbindelse med grupper af begrænsninger. Hvis man ønsker en begrænsning skal gælde for alle elementer i en mængde $S$, kan man skrive $\forall i \in S$.

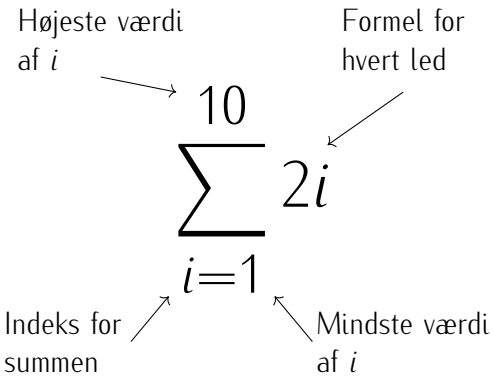
$\exists$ Logisk kvantor, som skal læses som "der eksisterer". Man kan med denne logiske kvantor koncist skrive sætningen "der eksisterer et element i mængden $S$" som "$\exists i \in S$".

$\wedge$ Operator, der skal læses som det logiske "og". Hvis man for eksempel har to variabler $x$ og $y$ og man ønsker at kommunikere, at både $x$ og $y$ skal være lig med 2, kan man skrive "$x = 2 \wedge y = 2$".

$\vee$ Operator, der skal læses som det logiske "eller". Her skal "eller" forstås som ikke-ekskluderende. Hvis igen to variabler $x$ og $y$ er givne, og man ønsker at kommunikere at enten skal $x$ være lig med 2 eller også (ikke-ekskluderende) skal $y$ være lig med 2, så kan man skrive $x = 2 \vee y = 2$. Dette betyder at enten er $x = 2$ og $y$ antager en anden værdi, eller $y = 2$ og $x$ antager en anden værdi, eller også er begge variabler lig med 2.

$\Rightarrow$ Denne operatorer en en såkaldt implikation og skal læses som "medfører". For eksempel kan sætningen "Hvis $x$ er større end eller lig med nul, strengt mindre end et og et heltal, da medfører det, at $x = 0$" skrives kort som "$x \geq 0 \wedge x < 1 \wedge x \in \mathbb{Z} \Rightarrow x = 0$".

Som med flere af de andre symboler findes der også en *ikke*-version af implikationen: $\not\Rightarrow$. Her skal $A \not\Rightarrow B$ læses som "$A$ medfører ikke $B$". Generelt gælder der ikke, at hvis $A \Rightarrow B$ så $B \Rightarrow A$. Som eksempel tag $A =$ "Det regner" og $B =$ "Vejen er våd"'. Her haves åbenlyst at $A \Rightarrow B$, men man kan ikke konkludere, at blot fordi vejen er våd, så må det også regne. Det kunne skyldes så meget andet. Altså haves, at $A \Rightarrow B$ men $B \not\Rightarrow A$.




**Figur A.1: Opdeling og forklaring af delelementerne i en sum opskrevet ved hjælp af $\sum$-notation.**

| | |
|---|---|
| $\Leftrightarrow$ | Dette symbol er en såkaldt bi-implikation, og skal læses som "hvis, og kun hvis". Hvis $A \Leftrightarrow B$ betyder det at $A \Rightarrow B \wedge B \Rightarrow A$. |

## A.2 Summer i kompakt form

I denne bog spiller summer en særdeles central rolle og det er derfor vigtigt, at læseren er fortrolig med den matematiske notation.

Formålet med enhver introduktion af matematisk notation er at øge læsbarheden og/eller muligheden for at udtrykke sig præcist. Det er også tilfældet med $\sum$-notationen ($\sum$ er en kapitæliseret version af det grænske bogstav $\sigma$ (sigma), hvorfor det også kaldes "store-sigma-notation") for summer. Summer med mange led, kan hurtigt blive besværlige at overskue. Tag for eksempel summen af alle lige tal som er større end eller lig med 0 og mindre end eller lig end 20 på følgende vis

$$0 + 2 + 4 + 6 + 8 + +10 + 12 + 14 + 16 + 18 + 20 \tag{A.1}$$

Denne sum kan man skrive kompakt på følgende måde

$$\sum_{i=0}^{10} 2i \tag{A.2}$$

Notationen skal læses som følger: "summen af $2i$ for alle $i$, som starter med $i = 0$ og slutter med $i = 10$". For en opdeling af summen i ligning (A.2) bedes læseren orientere sig i Figur A.1. Det første led i summe opnås ved at sætte $i = 0$ (den mindste værdi for indekset) og så evaluere formlen: $20 = 2 \cdot 0 = 0$ hvilket stemmer overens med første led i summen i (A.1). Herefter evalueres formlen for

$i = 1$ hvilket giver næste led i summen: $2i = 2 \cdot 1 = 2$. Dette stemme igen overens med summen i (A.1). Således fortsættes for alle heltallige værdier af $i$ indtil $i$ når sin højeste værdi, $i = 10$, som giver sidste led i summen $2i = 2 \cdot 10 = 20$. Dermed fås, at

$$0 + 2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20 = \sum_{i=0}^{10} 2i$$

Man siger, at man "summerer udtrykket $2i$ over alle $i$ fra $i = 1$ til $i = 10$".

