FFI RAPPORT

LINEÆRPROGRAMMERING OG MIXED INTEGER PROGRAMMERING I STRUKTURANALYSER – Grunnlag og erfaringer

SUNDFØR Hans Olav

FFI/RAPPORT-2006/00241

LINEÆRPROGRAMMERING OG MIXED INTEGER PROGRAMMERING I STRUKTURANALYSER – Grunnlag og erfaringer

SUNDFØR Hans Olav

FFI/RAPPORT-2006/00241

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norge

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

P O BOX 25
NO-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2006/00241	UNCLASSIFIED	54
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFI-I/1004/918	-	

4) TITLE

LINEÆRPROGRAMMERING OG MIXED INTEGER PROGRAMMERING I STRUKTURANALYSER – Grunnlag og erfaringer

Linear Programming and Mixed Integer Programming in defence planning – basic introduction and experience

5) NAMES OF AUTHOR(S) IN FULL (surname first)

SUNDFØR Hans Olav

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Operations Research	a) Operasjonsanalyse
b) Optimisation	b) Optimering
c) Linear Programming	c) Lineær Programmering
d) Mixed Integer Programming	d) Mixed Integer programering
e) Defence Planning	e) Forsvarsplanlegging

THESAURUS REFERENCE:

8) ABSTRACT

The basic concept of optimisation – to find the best solution to a decision-problem, given a set of decisions, an objective and a set of constraints – is at the very hart of decision support and operations research. It should therefore have a central place also in defence planning, which is a major area of application of operations analysis.

This report provides a brief overview of basic principles and theory of optimisation in general and linear programming and mixed integer programming in particular. It then presents and provides practical experience from the application of LP and MIP on three different defence planning issues under the NATO DRR process. The three projects are DRR fulfilment, which is matching of generic capability requirements with specific units' capabilities, an air-recce optimisation model and a model to pre-process aerospace-related generic requirements for fulfilment.

High level COTS modelling and implementation software was used in the three projects, and it is claimed that a sufficiently skilled analyst effectively can implement her own models within a matter of hours or days as part of the main analysis without taking on major costs in software development.

This report is prepared for project GOAL, and is a part of a series of survey reports covering different OR methods.

9) DATE	AUTHORIZED BY This page only	POSITION
2006-01-27	Jan Erik Torp	Director

ISBN 82-464-1002-4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

LINEÆRPROGRAMMERING OG MIXED INTEGER PROGRAMMERING I STRUKTURANALYSER – Grunnlag og erfaringer

1 INNLEDNING

Optimering er en samlebetegnelse på matematiske metoder for å *finne den beste løsningen* på et veldefinert beslutningsproblem. Det sier seg selv at dette kan være relativt krevende, og derfor finnes det ikke en enkelt metode som kan løse alle optimeringsproblemer (eller beslutningsproblemer) effektivt. For noen problemer finnes det metoder som finner den beste løsningen blant absolutt alle mulige løsninger, og beviser at den er den beste løsningen. For andre problemer finnes det metoder som er egnet til å lete seg frem til gode løsninger, og noen av disse metodene forteller også hvor gode løsningene er, mens andre ikke gjør dette. For andre problemer igjen finnes det i beste fall en metode for å finne en løsning som ikke er direkte gal, men disse metodene er strengt tatt i grenseland for hva som kalles optimering.

Et trivielt eksempel på et matematisk optimeringsproblem er en tyv som bryter seg inn i et kjøpesenter og kan fylle opp bilen sin med sprit og sigaretter, men som har en liten og skral bil. Bilen har liten plass, og den tåler ikke så mye vekt, men tyven har dårlig råd og er avhengig av å få med seg størst mulig verdier. På svartebørsen er en kasse sprit bedre betalt enn en like stor kasse sigaretter, men en kilo sigaretter er bedre betalt enn en kilo sprit. Hvis vår venn fyller bilen med sigaretter og slik utnytter hele volumet, så får han ikke utnyttet vektkapasiteten sin, og det ville lønne seg å bytte ut noen sigarettkasser med samme volum sprit. Hvis han tar med så mye sprit han kan, så utnytter han ikke volumet sitt, og det ville lønne seg å bytte ut noe av spriten med samme vekt i sigaretter. Det er åpenbart at det finnes en løsning som er den beste, og som ligger et sted mellom det å ta med bare sigaretter og bare sprit. Men hva er den beste løsningen?

Stilt overfor en slik utfordring vil de fleste tyver måtte gi tapt, ubelest som de gjerne er. De ville bli sittende og fundere over livets mysterier helt til politiet kommer og finner dem. De fleste tyver blir da også tatt før eller senere. En tilstrekkelig velskolert og oppvakt tyv derimot, ville forstå at dette kan formuleres som et lineært optimeringsproblem, som lar seg løse i løpet av millisekunder på en medbrakt laptop. Siden politiet ikke rykker ut i løpet av millisekunder, kommer han seg derfor lett unna med en optimal blanding av sprit og sigaretter, som er den kombinasjonen som gir maksimale verdier lastet inn i bilen.

Et annet trivielt eksempel er det velkjente markedskrysset[1] fra makroøkonomien, og de tilsvarende modellene for en monopolsituasjon eller en situasjon med få tilbydere. I en

[1] I et perfekt og statisk marked med mange tilbydere og mange etterspørrere vil samfunnets marginalkostnad ved produksjon av en enhet av et produkt være en økende funksjon av volumet. Tilsvarende vil brukernes samlede grensenytte for en enhet av produktet være en avtagende funksjon av omsatt volum. Markedskrysset er det punktet (pris og volum) hvor disse kurvene krysser hverandre, og i et perfekt marked vil dette være omsatt volum og markedspris. Funksjonene illustreres gjerne forenklet som rette linjer i kryss.

monopolmodell kan én tilbyder bestemme prisen på en vare og kjøperne vil kjøpe den hvis de ikke synes den er for dyr. Hvis prisen settes veldig høyt, er det nesten ingen som ikke synes det er for dyrt, og volumet blir så lite at tilbyderen nesten ikke tjener penger. Dersom prisen settes lik produksjonskostnaden eller mindre, blir volumet stort, men det blir ingen profitt på hver enhet, og derfor tjener ikke tilbyderen penger her heller. For en offentlig myndighet som skal kontrollere markedet, blir det avgjørende å forstå hvor prisen og volumet i en monopolsituasjon vil ligge, og da spør man seg vanligvis hvor tilbyderen vil tjene mest penger. Dette vil være på en pris som ligger et sted mellom den høyeste prisen han kan få i markedet og produksjonsprisen, men hvor vil denne prisen ligge? Og hvor vil den ligge i en annen modell med for eksempel to eller med tre tilbydere? Eller med mange tilbydere (markedskrysset)? Alle disse problemstillingene er optimeringsproblemer.

De to eksemplene over illustrerer to måter å bruke optimering på i beslutningsproblemer.

- I det første eksempelet prøver beslutningstakeren å finne den aller beste løsningen på sitt eget beslutningsproblem.
- I det andre eksempelet prøver en beslutningstaker å forstå konsekvensene av sin beslutning ved å spørre hvordan andre beslutningstakere vil respondere på beslutningen *dersom de agerer optimalt*.

Disse to spørsmålsstillingene kunne vært kombinert i det siste eksempelet ved at man spør seg ”hva er den beste løsningen på mitt problem – blant alle de alternativene jeg kan velge mellom – gitt at alle andre oppfører seg slik som er optimalt for dem?” Dette vil også være et rent optimeringsproblem selv om det er brukt i en spillteoretisk kontekst.

Optimering kan altså svare på spørsmålet ”hva er den beste løsningen for meg?” i en situasjon med uendelig mange teoretisk mulige løsningsalternativer. Optimering kan også svare på spørsmålet ”hvordan vil verden der ute reagere på mine handlinger hvis alle bare tenker på seg selv? – eller hvis alle bare tenker på sin syke mor? – eller hvis alle bare tenker på noe annet konkret og veldefinert?” En optimering forutsetter at man har et uttrykk for hva slags løsninger som er mulig – tyven måtte vite hva slags produkter han kunne velge mellom – men det forutsetter ingen eksplisitt definisjon av svaralternativer. Man må også ha et uttrykk for hvilke løsninger som er gode, eller en måte å finne godheten til en konkret løsning. Enhver matematisk metode som besvarer denne typen spørsmål vil per definisjon være en optimering, så ifølge den samme definisjonen er det ingen andre matematiske metoder som kan svare på denne typen spørsmål.

Det er ikke dermed sagt at matematisk optimering er den eneste måten å besvare viktige beslutningsspørsmål på. Det å finne gode løsninger er grunnleggende for all operasjonsanalyse, og som det vil bli diskutert senere, kan man prinsipielt se på subjektive vurderinger innenfor en analyse som en del av en overordnet optimeringsmetodikk. Der man forstår et beslutningsproblem godt nok til å uttrykke hele problemet matematisk, er imidlertid optimering måten å løse det. Ofte mangler det en grunnleggende forståelse for sammenhengen mellom nytten av forskjellige delmål, og da kan ikke optimering brukes direkte. En flermålsanalyse er en prosess for å etablere en slik sammenheng.

I operasjonsanalysesammenheng vil det være problemer som kan løses praktisk med optimeringsteknikker, og problemer som ikke kan løses på denne måten. Hvis man vet godt hva beslutningsproblemet er – hva man ønsker å oppnå, hvilke forhold man skal ta beslutning om, og hvordan systemet fungerer – men man ikke vet hva løsningen er, så vil et optimeringsresultat være et sterkt og troverdig resultat. En optimeringsteknikk kan systematisk ta for seg et uendelig rom av løsninger og finne den beste løsningen innenfor hele dette rommet uten å fokusere på a priori forestillinger om hvor løsningen vil ligge. Selv om man vet omtrent hvor den optimale løsningen ligger ut fra omtrentlige betraktninger, vil en optimering tjene to viktig formål:

1. Den vil gi en kontroll på den kunnskapen man har fra før.
2. Den vil gi kunnskap om eksakt hvor den optimale løsningen ligger.

Optimering er et omfattende og ganske komplekst fagområde, og man kan tilnærme seg det på mange nivåer: Før en metode kan få praktisk anvendelse må det ligge en grunnleggende forståelse for prinsippene i grunnen. Ut fra dette må det utledes matematisk teori og i de fleste tilfeller må det utvikles effektive løsningsalgoritmer. På toppen må det ligge en forståelse for hvordan virkelighetens problemer kan avbildes i en matematisk modell. Situasjonen i dag er at begge de to nivåene i midten – matematisk teori og algoritmer – kan fås innebygget i effektive programvarepakker. En analytiker som forstår de grunnleggende prinsippene kan derfor fokusere på modelleringsnivået, og trenger ikke ha mer enn en vag formening om hva som finner sted i de lavere nivåene for å kunne anvende optimering på praktiske problemer. Dette krever ikke all verden av innsats. For lineære problemer, som er fokus for denne rapporten, er det om ønskelig mulig å gå rett i gang med et reelt og anvendt problem etter prøve- og feilemetoden. Undertegnede er selv temmelig selvlært på optimering etter prøve- og feileprinsippet, og teoretisk lider nok rapporten litt under dette, men til gjengjeld er det å håpe at den fokuserer på forhold som er av betydning for praktisk anvendelse.

