Optimeringsmetoder innen operasjonsanalyse – en oversiktsstudie

Maria F. Fauske

Forsvarets forskningsinstitutt (FFI)

15. januar 2008

FFI-rapport 2008/00123

1068

ISBN 978-82-464-1314-3

Emneord

Operasjonsanalyse

Optimering

Simplex

Heltallsprogrammering

Heuristiske metoder

Godkjent av

Stein Malerud	Prosjektleder
Espen Skjelland	Forskningssjef
Espen Skjelland	Avdelingssjef

Sammendrag

Denne rapporten er en oversiktsstudie av optimeringsmetoder, utarbeidet av prosjekt 1068 ”Metoder og modeller for analyse av freds- og lavintensitetsoperasjoner”. Bruk av optimeringsmetoder er veletablert innen operasjonsanalyse. Imidlertid har slike metoder i liten grad vært benyttet for analyseformål ved instituttet de senere årene. Rapporten skal derfor bidra til å øke bevisstheten rundt hvilke muligheter optimering gir for slike formål. Ulike typer optimeringsmodeller og hvilke løsningsmetoder som egner seg for disse blir gjennomgått. Rapporten belyser også hvordan optimering skiller seg fra andre metoder innen operasjonsanalyse. Det blir beskrevet i hvilke sammenhenger optimering egner seg som analyseverktøy, samt hva som ofte er de største utfordringene ved bruk av optimering. Forutsetninger for at optimering skal være et effektive hjelpemiddel i analyser, er at forskerne har erfaring i å bruke det og at de har tilgang på gode verktøy som lett lar seg integrere med andre applikasjoner.

English summary

This report has been written as a part of project 1068 Methods and models for analyzing peace and low intensity operations. Optimization is a well established tool within the area of operations research. However, optimization is not used to a large extent for analytic purposes at the Institute. The purpose of this report is to show some of the possibilities that lie in the use of optimization, especially within the type of problems that often form the basis of analyses performed at the Institute. Different types of optimization models are described, in addition to some of the methods that are commonly used to solve these problems. The report also speaks of the differences between optimization and other common methods within the area of operations research. Some of the challenges in the use of optimization are described. Experience is essential for optimization to be an effective tool for analysis. It is also necessary with access to quality tools which are easily integrated with other applications.

Innhold

1 Innledning

Denne oversiktsrapporten om optimeringsmetoder er utarbeidet av prosjekt 1068 ”Metoder og modeller for analyse av freds- og lavintensitetsoperasjoner” (GOAL II) og inngår i en serie med oversiktsstudier av sentrale metoder innen operasjonsanalyse (OA). Andre rapporter i denne serien omhandler problemstrukturerende metoder [1], flermålsanalysemetoder [2] og simuleringsmetoder [3].

Bruk av optimeringsmetoder er veletablert innen OA – sågar settes det ofte likhetstegn mellom optimering og OA. Metoder som lineærprogrammering og heltallsprogrammering har stor utbredelse, og det finnes mange verktøyer på markedet som støtter disse metodene. Optimeringsmetoder har, i alle fall de senere årene, i liten grad vært benyttet for analyseformål ved instituttet. Det kan være mange årsaker til dette, men en viktig grunn er nok at vi har hatt kunnskap og tradisjon for å utvikle simuleringsmodeller og har derfor i liten grad vurdert å formulere problemene som optimeringsproblemer. En annen årsak er nok å finne i kompleksiteten i problemstillingene, og at optimering har vært ansett å være best egnet for avgrensede og velformulerte problemer.

Bakgrunnen for denne rapporten er å øke bevisstheten rundt hvilke muligheter optimering gir innenfor problemstillinger som kan være aktuelle for avdelingen og instituttet. Rapporten går gjennom ulike typer optimeringsmodeller og hvilke løsningsmetoder som egner seg for disse. Ved å følge OA-prosessen gjengitt i Figur 2.1 hvor problemstrukturering vektlegges innledningsvis, skulle det være mulig å identifisere problemstillinger som er egnet for optimering. Videre er det ofte ønskelig å adressere komplekse problemer med flere, komplementære metoder ut i fra det at ulike metoder er egnet til å studere ulike aspekter ved problemene.

Rapporten starter i kapittel 2 med å gi en generell innføring i temaet optimering, og hvorfor dette er et nyttig verktøy innen operasjonsanalyse. Videre beskrives det i kapittel 3 hvordan optimeringsmodeller kan klassifiseres ut fra ulike egenskaper ved modellene. Dette gir opphav til forskjellige typer optimeringsproblemer, og beskrivelse av disse følger i kapitlene 4 til 7. Kapittel 8 omhandler temaet heuristiske metoder som ofte må tas i bruk fordi eksakt optimering ikke er mulig. Kapittel 9 omtaler usikkerhetshåndtering innen optimeringsteori. Rapporten avsluttes med oppsummering og konklusjoner i kapittel 10.

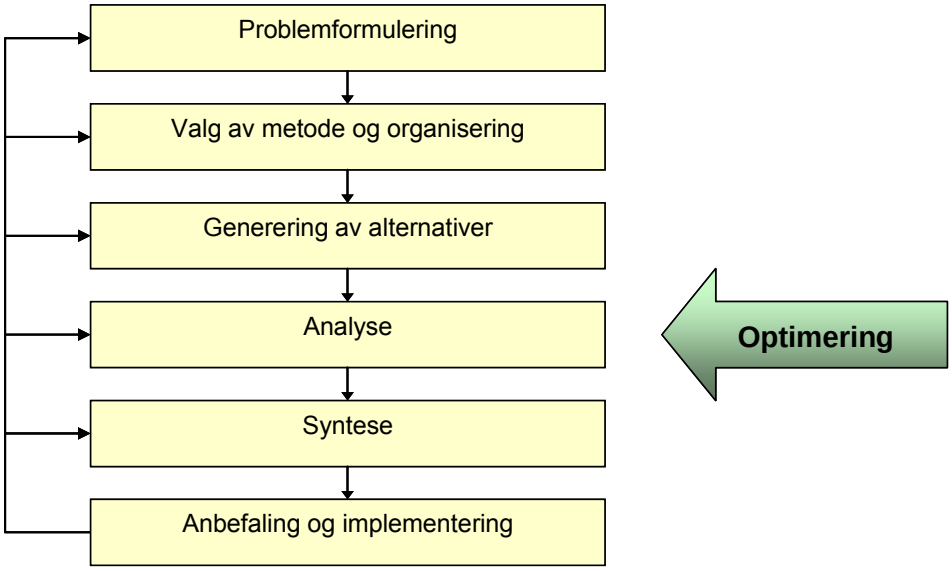
I hvert av kapitlene 4-6 samt 8 er det forsøkt å gi et illustrerende eksempel på et typisk problem som hører inn under den typen modeller kapitlet omhandler. Det brukes et gjennomgående tema for eksemplene som er valgt med tanke på at det skal gjenspeile tematikken i prosjektet. Prosjektet fokuserer på freds- og lavintensitetsoperasjoner, og har valgt Norges deltagelse i Afghanistan som en konkretisering av dette vide feltet. Lettfattelig og pedagogisk illustrasjon av metode er hovedhensikten med eksemplene. Dette er derfor vektlagt mer enn etterstrebelse av realisme i historien.

2 Hva er optimering?

Innen naturvitenskapen bruker man matematikken som hjelpemiddel til å analysere det universet vi lever i. Å bruke matematikk til å optimere lærer vi oss allerede når vi på skolen utfører vår første derivasjon. Optimering handler nettopp om å finne maksimums- og minimumsverdier på definerte problemer. I denne rapporten skal vi se på optimeringsmetoder som ofte anvendes innen OA, med fokus på metoder som kan være relevante for forsvarsrelaterte problemstillinger.

2.1 Optimering og operasjonsanalyse

Slagordet[1] ”Operations research is the science of better” illustrerer hvordan OA skal bidra til bedre beslutninger ved å fremskaffe et godt beslutningsgrunnlag. Tradisjonelt har OA handlet nesten utelukkende om bruk av kvantitative metoder, men i dag er det også vanlig å inkludere mer kvalitative metoder, eller såkalte ”myke” OA-metoder, i beslutningsstøtten. Optimering har alltid vært en sentral, kvantitativ metode innen OA. I noen miljøer har man satt likhetstegn mellom OA og optimering. Det korrekte er imidlertid at OA-prosessen er en prosess der optimering er et verktøy under analysedelen. Dette er illustrert i Figur 2.1. En beskrivelse som plasserer begrepene optimering, operasjonsanalyse og beslutningsstøtte i forhold til hverandre, finnes i [4].



Figur 2.1 Arbeidsprosess i operasjonsanalyse

Optimering er bare én av flere metoder som passer inn under analysesteget i prosessen. Andre metoder som hører hjemme her, kan være simulering og flermålsanalyse. Simulering brukes for å gjøre eksperimenter med en modell av et system [3]. Flermålsanalyse søker å finne den løsningen som best stemmer overens med beslutningstakers preferanser, og ikke nødvendigvis den optimale løsningen i matematisk forstand [2].

Målet med optimering er å finne en matematisk optimal løsning. Selve optimeringsanalysen omfatter flere steg. Første steg dreier seg om å formulere den matematiske modellen, med

[1] Slagordet er hentet fra ”The guide to operational research” utarbeidet av det britiske OR Society

utgangspunkt i problemformuleringen man kom frem til i starten av OA-prosessen. I praksis løses alle optimeringsproblemer innen OA i dag ved hjelp av datamaskiner. Derfor blir neste steg å implementere modellen med egnet programvare for optimering. Siste del av analysen handler om å gjøre selve kjøringen, der man kommer fram til optimal løsning på problemet slik det er definert i modellen, dersom en slik løsning finnes.

2.2 Hvorfor optimering?

Optimering er i dag svært utbredt innenfor forskning, økonomi og industri, hvor ulike optimeringsmetoder benyttes innenfor bl.a. transport, logistikk, utnyttelse av ressurser, nettverksplanlegging og billedanalyse. Innenfor slike områder finnes det ofte uttalige alternative løsninger, og det kan være en stor utfordring å finne den beste. Historisk har det eksistert noe skepsis overfor optimering, noe som blant annet har hatt årsak i manglende eller svak regnekapasitet. I dag er dette imidlertid ikke en begrensning i like stor grad, og for problemer som lar seg løse ved hjelp av optimering, er dette et uovertruffent verktøy dersom man ønsker å finne den matematisk beste løsningen på problemet. Noe av det som kjennetegner problemer hvor optimering ofte er en egnet løsningsmetode er:

- Det relevante problemet kan konkretiseres, og eventuelt begrenses, tilstrekkelig til å kunne beskrives i en matematisk modell
- Det er mulig å definere hva man ønsker å optimere, og hva som vil gjøre én løsning bedre enn en annen løsning
- Det er mulig å skille ulike beslutningsalternativer
- Problemet er for komplekst til å løses ”manuelt”, spesielt handler dette om at antallet mulige løsninger er stort
- Problemet er tilstrekkelig viktig til at det kan settes av nok tid og ressurser til å løse det

Å utvikle en optimeringsmodell kan være tidkrevende, og det krever trening for å bli god til å se hvordan et problem beskrevet med ord kan representeres matematisk. I mange tilfeller opplever man at optimeringsmodellen vil måtte begrenses i relativt stor grad i forhold til det virkelige problemet. Det kan være hensyn som må utelates fordi de er for vanskelige å kvantifisere, eller for å forhindre at modellen blir for kompleks. Komplekse optimeringsmodeller krever ofte stor regnekraft, og selv om det stadig utvikles raskere datamaskiner finnes det problemer som faktisk ikke lar seg løse innenfor overkommelig tid. Dette vil omtales i kapittel 8.

Den store fordelen ved å kunne bruke optimering, er at man kan vite at ingen annen metode ville komme fram til en bedre løsning på problemet. Det ligger også en del fordeler i selve modelleringsprosessen. Man får økt forståelse for problemet ved at man må tenke strukturert rundt det, og ved at man tvinges til å beskrive og tallfeste sammenhenger.

Optimeringsmodellen gir en utvetydig formulering uten rom for tolkninger. Det vil imidlertid ofte være slik at de virkelige beslutningstakere ikke selv er i stand til å ”lese” formuleringen av optimeringsmodellen, da denne kan være relativt komplisert. Slik sett er god kommunikasjon

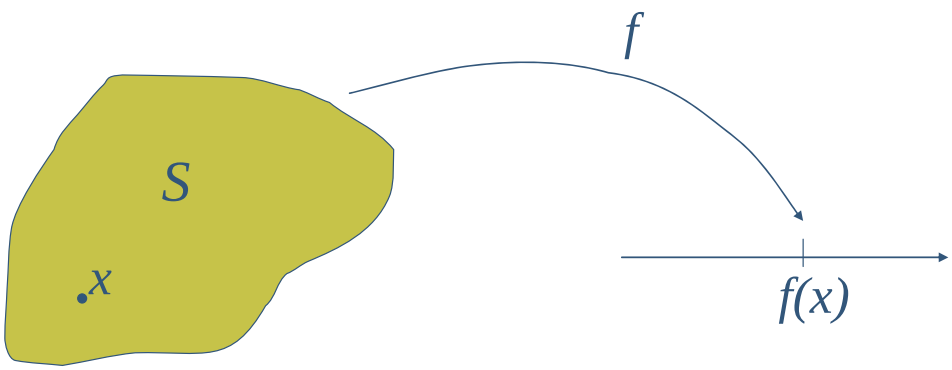
mellom beslutningstakerne og analytikerne svært viktig, slik at beslutningstakerne i tilstrekkelig grad forstår og kan si seg enig i formuleringen.

Noe som ofte vil være vanskelig å forholde seg til for beslutningstakere, er det at man med optimeringsmodellen i utgangspunktet ikke kan optimere flere enn ett aspekt om gangen. Eksempelvis vil en prosjektleder gjerne minimere både tidsbruk og kostnader ved prosjektet sitt, men disse to ønskene er ofte ikke mulig å tilfredsstille samtidig. Slike problemer, samt hva man kan gjøre for å løse dem, er omtalt nærmere i kapittel 3.4. En konsekvens av denne begrensningen er at beslutningstakere tvinges til å tenke nøye gjennom hva som er viktigst for dem, og hvordan de vil prioritere forskjellige mål.

2.3 Optimeringsmodellens byggesteiner

Generelt består optimeringsmodeller av:

- En objektfunksjon som beskriver hva vi ønsker å optimere
- Et sett av variable som påvirker verdien av objektfunksjonen
- Et sett av beskrankninger som setter begrensninger for hvilke verdier variablene kan ta



Figur 2.2 Sammenheng mellom elementene i en optimeringsmodell

Sammenhengen mellom de forskjellige elementene vises i Figur 2.2. Beskrankningene definerer problemets løsningsrom *S*. Det vil si at hvis beslutningsvariablene eksempelvis har definisjonsmengden ”alle reelle tall”, og en beskrankning som sier at variablene ikke skal være mindre enn 0, vil løsningsrommet bli ”alle reelle tall større eller lik 0”. Variabelen *x* kan altså ta verdier fra *S*, og objektfunksjonen *f* gir en tallverdi for hver *x*. Selv om *x* i de aller fleste tilfeller vil være et tall, kan den i prinsippet være ”hva som helst”, så lenge funksjonen *f* er i stand til å tilordne en verdi til den. Eksempelvis kan *x* være en vektor eller en funksjon. Målet med optimeringen er å finne den eller de verdier av *x* som gir optimal verdi for objektfunksjonen. Formelt kan man si at [5]:

En optimal løsning for beslutningsvariablene gir en objektfunksjonsverdi som er minst like god som enhver annen mulig løsning vil gi.