For summer er der flere måder at udtrykke hvilke værdier indekset $i$, skal kunne tage. Når man ønsker at summe over alle heltalsværdier mellem en nedre og en øvre grænse, som i eksemplet ovenfor, er det ofte belejligt at benytte den anførte notation. Men hvis man nu ønsker at summe over en *indeksmængde*, som ikke er så regulær, som i eksemplet, kan man benytte mængdenotation for indekset. Tag for eksempel eksemplet hvor det ønskes at summe følgende *delmængde* af de hele tal $\{2, 8, 10, 18\}$. Da kan man skrive summen som

$$2 + 8 + 10 + 18 = \sum_{i \in \{1,4,5,9\}} 2i$$

Man kan læse denne måde at opskrive summe på som :"summér udtrykket $2i$ for alle indeks $i$ i mængden $\{2, 8, 10, 18\}$". Denne notation er ofte snedig, når man har mængder, der for eksempel ikke er givet ved en sammenhængende række af tal.

Man skal her gøre sig klart, at indekset $i$ blot er en stedfortræder for tallene i mængden eller mellem den nedre og øvre grænse. Man kunne for så vidt lige så godt have bruge bogstavet $j$, $k$ eller et helt andet symbol. Det vil sige, at

$$\sum_{i=0}^{10} 2i = \sum_{j=0}^{10} 2j = \sum_{✠=0}^{10} 2✠$$

### A.2.1 Summer over variabler med et og flere indeks

Når man arbejder med variabler af høje dimensioner, benytter man sig ofte af indeks på variablerne. Hvis man eksempelvis har en variabel $x \in \mathbb{R}^n$, så er dette en vektor i et $n$-dimensionelt rum, hvorfor der reelt er $n$ variabler udtrykt ved $x$. Den $i$'te indgang i vektoren $x$ vil oftest skrives som $x_i$. Det vil sige, at hvis $n = 10$ betyder det, at der er 10 indgange i vektoren $x$ og man skriver de individuelle

indgange som $x_1, x_2, x_3, \ldots, x_{10}$. Eller mere kompakt; $x_i$ for alle $i = 1, \ldots, 10$. Man kan summe disse 10 variabler sammen som følger:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = \sum_{i=1}^{10} x_i$$

Det skulle gerne her blive klart, at i tilfælde hvor $n$ er meget stor giver $\sum$-notationen en særdeles effektiv måde at opskrive summen.

Inden for optimering, er det ofte belejligt at benytte variabler, som har to eller flere indeks. Et eksempel, som mange er bekendte med er det lineære transportproblem hvor man benytter variabler $x_{ij}$, som angiver hvor meget af et givet produkt, der flyttes fra fabrik $i$ til kunde $j$. Hvis man tager et simpelt eksempel med fem fabrikker og 8 kunder, kan man udtrykke den samlede mængden, der transporteres fra fabrik nummer 1 på følgende vis

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} + x_{1,6} + x_{1,7} + x_{1,8}$$

Altså, første led angiver, hvad der flyttes fra fabrik 1 til kunde 1, næste led angiver hvor meget, der flyttes fra fabrik 1 til kunde 2 og så fremdeles. Når alle disse mængder lægges sammen haves den samlede mængde, der sendes ud af fabrik 1. Dette kan skrives kompakt på følgende måde

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} + x_{1,6} + x_{1,7} + x_{1,8} = \sum_{j=1}^{8} x_{1,j}$$

Det vil altså sige, at man fastholder det første indeks på $x_{ij}$-variablerne og så varieres det andet indeks.

Vil man derimod beregne dem samlede mængde der sendes til kunde 1, kan dette gøres som følger:

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} + x_{5,1}$$

Her er logikken den samme som ovenfor, hvor vi blot fastholder det andet indeks (kundeindekset) og varierer fabriksindekset. Denne sum kan naturligvis også skrives kompakt:

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} + x_{5,1} = \sum_{i=1}^{5} x_{i,1}$$

Læseren gøres nu igen opmærksom på, at bogstavet der benyttes som indeks, er irrelevant, men placeringen er vigtig. Det vil sige, at

$$\sum_{i=1}^{5} x_{i,1} = \sum_{j=1}^{5} x_{j,1} = \sum_{\beta=1}^{5} x_{\beta,1} \neq \sum_{i=1}^{5} x_{1,i} \tag{A.3}$$