2 RAPPORTENS FORMÅL

Hensikten med denne rapporten er å gi en introduksjon til bruk av Lineær Programmering (LP) og Mixed Integer Programming (MIP) i operasjonsanalyse- og strukturanalyseproblemer med vekt på praktiske erfaringer. I løpet av nesten fire år med arbeid på NATO Defence Requirements Review (DRR) ved NC3A brukte vi LP/MIP ved flere anledninger på flere typer problemer og i prosjekter med svært forskjellige rammeforutsetninger. Tre prosjekter er nevnt her, og alle disse var krevende på sin måte. Det ene prosjektet består av et usedvanlig stort søkeproblem på grensen til å overstige det som kan håndteres med dagens teknologi, mens de to andre utmerket seg ved meget stramme tidsrammer. Det er erfaringer og forståelse fra praktisk bruk i disse og andre mindre prosjekter som forsøkes formidlet i denne rapporten.

Det har vært en forutsetning for utarbeidelsen av rapporten at den skal oppsummere eksisterende erfaringer og kunnskap. Det er ikke gjort noe nytt vitenskapelig arbeid i forbindelse med utarbeidelsen av dokumentet, og der forståelsen er tynn, er det denne tynne forståelsen som er videreformidlet. Av samme grunn er også bibliografien begrenset. Den inneholder referanse til

en lærebok (1) som om ønskelig vil gi en detaljert innføring i bl.a. SIMPLEX og assosierte løsningsalgoritmer, og en referanse til ILOG (2), som er leverandøren av den programvaren som er omtalt.

I alle de aktuelle prosjektene har det vært brukt COTS-programvare for å håndtere lavnivå løsninger. Rapporten omfatter derfor topplaget av optimeringen med modellering og høynivå implementering. Velkjente algoritmer som SIMPLEX og praktisk håndtering av minne og søketrær ligger nede i programvaren og er ikke omtalt. Det er en fordel å ha en formening om hvordan en LP- eller MIP-algoritme jobber, men siden dette faktisk ikke er et tema, kan rapporten godt leses uten slik kunnskap.

Rapporten gir først en introduksjon til optimering i sin alminnelighet og noen synspunkter på det prinsipielle forholdet mellom optimering, operasjonsanalyse og beslutningsstøtte. Deretter presenteres de to klassene av optimeringsproblemer som rapporten omhandler – MIP og LP – i noe større detalj. Samtidig tas viktige generelle og praktiske forhold så som problemstørrelse, linearisering, programvare og kostnader opp. I siste del av rapporten presenteres tre konkrete prosjekter, deriblant DRR fulfillment, som tilskyndet bruken av optimering i DRR.

Et viktig aspekt ved rapporten er vurderinger om hvor optimering, og da spesielt MIP-/LP-optimering, er en hensiktsmessig tilnærming til et problem. Dette er adressert både i den generelle delen av rapporten og i tilknytning til de tre prosjekteksemplene. Erfaringene mht anvendelighet oppsummeres også i et avsluttende kapittel. Siden det var viktig å formidle så mange praktiske erfaringer som mulig, går rapporten i relativt stor detalj på små trivielle utfordringer og komplikasjoner som oppsto ved praktisk bruk, samt betraktninger rundt disse. Den er nok mer detaljert på dette området enn en normal vitenskapelig rapport ville vært.

3 OPTIMERING SOM METODE

3.1 Noen sentrale begreper

Optimering forutsetter at man vet hva det skal tas beslutning om, hva man ønsker å oppnå, og hva som er mulig. Man må vite hvordan systemet det skal tas beslutning om fungerer – forstå de direkte konsekvensene av enkeltvalgene som gjøres. På den annen side trenger man ikke forstå de totale konsekvensene for systemet, eller ha oversikt over de konkrete konsekvensene av alle mulige beslutningsalternativer – det er nettopp denne helhetsforståelsen optimeringen skal tilføre. Dette avsnittet går gjennom noen sentrale begreper som gjør det mulig å beskrive beslutningsproblemet og beskrive hva som kommer ut av optimeringen.

Optimeringen skal ta en beslutning om en del forhold, og disse forholdene er beskrevet gjennom ***beslutningsvariabler*** (eller bare ***variabler)***. Variablene kan ses som dimensjoner i et rom som her er kalt ***løsningsrommet***. Løsningsrommet omfatter alle kombinasjoner av alle mulige og umulige kombinasjoner av beslutningsvariablene. En delmengde av dette rommet er de mulige

eller lovlige løsningene, og dette kalles her for ***domenet*** til optimeringsproblemet. Skal man fylle et bagasjerom på 1000 liter med sigaretter og sprit, så er 5000 liter av hver en del av løsningsrommet, men ikke en del av domenet. Domenet er begrenset av diverse forhold (så som totalt volum på bagasjerommet i eksempelet over), og alle slike begrensninger kalles ***beskrankninger***. En ***lineær beskrankning*** sier at en vektet sum av variablene må være større eller mindre enn en bestemt verdi, mens en ***heltallsbeskrankning*** sier at en konkret variabel bare kan ta heltallsverdier.

Man kan skille mellom kontinuerlige og diskrete valg. Skal man plassere et hus på en tomt, så kan man forskyve det så mye eller lite man vil innen visse grenser – man har løsninger med en meters avstand, en centimeters avstand, en millimeters avstand eller en mikrometers avstand – løsningene ligger tett i tett og utgjør et kontinuum. Skal man derimot velge hvilken tomt huset skal stå på, så ligger løsningene atskilt – avstanden kan være stor eller mindre stor, men de er definitivt atskilt. Et valg mellom adskilte (diskrete) alternativer kalles her et ***diskret valg*** mens et valg innenfor et kontinuum av alternativer kan kalles et ***kontinuerlig valg***.

Hva man ønsker å oppnå er beskrevet gjennom en ***objektfunksjon*** "$f([v])$", og et ***objektuttrykk*** "$\max(f([v]))$" eller "$\min(f([v]))$" som beskriver godheten til en løsning. Objektfunksjonen må være entydig, slik at en løsning som gir høyere verdi på objektfunksjonen enn en annen løsning alltid er vurdert som en bedre løsning enn denne andre løsningen, eller omvendt. Konvensjonen er at objektfunksjonen tolkes som kostnad, slik at en lav verdi på objektfunksjonen betyr en bedre løsning, og objektuttrykket er et minimumsuttrykk.

Et ***optimum*** er den løsningen (kombinasjonen av variabelverdier) som gir den høyeste objektfunksjonen. Optimumet er altså ikke den høyeste verdien av objektfunksjonen. Hvis et hus skal plasseres på en tomt for å gi maksimal verdi, så er kanskje den beste plasseringen midt på tomta, fordi salgsverdien da er 4 millioner kroner, mens den alle andre steder er mindre enn dette. Da er det 'midt på tomta' som er optimum og ikke '4 millioner kroner'. Med 'optimum' menes normalt et ***globalt optimum***, altså den løsningen innenfor hele domenet som gir den høyeste objektfunksjonsverdien. Et ***lokalt optimum*** er en løsning hvor ingen andre løsninger i dens umiddelbare nærhet har en høyere objektfunksjonsverdi. Lokale optima er litt gufne, for ut fra lokale betraktninger ser de jo veldig mye ut som et optimum, men det er de altså ikke nødvendigvis. I et helt diskret valg er det ingen løsninger som har noen andre løsninger i sin umiddelbare nærhet, og da blir alle løsningene lokale optima. Derfor kan diskrete problemer være vanskeligere å løse enn kontinuerlige problemer.

I den matematiske løsingen av et optimeringsproblem kan man ta en ***analytisk tilnærming*** eller en ***søketilnærming***. En ***analytisk tilnærming*** tar utgangspunkt i spesielle egenskaper som man vet at den optimale løsningen må ha, og prøver å finne den direkte. Et velkjent eksempel er et domene uten beskrankninger og en objektfunksjon som er deriverbar overalt i domenet. Da vil den deriverte mht alle variablene være lik null i optimumet, og en naturlig analytisk tilnærming vil være å formulere dette som et likningssett som løses. En søketilnærming består i mer generelt å søke eller lete blant de mulige løsningene for å finne en best mulig løsning, og ideelt

sett ende på den aller beste løsningen (optimum).

En spesiell type søk er ***tresøk***, som er søk innenfor en logisk trestruktur. Tresøk er et stort område i seg selv, og skal bare gis en minimalistisk beskrivelse her. Et tre består av noder, der hver node stammer fra én og bare én foreldrenode og kan forgrenes til én eller flere undernoder. En node uten undernoder kalles en **bladnode**. Et tresøk er aktuelt i et problem med mange diskrete valg, og i dette tilfellet er søketreet på mange måter algoritmens beslutningstre. Formelt er nok søketreet en ufullstendig instansiering av beslutningstreet. Rotnoden er situasjonen der ingen beslutninger er tatt, hvor man ser på alternative beslutninger på et delproblem, typisk én diskret variabel. Utvidelse av delløsningen i noden med de alternative (lovlige) verdiene for denne variabelen gir undernoder. I hver node ligger noen variabler fast – de delproblemene som allerede er adressert, mens man ser på alternative valg for et nytt delproblem for å danne forgreningene ut fra noden.

3.2 Generisk optimeringsproblem

Et optimeringsproblem er gitt ved en objektfunksjon og et domene. Optimeringen består i å finne det punktet i domenet som gir den beste (høyeste eller laveste) objektfunksjonen. Domenet er gitt ved et antall beslutningsvariable [v] og begrensninger på hvilke verdier beslutningsvariablene kan ta alene eller sammen – beskrankninger. Intuitivt definerer beslutningsvariablene et rom, og beskrankningene definerer en form i dette rommet. Denne formen (domenet) kan være sammenhengende, splittet eller diskret. I de fleste praktiske sammenhenger vil domenet ha færre dimensjoner enn rommet utspent av [v], (lavere dimensjonalitet enn n i oppsettet under) og vil bestå av et endelig antall kontinuerlige (sammenhengende) former. På generisk form kan beslutningsvektoren, optimeringskriteriet og beskrankningene skrives som følger:

beslutningsvektor	$[v] = [v_1, \quad v_2, \quad v_3, \quad v_4, \quad ....., \quad v_n]$
objektuttrykk	$\min(f([v_1, \quad v_2, \quad v_3, \quad v_4, \quad ....., \quad v_n]))$
beskrankninger	$[G] = [g_1([v]), \quad g_2([v]), \quad g_3([v]), \quad ....., \quad g_m([v])] \geq [0]$

Der f og g_i er vilkårlige funksjoner. Objektfunksjonen trenger ikke være formulert som et min-uttrykk, men det vil alltid være ekvivalent til et. Likeledes kan beskrankningene være uttrykt på mange måter, og skjønt de i prinsippet kan omformes til større-enn-null-uttrykk slik som her, er ikke alltid dette det mest intuitive eller hensiktsmessige.