Ut fra en slik definisjon av optimal løsning, kan man si at [5]:

Den optimale verdien i en optimeringsmodell er objektfunksjonsverdien til enhver optimal løsning.

Med andre ord er det slik at en optimeringsmodell kan ha flere optimale løsninger, men bare én optimal verdi.

2.3.1 Objektfunksjonen

Det å finne optimal verdi for objektfunksjonen handler om å finne enten minimumsverdi eller maksimumsverdi. Hvis objektfunksjonen uttrykker en form for kostnad brukes minimering. Motsatt brukes maksimering hvis objektfunksjonen uttrykker en form for gevinst.

Det finnes problemer som ikke har noen objektfunksjon, der det eneste målet er å finne en eller annen løsning som tilfredsstiller beskrankningene. I slike tilfeller er det i prinsippet ingenting som faktisk optimeres, det er heller en sjekk på om det finnes noen mulig løsning og hva den løsningen i så fall kan være. Et morsomt eksempel på denne typen problem, finner man i [6] der det er satt opp en modell for å løse sudokuspill. Her er det ingen objektfunksjon som skal maksimeres eller minimeres, man ønsker bare å finne løsningen på sudokuspillet.

I noen tilfeller vil det være flere mål man ønsker å optimere. Ofte er det slik at variabelverdiene som gir optimal løsning for ett av målene, ikke gir optimal løsning for det eller de andre målene. Det gjør det vanskelig å definere hva som vil være optimal løsning for slike problemer. Problemer med flere objektfunksjoner må derfor behandles på litt andre måter enn problemer med bare én objektfunksjon. Dette er beskrevet nærmere i kapittel 3.4.

2.3.2 Variablene

Variablene kalles ofte beslutningsvariable og beskriver de forskjellige alternativene beslutningstakeren kan velge mellom for å oppnå en best mulig løsning på problemet. Det må alltid være minst én beslutningsvariabel tilstede i et optimeringsproblem.

2.3.3 Beskrankningene

Beskrankningene setter begrensninger for hvilke verdier variablene kan ta. Beskrankningene er beskrevet i form av ligninger eller ulikheter. Heltallsbeskrankninger sier at en variabel bare kan ta heltallsverdier. Kontinuerlige beskrankninger setter nedre og/eller øvre begrensninger på variablene. En beskrankning er lineær dersom den er en vektet sum av variable i første potens, hvis ikke er den ulineær.

Det finnes optimeringsproblemer som ikke har noen beskrankninger. Såkalt ubegrenset optimering er en egen gren innen optimeringsteori, og denne vil bli omtalt nærmere i kapittel 3.3.

2.3.4 Lokale og globale optima

Optimal løsning er hittil omtalt som en løsning innenfor løsningsrommet som gir den høyeste verdien for objektfunksjonen. En slik formulering gir et såkalt globalt optimum. Det kan imidlertid også eksistere lokale optima, det vil si løsninger der ingen andre løsninger i *nærheten* gir høyere objektfunksjonsverdi. I mange tilfeller kan dette bli problematisk, da det kan være umulig for løsningsalgoritmen å vite om et optimum den har funnet er lokalt eller globalt. Dette gjelder imidlertid bare for noen typer optimeringsproblemer, og i mange tilfeller finnes det metoder for å sjekke om man kan garantere at et optimum faktisk er et globalt optimum. Kapittel 3 vil omtale dette temaet nærmere, og knytte det til tilfeller hvor man kan risikere at det blir et problem.

2.4 Generell formulering av optimeringsproblemer

Optimering betegnes ofte som matematisk programmering. Med programmering mener man dermed her noe annet enn dataprogrammering. I denne rapporten vil begrepet programmering brukes i sammenhenger hvor det er et etablert uttrykk, ellers vil ordet optimering brukes.

Den generelle formen av en optimeringsmodell kan uttrykkes som følger:

$$\min \textit{eller} \max \; f(x_1,...,x_n) \tag{2.1}$$

s.a.

$$g_i(x_1,...,x_n) \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \qquad i = 1,...,m \tag{2.2}$$

der f og g er funksjoner av beslutningsvariablene $x_1,...,x_n$, og $b_1,...,b_m$ er konstanter. En slik generell formulering ser forholdsvis enkel og oversiktlig ut, men bak det hele skjuler det seg et uttall av ulike typer optimeringsproblemer. Disse kan ha svært forskjellige egenskaper, og det å løse dem krever et vidt spekter av algoritmer og metoder. Ofte er den største utfordringen ved å sette opp optimeringsmodeller det å kjenne igjen hva slags type problem man har med å gjøre. Kapittel 3 vil derfor gå gjennom forskjellige klassifiseringer av optimeringsproblemer.

2.5 Dualitet

Ethvert *lineært* optimeringsproblem har et tilhørende dualproblem. Dualproblemet kan defineres direkte fra det originale problemet, eller primalproblemet. Dersom primalproblemet er et maksimeringsproblem, vil dualproblemet være et minimeringsproblem, og visa versa. Primal- og dualproblemet henger så nøye sammen, at hvis man finner optimal løsning for det ene problemet, vil man enkelt kunne bruke denne løsningen til å finne optimal løsning for det andre problemet. I optimum er dessuten optimal *verdi* for problemene den samme. Disse sammenhengene kan være nyttige da det vil være tilfeller hvor dualproblemet vil være mer hensiktsmessig å løse enn primalproblemet. I tillegg inneholder dualproblemet viktig informasjon som kan brukes til følsomhetsanalyse. Dette gjelder imidlertid bare for kontinuerlige, lineære problemer. Hvorfor

dualproblemet ikke kan brukes til følsomhets-analyse for andre typer problemer er illustrert i [7]. Dualitet og følsomhetsanalyse vil bli beskrevet nærmere i kapittel 9. Dualproblemet konstrueres fra primalproblemet på følgende måte:

Gitt maksimeringsproblemet

$$\max Z = c_1 x_1 + c_2 x_2 + c_3 x_3 \tag{2.3}$$

s.a.

$$a_{i1} x_1 + a_{i2} x_2 + a_{i3} x_3 \le b_i \qquad for\ i = 1,...,4 \tag{2.4}$$

$$x_1, x_2, x_3 \ge 0 \tag{2.5}$$

For dualproblemet gjelder:

1. For hver variabel i primalproblemet, har dualproblemet en beskrankning
2. For hver beskrankning i primalproblemet, har dualproblemet en variabel
3. Høyresiden i primalproblemets beskrankninger (b_i) gir koeffisientene i dualproblemets objektfunksjon
4. Koeffisientene i primalproblemets objektfunksjon gir høyresiden i dualproblemets beskrankninger
5. En rad av koeffisienter i primalproblemets beskrankninger gir en kolonne av koeffisienter i dualproblemets beskrankninger

Dualproblemet til modellen over kan formuleres som:

$$\min W = b_1 y_1 + b_2 y_2 + b_3 y_3 + b_4 y_4 \tag{2.6}$$

s.a.

$$a_{1j} y_1 + a_{2j} y_2 + a_{3j} y_3 + a_{4j} y_4 \ge c_j \qquad for\ j = 1,...,3 \tag{2.7}$$

$$y_1, y_2, y_3, y_4 \ge 0 \tag{2.8}$$

I optimum vil altså $Z = W$.

2.6 Løsningsmetoder for optimeringsproblemer

Det er allerede nevnt at det finnes et uttall løsningsalgoritmer for optimeringsproblemer. Det er også slik at man av og til ikke kan finne eksakt løsning på problemet innenfor overkommelig tid, og dermed må bruke tilnærmingsmetoder for å finne løsninger som er ”gode nok”. De neste to avsnittene tar for seg disse to forskjellige måtene å løse optimeringsproblemer på.

2.6.1 Eksakte metoder

Dersom det er mulig *og* praktisk gjennomførbart å løse et optimeringsproblem eksakt, så gjør man det. Hvorvidt dette kan gjøres avhenger som tidligere nevnt av problemets kompleksitet. I noen tilfeller er det mulig å løse problemet analytisk. Gjennom analytiske metoder kan man finne konkrete løsninger som beviselig er optimale, uten at man trenger å være innom andre løsninger på veien. Vanlig derivasjon av en todimensjonal graf er et eksempel på en analytisk metode. I andre tilfeller er det nødvendig, eller mer hensiktsmessig, å løse problemet ved hjelp av søk. En søkealgoritme fungerer slik at den systematisk prøver forskjellige løsninger, mens den hele tiden holder orden på hvilken løsning som så langt har gitt best objektfunksjonsverdi. Det finnes et uttall forskjellige søkealgoritmer. Det som hovedsakelig skiller dem fra hverandre, er hvilke kriterier som brukes for å avgjøre hvordan man skal gå videre for å lete etter neste løsning. Denne forskjellen er imidlertid helt essensiell, da det vil variere fra problem til problem hvilke metoder som vil fungere.

2.6.2 Heuristiske metoder

For problemer som ikke lar seg løse eksakt innen rimelig tid, er det derfor nødvendig å effektivisere søkealgoritmene slik at de blir ”smartere” når de leter etter løsninger. Slike heuristiske metoder er ikke garantert å finne optimal løsning. De finner en *mulig* løsning, og de finner en løsning som ofte er mye bedre enn de aller fleste andre mulige løsninger. Noen ganger finner de også optimal løsning, uten at det kan bevises.

Ofte vil feilen ved en heuristisk løsning i forhold til en eksakt løsning være forsvinnende liten i forhold til feil som er resultat av dårlig kunnskap om usikre inputdata, eller eventuelle feil og begrensninger som ligger i selve modellformuleringen. Heuristiske metoder vil derfor ofte være en god og praktisk fremgangsmåte som gir tilstrekkelig gode løsninger på problemene.

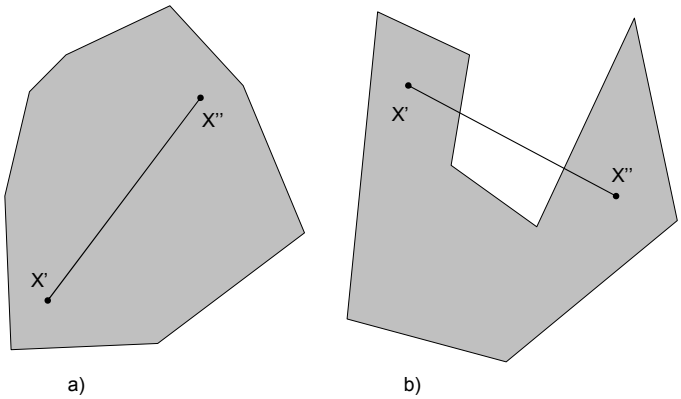
3 Klassifisering av optimeringsmodeller

Optimeringsproblemer kan klassifiseres ved hjelp av flere egenskaper. I det følgende er det beskrevet egenskaper som er relevante for de fleste problemformuleringer innen optimeringsteori.

3.1 Lineær – ulineær

En optimeringsmodell er lineær dersom både objektfunksjonen f og beskrankningene $g_1,...,g_m$ fra (2.1) og (2.2) er lineære i beslutningsvariablene. Hvis f eller minst én av $g_1,...,g_m$ er ulineære i beslutningsvariablene, er optimeringsmodellen ulineær.

Dersom et problem er lineært og samtidig kontinuerlig, vil det ha en del gunstige egenskaper. Løsningsrommet til problemet vil da alltid være kontinuerlig og konvekst. Man sier at løsningsrommet er en konveks mengde. Konvekse mengder garanterer at et optimum er et globalt optimum for lineære funksjoner. Figur 3.1 viser et eksempel på en konveks (a) og en ikke-konveks (b) mengde i to dimensjoner. En mengde er konveks hvis det er slik at en linje man trekker mellom ethvert par av punkter i løsningsrommet også ligger innenfor løsningsrommet.



Figur 3.1 Eksempel på a) konveks mengde og b) ikke-konveks mengde

Ulineære optimeringsproblemer har ikke alltid konvekse løsningsrom slik som lineære, kontinuerlige problemer, noe som gjør at det kan være umulig å finne globalt optimum. Hvis det er mulig å bevise at en *funksjon* er konveks eller konkav, er man garantert at det ikke finnes noen lokale optima. Når funksjonen er konkav finnes det et globalt maksimum. For konvekse funksjoner har man et globalt minimum. En grundig forklaring av konveksitet og konkavitet finnes i [8].

En annen forskjell på lineære og ulineære optimeringsproblemer, er at for ulineære problemer vil det av og til forekomme at problemet ikke har beskrankninger, bare en objektfunksjon. Dette vil omtales nærmere i kapittel 3.3.

Selv om ulineære problemer også passer inn i den generelle formuleringen av optimeringsmodeller, finnes det så mange forskjellige problemer at ingen enkelt algoritme kan løse dem alle. Kapittel 7 vil gå nærmere inn på forskjellige typer ulineære problemer.

Siden lineære problemer som regel er betydelig lettere å løse enn ulineære, er det fristende å forsøke å bruke en lineær formulering av et problem, selv om det ikke i utgangspunktet er lineært. Konsekvensen av å gjøre en slik forenkling, er som regel at resultatene blir helt feil. En grundig beskrivelse av lineærprogrammering, samt hvilke problemer som lar seg linearisere, er gitt i [4].

3.2 Kontinuerlig – diskret

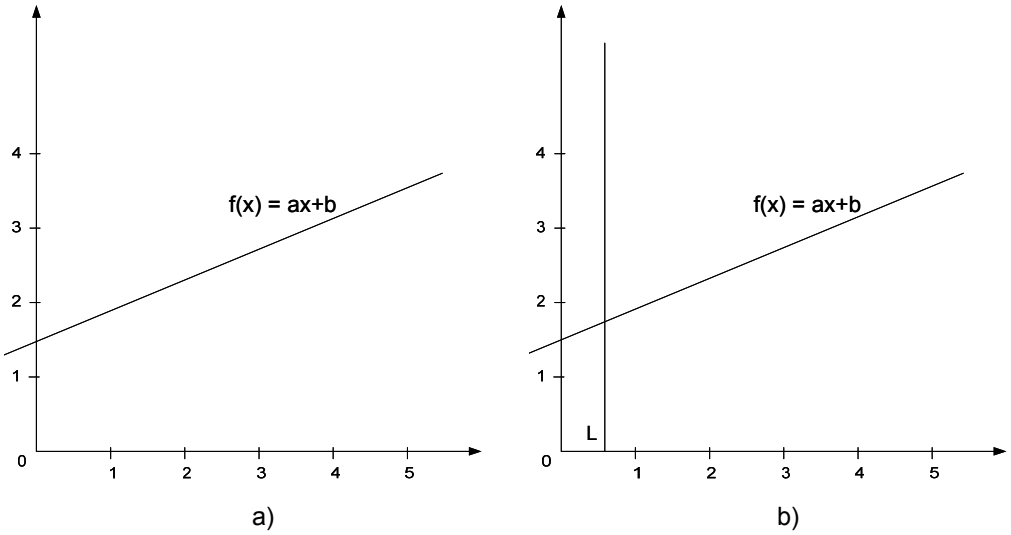
Hvorvidt et problem er kontinuerlig eller diskret, handler om hva slags verdier variablene kan ta. En variabel er kontinuerlig hvis den kan ta enhver verdi i et spesifisert intervall. Definisjonen på et slikt intervall kan for eksempel være ”alle positive, reelle tall”. En variabel er diskret dersom den kan ta verdier fra et fast, tellbart sett av verdier. Et eksempel på et slikt sett kan være ”alle positive heltall”. Diskret optimering kalles ofte heltallsprogrammering, da de diskrete verdiene variablene kan ta stort sett alltid er heltall. Det finnes også mange problemer som har både kontinuerlige og diskrete variable, såkalt *blandet heltallsprogrammering*. Veldig vanlig er også

problemer der variablene er binære, det vil si de kan ta verdien 0 eller 1. Binære variable gjenspeiler ofte hvorvidt en hendelse inntreffer eller ikke.