Det vil sige, man kan skifte navn på indekset uden at ændre summen, men man kan ikke ændre placeringen af indekset uden at ændre summen. For at overbevise sig selv om rigtigheden af dette udsagn, opfordres læseren til at skrive summerne i (A.3) ud eksplicit (der er fem led i hver af summerne). Det er derfor vigtigt at holde sig for øje hvilke index, der bliver summet over, og ikke så vigtigt hvilket "navn" der bruges for indekset. Det kan nogle gange hjælpe på forståelsen at tænke på variabler med to indeks som en matrix af variabler. Det vil sige, at variablen $x_{ij}$ angiver variablen i den $i$'te række og den $j$'te søjle af en $n \times m$ matrix (hvor $n$ angiver antallet af rækker og $m$ antallet af søjler):

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix}$$

Det vil altså sige, at fastholdes det første indeks mens man summer over det andet indeks, så svarer det til at summe variablerne sammen i en given række. Omvendt, hvis andet indeks fastholdes mens man summer over det første, svarer det til at summe variabler i en given søjle.

Man er naturligvis ikke nødsaget til at blot at summe over et indeks; man kan uden problemer summe summer sammen. Hvis man kigger tilbage på eksemplet med 5 fabrikker og 8 kunder anført ovenfor, så kunne man være interesseret i at vide hvad den totale transporterede mængede var. En måde at gøre dette på er at betragte mængden sendt ud fra en givet fabrik $i$ (givet ved $\sum_{j=1}^{8} x_{ij}$) og så summe denne mængde sammen over alle de fem fabrikker. Det ville se ud som følger

$$\sum_{i=1}^{5} \left( \sum_{j=1}^{8} x_{ij} \right)$$

Når man arbejder med dobbelt-summer som den ovenfor anførte, plejer man at

udelade parentesen, da denne er underforstået og man skriver i stedet blot

$$\sum_{i=1}^{5}\sum_{j=1}^{8} x_{ij}$$

Når man har sådan en dobbelt, triple, eller endnu flere samlede sumtegn, er den nemmeste måde at overskue hvad disse summer betyder, at ekspandere summerne indefra og ud efter:

$$\begin{aligned}\sum_{i=1}^{5}\sum_{j=1}^{8} x_{ij} &= \sum_{i=1}^{5}\left(x_{i,1}+x_{i,2}+x_{i,3}+x_{i,4}+x_{i,5}+x_{i,6}+x_{i,7}+x_{i,8}\right)\\ &= (x_{1,1}+x_{1,2}+x_{1,3}+x_{1,4}+x_{1,5}+x_{1,6}+x_{1,7}+x_{1,8})\\ &+ (x_{2,1}+x_{2,2}+x_{2,3}+x_{2,4}+x_{2,5}+x_{2,6}+x_{2,7}+x_{2,8})\\ &+ \ldots\\ &+ (x_{5,1}+x_{5,2}+x_{5,3}+x_{5,4}+x_{5,5}+x_{5,6}+x_{5,7}+x_{5,8})\end{aligned}$$

Læseren kan nu notere sig, at man også kunne have målt den totale transporterede mængde i eksemplet ved at først betragte hvor meget, der sendes til en kunde $j$, for så at summe denne mængde sammen over alle kunderne. Det ville give summen

$$\sum_{j=1}^{8}\left(\sum_{i=1}^{5} x_{ij}\right)$$

Da den samlede mængde sendt i eksemplet nødvendigvis bliver nødt til at være den samme uanset om man betragter situationen fra fabrikkernes eller kundernes synspunkt, er det derfor etableret at

$$\sum_{j=1}^{8}\sum_{i=1}^{5} x_{ij} = \sum_{i=1}^{5}\sum_{j=1}^{8} x_{ij}$$

Det vil sige, at summe ikke afhænger af i hvilken rækkefølge sumtegnene står (hvilket passer fint med vores intuition om, at addition er kommutativ). Faktisk gælder der *for alle endelige summer* at rækkefølgen på sumtegnene kan ombyttes. Det vil sige, at for $n, m < \infty$ haves

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} = \sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij}$$

Som en sidste lille krølle, kan dobbeltsummer opfattes i samme struktur som illustreret i Figur A.1. For at indse dette, betragt dobbeltsummen

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} x_{ij}\right) \tag{A.4}$$

Lad nu $y_i = \sum_{j=1}^{m} x_{ij}$ hvorved (A.4) omskrives som

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} = \sum_{i=1}^{n} y_i$$

Herved er udtrykket reduceret til igen at beskrive "summér udtrykket $y_i$ for alle indeks $i$ fra $i = 1$ til $i = n$". For at opskrive summe eksplicit, skal udtrykket $y_i$ her udskrives som en sum i sig selv.