3.3 Søkeproblemer og analytisk løsbare problemer

Det finnes en del teknikker for å gjennomføre en optimering. Noen går på strukturen i problemet og beviser optimalitet av en konkret løsning, uten nødvendigvis å være innom andre løsninger på veien. Dette fordrer at problemet oppfyller konkrete kriterier. Dette kan vi kalle analytiske optimeringsmetoder, og dersom et problem kan løses analytisk, vil en slik tilnærming som regel være svært effektiv, og naturligvis gi veldig sterk kunnskap om løsningen.

Der man ikke har en analytisk løsningsmetode, kan man bruke en søkealgoritme. En søkealgoritme består i at man finner seg en løsning og leter mer eller mindre på måfå etter en bedre løsning. En typisk søkealgoritme kan ha følgende elementer:

1. Husk beste løsning funnet så langt og objektfunksjonen til denne.
2. Finn en ny løsning.
3. Beregn objektfunksjon av ny løsning.
4. Dersom ny løsning er bedre enn beste løsning, sett ny løsning som beste løsning.
5. Finn ut på hvilken måte løsningen bør endres for å bli bedre.
6. Sammenlikn objektfunksjon av løsningen med beste løsning og utsiktene til å forbedre løsningen ved å endre den i tråd med steg 5. Bestem om neste løsning skal følge heuristikken i steg 5. eller starte i vilkårlig nytt punkt.
7. Begynn forfra.

Avhengig av hvordan en søkealgoritme er utformet kan den i prinsippet anvendes på et hvilket som helst domene og med en hvilken som helst objektfunksjon, men den er ikke nødvendigvis veldig effektiv. Hvis man ikke stiller krav om at løsningen skal finnes i endelig tid, vil nesten enhver rimelig algoritme finne et optimum, men dette har naturlig nok ingen praktisk betydning.

Kompleksitet er det store problemet i all praktisk optimering siden et reelt søkeproblem gjerne har noen tusen variable. Om man skulle ønske å dekke domenet med et tilfeldig søk, må man i hvert fall kunne velge en lav og en høy verdi for hver variabel i forskjellige kombinasjoner. To opphøyd i tusen er imidlertid et meget stort tall, og livet er kort, så tilfeldige søk har liten praktisk anvendelse. Dermed blir heuristikken – steg 5 og 6 i algoritmen over – essensiell.

Ved søk i et semikontinuerlig domene og med en lokalt kontinuerlig og deriverbar objektfunksjon, vil man bruke derivasjon i steg 5, og dette vil relativt effektivt lede løsningen mot et lokalt optimum. Hvis det forventes mange lokale optima, blir det viktig å kunne hoppe videre til en helt annen løsning når det lokale optimumet er funnet.

I et helt diskret domene vil man typisk søke i et tre av alternative løsninger eller delløsninger. Et typisk tresøk har en tom løsning som rot og komplette løsninger som bladnoder. En node er en delløsning, og ekspansjonen av en node består av utvidelsene av den. Bygger og analyserer man hele treet, så har man naturlig nok kontroll på det, men er det tusen variable som alle kan ta to verdier hver, så blir det igjen to opphøyd i tusen bladnoder, og derfor analyserer man ikke hele treet, og man bygger det ikke heller.

Det er derfor essensielt å finne ut hvilke noder som er lovende og som skal ekspanderes – man må kunne beregne en tilnærmet objektfunksjon for en delløsning – en heuristikk – og så ekspandere de mest lovende delløsningene. Det å beregne forventet objektfunksjon når mange av variablene er ukjente er ikke nødvendigvis trivielt. Kan man gjøre dette korrekt hver gang, så har man en metode for å konstruere riktig løsning i første forsøk, og jo mer usikker heuristikken er, jo lengre omveier vil man gå i søket.

I et oppstykket, lokalt sammenhengende domene vil et tresøk være en naturlig fremgangsmåte

for å generere uavhengige løsninger hvoretter man søker seg frem mot et lokalt optimum ved en tilnærming basert på derivasjon.

3.4 Optimering og operasjonsanalyse

Noen liker begrepet nyttefunksjon. Dersom vi sier at *f* er nyttefunksjonen og domenet er det reelle beslutningsrommet definert ved beslutningsvariable, avhengigheter og begrensninger, er det ikke vanskelig å se at det generelle optimeringsproblemet er ekvivalent med et generelt beslutningsproblem. Mange vil definere OA til å være beslutningsstøtte, og i så fall vil det rundt ethvert OA-problem ligge et optimeringsproblem. Det OA-problemet man studerer trenger imidlertid ikke være hele beslutningsproblemet eller optimeringsproblemet. Det er ingen ting i vegen for at et OA-prosjekt kan ha til oppgave å beregne objektfunksjonen eller bare noen ledd i objektfunksjonen for noen konkrete løsningskandidater og derved støtte en overordnet beslutningsprosess som i sin tur (i prinsippet) er ekvivalent til en optimeringsalgoritme – den være seg god eller dårlig.

Dersom [v] beskriver en forsvarsstruktur (gjerne i tusenvis av enkeltvariable), *f* er ytelsen (Measure of Policy Effectiveness) til strukturen i en valgt blanding av scenarier, mens økonomi, arv, fysiske og praktiske begrensninger er uttrykt gjennom [G], så har man den typiske strukturanalyse.

Selv om ethvert beslutningsproblem er ekvivalent med et optimeringsproblem, er det ikke gitt at det er hensiktsmessig å formulere hele problemet formelt. Om man skulle klare å få det til, er det heller ikke gitt at de formelle optimeringsteknikkene vi kjenner er spesielt effektive til å finne gode løsninger slik som nyttefunksjonen og beskrankningene er formulert.

En typisk søkeprosedyre for en normalt uryddig objektfunksjon og domene må beregne objektfunksjonen til mange tusen løsningskandidater. Samtidig vet vi at en mye brukt objektfunksjon i militære strukturanalyser er en relativt enkel funksjon av utfallsvariablene på en stridsutfallsberegning (stridssimulering), eller et utvalg av disse dersom enkeltberegningene har et stokastisk element. Når vi vet at en slik utfallsberegning i gode gamle dager gjerne var satt opp til å ta flere timer i ren beregningstid, vil beregningstiden for optimeringen fort bli uhåndterbar. Likeledes kan man ha beskrankninger som er uklart definert – for eksempel at løsningen må se realistisk ut. Selv om realismen nok bunner i relativt harde sammenhenger som kunne vært formulert stringent, så er det ikke alltid bryet verdt siden det ofte vil være mye enklere og sikrere å nyttegjøre seg eksisterende kunnskap gjennom konkrete vurderinger enn ved å utvinne den på generell form. Det er formelt sett ingen ting i vegen for at man kan ha en mann i loopen i en optimering, men det begrenser naturlig nok muligheten for å benytte seg av formelle teknikker.

I en virkelig analyse vil man ha en imperfekt modell av problemet, og man vil ha et imperfekt søk. Man kunne i teorien ønsket seg en perfekt realistisk modell og et komplett søk, men både realisme og søkeomfang bidrar til å øke kompleksiteten og dermed beregnings- og analysetiden, og det må foretas en avveining mellom disse to. Det hjelper lite med en perfekt modell hvis man

likevel lander på en tilfeldig løsningsvektor innenfor domenet, og det er tilsvarende unyttig å ha funnet den eksakte løsningen på galt problem. Denne avveiningen er vanskelig, og blir lett styrt av a priori preferanser. Det er mange eksempler på forsvarsanalyser som har brukt årsverk på å studere ytelsen til ett eller to løsningsalternativer. Tilsvarende er mange optimeringsprosjekter så forenklet at løsningen ikke kan brukes til reell beslutningsfatning.

Forsøk på OPKOLA og senere SLADI ved FFI viste at et noe forenklet forsvarsanalyseproblem kunne formuleres innenfor en optimeringsløkke (3), (4). Dagens computere har vesentlig øket kapasitet i forhold til de som ble brukt den gang, og det skulle i dag være mulig å håndtere modeller med samme realisme som de tradisjonelle utfallsberegningsmodellene innenfor en formell søkeprosedyre uten at løsningstiden blir uhåndterbar. Det er derfor grunn til å tro at optimering vil ha et økende potensial innen strukturanalyser i fremtiden

4 LP OG MIP

To sentrale klasser av optimeringsproblemer er lineære programmeringsproblemer og Mixed Integer Programming problemer. LP-klassen av problemer er viktig av to grunner:

- Den beskriver svært mange reelle beslutningsproblemer.
- Den har alltid en effektiv løsningsmetode.

I tillegg til de mange problemene som reelt er lineære, er det en stor gråsone av problemer som kan tilnærmes godt med et lineært problem. Dette betyr ikke at alle problemer kan analyseres ved lineariseringer, og spesielt ikke ved kontinuerlige lineariseringer. Det kan argumenteres for at LP i en periode ble forsøkt markedsført som selve ”Optimeringsmetoden for alle problemer”, og dette har siden bragt den litt i vanry.

Et MIP-problem er langt mer komplekst – det er egentlig et diskret søkeproblem der likheten til et LP-problem brukes til å lage en usedvanlig god heuristikk. MIP-problemer lar seg derfor løse størrelsesordener mer effektivt en diskrete søkeproblemer i sin alminnelighet.

Eksempelet med tyven som skulle fylle opp bilen med sprit og sigaretter kan formuleres som følger:

$$
\begin{array}{ll}
\textit{beslutningsvektor} & [v] = [\textit{spritmengde},\quad \textit{sigarettmengde}] \\
\textit{objektuttrykk} & \max\left(250\textit{kr}\cdot\textit{spritmengde} + 200\textit{kr}\cdot\textit{sigarettmengde}\right) \\
\textit{beskrankninger} & \begin{cases} 100\textit{l}\cdot\textit{spritmengde} + 300\textit{l}\cdot\textit{sigarettmengde} \le 1000\textit{l} \\ 200\textit{kg}\cdot\textit{spritmengde} + 100\textit{kg}\cdot\textit{sigarettmengde} \le 800\textit{kg} \end{cases}
\end{array}
$$

Dette er et rent lineært problem. Hvis vi derimot forutsetter at sprit og sigaretter kun kan lastes inn i hele kasser, så vil de to variablene bli heltallsvariable. Dette er to tilleggsbeskrankninger, som gjør problemet til et rent heltallsproblem (IP eller Integer Programming). Hvis sprit kun kan lastes inn i hele flasker – noe som jo er rimelig – mens den siste sigaretten om nødvendig kan hakkes i småbiter for å få plass, så har vi en heltallsvariabel og en kontinuerlig variabel, og dette

er i prinsippet en MIP. Det er liten praktisk forskjell på å løse et MIP-problem og et IP-problem.

4.1 Lineær Programmering (LP)

Et lineært programmeringsproblem er et optimeringsproblem der variablene er reelle tall, mens beskrankninger og objektfunksjon er lineære i variablene. Dette er et ganske sterkt krav, men som vist under er det ikke fullt så sterkt som det kan høres ut. Det er kun funksjonene som skal være matematisk lineære, og de skal kun være lineære i beslutningsvariablene.