Kontinuerlige, lineære problemer er generelt lettere å løse enn diskrete. For diskrete problemer har man blant annet at enhver løsning i prinsippet er et lokalt optimum, siden det ikke finnes noen andre løsninger "i nærheten". Dersom man har å gjøre med et heltallsproblem som er en diskret variant av et lineært problem, kan man begynne med å løse problemet som et kontinuerlig problem og bruke løsningen som en informasjon om hvor man bør lete etter optimal diskret løsning. Denne prosessen beskrives i [4].

3.3 Med beskrankninger – uten beskrankninger

Figur 3.2 viser en lineær funksjon uten beskrankninger. Det er ikke mulig å finne noe optimum, siden funksjonen er uten grenser. Hvis man derimot innfører en beskrankning som vist i Figur 2.1, som sier at $x \geq L$, får funksjonen et minimum i $x = L$.



Figur 3.2 a) Lineær funksjon uten beskrankninger b) Lineær funksjon med beskrankning L

Når det gjelder ulineære funksjoner uten beskrankninger, er det relativt ukomplisert hva som garanterer at et optimum er et globalt optimum. En nødvendig betingelse for at en løsning er optimal, er at den deriverte av funksjonen er lik 0 i det aktuelle punktet. Dette er også en tilstrekkelig betingelse dersom objektfunksjonen er konkav eller konveks [8]. Konkave funksjoner har globalt maksimum, og konvekse funksjoner har globalt minimum.

Når det finnes beskrankninger er situasjonen noe mer komplisert. Nødvendig betingelse for at en løsning er optimal, er da at problemet tilfredsstiller de såkalte Karush-Kuhn-Tucker-betingelsene (KKT). Disse er gjengitt i blant annet [5], [8] og [9]. KKT er også tilstrekkelige betingelser for at et optimum er et globalt optimum dersom objektfunksjonen er enten konkav eller konveks, *og* løsningsrommet for problemet er en konveks mengde.

Optimering uten beskrankninger er tett knyttet til ulineær optimering. Derfor vil dette temaet bli behandlet videre i kapittel 7.

3.4 Én objektfunksjon – flere objektfunksjoner

I mange beslutningssituasjoner ønsker man å legge flere kriterier til grunn for beslutningen. Dette innebærer at man ønsker å finne det alternativet som gir maksimal løsning over alle kriteriene. Det finnes flere måter å håndtere dette på. Blant annet kan man forsøke å formulere én objektfunksjon og la de andre kriteriene representeres via beskrankninger. Der dette er mulig er det absolutt det mest hensiktsmessige siden optimeringsproblemer med bare én objektfunksjon er mye lettere å ha med å gjøre enn de med flere.

Imidlertid hender det at problemer *må* formuleres ved hjelp av flere objektfunksjoner. Det vil i de fleste slike tilfeller ikke være mulig å finne en løsning som er optimal med hensyn på alle mål, men man kan komme fram til en løsning som er god nok og som best mulig balanserer de ulike behovene. Det finnes flere metoder for å løse slike flerkriterieproblemer. Slike metoder innebærer at det må innføres subjektive vurderinger i forhold til viktigheten av kriteriene, slik at man ikke lenger vil finne et matematisk optimum. Tre vanlige slike metoder er 1) prioritering av objektfunksjoner, 2) vektet sum og 3) goalprogrammering. Under omtales metodene kort. For en mer grundig innføring i flerkriterieoptimering, se [5].

Prioritering av objektfunksjoner

Med denne metoden settes det en prioritet på objektfunksjonene. Problemet optimeres med den viktigste objektfunksjonen først, så optimeres den nest viktigste under forutsetning av at den første oppnår sin optimale verdi. Slik fortsetter man til man har vært gjennom alle objekt-funksjonene. Jo lenger ut i rekken man kommer, jo mindre er sjansen for at den aktuelle objektfunksjonen oppnår sin optimale verdi. Derfor er det helt essensielt for denne metoden at det er mulig å gjøre en prioritering av objektfunksjonene.

Vektet sum

Denne metoden handler om å slå sammen objektfunksjonene til én ved hjelp av vekter. Dette gir en mer balansert løsning enn metoden med prioritering av objektfunksjoner. Det som gis størst vekt vil naturligvis ha størst betydning for hva som blir optimal løsning. Slik sett må man også med denne metoden foreta en prioritering.

Goalprogrammering

Den mest brukte metoden for optimering med flere objektfunksjoner, er goalprogrammering. Det kommer blant annet av at det er det alternativet som er lettest å implementere. Prinsippet går ut på at det defineres terskelverdier som objektfunksjonene *helst* skal være bedre enn, og så formuleres optimeringsproblemet slik at en prøver å komme så nær disse målverdiene som mulig. Goalprogrammering kan studeres videre i [5], [8] og [9].

3.5 Deterministisk – stokastisk

I deterministiske optimeringsmodeller antar man at verdiene på inndata er kjent. Det tas med andre ord ikke hensyn til usikkerhet i dataene. For virkelige problemstillinger vil det nesten alltid være slik at én eller flere av inputverdiene er usikre. Eksempelvis vil man i en planleggings-prosess ofte bruke data som sier noe om en fremtidig utvikling. I slike tilfeller vil det være en betydelig feilkilde i modellen dersom man tar for gitt at dataene er deterministiske og kjent.

Når ett eller flere dataelementer i en optimeringsmodell er representert ved en stokastisk variabel, har man å gjøre med en stokastisk modell. Stokastiske modeller tar med andre ord høyde for usikkerheten i problemet som modellen representerer. Stokastiske modeller er beskrevet nærmere i kapittel 9.2.

3.6 Oppsummering

Klassifiseringene som er gjort i dette kapitlet, gir opphav til optimeringsproblemer som er meget forskjellige i natur, og som krever et bredt spekter av løsningsalgoritmer. I de følgende kapitlene vil de vanligste typene optimeringsmodeller bli omtalt, og det vil bli gitt eksempler på hvordan de kan anvendes.

4 Lineærprogrammering

Når optimeringsproblemer er lineære, kontinuerlige og begrensede, kalles de lineær-programmeringsproblemer (LP-problemer). Alle slike problemer formuleres etter samme mønster og løses med samme løsningsalgoritme. Lineærprogrammeringen oppsto under andre verdenskrig, da den ble tatt i bruk som matematisk modell for å beregne hvordan man kunne redusere egne kostnader, samtidig som man skulle påføre fienden størst mulig tap. Etter andre verdenskrig spredte anvendelsen av lineærprogrammering seg til industrien generelt, hvor den de siste 50 årene hatt enorm betydning. I dag har anvendelsen også spredt seg til andre sektorer i samfunnet.

4.1 Anvendelsesområder

Generelt kan man si at alle LP-problemer dreier seg om at det skal utføres en rekke aktiviteter som krever tilgang til begrensede ressurser. Et av de mest brukte eksemplene i lærebøker om optimering, er produksjonsplanleggingsproblemet. I et slikt problem vil typiske aktiviteter være ”produser mengden *x* av produkt 1” og ”produser mengden *y* av produkt 2”. Nivåene på *x* og *y* bestemmer hvor mye av tilgjengelige råvarer som kreves i produksjonen av de to produktene. Objektfunksjonen vil gjerne uttrykke gevinsten når man selger produktene, og beskrankningene vil sørge for at man ikke bruker mer av ressursene enn man har tilgjengelig.

I militær sammenheng har lineærprogrammering tradisjonelt blitt brukt til å løse problemer som for eksempel hva slags sammensetning av våpen man bør bruke i en gitt situasjon, eller i problemer som handler om at forsyninger skal transporteres fra en base og ut til et sett av

destinasjoner. For våpenmiksproblemet uttrykker objektfunksjonen for eksempel ”antall destruerte fiendtlige fly”. Beskrankninger kan uttrykke krav om å overholde budsjettet, ikke bruke mer mannskap enn tilgjengelig og ikke bruke flere av hver våpentype enn man har tilgjengelig. I transportproblemet er målet å minimere transportkostnad, samtidig som man sørger for at etterspørselen etter forsyninger blir tilfredsstilt. Denne typen problemer er tema i [10]. Andre vanlige anvendelser av lineærprogrammering er:

- Jordbruk, planlegging av arealutnyttelse
- Oljeraffinering, bestemme produksjonskombinasjon av ulike raffinerte produkter
- Byplanlegging, type bebyggelse man bør satse på
- Finans, minimering av risiko i investeringer

Noen ganger vil det være slik at man ønsker at optimeringsmodellen skal gi løsninger som er heltall. Slik kan det for eksempel være i produksjonsplanleggingsproblemet over. Hvis optimal løsning gir desimaltall kan det i mange tilfeller være tilfredsstillende å runde av dette tallet til et heltall, men det er ikke alltid slik. I verste fall kan det gi svar som er langt fra optimale, eller løsninger som ikke er gyldige. Har man behov for heltall i svaret bør man vurdere å formulere modellen som et heltallsproblem, altså diskret istedenfor kontinuerlig. Mer om slike problemer kommer i kapittel 5.

4.2 Simplex-metoden for løsning av LP-problemer

Selv om det finnes flere løsningsalgoritmer for LP-problemer, er Simplex eller algoritmer som bygger på Simplex det som er desidert mest brukt. Simplex-algoritmens virkemåte kan illustreres gjennom et eksempel. Et LP-problem kan se ut som følger:

$$\max z = 3x_1 + 2x_2 \tag{4.1}$$

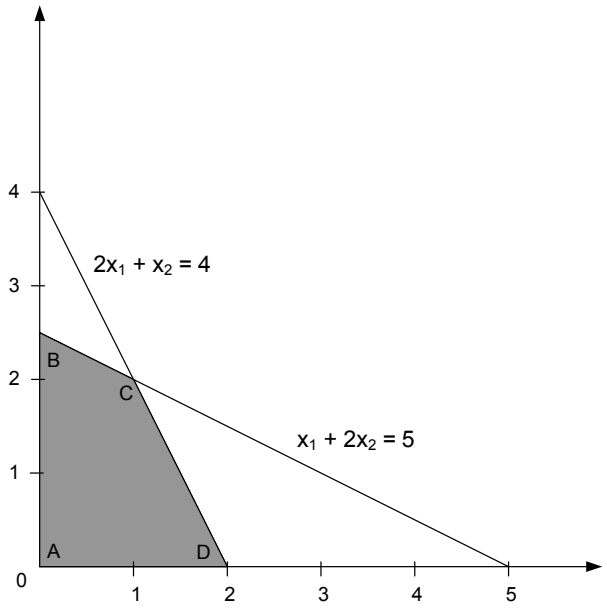
s.a.

$$2x_1 + x_2 \leq 4 \tag{4.2}$$

$$x_1 + 2x_2 \leq 5 \tag{4.3}$$

$$x_1, x_2 \geq 0 \tag{4.4}$$

Grafisk kan man fremstille dette problemet som vist i Figur 4.1.



Figur 4.1 Grafisk fremstilling av et LP-problem

Simplex-metoden utnytter noen gunstige egenskaper ved LP-problemet. Det er nemlig slik at optimum alltid ligger i et hjørnepunkt. Dette er en konsekvens av problemets linearitet og konveksitet, som ble omtalt i kapittel 3.1. Simplex utnytter dette ved å bevege seg fra hjørnepunkt til hjørnepunkt i løsningsrommet. I hjørnepunktet sjekker den om objekt-funksjonen øker langs noen av kantene. Hvis den gjør det, beveger den seg videre i den retningen hvor objektfunksjonen øker mest. Hvis derimot objektfunksjonen i hjørnepunktet ikke øker langs noen av kantene, vet man at man har funnet optimum. Optimum i
Figur 4.1 ligger i punkt C der $x_1 = 1$ og $x_2 = 2$.

For små problemer er det forholdsvis enkelt å finne en løsning for hånd ved hjelp av Simplex. Da bruker man noe som kalles tablåmetoden. Mer om Simplex og tablåmetoden kan man finne i [5], [8] og [9].

4.3 Eksempel på anvendelse

Dette og etterfølgende eksempler vil knyttes til fredsoperasjoner i Afghanistan. Det er forsøkt å holde graden av realisme i eksemplene så høy som mulig, men lettfattelig og pedagogisk illustrasjon av metode har vært førsteprioritet.

En internasjonal non-governmental organisation (NGO) i Faryab-provinsen har bestemt seg for å bygge to nye skoler i regionen. Den ene skolen skal bygges i et område hvor det stort sett bor uzbekere, mens den andre skal bygges i en landsby hvor det utelukkende bor pashtunere.

Det er ønskelig at man bruker lokale leverandører ved anskaffelse av bygningsmateriale til skolene. Det finnes fire lokale materialleverandører i områdene rundt de to landsbyene hvor

skolene skal bygges. Leverandørene tilhører tre ulike etniske grupper, og av politiske årsaker ønsker NGO-en å kjøpe noe materiale fra alle leverandørene.

Før anskaffelse av materialet, blir leverandørene bedt om å oppgi pris og leveringskapasitet på materialet. I tillegg beregner NGO-en antatt risiko ved å kjøpe materiale fra hver leverandør. Denne risikoen er en funksjon av blant annet:

- Leveringssikkerhet: Det kan være en usikkerhet knyttet til hvorvidt leverandørene kommer til å levere den mengden materiale som de har oppgitt.
- Risiko under transport: Det vil være en risiko forbundet med selve transporten av materialet fra en leverandør til en skole. Størrelsen på denne risikoen avhenger av mengde transportert materiale, avstand mellom leverandør og skole, og områdene som materialet må transporteres gjennom.
- Kvalitet på materialet: De forskjellige leverandørene leverer materiale med noe varierende grad av kvalitet.

For enkelthets skyld antar man at risikoen øker proporsjonalt med mengden materiale som kjøpes fra hver leverandør. På grunn av transportrisikoen, vil risikoen også variere med hvilken skole materialet skal leveres til.

De ansvarlige for materialanskaffelse til skolene har behov for en modell som forteller dem hvor mye materiale de skal kjøpe fra hver leverandør. De ønsker å minimere total ”kostnad”, som er en sum av pris og risiko. Samtidig setter de som krav at de fra hver leverandør skal kjøpe minst 20 % av deres respektive leveringskapasitet.

Inndata for modellen er gitt i tabellene under. Materialbehovet til skolene standardisert, slik at skole 1 har behov for 100 materialenheter, og skole 2 har behov for 130 materialenheter. Leverings-kapasiteten til leverandørene er dermed også gitt som antall materialenheter.

Skole	**Behov (b)**
1	100
2	130

Tabell 4.1 Materialbehov for skole 1 og 2

Leverandør	Pris pr. materialenhet (p)
1	5,5
2	7
3	4
4	5

Tabell 4.2 Leverandørenes materialpris

Risikokostnad (r)	Skole	
Leverandør	1	2
1	10	5
2	5	10
3	10	15
4	10	15

Tabell 4.3 Risiko ved å kjøpe materiale fra leverandør i til skole j

Leverandør	Leveringskapasitet (k)	Kjøpskrav (20 %)
1	70	14
2	100	20
3	90	18
4	60	12

Tabell 4.4 Leverandørenes leveringskapasitet

Både det å gjenkjenne hva slags type problem man har med å gjøre, og det å formulere problemet matematisk, krever erfaring – spesielt hvis problemet er stort og uoversiktlig. Her har vi å gjøre med et LP-problem. Hvis man ikke ser det ut fra beskrivelsen av problemet, vil det i hvert fall bli tydelig når man har satt opp problemet på matematisk form.