## A.3 Lineær programmering og blandet heltalsprogrammering

For at det er fuldstændig klart for læseren hvad der menes med lineære funktioner, gives her en formel definition af en sådan:

**Definition A.1** (Lineær funktion)**.** En funktion $f : \mathbb{R}^n \to \mathbb{R}$ er *lineær* hvis den opfylder følgende to krav

1. For $x, y \in \mathbb{R}^n$ skal der gælde, at $f(x + y) = f(x) + f(y)$
2. For $x \in \mathbb{R}^n$ og $\alpha \in \mathbb{R}$ skal der gælde, at $f(\alpha x) = \alpha f(x)$

Det følger umiddelbart fra denne definition, at $f$ er på formen $f(x) = a_1x_1 + a_2x_2 + \cdots a_nx_n$ for $a_i \in \mathbb{R}$ for alle $i = 1, \ldots, n$.

Det er altså ikke tilladt for en lineær funktion, at have for eksempel et produkt af to variabler, division med variabler eller eksempelvis udtryk på formen $e^x$.

Et lineært program er et optimeringsproblem på formen

$$\begin{aligned} \max\ & f(x) \\ \text{s.t. } & g(x) \leq b, \\ & x_i \geq 0, \qquad\qquad \text{for alle } i = 1, \ldots, n \end{aligned}$$

hvor objektfunktionen $f : \mathbb{R}^n \to \mathbb{R}$ og vektor-funktionen $g : \mathbb{R}^n \to \mathbb{R}^m$ er lineære og $b \in \mathbb{R}^m$ er en $m$-dimensional vektor. *Beslutningsvariablerne* $x_i$ er de

værdier, som man søger at finde, der kan maksimere *objektfunktione* $f$ givet at *begrænsningerne* $g(x) \leq b$ og $x_i \geq 0$ for alle $i = 1, \ldots, n$ er overholdt. Det siges, at en løsning $\tilde{x} \in \mathbb{R}^n$ er *brugbar* (feasible på engelsk) hvis $g(\tilde{x}) \leq b$ og $\tilde{x} \geq 0$. Det vil sige, hvis $\tilde{x}$ opfylder alle begrænsningerne til problemet.

I det $f$ og $g$ er lineære kan et lineært program også skrives på formen

$$\begin{aligned} &\max\ c^T x \\ &\text{s.t. } Ax \leq b, \\ &\qquad x_i \geq 0, \quad \text{for alle } i = 1, \ldots, n \end{aligned} \tag{A.5}$$

Hvor $c$ er en $n$-dimensional vektor og $A \in \mathrm{Mat}_{m\times n}(\mathbb{R})$ er en matrix med $m$ rækker og $n$ søjler. For nemheds skyld skrives "$x_i \geq 0$ for alle $i = 1, \ldots, n$" ofte blot som $x \geq 0$ hvilket indikerer, at hvert element i vektoren $x$ er ikke-negativt. Denne konvention vil også bruges her.

Man kan her notere sig, at man kan opskrive et ækvivalent lineært program hvor begrænsningerne er udtrykt ved ligheder. Dette ses i

**Proposition A.1.** Lad to lineære programmer være givet ved

$$\begin{aligned} P1 :\ &\max\ c^T x \\ &\text{s.t.: } Ax \leq b \\ &\qquad x \geq 0 \end{aligned} \qquad\qquad \begin{aligned} P2 :\ &\max\ c^T x \\ &\text{s.t.: } Ax + s = b \\ &\qquad x, s \geq 0 \end{aligned}$$

Lad nu $\tilde{x}$ være en optimal løsning til $P1$. Da er $\bar{x} = \tilde{x}$ og $\bar{s} = b - A\tilde{x}$ en optimal løsning til $P2$. Endvidere gælder, at hvis $(\bar{x}, \bar{s})$ er en optimal løsning til $P2$, da er $\tilde{x} = \bar{x}$ en optimal løsning til $P1$.

*Bevis.* Overlades til læseren. □

Beslutningsvariablerne som indførtes i programmet $P2$ i Proposition A.1 kaldes ofte *slack variabler* da de så at sige måler slacket mellem venstresiden og højre siden i begrænsningerne $Ax \leq b$.

Resultatet fra Proposition A.1 gør, at når man behandler et generelt/generisk lineært program, så er det oftest ikke vigtigt om man taler om et program med ligheds- eller ulighedsbegrænsninger. Derfor vil resten af denne bog ikke skelne mellem hvilken relation der står mellem venstre og højresiden i begrænsningerne.