- Funksjonene kan gjerne ha konstantledd – de trenger ikke uttrykke proporsjonalitet.
- Det er ingen begrensninger på hvordan funksjonene kan avhenge av inngangsparametre til optimeringen.
- Det er ingen begrensning på hvordan inngangsparametrene er fremkommet.
- Sammenhengene som er representert kan godt inneholde gjensidige avhengigheter. (Kjedede enveis-avhengigheter omtales noen ganger som ”logisk lineære”, men dette har lite med matematisk linearitet å gjøre).

Et LP-problem har noen meget behagelige egenskaper:

- Ingen lokale optima.
- Kontinuerlig og konvekst domene i rommet utspent av beslutningsvariablene.
- Alle avgrensninger til domenet er ”plan” (ikke nødvendigvis i to dimensjoner).

Et LP-problem har alltid et optimum i et hjørne av domenet, og siden man ikke trenger være redd for lokale optima, kan man lete seg frem langs kantene til domenet helt til man finner et hjørne hvor objektfunksjonen ikke øker langs noen av kantene. Da er dette et globalt optimum. Hvis objektfunksjonen er uforandret langs en eller flere kanter ut fra denne løsningen, sa vil hele denne avgrensningsflaten eller avgrensningskanten utgjøre globalt optimale løsninger, ellers er hjørneløsningen unik.

En LP kan løses effektivt med en computer. Et problem med noen hundre tusen variable løses typisk i løpet av i størrelsesorden 0,1 sekund. Den vanligste algoritmen for å løse en LP kalles SIMPLEX (1), men i praksis er alle løsningsalgoritmer for lineære problemer tilnærmet ekvivalente.

4.2 Mixed Integer Programming (MIP)

Et Mixed Integer (MIP) Problem er et LP-problem der domenet er diskret i visse variable, formulert slik at disse variablene kun tar heltallsverdier. I mange situasjoner vil en overvekt av disse variablene kun ta binære verdier. En MIP har egentlig ingen av de gode egenskapene til en LP. Den har et semi-diskret domene og den har i særdeleshet masser av lokale optima.

En MIP er egentlig et helt normalt søkeproblem, men den har en gunstig egenskap: Dersom problemet bare er en diskret variant av et helt lineært problem, vil LP-versjonen av det utgjøre en usedvanlig god heuristikk. Hvis man glemmer alle eller noen av heltallsbegrensningene, så vil problemet alltid være effektivt løsbart, og disse ”relakserte” løsningene vil gi viktig

informasjon om hvor man bør lete etter en optimal heltallsløsning.

En node i søketreet vil ha verdier for noen av heltallsvariablene, og hvis man ignorerer heltallsbegrensningene på de resterende heltallsvariablene, får man et meningsfylt estimat på objektfunksjonen av en fullført gren basert på en ufullstendig node. Den såkalt relakserte løsningen vil også gi mange andre indikasjoner om hvordan den optimale heltallsløsningen kan se ut.

Estimatet på objektfunksjonen vil alltid være et optimistisk estimat siden domenet for heltallsløsninger kun er en delmengde av domenet for det kontinuerlige problemet. Estimatet for en node gir altså en nedre grense for objektfunksjonen i deltreet under noden (eller øvre grense dersom objektuttrykket er et max-uttrykk). Dette gjør at man kan avslutte søket i grener som ikke kan gi bedre resultat enn hittil beste verdi for en lovlig løsning, og det gjør at man alltid har en grense for hvor god beste løsning kan være for hele søketreet. Basert på disse estimatene får man alltid en god sammenlikning mellom noder på samme dybde, og man må gjøre et valg om hvor aggressivt man vil søke nedover i forhold til å gå tilbake og søke etter mer lovende grener. Den laveste verdien for objektfunksjonen på den relakserte løsningen over alle ikke-ekspanderte noder i søketreet er den nedre grensen for objektfunksjonen i hele søketreet. Differansen mellom denne nedre grensen og objektfunksjonen i beste heltallsløsning kalles optimalitetsgapet, og kan uttrykkes enten prosentvis eller absolutt. Når man har funnet en løsning som ligger tilstrekkelig nær grensen for hvor gode løsninger som kan finnes, så har man et godt grunnlag for å si seg fornøyd og man kan avslutte søket. Det er vanlig å bruke en optimalitetsgap grense som kriterium for å avbryte et MIP-søk.

Den relakserte løsningen vil si mye om hvilke verdier de forskjellige variablene kan forventes å ta, og hvilke heltallsbeskrankninger som er viktige og mindre viktige – en variabel som ender opp midt mellom 0 og 1 vil være sterkere påvirket av en heltallsbeskrankning enn en som ender opp svært nær eller på 0 eller 1. Det lønner seg som regel å avklare de signifikante valgene før de mindre signifikante, og derfor gir dette viktig informasjon til styring av søket.

Den informasjonen man kan få ut av den relakserte løsningen, kombinert og brukt på forskjellige måter gjør egnede MIP-søk størrelsesordener mer effektive enn andre tilsvarende komplekse søkeproblemer. Egnetheten av et MIP-søk avhenger egentlig ene og alene av om den kontinuerlige modellen er en god tilnærming til heltallsmodellen. Noen enkle erfaringer vil bli oppsummert i kapittel 8.

5 EN GENERISK LP/MIP-MODELL

I definisjonen av et LP/MIP-problem er det en del informasjon som må håndteres og representeres, og det vil være stor forskjell på datastrukturen mellom forskjellige modeller. Siden beregningen har et bestemt format vil imidlertid også formatet for hvordan man kan formulere et LP/MIP-problem være relativt fast.

En LP- eller MIP-modell formuleres normalt i form av fem komponenter:
Mengder – som beskriver hva modellen består av. Mengdene brukes til å indeksere parametere og variable og som indeksmengder for beskrankninger og summering i objektfunksjonen. De brukes kun til dette. Basismengdene beskriver modellen, mens avledede mengder er delmengder og produktmengder eller kombinasjoner av disse, som beregnes ut fra inngangsverdiene. For eksempel vil en graf ha en grunnmengde av noder og kanter, og for indekserings- og beregningsformål vil det være interessant for hver node å definere mengden av kanter som går ut fra denne noden. Sistnevnte er da en avledet mengde.
Parametre – globale parametre eller parametre indeksert over mengder eller kombinasjoner av mengder. Parametre er alle tall i modellen som selve optimeringen ikke skal ta en beslutning om. Parametrene er ekvivalent til (eller bestemmer) koeffisientene i en LP-matrise, men trenger ikke være formulert slik.
Variabler – er de tallene som skal optimeres. Variablene kan være globale eller indeksert over mengdene. En variabel kan enten være kontinuerlig eller ha en heltallsbeskrankning.
Beskrankninger – sier hvilke kombinasjoner av variable og parametre som er lovlige. Beskrankninger tar form av ulikheter eller likheter, og er alltid ekvivalent til en mengde av større-enn-null uttrykk. Man vil typisk ha klasser av beskrankninger indeksert over mengder, og beskrankningene vil gjerne ha ledd som igjen summeres over mengder.
Objektfunksjonen – er den funksjonen som skal anta minimal verdi (eller ekvivalent maksimal verdi). Objektfunksjonen vil direkte eller indirekte være en funksjon av alle variablene, og vil typisk bestå av ledd som skal summeres over enkelte av mengdene.

Tyveriproblemet kan nå formuleres litt mer generelt:

- Den viktigste mengden er produktmengden, som består av aktuelle produkter – sprit og sigaretter.
- Det er tre klasser av parametere som indekseres over denne mengden:
    - Priser på svartebørsen.
    - Vekt per enhet.
    - Volum per enhet.
- Variablene er mengden av hvert produkt som skal stjeles, igjen indeksert over produktmengden.
- Beskrankningene er vektbegrensning og volumbegrensning, og de blir nå en sum over produktmengden.
- Objektfunksjonen uttrykker verdien av varene på svartebørsen, som også er en sum over produktmengden. Objektuttrykket er et max-uttrykk over objektfunksjonen.

Med denne modellen kan vår mann løse opplastingsproblemene sine i alle kjøpesentre han bryter seg inn i bare ved å legge inn pris, vekt og volum for nye produkter. I realiteten vil denne fleksibiliteten være en forutsetning for at modellen skal være anvendelig.

Modellen kunne vært generalisert ytterligere. For eksempel kunne man sett på en mengde av begrensende faktorer, her instansiert ved vekt og volum, men dette er kanskje litt søkt. Da er det mer naturlig å huske på at vår venn tyven er en ekspansiv kar, som ser muligheten for å tjene

mer penger ved å ansette eller underkontrahere flere tyver og dermed kunne få med seg flere biler. Det er derfor naturlig å legge inn bilene som en mengde med vekt- og volumbegrensninger indeksert over denne mengden. Det vil være hensiktsmessig å legge inn denne generaliseringen selv om bilmengden i utgangspunktet bare har ett element. Det blir nå vekt- og volumbeskrankninger for hvert element i denne mengden (indeksert over bilmengden), og variablene blir mengden av hver varekategori som går inn i hver bil, altså indeksert over det kartesiske produktet av produktmengden og bilmengden.

Når grunnmodellen først er på en rimelig generisk form, er det på tide å se på faktorer i det virkelige problemet som kunne vært implementert for å gjøre modellen mer generelt gyldig. Tyveriet var kanskje motivert av en bestilling fra en heler eller et krav fra en mafioso om å skaffe visse varer.

- Krav fra mafiosoer må man oppfylle, ellers går det galt. For de aktuelle varekategoriene blir derfor dette en minimumsbeskrankning på summen over alle bilene av mengden av denne varen.
- Det vil normalt kaste bedre av seg å oppfylle bestillinger enn å stjele varer som det ikke er truffet avtale om. Dermed vil verdien av total varemengde for visse varer være høyere innenfor bestillingen enn etter at bestillingen er oppfylt. Her må det defineres samlevariable som definerer totalvolum over alle bilene, men splittet på det som er innenfor en bestilling og det som er utenfor bestilling.

Med noen flere slike utvidelser vil modellen begynne å nærme seg kompleksiteten til en liten modell man vil kunne finne i praktisk anvendelse – ikke bare blant tyver.

Det viser seg stadig hensiktsmessig å definere en LP/MIP-modell rundt mengder, men det er ingenting som rent formelt fordrer dette, og beregningen på lavt nivå vil til slutt operere med selvstendige enkeltvariable. Et generelt problem lar seg aldri formulere gjennom enkeltvariable – kun meget små og spesifikke og svært velformulerte problemer kan praktisk formuleres på den måten. Hvis man strever med å få tak på og få modellert et problem, vil identifisering av viktige mengder og definisjon av modellen rundt disse være en god tilnærming. Definisjonen kan gjerne gå i den rekkefølgen som er beskrevet over. Dersom problemet faktisk er generisk – dersom elementene i mengdene ikke er entydig definert, er definisjon rundt mengder definitivt den beste tilnærmingen.

6 PROBLEMER SOM LAR SEG LINEARISERE

Det er naturligvis ikke noe mål i seg selv å modellere et problem på en bestemt måte. Styrken til LP-modellering består nettopp i at man bruker de lineære forutsetningene for alt de er verdt, og da kan man oppnå sterke resultater. Dersom disse forutsetningene svikter, blir imidlertid resultatene desto galere. Som alltid skal man ta utgangspunkt i problemet man skal løse, og LP/MIP-modellering er kun ett av flere verktøy, og skal benyttes kun der det passer.