Før man starter å formulere selve problemet, kan det være hensiktsmessig å definere inndataparametrene. I dette eksempelet har vi følgende parametre:

b_j – materialbehov for skole j
p_i – materialpris for leverandør i
r_{ij} – ”risikokostnad” ved å kjøpe materiale til skole j fra leverandør i
k_i – leveringskapasitet til leverandør i

Når man skal formulere problemet matematisk, kan det være gunstig å starte med å identifisere beslutningsvariablene. I dette tilfelle ønsker vi å bestemme verdien på mengden materiale hver skole skal få fra hver leverandør. Dette kan representeres som beslutningsvariable på følgende måte:

x_{ij} – mengden materiale som kjøpes fra leverandør *i* og brukes til å bygge skole *j*

Neste steg kan være å formulere objektfunksjonen for problemet. Vi ønsker å minimere kostnadene våre, det vil si summen av innkjøpskostnad og risikokostnad. Objektfunksjonen kan da defineres slik:

$$z = \sum_{i=1}^{4}\sum_{j=1}^{2} p_i x_{ij} + \sum_{i=1}^{4}\sum_{j=1}^{2} r_{ij} x_{ij} \qquad (4.5)$$

Det som gjenstår før modellen er komplett, er å definere beskrankningene. Løsningsrommet til beslutningsvariablene x_{ij} er blant annet begrenset av at de ikke kan være negative. Vi kan ikke kjøpe en negativ mengde materiale fra leverandørene. Dermed får vi en beskrankning som sier at:

$$x_{ij} \geq 0 \qquad for\ i = 1,...,4\ og\ j = 1,2 \qquad (4.6)$$

For å kunne bygge skolene, er vi avhengige av å kjøpe nok materiale til å dekke behovet. Dette kan formuleres som en beskrankning på følgende måte:

$$\sum_{i=1}^{4} x_{ij} \geq b_j \qquad for\ \ j = 1,2 \qquad (4.7)$$

Siden det er slik at materialbehovet til skolene kan være større enn leveringskapasiteten til én enkelt leverandør, må det formuleres en beskrankning som sørger for at x_{ij} for hver leverandør i ikke er større enn denne leverandørens leveringskapasitet:

$$\sum_{j=1}^{2} x_{ij} \leq k_i \qquad for\ i = 1,...,4 \qquad (4.8)$$

I tillegg er det slik at hver leverandør skal få selge oss minst 20 % av leveringskapasiteten sin. Denne beskrankningen formuleres som:

$$\sum_{j=1}^{2} x_{ij} \geq 0.2k_i \qquad for\ i = 1,...,4 \qquad (4.9)$$

Etter å ha formulert problemet matematisk ser vi at både objektfunksjonen og beskrankningene er lineære, og at beslutningsvariablene er kontinuerlige. Dette betyr at det er et LP-problem. For å løse dette problemet kan det implementeres i optimeringsprogramvare som for eksempel AMPL [11]. Imidlertid er det slik at små og oversiktlige problemer ofte løses like greit i MS Excel ved hjelp av Solver-funksjonen.
Tabell 4.5 viser hvordan det vil være mest optimalt å fordele innkjøpet av materiale mellom de fire leverandørene, gitt kriteriene man hadde på forhånd.

Leverandører	Skoler 1	2	SUM
1	0	70	70
2	97	3	100
3	0	48	48
4	3	9	12
SUM	100	130	

Tabell 4.5 Optimal fordeling av innkjøpsmengder fra de forskjellige leverandørene

5 Heltallsprogrammering

Dette kapitlet vil begrense seg til heltallsmodeller som er lineære. En lineær heltallsmodell er i prinsippet et LP-problem med diskrete variable. Det har noen gunstige effekter i forhold til løsningsmetoder, som vi vil se i de følgende avsnittene.

5.1 Problemtyper

I de fleste tilfeller er det slik at når man står overfor et heltallsproblem, vil dette problemet i stor grad ligne på andre problemer som har vært formulert før. Slike fellestrekk gjør at man kan dele heltallsmodeller inn i forskjellige kategorier. Dette er etablerte kategorier som man finner igjen i de fleste kilder for optimeringsteori. Under beskrives de forskjellige typer heltallsproblemer gjennom en inndeling lik den man kan finne i blant annet [5]. Det anbefales å lese denne eller andre kilder, for eksempel [7] eller [9], for en mer omfattende innføring i heltallsprogrammering.

Budsjetteringsmodeller

Dette er en type problemer hvor målet er å velge en optimal samling av objekter, investeringer e.l. der man har et budsjett som skal overholdes, eller et sett av ressurser som ikke kan overforbrukes. Et velkjent budsjettproblem er ryggsekkproblemet. Der har man en ”ryggsekk ”med begrenset kapasitet. Denne skal man fylle så godt man kan med objekter. Hvert objekt har en viss verdi dersom det blir valgt ut, men det har også en vekt. Målet er å maksimere verdien av innholdet i sekken, uten å gå utover sekkens vektkapasitet. Dette er den enkleste formen for budsjetterings-problem og brukes selvsagt helst i overført betydning. En noe mer komplisert variant av ryggsekkproblemet, har man når objektene man skal velge mellom for eksempel er investeringsprosjekter. I slike tilfeller kan det eksistere krav som for eksempel at prosjektene er gjensidig utelukkende, eller det kan eksistere avhengigheter mellom dem. Det kan også være et tidsperspektiv man må ta hensyn til.

Set covering, set packing og set partitioning

I mangel på gode norske ord, vil de engelske benevnelsene set packing, set covering og set partitioning bli brukt for denne kategorien av problemer. Disse tre typene problemer har til felles at variablene definerer hvorvidt elementer i forskjellige delmengder hører med til løsningen på problemet. Dette kan forklares gjennom et eksempel.

Vi tar utgangspunkt i at FFI skal gå til innkjøp av optimeringsprogramvare. Det finnes fire forskjellige typer programmer som inneholder forskjellige løsningsalgoritmer. Tabell 5.1 viser hva slags algoritmer de forskjellige programmene dekker.

	Program 1	**Program 2**	**Program 3**	**Program 4**
Lineærprogrammering	X		X	X
Heltallsprogrammering	X	X		X
Ulineær programmering			X	X
Objektfunksjonsverdi	2	3	5	10

Tabell 5.1 Alternative optimeringsverktøy og algoritmene de inneholder

Objektfunksjonsverdien kan for eksempel representere innkjøpskostnad, slik at man får et minimeringsproblem, eller kvalitet på programmene, noe som gir et maksimeringsproblem. Med dette utgangspunktet kan man definere et set covering-, set packing- eller set partitioning-problem på følgende måter:

Set covering innebærer at man definerer beskrankninger som sier at vi skal kjøpe inn en kombinasjon av programmer som gjør at *minst ett* program dekker LP, *minst ett* dekker heltallsprogrammering og *minst ett* dekker ulineær programmering. Hvis objektfunksjonsverdien representerer innkjøpskostnad, vil optimal løsning være å kjøpe program 1 og 3.

For set packing sier beskrankningene at i kombinasjonen av programmer vi kjøper inn, skal *maksimalt ett* program dekke hver av de forskjellige algoritmetypene. Hvis objektfunksjonsverdien representerer kvaliteten på programmene, vil optimal løsning være program 4.

For set partitioning sier beskrankningene at i kombinasjonen av programmer skal *akkurat ett* dekke hver av de forskjellige algoritmetypene. Hvis objektfunksjonsverdien representerer innkjøpskostnad, vil optimal løsning være program 2 og 3.

Allokeringsproblemer

Allokeringsproblemer handler om å allokere én type objekter til en annen type objekter, for eksempel personell til oppgaver. Det defineres en variabel som er lik 1 dersom et objekt av den ene typen settes sammen med et objekt av den andre typen, og 0 ellers. Det kan eksistere flere enn to typer objekter i denne typen problemer, men to er det mest vanlige. I allokeringsproblemer kan det finnes avhengigheter som gjør at problemet blir ulineært istedenfor lineært. I slike problemer er det vanskelig å finne globalt optimum, og man må ta i bruk heuristiske metoder.

Ruteproblemer

Generelt handler ruteproblemer om at man ønsker å finne den beste ruten mellom et sett av lokasjoner. I noen typer ruteproblemer har man et depot som man stadig må innom, slik at problemet handler om å finne mange ruter. Ruteproblemer er symmetriske dersom veien mellom

to lokasjoner er like lang fram og tilbake, og asymmetriske hvis ikke. Et av de mest kjente ruteproblemer er det såkalte Traveling salesperson-problemet, se for eksempel [5] eller [9].

Nettverksdesign
Heltallsprogrammering kan brukes innen nettverksdesign til å bestemme hvilke veier mellom noder i et nettverk man bør bygge eller eventuelt ”åpne”. Metodene er relevante for alle nettverk hvor det er noe som skal ”flyte”. Eksempler er telenett, kraftnett og kloakknett. I nettet finnes et sett av noder som har en gitt etterspørsel. For eksempel kan denne etterspørselen være husstanders strømforbruk. Objektfunksjonen representerer som regel faste kostnader ved å opprette en vei i nettet, samt driftskostnader ved å holde veien åpen. Beskrankningene sørger både for at flyten i nettet ikke overskrider kapasiteten på veiene, og at etterspørselen til nodene blir tilfredsstilt.

Scheduling
Scheduling handler om å allokere ressurser over tid. Man har typisk et sett av jobber som skal prosesseres, og ressurser som kan prosessere dem. Et eksempel er en fabrikk som produserer et eller annet produkt som må gjennom forskjellige produksjonsstadier. Ressursene kan bare prosessere én jobb av gangen. Man kan ha krav til at visse jobber må prosesseres før andre jobber, og man kan ha tidsfrister for når visse jobber må være ferdig. Det er også ofte slik at noen jobber krever helt spesielle ressurser, kanskje også i en gitt rekkefølge. For schedulingproblemer er det vanlig at man ønsker å minimere den tiden det tar å bli ferdig med alle jobbene.

5.2 Løsningsmetoder

Den største utfordringen ved diskrete problemer er at det ofte finnes veldig mange lokale optima. En metode som er mye brukt for å løse lineære heltallsproblemer, er den såkalte Branch and bound-algoritmen (BB), eller varianter av denne. Gode beskrivelser av hvordan denne algoritmen fungerer, finnes i [4] og [5].

Noen problemer er av en slik karakter at man kan løse dem ved hjelp av dynamisk programmering. Et eksempel på et slikt problem er gitt i kapittel 5.3.

Av og til er det imidlertid slik at heltallsproblemer er uløselige innenfor rimelig tid. For slike problemer, samt for ulineære heltallsproblemer, må man ty til heuristiske metoder for å finne løsninger som er tilnærmet optimale. Noen slike heuristiske metoder omtales i kapittel 8.

5.3 Eksempel på anvendelse

Vi tar igjen turen til Afghanistan, til det norskledede PRT-et i Meymaneh hvor Sjef PRT har fått en utfordring hvor optimering viser seg å være til stor hjelp.

Det er mandag morgen i PRT Meymaneh. Sjef PRT sitter som vanlig med en lang liste over sikkerhetsoppgaver som bør utføres i området. Han har begrenset med ledige ressurser og må

prioritere de viktigste oppgavene. Han velger ut fire oppdrag som han ønsker å finne ut om han klarer å bemanne med et tilstrekkelig antall soldater:

1) Gjennom etterretning har man fått sterke indikasjoner på at noen planlegger å angripe guvernørens bolig i løpet av uken. Det er ønskelig at PRT-et bidrar med soldater til vakthold for å sikre guvernørens hjem og familie.
2) Man ønsker å starte opp byggingen av en lenge planlagt radiomast i regionen. Denne er en del av ISAF sitt arbeid med å bygge radiomaster rundt omkring i hele landet for å bedre informasjonsspredningen til folket. Da det er mange som ønsker å sabotere dette arbeidet, kan ikke byggingen starte uten skikkelig vakthold. Oppstart av byggingen har allerede blitt utsatt flere ganger grunnet manglende ressurser.
3) Et av PRT-ets Military Observer Team (MOT) er ute på oppdrag og har behov for etterforsyninger. De befinner seg ca. en halv dags reise fra PRT-et. Normalt pleier forsyningene å bli sendt med drosje, men grunnet økte sikkerhetstrusler på veiene i området, er mange av de lokale drosjeeierne skeptiske til å ta turen. Denne gangen må derfor PRT-ets egne soldater kjøre ut med forsyninger til MOT-et.
4) En skole for jenter er nettopp startet opp i regionen. Man frykter at opprørsgrupper som er imot at jenter utdannes skal aksjonere mot skolen. Mange foreldre sier at de vil vurdere å holde jentene hjemme hvis ikke PRT-et stiller med soldater til vakthold.

Sjef PRT har 5 lag som er klar for å sendes ut på oppdrag. Målet til Sjef PRT er å avgjøre hvor mange lag han skal allokere til hvert oppdrag, slik at han maksimerer verdien av å utføre oppdragene.

Man har estimert verdiene av å sende ut forskjellig antall lag på oppdragene. Disse er gitt i tabell Tabell 5.2. Verdiene er regnet ut basert på flere inputfaktorer. Eksempler er viktigheten av oppdraget, hvor godt oppdraget kan utføres med et gitt antall lag, hvor stor risiko det er forbundet med å utføre oppdraget o.l.

Antall lag	Oppdrag 1	Oppdrag 2	Oppdrag 3	Oppdrag 4
0	0	0	0	0
1	9	8	15	7
2	17	16	15	10
3	23	18	15	10
4	23	18	15	10
5	23	18	15	10

Tabell 5.2 Estimert verdi av å sende forskjellig antall lag ut på oppdragene

Dette problemet kan formuleres som et ryggsekkproblem. Man kan si at ”ryggsekken” har en kapasitet på 5 lag. Det å sende ut et lag på et oppdrag har en verdi, men også en ”vekt” som tilsvarer antall lag man bruker opp på dette oppdraget. Målet er å maksimere total verdi, samtidig

som man ikke overskrider kapasiteten på 5 lag, det vil si at man ikke sender ut flere lag på oppdrag enn man har tilgjengelig.

Inndata for modellen er gitt ved:

W – antall tilgjengelige lag
r_{ij} – verdien av å sende ut i antall lag på oppdrag j

Ryggsekkproblemer er heltallsproblemer, og i dette tilfellet vil variablene være gitt som:

$$x_{ij} = \begin{cases} 1, hvis\ i\ antall\ lag\ sendes\ ut\ på\ oppdrag\ j \\ 0, ellers \end{cases}$$

Målet med modellen er å maksimere verdien av å sende ut lag på oppdrag. Objektfunksjonen formuleres dermed som:

$$z = \sum_{i=0}^{5} \sum_{j=1}^{4} r_{ij} x_{ij} \tag{5.1}$$

x_{ij} fungerer som en ”av/på”-knapp”, slik at verdien av r_{ij} slår inn når i og j er slik at x_{ij} er lik 1. Det at x_{ij} er binær og bare kan ta verdiene 0 og 1 må formuleres i en beskrankning:

$$x_{ij} = 0\ eller\ 1 \qquad for\ i = 0,...,5\ og\ \ j = 1,...,4 \tag{5.2}$$

Vi har også behov for en beskrankning som sørger for at det ikke allokeres flere lag til oppdrag enn antall tilgjengelige lag:

$$\sum_{i=0}^{5} \sum_{j=1}^{4} i x_{ij} \leq W \tag{5.3}$$

Det skal bare allokeres lag til oppdrag én gang. Det vil si at det for eksempel ikke kan allokeres både to lag og tre lag til et og samme oppdrag. Dette kravet formuleres slik:

$$\sum_{i=0}^{5} x_{ij} \leq 1 \qquad for\ \ j = 1,...,4 \tag{5.4}$$

Formuleringen som her er gitt av dette problemet, er en heltallsmodell som kan løses ved hjelp av programvare som for eksempel AMPL [11]. Løsningen under for problemet er gitt i Tabell 5.3.