Det er ikke altid så belejligt, at opskrive sine lineære programmer i den såkaldte matrixform som ovenfor anvist. I stedet kan det være belejligt at opskrive

begrænsningerne en anelse mere eksplicit ved hjælp af summer (for at gøre dette, skal man genkalde sig hvorledes man multiplicerer matricer og vektorer). Lad matricen $A$ være givet ved

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Da kan begrænsningerne $Ax \leq b$ opskrives på følgende måde

$$\begin{aligned} Ax &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\ &= \begin{pmatrix} a_{11}x_1 & a_{12}x_2 & \dots & a_{1n}x_n \\ a_{21}x_1 & a_{22}x_2 & \dots & a_{2n}x_n \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}x_1 & a_{m2}x_2 & \dots & a_{mn}x_n \end{pmatrix} \\ &= \begin{pmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \sum_{j=1}^{n} a_{2j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{mj}x_j \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} = b \end{aligned}$$

På denne måde bliver det formodentligt mere tydeligt, hvad der menes med notationen $Ax \leq b$, nemlig, at indgang $i$ i vektoren $Ax$ skal være mindre end eller lig med den $i$'te indgang i vektoren $b$, *for alle* $i = 1, \dots, m$. I den kommende sektion, vises det, hvorledes sådanne *systemer* af begrænsninger kan opskrives ved hjælp af "for alle" symbolet $\forall$.

### A.3.1 Systemer af lineære begrænsninger og $\forall$-tegnet

Det er ofte sådan, at en optimeringsmodel indeholder begrænsninger (uligheder og ligheder), som skal være gældende for en gruppe af enheder. Det kunne for eksempel være tilfældet, at "$x_i$ variablerne skal være binære *for alle* $i = 1, \dots, 10$" eller det kan være, at "kapacitetsbegrænsningerne skal være overholdt *for alle* produktionsfaciliteter".

For at tage udgangs punkt i et kendt eksempel, betragt igen eksemplet fra tidligere

$$\begin{pmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \sum_{j=1}^{n} a_{2j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{mj}x_j \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Denne matrix-notation er en anelse ufleksibel og fylder også en del på skrift. Det er her $\forall$-tegnet kommer til at spille en rolle, idet ulighederne ovenfor ved brug af $\forall$ kan udtrykkes som

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad \forall i = 1, \ldots, m \tag{A.6}$$

Matematikken i (A.6) skal læses på følgende måde (fra venstre mod højre): "summen over alle $j$ fra 1 til $n$ over $a_{ij}x_j$ skal være mindre end eller lig med $b_i$ *for alle* $i = 1, \ldots, m$". Man skal altså således tænke, at der er lige så mange begrænsninger i spil, som der er indeks efter $\forall$-tegner – i dette tilfælde er der $m$ begrænsninger. På denne måde kan det lineære program i matrix form fra (A.5) opskrives som

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j & & \\ \text{s.t.:} \quad & \sum_{j=1}^{n} a_{ij}x_j \leq b_i, & & \forall i = 1, \ldots, m \\ & x_j \geq 0, & & \forall j = 1, \ldots, n \end{aligned}$$

*Eksempel* A.1. Betragt tilfældet hvor en lineær programmeringsmodel har følgende tre begrænsninger

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 &\leq b_2 \\ a_{31}x_1 + a_{32}x_2 &\leq b_3. \end{aligned}$$

Disse tre begrænsninger kan opskrives mere kompakt på én linje på følgende måde

$$a_{i1}x_1 + a_{i2}x_2 \leq b_i, \quad \forall i = 1, 2, 3$$

Man kan altså tænke det på følgende måde: hver værdi af indekset specificeret efter $\forall$ generere en begrænsning, hvor indekset (i dette tilfælde $i$) fastholdes: én begrænsning hvor $i = 1$, én begrænsning hvor $i = 2$ og én begrænsning hvor $i = 3$.

Det forholder sig sådan, at man ofte bliver nødt til at specificere værdien af mere end et indeks efter for alle tegnet. Hvis man for eksempel har specificeret en variabel $x_{ij}$ der skal være ikke-negativ for alle kombinationer af $i = 1, \ldots, 5$ og $j = 1, \ldots, 8$ (som i eksemplet med transportproblemet ovenfor) kan dette gøres på følgende måde

$$x_{ij} \geq 0, \quad \forall i = 1, \ldots, 5, \ j = 1, \ldots, 8 \tag{A.7}$$

Dette skal læses som "$x_{ij}$ skal være ikke-negativ for alle *kombinationer af* $i$ og $j$ hvor $i = 1, \ldots, 5$ og $j = 1, \ldots, 8$". Det vil altså sige, at der faktisk er $5 \times 8 = 40$ begrænsninger gemt i (A.7).

Vi vil slutte denne generelle introduktion til lineære optimeringsproblemer af med en observation, der ofte overses af læsere, som ikke er så erfarne ud i optimering. Lad et generelt optimeringsproblem være givet ved

$$\begin{aligned} &\max \ f(x) \\ &\text{s.t.: } g(x) \leq b \end{aligned} \tag{A.8}$$

hvor $f : \mathbb{R}^n \rightarrow$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ og $b \in \mathbb{R}^m$ (det er her uden betydning om $f$ og $g$ er lineære eller ej. Det eneste krav er, at der eksistere en optimal løsning til (A.8)). Da gælder, at hvis $x^*$ er optimal til (A.8) da er $x^*$ også optimal til

$$\begin{aligned} &\max \ f(x) + k \\ &\text{s.t.: } g(x) \leq b \end{aligned} \tag{A.9}$$

hvor $k \in \mathbb{R}$ er en konstant. Det vil sige, at det at lægge en konstant til objektfunktionen i et optimeringsproblem, ikke ændrer på mængden af optimale løsninger. For at indse dette, betragtes følgende argument. Lad $x^*$ være en optimal løsning til (A.8) og antag for modstrid, at $x^*$ *ikke* er optimal for (A.9). Først konstateres, at $x^*$ er *brugbar* for (A.9) da $x^*$ er optimal til (A.8) og fordi begrænsningerne ikke er ændret fra (A.8) til (A.9).