Av denne grunn skal man ta forutsetningene om linearitet svært alvorlig, og spesielt forsikre seg

om at det konkrete fenomenet man ønsker å studere ikke er et resultat av genuine ikke-lineariteter. Når dette er sagt, så er det ofte flere problemer som er lineære eller lineariserbare enn det man ved første øyekast skulle tro. Med hensyn til anvendelighet av LP/MIP-modellering, er det i hovedsak fire grupper av problemer som metoden er egnet for:

- Genuint lineære problemer/systemer
- Lineære delproblemer/delsystemer i et ikke-lineært problem
- Stykkevis lineære eller lineariserbare problemer
- Diskrete problemer (herunder allokeringsproblemer)

Det er alltid kjekt å vite at det er en tilnærmet en-til-en sammenheng mellom det problemet man studerer og modellen man bruker, og derfor er det behagelig å jobbe med hva vi kan kalle genuint lineære problemer. Hvis problemet er lineært, kan det avbildes direkte over i en lineær modell. Produksjonsplanlegging i et raffineri er et klassisk eksempel som ofte benyttes i LP-lærebøker. Det er en klart lineær sammenheng mellom hva som går inn og hva som kommer ut av hver prosess, og de fleste prosessene har maksimums- og eventuelt minimumsbeskrankninger. Skal man gjøre det riktig avansert, tar man med en liten heltallsbeskrankning og -kostnad dersom man ønsker å stoppe en prosess temporært for å gå under min-beskrankningene. Eventuelt kan noen knekkpunkter introduseres dersom raffineriet må kjøpe ekstern kapasitet. Mer generelt er dette modellen for de fleste produksjonsprosesser der man har produksjon basert på forskjellige innsatsfaktorer med faste og variable kostnader. Innen gitte rammer er det som regel en lokalt lineær sammenheng mellom hva man setter inn av innsatsfaktorer og hva som kommer ut av slike prosesser.

I et produksjonssystem som angitt over vil det som regel også være ikke-lineære faktorer. Eksempelvis vil endringer i produksjonsvolumet kunne innvirke på de ansattes motivasjon, positivt eller negativt. Dette er særlig tilfelle når endringstakten passerer en grense for hva som gir normal arbeidsbelastning eller når arbeidsplassen blir mer eller mindre trygg. Dette kan føre til en ikke-lineær sammenheng mellom innsatsfaktorer og produksjon dersom de ansattes produktivitet er viktig. Produksjonssystemet er dermed ikke nødvendigvis lineært som system, og dynamikken i endringsprosesser kan være dominert av ikke-lineære faktorer. Dersom det imidlertid ikke er dynamikken i en slik volumendring man studerer (f eks i en langsiktig studie av ønsket gjennomsnittlig produksjonsnivå), kan man fortsatt ha et lineært analyseproblem. Et lineært delproblem/delsystem virker ikke fullt så behagelig som et system som er lineært i sin helhet, fordi man må forsikre seg om at det problemet man studerer ikke er dominert av de ikke-linearitetene som ligger i systemet som helhet. Imidlertid vil alle systemer være del av en verden med ikke-linariteter, og derfor vil det uansett være hensiktsmessig å undersøke slike avhengigheter nøye.

Funksjoner som ikke er lineære kan ofte tilnærmes ved en lineær funksjon. Bl a kan alle stykkevis deriverbare funksjoner tilnærmes så tett man vil med tilstrekkelig mange lineære biter. (Dette følger fra definisjonen av deriverbarhet). Enhver stykkevis lineær beskrankning kan dermed tilnærmes med et tilstrekkelig antall lineære beskrankninger, og eventuelt heltallsbeskrankninger dersom domenet ikke blir konvekst. På denne måten kan de aller fleste

problemer i prinsippet lineariseres, men det koster i form av kompleksitet. Dette er sjelden noe problem for en rimelig oppstykket funksjon som ikke krever heltallsvariable, men dersom det fremstår som avgjørende at modellens representasjon blir liggende nært den reelle krumningen, og modellen ender opp som et stort heltallsproblem må man spørre seg om linearisering er en hensiktsmessig måte å modellere problemet på. Generelt er knekkpunkter i det reelle problemet uproblematiske – modellen er fortsatt grunnleggende lineær, også om det er mange knekkpunkter. Et avgjørende spørsmål blir om funksjonene i det reelle problemet er krumme, og det nettopp er krumningen man egentlig ønsker å modellere.

Problemer knyttet til diskrete valg er en viktig klasse av lineariserbare problemer, og allokeringsproblemer er en viktig underklasse. Dersom man skal velge mellom fire beslutningsalternativer, for eksempel å reise til Afrika, reise til Amerika, gå i svømmehallen eller å spise middag, så kan man se valget som én beslutningsvariabel. I dette eksempelet er det liten grunn til å tro at utfallet av beslutningen er lineært i de fire alternativene, uansett hvordan man ordner dem. Dersom man imidlertid ser dette som fire beslutningsvariable med verdier fra null til én, så har man et lineært problem. Summen av dem må være én, siden det var en forutsetning at man måtte velge *mellom* dem, og man må vurdere om delvise valg er mulig. Det er kanskje mulig å spise *litt* middag, og i så fall kan middagsvalget ta mange forskjellige verdier, men det er ikke mulig å reise *litt* til Amerika – da blir man våt, og det vil man jo ikke bli. Dermed blir beslutningsvariabelen for å reise til Amerika en heltallsvariabel som kun kan ta verdiene 0 og 1.

Det generelle allokeringsproblemet er tilsvarende: Man har n forskjellige ressurser som kan allokeres til m forskjellige steder i k forskjellige modi, og der hver ressurs bare kan allokeres én gang og i én modus (eventuelt oppstykket). Dette modelleres med $n \times m \times k$ beslutningsvariable, som representerer alle de forskjellige mulige delallokeringene. Uansett hvordan utfallet av disse allokeringene varierer med allokeringsalternativene, vil dette bli et lineært problem. Om variablene er heltallsvariable eller kontinuerlige variable avhenger av om ressursene må allokeres i sin helhet eller om de kan stykkes opp. Ved å legge på tilleggsvariable kan man f eks ha beskrankninger som sier at en ressurs må gå i sin helhet til en destinasjon men kan gjøre dette i enhver (reell) blanding av modi.

Med boolske allokeringsvariable vil denne typen problemer alltid være lineære, men med kontinuerlige allokeringsvariable, må funksjonene også være lineære i enkeltvariablene. Et allokeringsproblem er normalt lineært også der det ikke er diskret. Mer generelt kan alle problemer som i sin helhet er diskrete lineariseres ved at det opprettes en heltallsvariabel for hvert alternativ.

7 PROBLEMSTØRRELSE

Problemstørrelse og kompleksitet er alle problemers mor når man ser på gjennomførbarhet av en optimering. Som kjent er et søkeproblem NP-komplett (NPC)[2] med mindre det er spesielle forhold som forenkler problemet. I forhold til det å finne en optimal løsning er en MIP intet unntak, men den gir mulighet for å eliminere store deler av forgreningen, og den oppfører seg som nevnt veldig pent når det gjelder å finne en *nær* optimal løsning. Et LP-problem er et sted mellom lineært og kvadratisk i antall variable, og blir sjelden noe praktisk problem.

For å få et godt gjennomførbart søk må man til enhver tid bevare hele søketreet (bortsett fra eliminerbare grener), og dette spiser minne. I tillegg tar det prosesseringstid hver gang en node undersøkes, slik at både søketid og minneforbruk øker sterkt som funksjon av problemstørrelse.

Inntil for få år siden representerte et par hundre heltallsvariable et meget stort problem. For 10 – 15 år siden var en LP med noen tusen variable et tilsvarende stort problem. I dag kan man håndtere flere millioner kontinuerlige variable og noen hundre tusen heltallsvariable. Da er det ikke snakk om å finne den optimale løsningen på MIP-problemet – det problemet er som nevnt NPC – men å finne en nær optimal løsning gitt at heuristikken til enhver tid leder søket i riktig retning.

Det følgende gir to eksempler på problemer med forskjellig kompleksitet:

Et klassisk lineært problem er raffineridrift hvor man tar inn råolje og skal produsere et lite antall sluttprodukter, og hvor det må opprettholdes en minimumsproduksjon innen forskjellige områder. Et slikt problem er i sin natur genuint lineært og meget lite – typisk 10 til 100 variable. (Siden man i dag har muligheten til å løse langt større problemer, vil selvsagt også en reell raffineriplanlegger modellere systemet sitt i stor detalj og ende opp med kompleks modell, svært forskjellig fra det som er referert her.)

Grunnvarianten av et allokeringsproblem har en variabel for hver kombinasjon av en ressurs og en mulig allokering/destinasjon. Dersom ressursene er enheter som må allokeres i sin helhet, blir de binære heltallsvariable. Med 500 diskrete ressurser og 200 mulige steder å allokere ressursene gir dette 100 000 heltallsvariable. Nå kan det være at man vil ta hensyn til spesielle forhold rundt en allokering i objektfunksjonen, og da må dette gjerne representeres ved variable som eksempelvis beskriver allokeringsmodus eller allokeringssituasjon. Ti forskjellige statuser på en slik variabel kan tidoble antall variable og i eksempelet over er man nå oppe i 1 000 000 heltallsvariable. I mange reelle situasjoner vil det være flere enn disse tre faktorene som bidrar til å multiplisere opp antall variable, og selv om man stykker opp problemet slik at det blir relativt få alternativer på hver faktor, ender man fortsatt opp med et stort antall variable.

[2] Non-Polynomial Complete – en klasse av prosesseringsproblemer som vil ha eksponentiell kompleksitet med alle kjente teknikker. Det er ikke bevist at søkeproblemer må ha eksponentiell kompleksitet, men det kan vises at dersom et NPC-problem kan løses i polynomisk tid, så kan alle NP-problemer løses i polynomisk tid.

8 MIP-SØKET OG UHENSIKTSMESSIGE LINEARISERINGER

Hensiktsmessigheten av en MIP-formulering avhenger av flere forhold. Dette avsnittet nevner to forhold som man fort får erfaring for ved bruken av MIP-modellering:

- Dersom MIP-problemet har lite til felles med det underliggende LP-problemet (den relakserte modellen), får man liten hjelp av heuristikken
- En søkealgoritme kan ikke nødvendigvis se de store linjene på egen hånd hvis ingen har fortalt den om dem. En rekke detaljbeslutninger kan låse hverandre på en slik måte at det blir umulig å få endret dem inkrementelt, og dette er en situasjon hvor søkealgoritmen også vil låse seg opp i sub-optimaliteter.