Oppdrag	Antall lag
1	2
2	2
3	1
4	0

Tabell 5.3 Optimal allokering av lag til oppdrag

Resultatet fra optimeringen ble at det sendes to lag ut på oppdrag 1 og 2, ett lag ut på oppdrag 3 og ingen lag ut på oppdrag 4.

Det å løse problemet på denne måten går bra når problemet er relativt lite, men som vanlig når man har å gjøre med heltallsproblemer, kan størrelse og kompleksitet bli et problem. Ryggsekkproblemer er en av flere typer problemer som er gunstige å løse ved hjelp av såkalt dynamisk programmering. Dynamisk programmering handler om at man deler opp et optimeringsproblem med flere variable i delproblemer, der hvert delproblem bare har én variabel. Problemet løses så ved hjelp av en rekursiv funksjon, slik at løsningen på ett delproblem brukes som input til neste delproblem. Hver delløsning er en del av det totale problemets optimale løsning. Når den rekursive funksjonen har gått gjennom alle stegene, vil den dermed ha funnet problemets optimale løsning.

Ikke alle problemer har en struktur som gjør at de kan formuleres slik at dynamisk programmering kan brukes. Det finnes ingen ”regler” for hvordan et problem skal se ut for at det skal kunne løses ved hjelp av dynamisk programmering, utover det at det må være flere enn én variabel. Når man får erfaring med å bruke metoden, vil man etter hvert kunne gjenkjenne problemer hvor denne metoden egner seg. Dynamisk programmering er beskrevet i blant annet [5], [8] og [9].

6 Nettverksmodeller

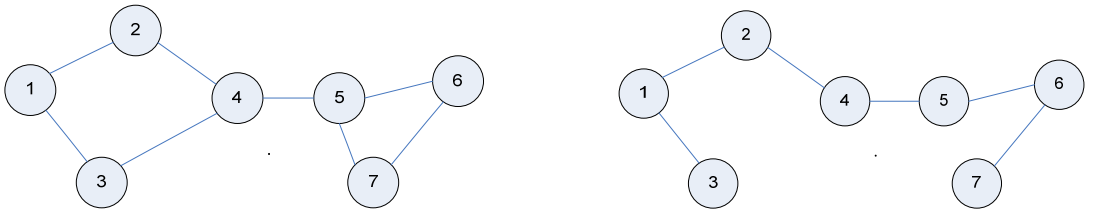
Nettverksmodeller er spesialvarianter av lineære optimeringsproblemer og får alltid et eget kapittel i optimeringsbøker. Grunnen er at disse modellene karakteriseres ved at de kan fremstilles som en graf, eller et nettverk, noe som gjør at man har noen meget effektive algoritmer for å løse problemene. Dette er særlig praktisk for diskrete problemer som i utgangspunktet er vanskelige å løse.

6.1 Problemtyper

Det finnes forskjellige typer nettverksproblemer. Dette kapitlet vil omtale noen av dem. For nærmere beskrivelse av løsningsalgoritmene for disse nettverksproblemene, se [5], [7] og [9].

Minimale spenntrær

Et spenntre er et tre som binder sammen alle nodene i et nettverk, uten sykler. Figur 6.1 a) er ikke et spenntre fordi det inneholder sykler, mens Figur 6.1 b) er et spenntre. I et nettverk der forbindelsene mellom noder representerer avstand mellom nodene, vil minimalt spenntre være det spenntreet som har korteste totale lengde på grenene. Algoritmer som finner minimalt spenntre kan blant annet brukes når man skal minimere kostnadene ved å bygge ut et nett der antall meter har mye å si for kostnadene. Dette kan eksempelvis gjelde kabelnett, telenett, strømnett eller kloakknett.



Figur 6.1 a) Tilfredsstiller ikke kravene til spenntre b) Spenntre

Korteste vei

Algoritmer for å finne korteste vei brukes når man har et eksisterende nett og ønsker å finne korteste vei mellom noder som befinner seg i nettet. Et typisk problem er å finne korteste vei mellom to byer, der det finnes flere alternative veier mellom byene. I tillegg til å finne korteste vei mellom to noder, er det mulig å finne korteste vei fra én node til alle andre og fra alle noder til alle andre noder. Disse problemstillingene krever forskjellige løsningsalgoritmer som demonstrert i for eksempel [5].

Korteste vei handler ikke nødvendigvis om avstand. Det å velge en vei gjennom nettverket kan eksempelvis representere det å velge en sekvens av aktiviteter. Slik kan korteste vei-algoritmer anvendes for å minimere total tidsbruk ved sekvensen av aktiviteter.

Kritisk vei

Algoritmer for å finne kritisk vei i et nettverk anvendes typisk på prosjektplanleggingsproblemer der man har aktiviteter som har krav til ressurser og tidsbruk, i tillegg til krav om at visse aktiviteter skal utføres før andre. Slike krav om rekkefølge representeres i et nettverk som viser hvilke aktiviteter som kan gå i parallell, og hvilke som må gå i serie. Kritisk vei er korteste avstand fra ”start” til ”mål” i prosjektet, og angir korteste mulige gjennomføringstid.

Minimal kostnad

Å finne minste kostnad ved flyt gjennom et nettverk er en typisk problemstilling for eksempel for distribusjon av varer fra fabrikker til varehus og kunder. Nettverket karakteriseres ved at varer flyter fra et visst antall forsyningsnoder (fabrikker) til et visst antall etterspørselsnoder (kunder). På veien gjennom nettverket kan det være krav til at varene skal passere gjennom bestemte

prosesseringsnoder før de kan sendes videre til kundene. Kantene mellom nodene representerer kostnadene ved å sende varer mellom de respektive noder, samt kapasitet – det vil si hvor mye varer man kan sende på en gang. Hvis det for eksempel er en bil som skal frakte varer fra et sted til et annet, vil man ha en viss kostnad per kilometer samt en øvre lastekapasitet. Poenget med å sette opp distribusjonsprosessen som et slikt nettverk, er å finne optimal distribusjonsrute for varene slik at total kostnad minimeres. Eksempel på et minimal kostnad-problem er gitt i kapittel 6.2.

Maksimal flyt

Denne typen problem handler om at man ønsker å finne ut hvor mye "varer" man kan sende ut fra en kilde gitt kapasitetene på kantene i nettverket. Det kan for eksempel gjelde en eller annen form for transportnett. Nettverkene karakteriseres av at de har én kilde og ett sluk. Slik sett er de en spesialvariant av nettverk for minimal kostnad, med kostnad 0 på alle kanter og etterspørsel 0 for alle noder. Dersom man innfører en ny kant fra sluk til kilde med kostnad -1 og ingen øvre skranke på kapasitet, kan maksimal flyt finnes ved at man løser problemet som et minimal kostnad-problem. Imidlertid finnes det også algoritmer som løser maksimal flyt-problemer uten at man gjør dem om til minimal kostnad-problemer.

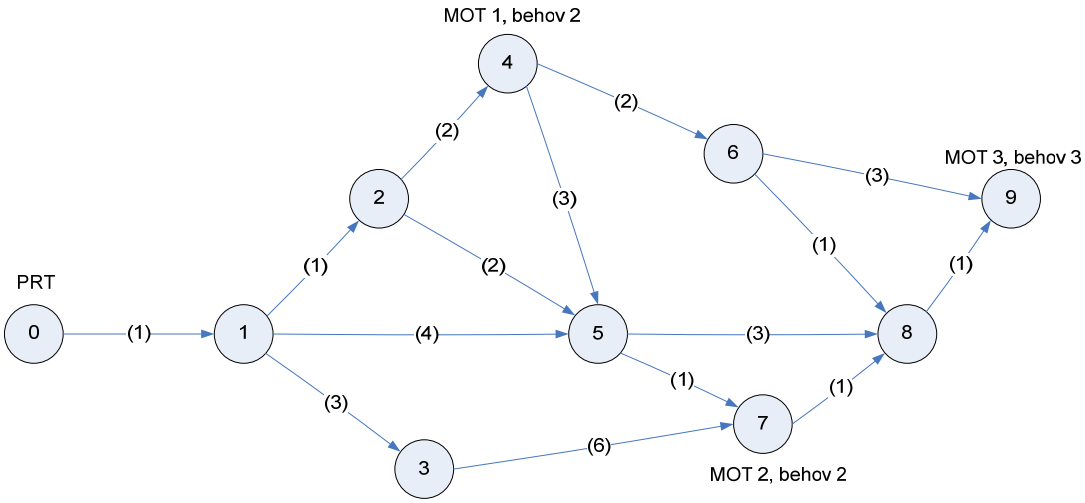
6.2 Eksempel på anvendelse

Vi skal igjen se på en problemstilling som vi tenker oss har oppstått i det norskledede PRT Meymaneh i Afghanistan. Fra PRT-et er det sendt ut tre MOT-er på oppdrag forskjellige steder i Faryab-provinsen. Disse har vært ute i en uke og har nå behov for etterforsyninger. MOT-enes behov for etterforsyninger angis i antall biler, og er gitt i Tabell 6.1.

MOT	Behov for etterforsyning (antall biler)
1	2
2	2
3	3
Totalt	7

Tabell 6.1 MOT-enes behov for etterforsyninger, oppgitt i antall biler som må kjøre til dem

Det er viktig for PRT-et at bilene bruker så kort tid som mulig på turene. Siden det finnes flere mulige kjøreruter til områdene hvor MOT-ene er utstasjonert, har PRT-et behov for en modell som kan beregne hvilke kjøreruter som vil medføre kortest mulig tidsbruk for bilene.
Figur 6.2 illustrerer et nettverk som viser kjøretider i timer for forskjellige kjøreruter i et område som dekker de tre lokasjonene hvor MOT-ene er utplassert. Nodene representerer lokasjoner der det er mulig å velge mellom flere enn én rute, eller der veier møtes.



Figur 6.2 Kjøretider (i timer) i området der MOT-ene er utstasjonert

Det å finne korteste kjøreruter for bilene er det samme som å beregne minimal kostnad, hvis man ser på kjøretidene som kostnader. PRT-noden er en forsyningsnode som sender ut biler. Nodene hvor MOT-ene befinner seg er etterspørselsnoder som har behov for besøk av et visst antall biler, som gitt i Tabell 6.1. Den matematiske formuleringen av problemet settes opp som et vanlig LP-problem. Beslutningsvariabel i modellen blir:

x_{ij} – antall biler som sendes fra node *i* til node *j*

I tillegg har vi at *V* er settet av alle noder og *A* er settet av alle definerte kanter. Målet med modellen er å minimere total tidsbruk for bilene. Objektfunksjonen som skal minimeres blir:

$$z = \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (6.1)$$

Der c_{ij} er kjøretiden fra node *i* til node *j*.

Vi trenger en beskrankning som sørger for at strømmen av biler inn og ut av nodene blir riktig. Det vil for eksempel si at strømmen inn i en etterspørselsnode må være større eller lik etterspørselen. Siden en bil ikke kjører videre til noen ny MOT etter at den har vært innom et MOT og levert fra seg forsyninger, må strømmen ut av noden være lik strømmen inn i noden, minus etterspørselen. For forsyningsnoder er det motsatt. I tillegg vil det for alle andre noder være slik at strøm inn i noden er lik strøm ut av noden. Dette kan formuleres generelt som:

$$\sum_{(i,k)\in A} x_{ik} - \sum_{(k,j)\in A} x_{kj} = b_k \qquad for\ alle\ k \in V \qquad (6.2)$$

der b_k er etterspørselen i node *k*. I forsyningsnoder er b_k negativ. I vårt eksempel vil b_0 være -7 siden det er 7 biler som skal sendes ut fra PRT-et med etterforsyninger.

Til slutt må vi huske at man ikke kan sende et negativt antall biler mellom noder:

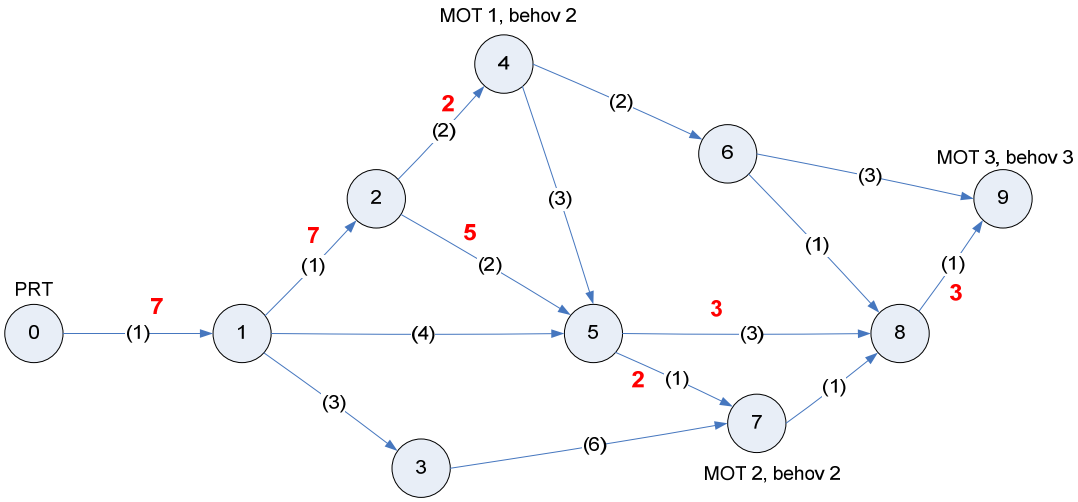
$$x_{ij} \geq 0 \qquad for\ alle\ (i,j) \in A \tag{6.3}$$

Det antas at man ikke har noen kapasitetsbegrensning på veiene. Det vil si at man kan kjøre så mange biler man vil på enhver strekning.

Et såpass lite nettverksproblem som dette, lar seg raskt implementere og løse i for eksempel MS Excel, ved hjelp av Solver. Det løses da som et vanlig LP-problem. Som nevnt tidligere, er det imidlertid lite effektivt å løse større nettverksproblemer som LP- eller heltallsproblemer, da det finnes algoritmer som er mer effektive. Dette gjelder særlig for heltallsproblemer. Vårt eksempel kunne godt vært formulert som et heltallsproblem, der man spesifiserte at mengden biler som sendes mellom noder må være heltall.

Algoritmer for å løse minimal kostnad-problemer, og dermed eksempelet over, kan studeres i kildene nevnt i innledningen til dette kapitlet.

Figur 6.3 viser løsningen for eksempelet vårt. De røde tallene viser hvor mange biler som kjører langs de forskjellige rutene. Biler som har vært innom et MOT og levert fra seg forsyninger, antar man at snur og kjører korteste vei hjem til PRT-et. Dette er ikke illustrert i nettverket.



Figur 6.3 Optimale kjøreruter for bilene i eksempelet

7 Ulineær programmering

Dette kapitlet vil gi en kort beskrivelse av de vanligste formene for ulineære problemer, sammen med omtale av hvordan disse kan løses. Grundigere beskrivelser av, og eksempler knyttet til, temaet ulineær programmering kan studeres i [7] eller [8].

Det finnes ikke én algoritme som kan løse alle typer ulineære problemer. Felles for løsningsmetodene er at alle baserer seg på søk. Forskjellen mellom dem er hvordan søkene utføres. Dette vil variere med hva slags type ulineært problem man har med å gjøre. Først og fremst kan man dele ulineære problemer i ubegrensede og begrensede problemer.