Da $x^*$ ikke er optimal for (A.8) (per antagelse), må der eksistere en anden løsning $\tilde{x} \neq x^*$, som er brugbar for (A.9) og hvor $f(\tilde{x}) + k > f(x^*) + k$. Lader man nu konstanterne gå ud mod hinanden fås uligheden $f(\tilde{x}) > f(x^*)$. Men da $\tilde{x}$ også er brugbar for (A.8) haves modstriden da $\tilde{x}$ nu er strengt bedre end $x^*$, som var antaget optimal. Dermed er $x^*$ optimal til (A.9).

### A.3.2 Blandet heltalsoptimering

Lineær programmering kan bruges til at formulere mange overraskende forskellige problemstillinger i den virkelige verden, men der er og grænser for, hvad der kan formuleres lineært. Hvis man udvider konceptet lineær programmering til at tillade, at nogle af variablerne kun må antage heltallige værdier, så får man et endog meget stærkt modelleringsværktøj. Og så længe man kun benytter lineære begrænsninger og objektfunktioner findes der også relativt effektive løsningsprocedurer til at håndtere de resulterende optimeringsproblemer.

Et såkaldt blandet heltalsoptimeringsproblem er på formen

$$\begin{aligned} \max\ & c^T \\ \text{s.t.:}\ & g(x) \leq b \\ & x_j \in \mathbb{N}_0, && \forall j \in N^I \\ & x \geq 0 \end{aligned}$$

hvor $N^I$ er en delmængden af indeksmængden for variablerne, altså $N^I \subseteq \{1, 2, 3, \ldots, n\}$. *I*'et i notationen $N^I$ står for *integer* da mængden $N^I$ angiver hvilke variabler, der skal være heltallige. Hvis $N^I = \{1, 2, 3, \ldots, n\}$ hvorved alle variabler skal være heltallige, da kaldes problemet blot et heltalsproblem (man undlader "blandet").

Man kan desværre ikke blot løse at blandet heltalsprogram ved at se bort fra heltalskravet og så afrunde den resulterende løsning. På nuværende tidspunkt findes der ingen *polynomielle* algoritmer til at løse blandede heltalsprogrammer, hvorfor man i værste fald skal forvente, at beregningstiden stiger eksponentielt i størrelsen på inputtet (antallet af variabler og begrænsninger). Det vil sige, at hver gang én variabel tilføjes problemet, fordobles beregningstiden i værste tilfælde. I mange praktiske situationer, er dette dog ikke tilfældet! Vi vil i den resterende del af Bilag A antage, at både $f$ og $g$ er lineære funktioner. I dette tilfælde betegnes problemet et lineært blandet heltalsproblem eller MILP fra den engelske *Mixed Integer Lineær Programming*.

For at se, at en afrundet løsning selv i simple tilfælde ikke nødvendigvis leder en optimal løsning betragtes nu et eksempel.

*Eksempel* A.2. Betragt det lille lineære heltalsprogram

$$\begin{aligned} \max\ & 3x_1 + 4x_2 \\ \text{s.t.:}\ & 8x_1 + 14x_2 \leq 189 \\ & 15x_1 + 11x_2 \leq 240 \\ & x_1, x_2 \in \{0, 1, 2, 3 \dots\} \end{aligned}$$

Hvis man løser den *lineære relaksering* (linear relaxation), som fremkommer ved at fjerne heltalskravet og løse problemet som et lineært program, da opnås den optimale løsning $x^{\text{LP}} = (10.5, 7.5)$ med objektfunktionsværdien 61.5. De mulige løsninger der kan fremkomme ved afrunding er givet neden for sammen med deres status mht. brugbarhed og deres tilhørende objektfunktionsværdier

| Løsning | Status | Objektfunktionsværdi |
|---|---|---|
| $(x_1, x_2) = (11, 8)$ | Infeasible | — |
| $(x_1, x_2) = (10, 8)$ | Infeasible | — |
| $(x_1, x_2) = (11, 7)$ | Infeasible | — |
| $(x_1, x_2) = (10, 7)$ | Feasible | 58 |

Det vil altså sige, at den bedste løsning der kan opnås ved at afrunde løsningen fra den tilhørende lineære relaksering er $(x_1, x_2) = (10, 7)$ med en objektfunktionsværdi på 58. Den optimale løsning til problemet at givet ved $x^* = (9, 8)$ med en objektfunktionsværdi på 59.