8.1 Lineariserte modeller som egentlig ikke er lineære

Hvis den kontinuerlige løsningen gir en dårlig indikasjon på hva som vil være en optimal heltallsløsning, nærmer søket seg et tilfeldig eller attpåtil villedet søk. Med en noenlunde realistisk problemstørrelse vil problemet da ikke være løsbart. Dersom modellen gir god mening som en kontinuerlig modell, men det tilfeldigvis er lagt et heltallskrav på noen av variablene, vil det være en nær sammenheng mellom kontinuerlig løsning og heltallsløsning. Et enkelt allokeringsproblem er et eksempel på dette: Man kan godt ha kontinuerlige allokeringsproblemer der hver ressurs kan fordeles på mulige destinasjoner i småbiter, og det er ingen grunnleggende forskjell mellom den kontinuerlige modellen og heltallsmodellen – en halv allokering er halvparten av en hel allokering.

Luftrekognoseringsmodellen som er beskrevet i kapittel 11 er et eksempel på det motsatte – enkelte heltallsvariable er brukt som brytere og gir bare mening for heltallsverdiene. De kontinuerlige løsningene representerer en oppførsel hos rekognoseringsflyene som er helt forskjellig fra den som ligger i heltallsløsningene. Heuristikken blir villedende på søket, og modellen bygger i praksis opp store deler av søketreet før den finner en rimelig god løsning. Modellen lot seg likevel bruke i praksis fordi kompleksiteten var svært liten.

8.2 Innbyrdes avhengigheter

Dersom flere variable er sterkt avhengig av hverandre uten at denne avhengigheten er eksplisitt ivaretatt, vil det skape problemer for søket. Det relativt intuitive resonnementet nedenfor gir godt samsvar med modellenes virkelige oppførsel, selv om det ikke bygger på den konkrete mekanismen i søkealgoritmen. Når søkealgoritmen backtracker opp gjennom søketreet for å utforske alternative grener prøver den i praksis å gjøre om et antall av de siste beslutningene som ble gjort i konstruksjonen av nåværende delløsning. (Alle noder i søketreet er delløsninger og når søket klatrer mot rota i søketreet, går det mot mindre komplette delløsninger). Det er rimelig at det vil være enklere å endre på få variabler enn på mange variabler, og at det er enklere å finne riktig punkt å forgrene seg ut fra dersom kostnaden på objektfunksjonen oppstår når den reelle beslutningen tas.

En velkjent allokeringsbeskrankning er at mange ressurser i en gruppe må allokeres til samme sted i en eller annen rolle, men der hver ressurs bare skal ha nøyaktig én rolle. Hvis ressursene

er modellert uavhengig og avhengighetene er beskrankninger mellom enkeltelementer, er det en lang veg fra det at et element i gruppen allokeres i en bestemt rolle og til det at alle elementene er allokert i forskjellige roller. Det virker ikke som denne beslutningen gis den oppmerksomheten den fortjener i et søk basert på en generisk søkealgoritme (så som C-PLEX) med mindre man relaterer den overordnede allokeringen av denne gruppen til en enkelt prioritert variabel.

Et annet problem som vil oppstå i dette eksempelet er at et element allokeres i feil rolle (eller modus). Vegen til å få omdefinert denne rollen fordrer at enheten tas ut av sin nåværende rolle for så å settes inn i en ny. Imidlertid vil enhver endring av disse to variablene hver for seg være ulovlig siden det skal være nøyaktig én allokering av hvert element i gruppen når først ett element er allokert. Et søk later til å ha lett for å rette opp uheldige beslutninger som kan endres inkrementelt. Gjensidige avhengigheter som fordrer at mange likeverdige variable endres simultant og koordinert later til å ende opp med de verdiene de tilfeldigvis dumpet ned på ved første forsøk. (Selv om algoritmene later til å fungere best der inkrementelle endringer ville vært mulig, er det ikke nødvendigvis slik at algoritmen vil håndtere dette gjennom inkrementelle endringer.)

Problemer som det over har vist forbedret oppførsel etter følgende tiltak:

- Kostnaden ved hele gruppeallokeringen knyttes til en enkelt allokeringsparameter som dermed får nødvendig fokus fra algoritmen
- Fjerning og reallokering av enkeltelementer gjøres til uavhengige beslutninger som ikke hver for seg vil bryte beskrankninger.

9 COTS PROGRAMVARE OG MODELLERINGSVERKTØY

Lineærprogrammering har vært en praktisk gjennomførbar optimeringstilnærming helt siden datamaskinenes barndom (og også før det, men da bare for helt spesielt interesserte). Det er derfor utviklet utallige små applikasjoner som støtter slike analyser, varierende fra de helt enkle som er gratis tilgjengelig, og opp til øverste hylle hvor det gis omfattende støtte for hele prosessen. Disse programmene er i dag relativt dyre. Det er ikke her gjort noe forsøk på et komplett søk gjennom tilgjengelig programvare, men ifølge andre som har forsøkt seg på dette, fantes det for et par år siden tre applikasjonspakker tilgjengelig som lå på omtrent samme ytelsesnivå, men vesentlig foran konkurrentene. C-PLEX var en av disse.

Her vil kun C-PLEX (med ”Concert Technology”) og det assosierte høynivå modelleringsverktøyet OPLStudio bli presentert, og presentasjonen tar et relativt overfladisk brukerperspektiv. Begge produktene leveres av ILOG, som har hovedkontor i Paris og utviklingsavdeling i USA.

C-PLEX har en løsningsalgoritme som tar en stor matrise av større-enn-null beskrankninger og heltalls beskrankninger samt en objektfunksjon, og som finner en løsning. Inngangsverdiene til løsningsalgoritmen er altså et formelt problem – en flat matrise – uten noen logisk

problemstruktur slik som skissert i kapittel 5. Normalt vil man ønske at optimeringsprogrammet henter data til beregningen fra en database eller en annen problembeskrivelse og omformer disse dataene til et formelt problem i tråd med en modell slik som beskrevet tidligere. Det er denne oppbyggingen av det formelle problemet basert på en modell og data som er implementeringen av modellen fra et brukersynspunkt. Brukerens håndtering av selve optimeringen vil for enkle problemer bestå i et enkeltstående kall av løsningsalgoritmen. I litt mer krevende problemer vil brukeren søke å tune løsningsalgoritmen ved hjelp av tilgjengelige kontrollparametre eller ved å legge inn egne spesialiserte steg i algoritmen.

9.1 C-PLEX (Concert Technology)

C-PLEX og Concert Technology er et C++ bibliotek med en løsningsalgoritme for LP og MIP. C-PLEX er løsningsalgoritmen og Concert Technology er biblioteket av funksjoner rundt det. Concert Technology gir viktig tilleggsfunksjonalitet utover det som ligger i C++ ved at basisfunksjonalitet man normalt vil ønske å bygge inn i en modell ligger forhåndsimplementert i biblioteket. Siden språket C++ benyttes skjer implementeringen på relativt lavt nivå. Dette gir brukeren stor grad av kontroll, men implementeringen kan til gjengjeld bli relativt omfattende. Antall linjer C++ kode blir lett en størrelsesorden større enn antall linjer i optimeringsmodellen. Datastrukturen er bygget opp rundt arrayer, slik at indeksering må skje ved matching av arrayer. Det er derfor ikke helt trivielt å utforme det formelle problemet basert på en generisk optimeringsmodell slik den er formulert. Har man lang erfaring med koding i C++ er det imidlertid ingen grunn til at dette skulle være spesielt avskrekkende.

Det man får igjen for en implementering i C++/Concert er god kontroll med hvordan problemet som sendes til optimeringsalgoritmen (C-PLEX) ser ut. Kallet av C-PLEX kan i stor grad detaljstyre hvordan algoritmen skal operere. Brukeren har kontroll på hvert eneste steg algoritmen tar og kan om ønskelig erstatte hele eller enkelte deler av algoritmen men en egen spesialisert algoritme. Skal man først bruke mye penger på kommersiell programvare som støttes og videreutvikles eksternt, er det åpenbart at man ikke bør bruke denne muligheten ukritisk.

9.2 OPLStudio

OPLStudio er et miljø for modellering, prototyping, implementering og debugging av blant annet optimeringsmodeller. For LP/MIP-bruk er dette miljøet strømlinjeformet mot å gi en intuitiv og enkel veg fra teoretisk modell til kjørbar modell. Det brukes et eget programmeringsspråk – OPL. ’OPL’ står for Optimisation Programming Language. Språket er så vidt vites en krysning mellom det ”gamle OPL” fra tidlig 90-tall og AMPL (A Mathematical Programming Language). Det er i noen grad videreutviklet gjennom de senere år, men kanskje ikke fullt så mye som man kunne ønsket.

OPL lar brukeren formulere modellen i kode omtrent eksakt slik den er formulert matematisk. Dette gjør det ekstremt lett å implementere en enkel matematisk modell som er ferdig utviklet til noe som kan brukes praktisk. Det gjør det også veldig enkelt å teste ut modeller som er under

utvikling frem mot en ferdig modell eller mot en prototyp applikasjon. Spesielt kan man indeksere parametre, variable og beskrankninger over nesten enhver mengde, inkludert kartesiske produkter, mengder av mengder og betingede delmengder. Implementering av en ferdig og relativt enkel modell i C++ vil gjerne være et spørsmål om én eller flere uker. Implementering av en tilsvarende modell i OPL tar typisk et par timer, og vesentlig mindre enn en dag. I praksis har man ikke alltid en helt ferdig modell når man begynner å implementere, og dermed vil det selvsagt ta lengre tid.

OPLStudio har en del svakheter ved bruk til LP/MIP:

- OPL gir ikke samme muligheter for å kontrollere løsningsalgoritmen som man får ved kall fra C++. Løsningsalgoritmen styres fra dialogbokser i OPLStudio og det kan ikke gis føringer i koden. Styringen blir dermed bare på globale parametre og kan i særdeleshet ikke relatere seg til spesifikke klasser av variabler eller beskrankninger.
- Språket har fortsatt noen barnesykdommer – spesielle formuleringer som feilsøkeren i kompilatoren reagerer på uten at det skulle være noen grunn til det.
- Lavnivåkoden som genereres fra OPL-koden er som all generert kode litt suboptimal. For svært store problemer, blir det mange runder med litt suboptimalitet i hver runde, og resultatet kan bli en ineffektiv prosedyre.

Noen av disse svakhetene er spesielt irriterende, siden det burde vært en veldig enkel sak å eliminere dem. Jeg mistenker at det brukes svært lite ressurser på å oppdatere OPLStudio. På tross av disse (til dels irriterende) svakhetene, er OPLStudio et meget kraftig implementeringsverktøy for egnede problemer, og gir helt nye muligheter for en analytiker til å gjennomføre hurtige optimeringer som del av en analyse i stedet for som et frittstående prosjekt.

OPLStudio kan brukes til utvikling av mer enn LP-modeller. Applikasjonen er trolig utviklet hovedsakelig med tanke på CP (Constraint Programming) med ILOG Solver som løsningsalgoritme, og mye av funksjonaliteten er spesialisert for å fungere sammen med denne algoritmen.

10 TOTAL ANALYSEKOSTNAD – VERKTØY, MODELLERING, IMPLEMENTERING OG ANALYSE

Hensikten med dette kapitlet er å kort beskrive de forskjellige elementene som bidrar til kostnaden til et prosjekt. Der det kan sies noe generelt om størrelsesorden på kostnaden er dette gjort. For de andre elementene omtales den konkrete kostnaden for de tre eksemplene som er presentert i de neste kapitlene.