7.1 Ulineær optimering uten beskrankninger

For ulineære problemer uten beskrankninger gjelder det å optimere objektfunksjonen over alle verdier av beslutningsvariablene. Søkeprosedyren blir forholdsvis enkel fordi det er lett å finne tillatte punkter. Utfordringen i søket begrenser seg dermed til å bestemme i hvilken retning man skal bevege seg i hver iterasjon, samt hvor langt man skal bevege seg i denne retningen. Man slipper altså å tenke på hvorvidt punktene er tillatte eller ikke. En generell beskrivelse av iterasjonene i søkeprosedyrene finnes i [7].

7.2 Ulineær optimering med beskrankninger

Ulineære problemer med beskrankninger er noe mer komplisert. Utfordringen er her å finne et lovlig startpunkt for søkemetoden, samt å finne søkeretning som er slik at man holder seg innenfor tillatt område.

Problemer med lineære beskrankninger

Dersom problemet har lineære beskrankninger, kan man bruke tilpasninger av Simplex-metoden til å finne optimal løsning. Eksempel på denne typen problem er såkalt kvadratisk programmering. Dette er en forholdsvis vanlig type ulineært problem, der objektfunksjonen inneholder produktet av to variable, eller én variabel opphøyd i annen potens. Kvadratiske problemer støter man blant annet på innen økonomi og kjemi.

Problemer med ulineære beskrankninger

For problemer som har ulineære beskrankninger skiller man mellom konvekse eller konkave problemer der man kan garantere at et optimum er globalt, og problemer som ikke er konvekse eller konkave, og dermed i utgangspunktet ikke lar seg løse. Separabel programmering er en type konveks programmering der objektfunksjon og beskrankninger er separable funksjoner i variablene. En konsekvens av dette er at man kan tilnærme disse ikke-lineære funksjonene som stykkevis lineære funksjoner [7]. Disse approksimasjonene kan løses ved hjelp av Simplex-metoden. Jo finere man kan linearisere funksjonene, jo nærmere kommer man det virkelige optimum.

For problemer hvor objektfunksjonen hverken er konveks eller konkav, finnes det algoritmer som finner *lokale* optima relativt greit. Selv om man i utgangspunktet ikke har noen garanti for at et optimum er globalt, finnes det noen typer problemer som er slik at man ved hjelp av noen triks kan løse dem likevel. Eksempler på dette er fraksjonelle programmeringsproblemer, det vil si problemer som har en objektfunksjon som består av en brøk med beslutningsvariable både over og under brøkstreken. Slike problemer kan transformeres til LP-problemer og løses ved hjelp av Simplex. Denne prosedyren kan studeres i [7].

De forskjellige typene ulineære problemer som er omtalt i dette kapitlet, er problemer hvor det er utviklet tilpassede algoritmer som gjør at man kan, eller er nær ved å, finne optimum. For andre typer ulineære problemer der dette ikke er mulig, kan det være en mulighet å ta i bruk heuristiske metoder for å forsøke å finne lovlige løsninger som er så gode som mulig. Heuristiske metoder er tema for neste kapittel.

8 Heuristiske metoder

Når man har å gjøre med heltallsproblemer eller ulineære, kontinuerlige problemer, må man ofte ta i bruk heuristiske løsningsmetoder fordi problemet ikke lar seg løse eksakt. Slike metoder garanterer ikke at løsningen man finner er optimal. Som regel kan man ikke engang si noe sikkert om hvor langt unna optimum løsningen er. Likevel vil heuristiske metoder ofte være nyttige da de i hvert fall finner en gyldig løsning, og ofte er denne løsningen også bedre enn veldig mange andre gyldige løsninger. Det *kan* naturligvis også hende at løsningen faktisk er optimal..

Mange problemer er slik at størrelse, målt ved antall variable, avgjør om det er løsbart eller ikke. Hvis det er mulig å lage en mindre variant av problemet som lar seg løse eksakt, kan dette være nyttig. Dersom man løser en slik liten variant både eksakt og ved hjelp av den heuristiske metoden man har tenkt å bruke på den større varianten av problemet, vil man få en pekepinn på hvor god den heuristiske metoden er. Selv om den heuristiske metoden finner en løsning på det lille problemet som viser seg å være optimal, kan man ikke uten videre konkludere med at den skal gjøre en like god jobb på større varianter av problemet.

For store, lineære heltallsproblemer kan det være nyttig å løse problemet som et LP-problem, hvor man har fjernet heltallsbeskrankningen. Dette vil gi en øvre eller nedre skranke for løsningen, slik at man i hvert fall vet at man ikke kan finne noen løsning som er bedre enn det. Det å finne slike øvre eller nedre skranker er viktig ved bruk av heuristiske metoder. Blant annet brukes det som en indikasjon på når man har funnet en så god løsning at det er greit å slutte å søke.

Generelt kan man si at hensikten med heuristiske metoder er å søke etter løsninger på en ”intelligent” måte. Dette innebærer at metodene skal prøve å unngå å søke gjennom absolutt alle løsninger (brute force), samt at de skal unngå å låse seg fast i lokale optima. I dette kapitlet omtales tre slike metoder.

8.1 Tabusøk

I utgangspunktet ble tabusøk utviklet for diskrete optimeringsproblemer. Dette er fortsatt det aller mest vanlige anvendelsesområdet, selv om det er gjort forsøk på å tilpasse algoritmen til kontinuerlige problemer. Noen av teknikkene som er blitt benyttet for å gjøre en slik tilpasning, er beskrevet i [12].

En heuristisk metode har to mål. Det første er å lete etter lokale optima. For å gjøre dette brukes vanligvis noe som kalles lokalt søk. Et lokalt søk starter med å finne en tilfeldig løsning, og så leter den etter bedre løsninger blant "naboløsningene". Dersom den finner en bedre løsing, hopper den videre til denne løsningen. Slik fortsetter den til ingen bedre løsninger finnes i nærheten. En slik algoritme vil låse seg fast i lokale optima, fordi et slikt punkt er det beste i umiddelbar nærhet. Tabusøk handler om å oppnå det andre målet med heuristiske metoder, nemlig å hindre fastlåsing i lokale optima. I et tabusøk får man lov til å gå videre fra et lokalt optimum til en løsning som ikke nødvendigvis er bedre. For hver gang man beveger seg til en ny løsning, føyer man forrige løsning til en "tabuliste" over løsninger man ikke får lov å gå tilbake til før det har gått et visst antall iterasjoner. Algoritmen sørger for hele tiden å holde rede på hvilken løsning som er den beste den har funnet så langt. Når man har nådd en forhåndsbestemt maksimaltid, stopper algoritmen.

I [5] kan man lese mer om lokalt søk og om tabusøk.

8.2 Simulated annealing

I likhet med tabusøk, forsøker simulated annealing å hindre fastlåsing i lokale optima. Når algoritmen skal flytte seg fra en løsning til en annen, velger den en tilfeldig løsning blant lovlige naboløsninger. Dersom den nye løsningen er bedre enn den nåværende, hopper algoritmen videre til den nye løsningen. Dersom den nye løsningen derimot er dårligere, vil algoritmen med en gitt sannsynlighet hoppe videre til den nye løsningen. Denne sannsynligheten er stor i begynnelsen, og så settes den lavere etter hvert som algoritmen leter gjennom nye løsninger. Algoritmen stopper når man når en forhåndsbestemt maksimaltid. Simulated annealing er også beskrevet i [5].

8.3 Genetiske algoritmer

Genetiske algoritmer er inspirert av evolusjonstanken og "survival of the fittest". Virkemåten til genetiske algoritmer kan variere, men de aller fleste inneholder de samme elementene. Prinsippet går ut på at man lager seg en såkalt populasjon av mulige løsninger på optimeringsproblemet, og ved å utføre visse operasjoner på disse løsningene, eller kromosomene som de kalles, skal de utvikle seg til å bli bedre løsninger. Operasjonene kalles seleksjon, krysning og mutasjon. Ved å utføre disse operasjonene går man fra én generasjon til neste generasjon i algoritmen.

Seleksjon handler om å finne de beste løsningene i populasjonen, og sørge for at disse løsningenes "gode egenskaper" føres videre til neste generasjon. Hvert kromosom har en viss godhet (fitness), det vil si en verdi som forteller hvor god denne løsningen er. Som regel vil dette være den verdien kromosomet gir på optimeringsproblemet. Poenget med seleksjon er å sørge for at de kromosomene med best fitness har størst sannsynlighet for å bli valgt ut for reproduksjon.

Krysning er den operasjonen som sørger for reproduksjon. Etter at det er gjort en seleksjon settes kromosomer sammen til par. Ut fra et slikt par dannes det to avkom, det vil si to nye løsninger.

Mutasjon av avkommene skjer med en viss, forholdsvis liten, sannsynlighet. Mutasjonen sørger for at løsningen forandres litt. Dette for å skape variasjon i befolkningen. Det kan være hensiktsmessig å tilføre nye, og forhåpentligvis bedre, egenskaper til avkommene.

Når disse operasjonene er utført på en generasjon, vil man igjen gjøre en seleksjon og utføre prosessene på nytt. De kromosomene som ikke blir valgt ut gjennom seleksjonen, lar man dø ut. Antallet kromosomer i populasjonen skal holdes på det nivået man startet med. Gode egenskaper vil gå i arv og gi bedre og bedre løsninger. Dårlige egenskaper vil dø ut litt etter litt. I hver generasjon vil kromosomet eller kromosomene med best fitness gi algoritmens hittil beste løsning på optimeringsproblemet. Algoritmen kjører til den når en betingelse for å avslutte. Alternative betingelser for å stoppe kan være at man har produsert et visst antall generasjoner, at man har nådd en forhåndssatt maksimaltid for kjøringen, at man har nådd et platå der løsningene ikke har forbedret seg på et visst antall generasjoner, eller at løsningen har nådd en viss verdi.

Genetiske algoritmer er lettere å forstå dersom man ser på et eksempel. Under beskrives det hvordan man kan bruke en slik algoritme for å løse et schedulingproblem. Denne typen problemer er ikke uvanlig å løse ved hjelp av genetiske algoritmer. For en grundigere innføring i genetiske algoritmer generelt, se [13].

8.4 Eksempel på anvendelse av genetisk algoritme

Når man bruker genetiske algoritmer er det vanlig å representere kromosomene ved hjelp av bitstrenger. Dette er beskrevet utførlig i [13]. I dette eksempelet settes modellen opp på en litt annen måte. Eksempelet og løsningsmetoden som demonstreres, bygger på arbeidet gjort i [14] og [15].

I dette eksempelet har Sjef PRT i Meymaneh åtte oppgaver som skal løses av tre forskjellige lag. For hver oppgave har Sjef PRT estimert oppgavens varighet, samt hvilke lag som vil være i stand til å gjennomføre den. Et lag kan bare utføre én oppgave om gangen. Det finnes krav til at noen oppgaver må gjennomføres før andre kan starte. I tillegg er det noen oppgaver som må utføres samtidig. Sjef PRT ønsker å finne en prosjektplan som minimerer total tidsbruk på gjennomføringen av oppgavene. Tabell 8.1 viser en oversikt over disse oppgavene.

Oppgave	Varighet (dager)	Må gjøres før	Må gjøres samtidig som	Kan utføres av
1	10			Lag 1, Lag 2
2	4		5	Lag 1, Lag 3
3	9	1		Lag 2
4	2			Lag 2, Lag 3
5	3		2	Alle
6	1			Lag 2
7	16			Lag 3
8	2	5		Lag 1, Lag 3

Tabell 8.1 Oppgaver og krav til rekkefølge og utførelse

Dette eksempelet er et schedulingproblem som kan sees på som et prosjektplanleggings-problem med begrensede ressurser. Slike problemer er mye studert i litteraturen der de går under navnet ”Resource constrained project scheduling”. Det er bevist at slike problemer i praksis er uløselige innenfor rimelig tid. Eksempelet som er vist her, er så lite at det er mulig å løse eksakt, men det skal ikke bli mye større før man er nødt til å ta i bruk heuristiske metoder.

Formålet med dette kapitlet er å vise bruk av en genetisk algoritme for å løse eksempelet over. Den grunnleggende algoritmen er gitt under. Etterpå følger nærmere beskrivelser av de forskjellige stegene.

G := 1;
generer initiell populasjon *POP* av kromosomer;
beregn fitness for kromosomene i *POP;*
WHILE G < antall generasjoner DO
    G := G + 1;
    produser barn *B* fra *POP* ved hjelp av krysning;
    mutér hvert barn i *B* med sannsynlighet p_m
    beregn fitness for hvert barn *B*;
    POP := *POP* U *B;*
    redusér populasjonen *POP* ved hjelp av seleksjon;
END

Før algoritmen forklares nærmere, bør det forklares hvordan hvert kromosom er bygd opp. Et kromosom består av en gyldig aktivitetsplan, det vil si en plan som overholder krav til rekkefølge og der lag er allokert til oppgaver de er i stand til å utføre. Tabell 8.2 viser et eksempel på en slik plan. Den vertikale plasseringen av en oppgave i tabellen gjenspeiler at en oppgave må starte senere enn, eller likt med, oppgavene som er plassert lenger opp.

Oppgave	Utføres av
3	Lag 2
4	Lag 3
8	Lag 3
7	Lag 3
1	Lag 2
5	Lag 2
2	Lag 1
6	Lag 2

Tabell 8.2 Eksempel på kromosom i den genetiske algoritmen

Siden målet med optimeringen vår er det skal ta kortest mulig tid å utføre alle oppgavene, vil kromosomets fitness være nettopp total tidsbruk for planen kromosomet representerer. Den finner man ved at starttiden til alle oppgavene settes så tidlig som mulig, gitt kravene om rekkefølge og om at et lag må bli ferdig med en aktivitet før det kan starte på en ny.

Første steg i algoritmen er å generere den initielle populasjonen *POP* av kromosomer. Det skjer ved at *n* antall kromosomer, der *n* er størrelsen på *POP*, genereres tilfeldig på følgende måte:

I mengden O_m plasseres oppgaver som er klare for schedulering, det vil si oppgaver som ikke har noen forgjengeroppgaver som må utføres først. Fra O_m trekkes en tilfeldig aktivitet som plasseres i første posisjon i kromosomet. Deretter oppdateres O_m slik at oppgaver som ble klare for schedulering fordi oppgaven som ble flyttet til kromosomet var deres eneste forgjenger, nå kommer med i mengden av oppgaver som kan scheduleres. Deretter trekkes en ny oppgave fra O_m som plasseres i andre posisjon i kromosomet, og igjen oppdateres O_m. Denne prosessen fortsetter til man har schedulert alle oppgavene.

For hver oppgave trekkes det så et lag fra mengden av lag som er i stand til å utføre den aktuelle oppgaven. Ved hjelp av rekkefølgen og allokeringen av lag, kan man nå regne ut fitness for kromosomet.

Når den initielle populasjonen er generert, går algoritmen inn i en løkke som varer helt til man har nådd ønsket antall generasjoner. Det første som skjer i løkken, er at det produseres en mengde barn *B* fra *POP* ved hjelp av krysning. Kromosomene i *POP* settes tilfeldig sammen til par. Når det utføres en krysning på parene, dannes det to nye kromosomer – to ”barn”. I dette eksempelet er det brukt såkalt to-punkts-krysning. Denne prosedyren er forklart i [14]. Gjennom slik krysning vil også de to nye kromosomenes aktivitetsplaner være gyldige i henhold til krav om rekkefølge. Dette er bevist i [14].

Hvert barn i den nye generasjonen skal så muteres. Med en sannsynlighet p_m vil en tilfeldig oppgave i aktivitetsplanen bytte plass med oppgaven som er plassert foran, dersom det ikke finnes

noen rekkefølgekrav mellom de to oppgavene. I tillegg er det slik at med en sannsynlighet p_m trekkes det en ny, tilfeldig oppgave som bytter lag med oppgaven foran, dersom dette ikke bryter med kravene om hvilke lag som kan utføre de to oppgavene.