Som det fremgår af Eksempel A.2 kan man ikke benytte den lineære relaksering til at løse et MILP direkte. Men man kan notere sig, at hvis man fjerner heltalskravene til variablerne og optimerer det resulterende lineære program, så opnår man et *optimistisk* bud på den optimale objektfunktionsværdi. Dette skyldes, at hvis begrænsninger fjernes, kan man vælge mellem flere brugbare løsninger, og dette giver mulighed for, at finde en (relakseret) løsning, som giver en bedre objektfunktionsværdi end heltalsprogrammet ville resultere i. Dette formaliseres i

**Proposition A.2.** Lad $z^{\text{LP}}_{\max} = \max\{c^T x : Ax \leq b, x \geq 0\}$ og $z^{\text{MILP}}_{\max} = \max\{c^T x : Ax \leq b, x_j \in \mathbb{N}_0 \forall j \in N^I, x \geq 0\}$. Da gælder, at $z^{\text{LP}} \geq z^{\text{MILP}}$.

Tilsvarende gælder der for $z^{\text{LP}}_{\min} = \min\{c^T x : Ax \leq b, x \geq 0\}$ og $z^{\text{MILP}}_{\min} = \max\{c^T x : Ax \leq b, x_j \in \mathbb{N}_0 \forall j \in N^I, x \geq 0\}$, at $z^{\text{LP}}_{\min} \leq z^{\text{MILP}}_{\min}$.

Det faktum at man overestimerer objektfunktionsværdien når man maksimerer den lineære relaksering (underestimerer i tilfælde af minimering) gør, at man kan løse et MILP ved hjælp af branch-and-bound som her kort skitseres:

Input: Et optimeringsproblem på formen $\max\{c^T x : Ax \leq b, x_j \in \mathbb{N}_0 \forall j \in N^I, x \geq 0\}$

Output: En optimal løsningsværdi $z^*$ eller et bevis for at ingen optimal løsning findes.

Step 0: Sæt en mængde af delproblemer $\mathcal{L}$ lig med det originale problem og sæt $LB = -\infty$.

Step 1: Hvis $\mathcal{L} \neq \emptyset$, vælg da et delproblem fra $\mathcal{L}$ og fjern det fra listen. Ellers gå til Step 6

Step2: Løs den lineære relaksering af delproblemet og sæt $z^{\text{LP}}$ lig med den resulterende objektfunktionsværdi. Hvis $z^{\text{LP}} = \infty$ gå til Step 6.

Step 3 - bounding: Hvis $z^{\text{LP}} \leq LB$ gå tilbage til Step 1 og vælg et nyt delproblem.

Step 4: Hvis alle variabler, som skal være heltallige(alle $x_j$ med $j \in N^I$), er heltallige i LP-løsningen, sæt da $LB = \max\{LB, z^{\text{LP}}\}$ og gå til Step 1. Ellers fortsæt til Step 5.

Step 5 - branching: Vælg en variabel $x_j$ hvor $j \in N^I$ og hvor $x_j$ antager en fraktionel værdi, $v \in \mathbb{Q} \setminus \mathbb{Z}$, i LP-relakseringen. Tilføje nu to nye delproblemer til $\mathcal{L}$: i det ene tilføjes begrænsningen $x_j \leq \lfloor v \rfloor$ til delproblemet og i det andet tilføjes begrænsningen $x_j \geq \lceil v \rceil$. Gå nu til Step 1.

Step 6: Hvis $LB = -\infty$ (infeasible) eller $z^{\text{LP}} = \infty$ (unbounded) findes ingen optimal løsning. Ellers er $z^* = LB$ en optimal løsningsværdi.

I tilfældet, hvor hver variabel skal være *binær* (det vil sige, at $x_j \in \{0, 1\}$ for alle $j \in \{1, \ldots, n\}$) haves, at man i Step 5 vil fikserer variablen $x_j$ til enten 0 eller 1. Dermed kan man præcist bestemme antallet af lineære programmer, der skal løses (i det værste tilfælde) til $2^{n+1}-1$, hvor $n$ er antallet af binære variabler. Det vil sige, at for $n = 2$ kan man risikere at skulle løse 7 lineære programmer. Dette antal stiger til 65.535 for $n = 15$. For et relativt lille MILP med $n = 100$ er dette steget til 2.535.301.200.456.460.000.000.000.000.000. Dette skal blot illustrere, at når

man bevæger sig fra lineær programmering til blandet heltalsprogrammering, stiger den forventede beregningstid markant! Heldigvis sorteres mange delproblemer fra i Step 3 i branch-and-bound algoritmen beskrevet ovenfor, og man vil sjældent se denne opførsel, hvis man har formuleres sine problemer fornuftigt.