Innenfor en prosjektramme vil valg av tilnærming til et problem blant annet være et kostnadsspørsmål. Kostnaden består av følgende elementer:

- Kompetanseoppbygging
- Kjøp av verktøy
- Modellering av problemet

- Implementering av modellen
- Analyse (datainnsamling og beregninger)

Kompetanseoppbyggingen kan veldig lett bli hovedkostnaden. En tilstrekkelig inkompetent person vil gjerne bruke flere år på å bygge opp basiskompetanse på matematisk modellering. Dersom dette skal gjøres i forbindelse med en konkret modelleringsjobb, vil han eller hun typisk modellere og implementere og forsøksvis analysere problemet flere ganger i løpet av kompetanseoppbyggingen. De fleste vil trenge minimum et par ukesverk og fortrinnsvis noen månedsverk på å bli grunnleggende kjent med LP- og MIP-teori og få grunnleggende modelleringsferdigheter. Å bli veldig flink vil alltid ta lang tid og kreve erfaring fra flere prosjekter.

Til mindre problemer og til helt enkeltstående analyser finnes det gratis eller billig modelleringsverktøy tilgjengelig. Til større problemer, eller dersom man ser for seg å bruke verktøyene flere ganger, kan avveiningen mellom dyre verktøy og lengre analysetid falle ut til verktøyenes fordel. Markedet for store optimeringspakker er tydeligvis begrenset, og de beste verktøyene er relativt dyre[3].

Modellering er det å forstå problemet og formulere en matematisk modell som representerer det. Svært komplekse problemer vil nødvendigvis ta lenger tid å modellere enn svært enkle problemer. Videre vil det ta lang tid å bygge en modell som er optimal i forhold til søketid og minneforbruk.

Modellen implementeres i et modelleringsspråk/-verktøy. Et høynivåverktøy som OPLStudio kan ta modellen omtrent som den er og bygger selv en eksekverbar modell, men med begrenset kontroll på utførelsen. Et lavere-nivå verktøy som C-PLEX lar brukeren gjøre mer av programmeringsjobben, men gir dermed også bedre kontroll på utførelsen.

I analysen skal analyseteamet gjøre den jobben som modellen er ment å understøtte. Inngangsdata til optimeringen genereres gjennom eksempelvis taktiske eller økonomiske analyser av den virkeligheten man modellerer. Til slutt skal modellen kjøres og resultatene skal analyseres, og alt dette tar tid og koster penger.

Som så ofte ellers må man gjøre avveininger. For en analyse som skal gjennomføres et lite antall ganger, og som ikke er spesielt krevende beregningsmessig, vil det være uhensiktsmessig å bruke veldig mye tid på å finpusse på modell og spesielt implementering for å øke beregningshastigheten. En iterasjon på modellering og implementering tar fort et par dager selv for mindre modeller, og en beregning er ganske stor hvis den tar like lang tid som dette. Videre må man gjøre betraktninger om hvorvidt man skal bruke tiden på å samle inn mer presise data

[3] En komplett C-PLEX utviklingslisens (som tillater koding) koster idag 17 100 Euro, og prisen har vært uforandret de siste årene. En OPLStudio lisens koster 11 400 Euro. Det selges billige undervisningslisenser, og for rene forskningsformål kan man få 30% rabatt, men det er tvilsomt om skarpe strukturanalyser kvalifiserer som forskning i denne sammenheng. En del data om applikasjonene vil være tilgjengelig på www.ilog.com, men prisinformasjon har kun vært tilgjengelig på individuell forespørsel.

eller på å gjøre en mer troverdig beregning basert på eksisterende data. Her hender det naturligvis at publikum har forskjellig toleranse for feil i datagrunnlag og feil i beregninger, og at hensynet til å få aksept for studien styrer ressursbruken.

I eksemplene i kapittel 11, 12 og 13 er det oppsummert omtrent hvor store ressurser som ble brukt på modellering, implementering og analyse. Dette er ment som eksempler på størrelsesorden av ressursbruk, og som det fremgår er det en vesentlig forskjell mellom ressursbruken til små prosjekter og store prosjekter.

11 DRR LUFTREKOGNOSERINGSMODELL

Defence Requirements Review (DRR) er NATOs forsvarsplanleggingsprosess med middels planleggingshorisont. Prosessen er delt i en kravanalyse hvor kapabilitetskrav utledes og en analyse av den mest hensiktsmessige måten å oppfylle disse kravene på.

Som en del av kravanalysen for det enkelte scenario skal det avklares hvor mange luftrekognoseringssortier som kreves per dag for å oppnå ønsket dekning over et antall vegstrekninger og områder. Hvilke områder og strekninger som skal dekkes avgjøres gjennom taktiske kartstudier. Dette er ikke noe veldig komplekst problem, og problemet hadde ikke stor synlighet, men det var blitt reist tvil om de skjønnsmessige resultatene som var lagt frem i prosessen. Derfor ville det være gunstig med et resultat som man visste at var optimalt – altså et tilstrekkelig og samtidig minimalt kapabilitetsbehov.

Modellen måtte inneholde en ruteplanlegger for hvert sortie. Det kan godt hende det finnes en god måte å modellere dette som en MIP, men noen slik modell ble ikke funnet innenfor dette prosjektet. Problemet ble modellert slik at søket ble avgrenset til realistiske løsninger bare når heltallskravet ble brukt for alt det var verdt. De kontinuerlige løsningene av denne modellen vil ha lite med det reelle problemet å gjøre, og optimeringsresultatet for den kontinuerlige modellen har svært lite med den optimale heltallsløsningen å gjøre. Modellen kunne nok vært pyntet vesentlig på med litt mer tid, men her vil det bli fokusert på erfaringene med modellen slik den var.

11.1 Generell beskrivelse av basismodell

Mengder:
Modellen starter med en mengde av tilgjengelige fly, en mengde rutesegmenter som skal flys, en mengde punkter for luft-til-luft tanking (AAR) og en mengde noder. Rutesegmentene ender i noder, og hensikten med noden er kun å være endepunkter for rutesegmenter. Flyene skal så få en rute hver som til sammen dekker alle rutesegmentene og der færrest mulig fly er brukt totalt. Flyene må etterfylle drivstoff i luften i forkant og etterkant av toktet for å kunne ta transportetappen fra base og tilbake til base. Resultatet av analysen er antall fly som er brukt. Ruten til det enkelte fly er bygget opp som en tidslinje med en gitt mengde rutesteg for å ordne rekkefølgen på flyruten og sikre at den er sammenhengende. I hvert rutesteg kan flyet dekke ett

rutesegment og det skal gjøre en transitt fra node til node (gjerne den samme). Av praktiske grunner var det to instanser av hvert rutesegment, ett for hver veg det kunne flys.

Variable:
(Variabelnavnene er de samme som de som er brukt i koden som er gjengitt som appendiks.)

- In-use variabler: for hvert fly, som beskriver hvorvidt dette flyet er brukt eller ikke. Boolsk variabel.
- Cover variabler: for hvert fly, hvert rutesteg og hvert rutesegment, som beskriver hvorvidt et fly flyr akkurat dette rutesegmentet på en bestemt plass i ruten sin. Boolsk variabel
- Transitt variabler: for hvert fly, hvert rutesteg, og hvert par av noder a og b, som beskriver hvorvidt flyet flyr i transitt fra a til b i dette rutesteget. Boolsk variabel
- Not-covered variabel: for hvert rutesegment som beskriver hvorvidt den ikke er dekket over hodet. Boolsk variabel

For å sikre at det finnes en løsning er det tatt med en slack-variabel (Not-covered variablene) med høy kostnad for rutesegmenter som ikke er dekket.

Objektfunksjon:
Objektfunksjonen er hovedsakelig summen av ”In-use-variabelen” over alle flyene samt kostnaden ved ikke å dekke et rutesegment. Noen mindre ledd er tatt med for å fokusere søket, bl a gis det en preferanse for å bruke flyene med lavest indeks før de med høyere indeks. Det gis også en preferanse for at et fly skal gjøre seg ferdig med sortiet først og deretter stå stille i stedet for å fly i transitt frem og tilbake mellom forskjellige noder. Begge disse forholdene er naturligvis uvedkommende i det egentlige problemet. I et søk som i praksis er tilfeldig, kan imidlertid algoritmen bruke nesten ubegrenset med tid på å finne ut om et fly skal hoppe frem og tilbake mellom a og b i syvende, åttende eller niende tidssteg. Siden tolkningen av objektfunksjonen er en kost-funksjon, er objektuttrykket et minimumsuttrykk.

Beskrankninger:

- Full dekning: for hvert rutesegment skal summen over alle fly og tidssteg av cover-variabelen samt av not-covered variabelen for rutesegmentet være større eller lik 1.
- Force pool control: for hvert fly, rutesteg og rutesegment skal cover variabelen for flyet, rutesteget og segmentet være mindre eller lik in-use variabelen til flyet. Flyet kan ikke gjøre noe med mindre det er i bruk.
- Unik deployering: for hvert fly og rutesteg skal summen av cover-variable over alle segmenter være mindre eller lik 1.
- Rute innenfor rekkevidde: for hvert fly skal summen av lengden av segmenter som dekkes og lenden av transitter være mindre eller lik rekkevidden til flyet.
- Konsistent og sammenhengende rute: for hvert fly, rutesteg og node skal flyet transitere til noden hvis og bare hvis den skal fly et segment som starter i noden. Summen av transittvariable for flyet *til* noden er lik summen av covervariable for segmenter ut fra noden. (Merk at flyene alltid tranisterer mellom noder, men gjerne med samme start og endenode.) Tilsvarende skal flyet transitere ut fra en node hvis og bare hvis den har

dekket et segment som slutter i noden.
- Fueling ved sortiets start og slutt: for hvert fly og for første og siste tidssteg skal summen av cover-variabelen over alle AAR punkter være større eller lik in-use variabelen.

De to siste klassene av beskrankninger tvinger inn en sammenhengende rute for heltallsløsninger. For kontinuerlige løsninger kan imidlertid flyet ha en liten konsistent rute som går slik den skal og en parallell rute som går i loop frem og tilbake over fjernere rutesegmenter. Denne feilen lar seg rette opp.

Hvis cover-variablene kan ta små verdier, slik den kan i den kontinuerlige modellen, så kan dette kompenseres med å fly samme rutesegment frem og tilbake mange ganger. Transittene mellom rutesegmenter trenger man imidlertid bare fly én gang, og kun med samme brøk som for rutesegmentene. Dermed eliminerer man i praksis transittkostnaden i den relakserte løsningen. Dette vil åpenbart gi dårlig veivisning for MIP-søket, og denne feilen ble det ikke funnet noen beskrankning som håndterte.

11.2 Utfordringer

Prosjektet hadde veldig begrenset tid tilgjengelig, og det var i særdeleshet ikke tid til å tilegne seg noen ny kompetanse. Utfordringen ble å få til en tilstrekkelig god løsning – produsere optimale resultater innenfor begrensede ressursrammer. Problemet – i hvert fall slik det ble modellert – er ikke typisk egnet for en MIP-løsning, men dette var den eneste tilnærmingen hvor vi hadde alle lisenser og nødvendig kunnskap tilgjengelig.