Til slutt regnes det ut fitness for alle kromosomene, som nå er dobbelt så mange som i den initielle *POP*. Ved hjelp av seleksjon kuttes halvparten av kromosomene fra denne mengden, slik at man sitter igjen med en ny populasjon *POP*. Seleksjonen kan foregå slik at den beste halvparten får lov til å overleve, men en mer vanlig måte å selektere på, er at hvert kromosom har en viss sannsynlighet for å "overleve", der denne sannsynligheten er bedre jo bedre kromosomets fitness er. Etter seleksjonen starter løkken på nytt, helt til spesifisert antall generasjoner er nådd.

Algoritmen over kan implementeres i det programmeringsspråk man måtte foretrekke. I Figur 8.1 vises resultatet etter en kjøring i Matlab. Størrelsen på *POP* var 50 og antall generasjoner var 5 under denne kjøringen. Resultatet fra denne kjøringen er at det krever 19 dager å gjennomføre alle oppgavene.

Aktivitet	Lag	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1	-	-	-	-	-	-	-	-	-	1	2	3	4	5	6	7	8	9	10	-	-	-	-
2	1	-	-	1	2	3	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	2	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	3	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	1	-	-	-	-	-	-	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	2	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
7	3	-	-	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	-	-	-	-	-
8	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figur 8.1 Beste aktivitetsplan funnet ved hjelp av genetisk algoritme

Siden dette problemet som nevnt er lite nok til å løses eksakt, er det interessant å sammenligne løsningen funnet ved hjelp av den genetiske algoritmen, med en løsning funnet ved hjelp av optimering. Problemet ble formulert som en heltallsmodell og løst ved hjelp av GLPK[2]. Resultatet fra optimeringen er vist i Figur 8.2.

Aktivitet	Lag	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1	-	-	-	-	-	-	-	-	-	1	2	3	4	5	6	7	8	9	10	-	-	-	-
2	1	1	2	3	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	2	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	2	-	-	-	-	-	-	-	-	-	1	2	-	-	-	-	-	-	-	-	-	-	-	-
5	1	-	-	-	-	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	2	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
7	3	-	-	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	-	-	-	-	-
8	3	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figur 8.2 Beste plan funnet ved hjelp av matematisk optimeringsmodell

Her ser vi at løsningen som ble funnet ved hjelp av den genetiske algoritmen, er like god som optimal løsning. Kjøretiden til optimeringen var på 13 minutter, mens kjøretiden til den genetiske algoritmen var ca. et halvt sekund. Dersom antall oppgaver i modellen økes til ti, øker kjøretiden for optimeringen til over en time, mens den genetiske algoritmens kjøretid har en økning som

[2] Se Appendix A for omtale av dette verktøyet

knapt er merkbar. Dette sier noe om hvor fort kompleksiteten i modellen øker med antall variable, og hvor mye mer effektiv den heuristiske algoritmen er.

Det er ikke alltid nødvendig å programmere genetiske algoritmer fra bunnen av. Det finnes flere verktøy som støtter genetiske algoritmer. Det finnes blant annet et tillegg til Matlab for genetiske algoritmer.

9 Usikkerhetshåndtering

Dette kapittelet tar for seg to viktige tema når det gjelder usikkerhetshåndtering innen optimeringsteori.

9.1 Følsomhetsanalyse av LP-modeller

Deterministiske modeller tar ikke hensyn til usikkerhet i inndata. Imidlertid kan man ved hjelp av følsomhetsanalyse si noe om konsekvensene av at verdiene på inndataene endrer seg. Interessante spørsmål kan være 1) hvordan forandrer optimal verdi seg hvis man endrer inndata, 2) hvor mye kan inndata variere uten at optimal verdi endres og 3) hvor mye kan inndata variere uten at løsningen blir ugyldig.

For LP-problemer kan man finne svar på slike spørsmål ved å se på problemets dualformulering. Dualproblemer ble omtalt i kapittel 2.5. Ved å se på dualproblemet til et optimeringsproblem, kan man finne de såkalte skyggeprisene, og det er disse som brukes i følsomhetsanalysen. Betydningen av skyggepriser kan illustreres ved hjelp av maksimeringsproblemet i kapittel 2.5. Denne formuleringen kan representere et produksjonsplanleggingsproblem der man skal produsere mengden x_j av produkt j, b_i representerer tilgang på råvare i og beskrankningene forteller hvor mye hvert produkt trenger av hver råvare. Objektfunksjonen kan representere inntekt fra salg av produktene. Inndata i denne modellen er mengden tilgjengelig råvare og salgspris.

I dualformuleringen til dette problemet, representerer y_i skyggeprisen til råvare i. Skyggeprisen forteller hva som skjer med objektfunksjonen dersom man øker tilgangen på råvare b_i. Det vil si at verdien til y_i i optimal løsning for dualproblemet forteller hvor mye objektfunksjonsverdien vil øke dersom man øker tilgangen på råvare b_i med 1. På denne måten kan man svare på spørsmål 1 som ble nevnt over. Dualproblemet og skyggepriser kan sammen med såkalt reduced cost også brukes for å svare på spørsmål 2 og 3, samt flere andre typer spørsmål rundt følsomhet. Metoder for dette kan studeres i [5] og [9].

I [7] argumenteres det for at følsomhetsanalyse av deterministiske modeller ikke egentlig sier noe om effekten av usikkerhet, og at følsomhetsanalyse ”kan kun brukes for å analysere effekten av endring i koeffisienter som ikke er usikre”. Det vises at følsomhetsanalyse egentlig er en form for scenarioanalyse, der man løser et antall deterministiske problemer, og at det ikke nødvendigvis er

noen sammenheng mellom dette og optimal løsning på det stokastiske problemet. Usikkerhets-håndtering og stokastiske modeller omtales i neste avsnitt.

9.2 Stokastiske modeller

Ved optimering under usikkerhet er ikke all nødvendig informasjon tilgjengelig idet en beslutning skal tas. Dette gjelder for eksempel når tid er en faktor, det vil si problemer der man skal planlegge noe som skal skje, eller kommer til å ha effekt, i fremtiden. Mange hevder at det ikke finnes deterministiske planleggingsproblemer, blant annet [16], da det alltid vil være usikkerhet rundt slike problemer. Det finnes problemer som lar seg studere greit ved hjelp av deterministiske *verktøy*, men det kan være vanskelig å vite om problemet man har med å gjøre er av en slik art. I [16] og [17] er det gitt eksempler som demonstrerer effekten av å bruke deterministiske verktøy for å løse problemer som egentlig er stokastiske.
Dersom man overser usikkerhetens betydning risikerer man å utelate enkelte kostnader, og/eller la gode beslutninger fremstå som mindre gode enn de faktisk er. Det er alltid en kostnad forbundet med usikkerhet – det å ikke vite med sikkerhet fører gjerne til at man må ta noen forholdsregler, for eksempel kjøpe forsikring, velge dyre robuste/fleksible løsninger, eller at man velger å "gamble" for så å oppleve at ikke alt går som ”planlagt”. Siden det er en kostnad forbundet med usikkerhet i beslutningsproblemer, vil det også ha en verdi å kunne fjerne usikkerhet: Informasjonen får således en verdi. *Forventet verdi av perfekt informasjon* gir en øvre grense på hva man kan forvente at mer informasjonen er verdt. Denne verdien er forbundet med wait-and-see-løsningen på problemet. Wait-and-see-løsningen er den forventede optimale løsningen dersom man kan vente med å ta alle beslutninger til all usikkerhet er avslørt. Dette betyr at man får vite *hva* usikkerheten er, men ikke at man er i stand til å kontrollere den.

Ved beregning av kjøreruter på linje med det som ble gjort i eksempelet med etterforsyning av MOT-er i kapittel 6.2, kan stokastisk programmering brukes til å håndtere usikkerhet. I slike kjøreruteproblemer kan det være usikkerhet forbundet med kjøretider som følge av vær, føreforhold og trafikk, eller det kan være usikkerhet i forbindelse med at etterspørselen i nodene kan variere. Dersom man har et problem som gjentar seg – det vil i denne forbindelse si at man har en fast rute som skal kjøres igjen og igjen over tid, for eksempel hver dag eller hver uke – kan stokastisk programmering brukes til å beregne den løsningen som i gjennomsnitt vil gjøre det best. Usikre data representeres da som stokastiske variable, hvor variablenes fordelinger estimeres ut fra data samlet opp over tid.

Man kan også bruke stokastisk programmering for problemer som ikke gjentar seg, men der en beslutning skal tas én gang. Et eksempel kan være planleggingsproblemet i kapittel 8.4. I dette eksempelet var aktivitetslengdene deterministisk bestemt. I virkeligheten vil det i mange tilfeller være usikkerhet knyttet til aktivitetslengdene. Det stokastiske beslutningsproblemet vil dermed handle om å finne ut hvordan man skal fordele ressursene slik at man får kortest mulig forventet prosjekttid. I [18] beskrives en del teori rundt ressurstildeling i stokastisk prosjektplanlegging, og det presenteres en modell for å løse et slikt problem. Det beskrives også hvordan heuristikker som for eksempel tabusøk kan brukes til å løse stokastiske problemer av denne typen. I [7] er et helt

kapittel viet til prosjektstyring, der blant annet forskjellen mellom deterministiske og stokastiske varianter diskuteres.

Den mest vanlige typen stokastiske problemer, er såkalte to-stegsmodeller. Det vil si modeller der tidsaspektet er viktig. Et steg i denne sammenhengen er et punkt hvor det er mulig og meningsfylt å ta en beslutning. I slike modeller er det altså viktig *når* en beslutning tas, og hva slags kunnskap man har på dette tidspunktet. I tillegg er det slik at en beslutning gjerne får en effekt i fremtiden, men at denne effekten er ukjent eller usikker ved beslutnings-tidspunktet. Et typisk eksempel på en to-stegs-modell, er et produksjonsplanleggingsproblem. I en deterministisk modell hvor man ikke tar hensyn til tiden, som i eksempelet omtalt i kapittel 9.1, antar man at den mengden varer man produserer er lik den mengden man faktisk får solgt. I virkeligheten er det ofte slik at man ved produksjonstidspunktet av en vare ikke kan vite hvordan etterspørselen etter denne varen vil bli. Det er med andre ord en usikkerhet knyttet til hvor mye man får solgt av varen man produserer.

Det typiske mønsteret for en to-stegsmodell, er at man i første steg tar en beslutning, og at effekten av denne beslutningen vil være avhengig av hendelser i fremtiden som er ukjente på beslutningstidspunktet. Når man etterpå får kjennskap til hva som skjer i fremtiden, kan man gjøre en ny beslutning for å bøte på eventuelle uheldige effekter av den første beslutningen. Resultatet fra en slik modell består dermed av én beslutning i første steg og en samling av beslutninger for andre steg, samt et sett av ”regler” som bestemmer hvilken av beslutningene i andre steg man bør velge gitt utfallet av de usikre hendelsene. Optimal løsning består av den beslutningen i første steg som vil gi best forventet resultat, samt verdien på det forventede resultatet.

I [19] behandles temaet beslutningsstøtte under usikkerhet. Her kan man blant annet finne eksempler som på en lettfattelig måte illustrerer hvordan man går fra deterministiske modeller som ikke tar hensyn til tid, til stokastiske flerstegsmodeller, og hva det er viktig å tenke på i den forbindelse. En god introduksjon til stokastisk programmering finnes også på [20].

10 Oppsummering og konklusjoner

Operasjonsanalyse assosieres av mange med optimering. Optimering bør derfor ha en sentral plass i ”verktøykassen” til alle som driver med OA. Sammenlignet med andre metoder som brukes innenfor OA har optimering den fordelen at den finner matematisk optimal løsning på problemene. En forutsetning for at dette også skal være en hensiktsmessig løsning, er at optimeringsmodellen i seg selv er en presis matematisk representasjon av problemet. I større analyseprosesser vil god problemstrukturering være en forutsetning for å kunne formulere gode optimeringsmodeller.

Kapittel 2 beskriver når det kan passe å bruke optimering. Et av de største hindrene for bruk av optimering som analyseverktøy er at det kreves tid og ressurser til modellformulering og implementering. Imidlertid kan det ofte være slik at optimering kan brukes som hjelpemiddel i en

beslutningsprosess hvor andre metoder utgjør hoveddelen av analysen. Det kan for eksempel være at man ønsker å utforske mulighetsrommet til et relativ lite delproblem. Blant annet kan optimering brukes til å sjekke mulige verdier på noen av inndataene til en større modell.

Det er lett å tenke at optimering bare kan brukes de gangene man kan finne, og er interessert i å bruke, den matematisk optimale løsningen på et problem. Imidlertid er det i mange tilfeller ikke nødvendig eller interessant å bruke den optimale løsningen direkte. Like vanlig er det at man ønsker å vite hva som er optimal løsning slik at man har noe å måle andre løsninger opp mot. Dette kan blant annet være interessant når man utforsker et system ved hjelp av simulering. Gjennom simulering kan man se mange forskjellige løsninger på et problem, og man kan se effekten av ulik input. I slike tilfeller kan det være nyttig å finne matematisk optimal løsning gitt forskjellige forutsetninger, for så å sammenligne resultatene fra simuleringene med denne. Slike analyser vil være relativt ressurskrevende for store og komplekse problemer, men for mindre problemer kan gevinsten av slike analyser være relativt stor i forhold til innsatsen.

En annen måte optimering og simulering kan kombineres på, er ved at optimering utføres som en del av simuleringen. Dette kan enten være optimering av inndata som mates inn i simuleringsmodellen, eller det kan være delproblemer som løses ved hjelp av optimering som et ledd i selve simuleringen. I slike tilfeller kan man si at optimeringen foreslår beslutninger, og at simuleringen viser effektene av beslutningen.

Selv om både simulering og optimering kan fungere som verktøy for å finne den beste løsningen på et beslutningsproblem, vil det gjerne heller være slik at de brukes som metoder for å generere ulike beslutningsalternativer, eller for å utforske potensialet til ulike beslutningsalternativer. Deretter vil alternativene måtte vurderes av beslutningstakeren, og i en slik prosess vil en metode som for eksempel flermålsanalyse være et aktuelt redskap. Ulike metoder innen operasjonsanalyse vil med andre ord gjerne brukes på forskjellig steg i beslutningsprosessen, og på forskjellig nivå.

Usikkerhetshåndtering er viktig innen operasjonsanalyse. I denne rapporten ble usikkerhet i forbindelse med optimering belyst gjennom kapittel 9. For mange vil det nok være en høy terskel forbundet med å ta steget fra deterministisk til stokastisk optimering. For større analyseprosesser bør likevel det stokastiske aspektet som regel tas med.