# Bibliografi


Aneja, Y. P. og K. P. K. Nair (1979). "Bicriteria Transportation Problem". I: *Management Science* 25.1, s. 73–78. DOI: 10.1287/mnsc.25.1.73.

Balakrishnan, N., B. Render og R.M. Stair (2020). *Managerial decision modeling with spreadsheets (customized book for Aarhus University)*. Prentice Hall Press.

Balinski, M.L. (1965). "Integer programming: methods, uses, computations". I: *Management science* 12.3, s. 253–313.

Cohon, J. L. (1978). *Multiobjective Programming and Planning*. Academic Presse, London.

Cooper, L. (1963). "Location-allocation problems". I: *Operations research* 11.3, s. 331–343.

Dantzig, G., R. Fulkerson og S. Johnson (1954). "Solution of a large-scale traveling-salesman problem". I: *Operations Research* 2.4, s. 393–410.

Davenport, T. og J. Harris (2007). *Competing on analytics: The new science of winning*. Harvard Business School Press. ISBN: 9781422103326.

Gavish, B. og S.C. Graves (1978). "The travelling salesman problem and related problems". I.

Hakimi, S.L. (1964). "Optimum locations of switching centers and the absolute centers and medians of a graph". I: *Operations research* 12.3, s. 450–459.

— (1965). "Optimum distribution of switching centers in a communication network and some related graph theoretic problems". I: *Operations research* 13.3, s. 462–475.

Handfield, R.B og C.B. Bozarth (2016). *Introduction to operations and supply chain management*. Edinburgh Gate, Harlow, England: Pearson Education Limited.

Khandani, A.E., A.J. Kim og A.W. Lo (2010). "Consumer credit-risk models via machine-learning algorithms". I: *Journal of Banking & Finance* 34.11, s. 2767–2787.

Koç, Ç. m.fl. (2016). "Thirty years of heterogeneous vehicle routing". I: *European Journal of Operational Research* 249.1, s. 1–21.

Miller, C.E., A.W. Tucker og R.A. Zemlin (1960). "Integer programming formulation of traveling salesman problems". I: *Journal of ACM* 7.4, s. 326–329.

Pochet, Y. og L.A. Wolsey (2006). *Production planning by Mixed Integer Programming*. Springer.

Tian, J. m.fl. (2021). "Modular machine learning for Alzheimer's disease classification from retinal vasculature". I: *Scientific Reports* 11.1, s. 1–11.

Williams, H.P. (2013). *Model building in mathematical programming*. The Atrium, Southern Gate, Chichester, West Sussex, England: John Wiley & Sons.

# Indholdsfortegnelse


Blandet heltalsoptimering, 206

Clustering, 33
- $k$-means, 35
  - Lloyds algoritme, 36
- Classifaction, 34
- Definition, 33
- Grafopdeling, 47
- Lokationsbaseret, 38

Lineær programmering, 201
Lokationsplanlægning, 53
- $p$-center, 57
- $p$-median, 55
- Covering, 61
- Faste omkostninger, 59
- Kapacitetsbegrænsninger, 60
- Max covering, 65
- Set covering, 62

Multikriterie optimering, 172
- $\varepsilon$-metoden, 178
- Ikke-domineret punkt, 174
- Løsning af, 178
- Pareto optimalitet, 174
- Repræsentation af efficient front, 180

Multiple traveling sales person problem, 126
- MTZ formulering, 129
- One commmodity network flow, 130

Netværksdesign, 67
- Fixed charge network flow problemer, 71
- Fixed charge transportation problem, 67

Notation, 193
- Logisk operator, 193
- Logiske kvantorer, 193
- Summer, 196
- Sumtegn, 196

Produktionsplanlægning, 77
- Master production scheduling, 85
  - Valg af maskine, 90
- Uncapacitated lot-sizing model, 78
  - Network flow, 81

Pyomo, 5
Python, 10
- Introduktion, 10
- Pyomo, 5
- Typer, 15

Python:Funktion, 11
Python:Variabel, 15
Python:Variabler, 15

Rutepanlægning

Trekantsulighed, 126
Ruteplanlægning, 117
*m*-TSP, 126
CVRP, 131
Heterogene, 135
Homogen, 132
MTZ formulering, 133
Network flow formulering, 134
Prize collection, 140
Traveling salesperson problem, 118
Åbne ruter, 136

Skemalægning
*n* jobs på *m* maskiner, 107
En maskine, 101
Minimer samlet tid, 109
Minimer seneste færdige job, 110
Skemaplanlægning, 99
Job shop scheduling, 100
Stokastisk optimering, 163
Deterministisk ækvivalent, 167
Recourse, 166
Scenarie-generering, 168
Two-stage recourse, 166

Traveling salesperson problem, 118
Gavish and Graves formulering, 123
MTZ, 120
Multiple traveling salesperson problem, 126
One commodity network flow, 123
TSP, 118
TSP
Multiple traveling salesperson problem, 126