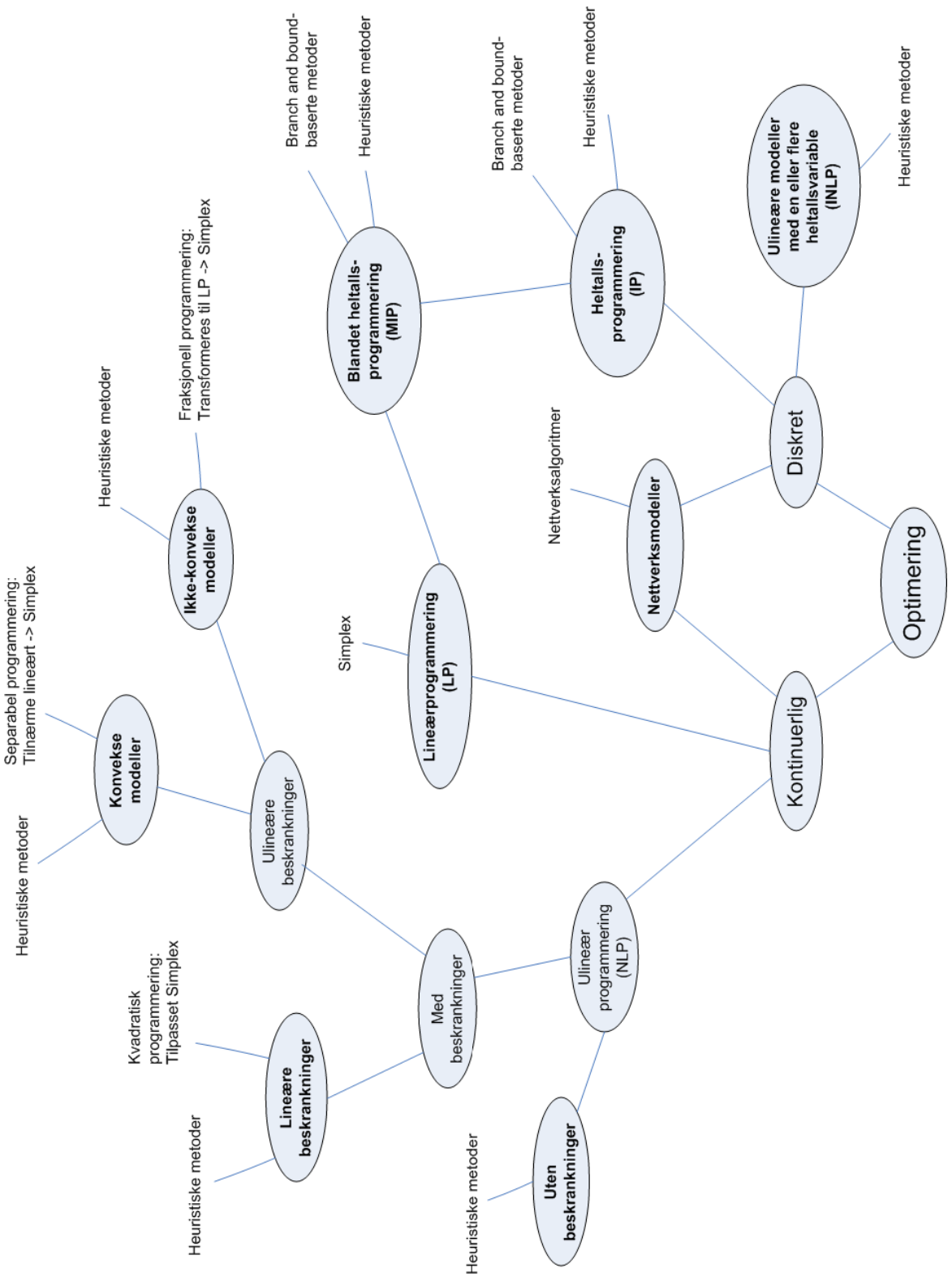
I denne rapporten er det beskrevet en rekke forskjellige typer optimeringsproblemer og tilhørende løsningsmetoder. Figur 10.1 oppsummerer hvordan optimeringsmodeller kan sorteres ut fra egenskapene beskrevet i kapittel 3. Det vil aldri bli helt riktig å klassifisere modeller slik som det er gjort i denne figuren, da det er mer overlapp og flytende grenser mellom modellene enn en slik illustrasjon gir inntrykk av. Likevel illustrerer figuren hovedtrekkene i spekteret av forskjellige modelltyper. I tillegg viser den noen av de vanligste løsningsmetodene for de ulike modelltypene, som beskrevet i kapitlene 4 til 8. Figuren illustrerer hvordan det er viktig å skille mellom kontinuerlige og diskrete problemer. Når man skal implementere og løse et optimeringsproblem, må det videre identifiseres hvorvidt problemet er lineært eller ulinært, fordi det har stor betydning

for valg av løsningsmetode. Lineære problemer lar seg løse med de aller fleste optimerings-verktøy. Dersom problemet er ulineært, vil man som Figur 10.1 viser ofte måtte bruke heuristiske løsningsmetoder. Det er også et viktig å poeng å identifisere om problemet har en nettverksstruktur som gjør at det er mulig å bruke en nettverksalgoritme til å løse det. Disse algoritmene er ofte svært effektive i forhold til alternative metoder, særlig dersom problemet har en eller flere heltallsvariable.
De aller beste solverene på markedet kan løse de fleste typer problemer. De har innebygd effektive heuristiske algoritmer og kan ofte gi meget gode svar selv på problemer som i utgangspunktet ikke lar seg løse eksakt. Dersom man har tilgang til slike solvere, blir det aller viktigste å formulere modellen så presist som mulig i forhold til det reelle problemet, og så la solveren vurdere hvilke algoritmer som gir best løsning.

Som nevnt i innledningskapitlet til denne rapporten, er optimeringsmetoder lite brukt for analyseformål ved instituttet. Verktøyet J-DARTS[3] er et av få eksempler på bruk av optimering ved avdelingen i nyere tid. Innenfor et område som for eksempel logistikk, bør det være stort potensial for bruk av optimeringsmodeller. Også for problemstillinger knyttet til materiellinvesteringer, ressursallokering, effektiv bruk av personell og utstyr, samt andre typer planleggingsoppgaver, vil det kunne være nyttig å bruke optimeringsmetoder. Hver enkelt forsker vil også kunne ha nytte av å bruke optimering i mindre skala, for eksempel som et verktøy til å utforske mulighetsrom innenfor analyser som i hovedsak bygger på andre metoder enn optimering.

Gevinsten ved å bruke optimering øker med erfaring. Dette gjelder både for den enkelte forsker og i et større analysemiljø. Kanskje den viktigste forutsetningen for etableringen av god optimeringspraksis, er at man har enkel tilgang på gode verktøy som lett lar seg integrere med andre applikasjoner. De fleste gode optimeringssolvere i dag har grensesnitt mot andre programmeringsspråk, som for eksempel Java og C++. De mest fleksible og effektive solverne er imidlertid relativt kostbare. Det finnes gratis prøveversjoner som hjelper en litt på vei, men disse har gjerne begrensninger på hvor store problemer som kan løses, samt at integrasjonsmulighetene med andre applikasjoner er begrensede eller fraværende. For små problemer der det er ønskelig med rask implementering og behovet for integrasjon med andre applikasjoner ikke er viktig, er Microsoft Excel sin Solverfunksjon ofte overraskende god. En omtale av ulike optimeringsverktøy er gitt i Appendix A.

[3] J-DARTS er et verktøy som kan brukes i forsvarsanalyse til å finne den styrkestrukturen som minimerer kostnadene, samtidig som den tilfredsstiller et fastsatt ambisjonsnivå [21].



Figur 10.1 Oversikt over ulike typer optimeringsmodeller

Referanser

[1] M. Gilljam and H. Ljøgodt, "Problem structuring methods - A survey and a case study", Forsvarets Forskningsinstitutt, FFI rapport 2005/00852, 2005.

[2] S. Malerud og T. Kråkenes, "Metoder for flermålsanalyse - en oversiktsstudie fra GOAL", Forsvarets Forskningsinstitutt, FFI rapport 2005/03041, 2005.

[3] T. Kråkenes, H. Ljøgodt og S. Malerud, "Simuleringsmetoder innen operasjonsanalyse - en oversiktsstudie", Forsvarets Forskningsinstitutt, FFI rapport 2007/00297, 2007.

[4] H. O. Sundfør, "Lineærprogrammering og mixed integer programmering i strukturanalyser - grunnlag og erfaringer", Forsvarets Forskningsinstitutt, FFI rapport 2006/00241, 2006.

[5] R. L. Rardin, *Optimization in operations research*, Prentice Hall, Inc., 1998.

[6] A. C. Bartlett and A. N. Langville, "An Integer Programming Model for the Sudoku Problem", *http://www.cofc.edu/~langvillea/Sudoku/sudoku2.pdf*, 2006.

[7] K. Jörnsten, S. Storøy og S. W. Wallace, *Operasjonsanalyse*, Cappelen Akademisk Forlag, 1999.

[8] F. S. Hillier and G. J. Lieberman, *Introduction to operations research*, McGraw-Hill, 2001.

[9] H. A. Taha, *Operations research - An introduction*, Pearson Education, Inc., 2007.

[10] N. K. Jaiswal, *Military operations research - Quantitative decicion making*, Kluwer Academic Publishers, 1997.

[11] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL - A modeling Language For Mathematical Programming*, boyd & fraser publishing company, 2002.

[12] F. Franzé and N. Speciale, "A tabu-search-based algorithm for continuous multiminima problems", *International Journal for Numerical Methods in Engineering*, vol. 50, pp. 665-680, 2001.

[13] M. Mitchell, *An introduction to genetic algorithms*, The MIT press, 1998.

[14] S. Hartmann, *Project scheduling under limited resources*, Springer-Verlag, 1999.

[15] S. Shadrokh and F. Kianfar, "A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty", *European Journal of Operational Research*, no. 181, pp. 86-101, 2006.

[16] S. W. Wallace, "The role of uncertainty in strategic planning", *Telektronikk*, vol. 94, no. 3/4, pp. 21-23, 1998.

[17] P. Kall and S. W. Wallace, *Stochastic Programming*, John Wiley & Sons, Chichester, 1994.

[18] I. Honve, "Ein modell for ressurstildeling i stokastisk prosjektplanlegging", Universitetet i Bergen, 2002.

[19] S. W. Wallace, *Decision making under uncertainty: The art of modeling,* Norwegian University of Science and Technology, 1999.

[20] Stochastic Programming Community Home Page, *http://www.stoprog.org/*, 2007.

[21] A. C. Hennum og S. Glærum, "Metoder for langtidsplanlegging - støtte til FS 07", Forsvarets Forskningsinstitutt, FFI rapport 2007/02174, 2007.

Forkortelser

BB	Branch and bound
GLPK	GNU Linear programming kit
INLP	Integer nonlinear programming
IP	Integer programming
ISAF	International Security Assistance Force
KKT	Karusch-Kuhn-Tucker
LP	Lineærprogrammering
MIP	Mixed integer programming
MOT	Military observer team
NGO	Non-governmental organisation
NLP	Nonlinear programming
OA	Operasjonsanalyse
PRT	Provincial reconstruction team

Appendix A Verktøy og ressurser

Her følger en omtale av noen optimeringsverktøy, samt en liste over optimeringsressurser på internett.

A.1 Gratis verktøy

De fleste kommersielle optimeringsverktøyene finnes i gratis student- eller prøveversjoner. Disse har som regel begrensninger som gjør at man ikke får lov til å løse problemer som er større enn noen hundre variable og beskrankninger. Med slike begrensninger vil man veldig fort stange hodet i veggen. Eksempel på slike gratisversjoner er:

AMPL (http://www.ampl.com/). Dette er et språk for å modellere optimeringsproblemer. AMPL må kommunisere med en solver for å løse optimeringsproblemet. En lang rekke solvere kan brukes sammen med AMPL, både gratis og kommersielle varianter. Noen eksempler på slike solvere er lp_solve og CPLEX, som begge vil bli omtalt lenger ned. Gratis studentversjon finnes av AMPL. Denne følger også med som tillegg til visse bøker som baserer seg på AMPL.

Xpress-MP (http://www.dashoptimization.com/). Dette er et verktøy som både består av grafisk brukergrensesnitt og optimeringssolvere. I tillegg har det grensesnitt til flere programmerings-språk, slik som Java og C++, slik at brukeren kan integrere solveren i sine egne applikasjoner. Gratis studentversjon finnes av Xpress-MP.

Optimisation Programming Language (http://www.ilog.com/). Utviklerne av markedets kanskje beste optimeringssolver – CPLEX – har sitt eget modelleringsspråk som kan brukes sammen med CPLEX. En gratis prøveversjon kan lastes ned fra websiden deres.

Lindo (http://www.lindo.com/). Lindo har blant annet et verktøy som baserer seg på integrasjon med Excel. Det finnes også et eget verktøy for å skrive og løse optimeringsproblemer, i tillegg til at man kan bruke solveren direkte i egne applikasjoner. Gratis prøveversjon kan lastes ned fra websiden.

I tillegg til gratisversjoner av kommeriselle optimeringsverktøy finnes det en lang rekke helt gratis programvare og solvere der ute. Disse har ingen begrensninger på problemstørrelse, så her er det kun maskinkraft og kompleksitet på problemet som vil ha betydning for hvilke problemer som lar seg løse. Dessverre er disse verktøyene ofte betydelig mindre effektive enn kommersielle verktøy. Her vil noen av den store mengden gratis optimeringssolvere nevnes:

lp_solve (http://lpsolve.sourceforge.net/). Dette er en solver som løser lineære problemer. lp_solve kan kalles fra de fleste programmeringsspråk, samt eksempelvis Excel og Matlab, og kan dermed greit integreres i egne applikasjoner. Den har også et grafisk grensesnitt som man kan bruke til å formulere og løse problemer.

GNU linear programming kit (http://www.gnu.org/software/glpk/glpk.html). Denne solveren støtter språket GNU Mathprog, som er et underspråk av AMPL. For de som kan AMPL og ønsker å bruke et gratis verktøy, er det lett å sette seg inn i GLPK. En ulempe med GLPK er at det ikke finnes noen enkel måte å behandle resultatene på. De gis ut som en tekstfil, og kan ikke enkelt skrives til for eksempel Excel eller en database. Man vil ha behov for et annet programmeringsspråk for å håndtere data til/fra modellen på en god måte.

NEOS server (http://www-neos.mcs.anl.gov/). NEOS server inneholder en rekke solvere som man kan benytte online. Det vil si at man laster opp optimeringsmodellen til serveren, som så løser det ved hjelp av en solver og sender resultatet tilbake via e-post. Modellen må formuleres på et format som støttes av den solveren man skal bruke. Det varierer hvorvidt solverene på serveren er kommersielle eller ikke, og hvorvidt det finnes begrensninger på problemstørrelse.

A.2 Kommersielle verktøy

Alle verktøyene nevnt i A.1 som finnes i student- og prøveversjoner finnes også som fullversjoner man kan kjøpe. I tillegg bør det nevnes tre verktøy som er høyst relevante for oss på FFI:

Matlab optimization toolbox. Dersom man har behov for å utføre optimeringer i Matlab-modeller, er dette et nyttig verktøy. Det er også et godt alternativ generelt som optimeringsverktøy for de som kjenner Matlab godt fra før. Verktøyet er et tillegg til Matlab som man må betale for.

CPLEX. Denne solveren anses som kanskje den beste optimeringssolveren på markedet. Den er meget kraftig, og vil ofte være flere hundre ganger raskere enn gratisverktøyene nevnt over. CPLEX har grensesnitt til de fleste programmeringsspråk, som for eksempel Java og C++. Det er dermed enkelt å integrere CPLEX i ens egne applikasjoner. Algoritmene i CPLEX gjør at denne solveren også yter veldig bra for problemer som ikke lar seg løse eksakt.

Excel. Solverfunksjonen til Excel er en add-in som man ikke behøver å betale ekstra for hvis man allerede har Excel installert. Man kan nesten bli overrasket over hvor nyttig Solver kan være. Det gjelder spesielt hvis man skal gjøre en optimering som ikke er en del av en større applikasjon, og der man bare har behov for å få et resultat raskt. Det krever nesten ingen innsats å sette opp modellen i Excel. I standardversjonen av Solver er det en øvre begrensning på 200 variable. Det er mulig å kjøpe en premiumversjon som tar større problemer.

A.3 Optimeringsressurser på internett

Under er det listet opp endel nyttige optimeringslenker.

NEOS Guide – http://www-fp.mcs.anl.gov/OTC/Guide/
Optimering i praksis (UiO) – http://folk.uio.no/optimer/
Mathematical programming society – http://www.mathprog.org/
Stochastic Programming Community – http://www.stoprog.org/