11.3 Prosess

Prosessen ble i sin helhet gjennomført av undertegnede med problemforklaring og oppklaringer fra Sigurd Glærum.

Problemet ble modellert og implementert i OPLStudio. Her var det nødvendig med et par iterasjoner etter overraskelser med hvor vanskelig det var å få tvunget flyrutene til å bli konsistente og sammenhengende. Spesielt var det vanskelig å unngå fritthengende løkker i ruten slik som beskrevet over. Det var også usedvanlig overraskende at OPL ikke har trigonometriske funksjoner. Dermed måtte jeg bruke en litt forenklet versjon av avstandsberegningen mellom punkter oppgitt ved koordinater basert på en spesifisert verdi for cosinus av breddegraden.

Data ble samlet inn ved kartanalyse. Det var gitt fra landdelen av scenariene hva som skulle rekognoseres. I de gitte scenariene ble dette konkretisert som rutesegmenter som skulle oppklares i forskjellige faser med identifiserte start og endepunkter. Forutsetningene i form av rutesegmenter, områdeoppklaringer og refuelingpunkter ble illustrert i kartene og siden presentert for oppdragsgiver (ACT). I prinsippet kunne de da ha kommet med endringsinnspill, men så vidt jeg vet var de såre fornøyde.

Data ble lagt i tabeller på et regneark, og for hver kjøring ble de aktuelle tabellene kopiert over i

en liten database som var laget for formålet. Det var en utfordring å huske på å bytte cosinus av breddegrad parameteren for hvert scenario, men jeg tror det ble riktig.

Kjøringene besto i at en optimering ble startet, og så fikk den gå til beste heltallsløsning lå innenfor samme heltallsintervall som grensen for den optimale løsningen – da ville denne heltallsløsningen være optimal siden objektfunksjonen nesten utelukkende er styrt av in-use variablene. Objektfunksjonsverdien for beste heltallsløsning var egentlig resultatet siden den angav antall fly som var brukt. Tallet ble skrevet ned på et ark, og beregningen ble terminert. Her var det med andre ord svært lite automatisk eksport av resultater. En kjøring tok i størrelsesorden en halv time. På dette tidspunktet lå enten beste verdi og grensen for den optimale løsningen innenfor samme heltallsintervall, eller maskinen hadde gått tom for minne.

11.4 Ressursbruk

Prosjektet ble i sin helhet gjennomført innenfor en uke, på omtrent fire dagsverk. I tillegg kom noe tid til å forklare resultatene senere. Ressursbruken fordelte seg omtrent som følger:

- Modellering: 1 dagsverk
- Implementering : 1 dagsverk
- Analyse og kjøring: 2 dagsverk

Tidsrammen var stram, og det skisserte forbruket ble oppfattet være større enn den tiden som var tilgjengelig for oppgaven.

11.5 Vurdering

Man er nødt til å se denne modellen fra to sider for å kunne vurdere prosjektet:

Som nevnt kan det godt hende at det finnes en hensiktsmessig MIP-modell for problemet. Hvis en slik finnes, er det mest sannsynlig et spørsmål om å legge til noen redundante beskrankninger som reflekterer heltallsproblemet inn i det reelle rommet. Den modellen som ble implementert i dette prosjektet hadde imidlertid som en vesentlig svakhet at den optimale heltallsløsningen og den optimale reelle løsningen var diametralt forskjellige. Mens en MIP normalt er kjennetegnet som et søkeproblem med en ekstraordinært god heuristikk, var altså dette en modell med en direkte villedende heuristikk. Dette gjør eksekveringen ekstremt ineffektiv hva angår CPU-tid og minne. På bakgrunn av dette er modellen brukt som et eksempel på et problem som ikke nødvendigvis er egnet til å implementeres som en MIP. Som beregningsprogram isolert sett, kan man altså se på denne modellen som mislykket, skjønt den til slutt finner det svaret den skal finne, og beviser optimalitet slik den skal.

Prosjektet som helhet hadde stramme tidsrammer, og handler i og for seg om et komplisert problem selv om det ikke er så stort. Når modellering, implementering og analyse tok i underkant av et ukeverk, var dette å strekke tidsrammene noe, så en mer tidkrevende tilnærming ville ikke vært aktuell. Fire dagsverk er ikke mye ressurser til et helt prosjekt av denne typen, og at det lyktes understreker styrken ved å bruke et høynivå matematisk modelleringsverktøy og å bruke kjent teori. Det må videre understrekes at selv om eksekveringen var litt smertefull for

datamaskinen, og resultatene kom ut på en hårete måte, så var det beviselig det optimale resultatet som ble funnet. Fra et prosjektperspektiv er det totalkostnaden og styrken i resultatet som teller, og slik sett var dette et svært vellykket prosjekt.

Så vidt vites har det oppstått en liten misforståelse om denne modellens fortreffelighet, og NC3A har gående en kontrakt på å implementere den eksisterende modellen uten bruk av lisensiert software (uten OPL og C-PLEX). Det må da brukes betydelige ressurser på re-implementering uten at eksekveringskvaliteten forventes å gå opp. Dette var nok ingen veldig god idé.

12 DRR AEROSPACE KVASIFULFILLMENT – OPTIMAL UTGLATTING AV STYRKEKRAV OVER FASER

I Defence Requirements Review beregnes styrkekravene for én fase av en operasjon om gangen, og alle problemstillinger knyttet til gjenbruk av styrker i senere faser skal i prinsippet håndteres i fulfillment. Behovene i den enkelte fase er dermed som hovedregel uavhengig av de andre fasene. Det er imidlertid et par viktige forhold som ikke håndteres av fulfillment-formatet, og som det er uhensiktsmessig å inkludere i fulfillment av hensyn til kompleksiteten.

- Bytte av operativ rolle for en multirolle avdeling *mellom* faser.
- Surging – avdelinger (fly-avdelinger) kan fly med høyere sortierate i begrensede tidsrom ved å kutte ned på bl a vedlikehold og hvile. Begrensningen i varighet er viktig og høyst reell.

Bytte av rolle mellom spesialiserte og mer generelle roller er håndtert i fulfillment, men for visse roller som er adskilte og uavhengige er dette et problem. Det klassiske eksempelet er et høyt luft-til-luft behov i en tidlig fase som går over til et høyt luft-til-bakke behov i en senere fase. Totalbehovet går kraftig ned ved at begge behovene støttes av de samme flyene. Surging vil likeledes redusere toppene i behovsprofilen ved at den flypakken som dekker det normale behovet ved normal sortierate også dekker kortvarige og høye behov ved temporært å øke sortieraten.

Disse to forholdene ble tatt hensyn til ved en forprosessering av kravene der deler av behovet for spesialiserte kapabiliteter erstattes av et eksplisitt behov for multirolle kapabiliteter, og der topper som kan avhjelpes ved surging flates ut.

Dette optimeringsproblemet er et rent LP-problem uten noen heltallsbeskrankninger, samtidig som det i utgangspunktet ikke er altfor stort. Dermed er det i praksis ingen grunn til å prøve å redusere kompleksiteten. Det kan hende det finnes modeller av dette problemet med færre variable enn den modellen som er brukt her, og den problemmodellen som ble funnet er relativt voluminøs hva angår variable. Det viste seg lett å gå vill i dette problemet, som ikke er fullt så lineært som det later til og derfor ikke så helt liketil å forenkle. Et grunnleggende ønske om ikke å bruke flere variable enn nødvendig var trolig grunnen til at det første forsøket gav en modell med feil. I en MIP-modell er det ofte bryet verdt å ta en del prøving og feiling og gjøre noen iterasjoner i forsøket på å slanke modellen, men det var åpenbart en dårlig idé i denne

situasjonen. Koden til OPL versjonen av applikasjonen er kompakt og relativt lesbar og kan trolig skaffes ved forespørsel til NC3A.

12.1 Generell beskrivelse av basismodell

Modellen er bygget opp som et kvasi-fulfillment. Basert på eksisterende behov allokeres det nye behov til gitte tidssteg som på samme måte som reelle styrker vil bidra til å møte de eksisterende behovene i det tidssteget de allokeres til og senere tidssteg innenfor samme deployment. For hver kapabilitetskategori som det kan beregnes behov for og for hvert tidssteg er det en allokeringsvariabel.

Videre er det for hver slik enkeltdeployering eller potensiell enkeltdeployering variable som beskriver den videre bruken av de allokerte ressursene i hvert senere tidssteg i form av rolle og surge-rate. Surge-raten er formulert som en variabel for non-surged bruk og en variabel for surged bruk innen hver rolle ressursen kan ta og for hvert tidssteg. (Antallet allokeringsvariable går altså som kvadratet av antall tidssteg, som antall kapabilitetskategorier og som antall roller for hver kapabilitetskategori.)

For å få et riktigst mulig bilde av behovene er det lagt inn en mulighet for å fjerne allokert kapabilitet (behov). Dette er lagt inn som én variabel per tidssteg og kapabilitetskategori.

Beskrankningene er bare problemets basis-beskrankninger: De opprinnelige behovene må dekkes av de nye behovene (minus reallokert behov), bruken av en allokering må ikke i noe tidssteg overstige det som er allokert, og surge-graden vektet med lengden av fasene må ikke overstige maksimal surging.

Objektfunksjonen styrer de fleste forhold rundt allokeringen.

- Allokeringskostnad: Hovedelementet i objektfunksjonen er kostnaden knyttet til allokert behov. Kostnaden varierer mellom de forskjellige kapabilitetskategoriene slik at kostnaden for en multirolle kapabilitet blir liggende mellom kostnaden på enkeltrollene og summen av kostnadene til enkeltrollene.
- Det gis en gevinst for å fjerne allokert behov. Denne er proporsjonal med allokeringskostnaden, men en størrelsesorden mindre slik at det ikke skal lønne seg å allokere et falskt behov for deretter å fjerne det.
- Det gis en kostnad på tidlig allokering. Denne er også proporsjonal med allokeringskostnaden, og må være liten også i sammenligning med besparelsen ved å allokere en multirolle kapabilitet i stedet for flere enkeltrolle kapabiliteter.

Etter at optimeringsproblemet er løst må de endelige behovene bygges opp. Disse består i de kumulative allokeringene fratrukket behov som fjernes.

12.2 Utfordringer

Dette problemet er i seg selv veldig enkelt å løse rent beregningsmessig. Vi hadde imidlertid

egentlig ikke tid til å gjøre det på det tidspunktet det måtte gjøres, og derfor ble det en utfordring å gjøre det helt riktig på aller første forsøk, eller å få noen andre til å overta utviklingen. Dette lyktes ikke. En kollega testet ut den første modellen, han brukte relativt mye tid på det (minst et ukeverk), fant at den foretok feilallokeringer, og jeg måtte deretter endre modellen.

Siden hele optimeringen er kontrollert av kost-koeffisientene i objektfunksjonen er det en utfordring (om enn ganske overkommelig) å tune modellen riktig. Her kan man enten gå på de fundamentale kostnadene bak problemet og derfra utlede korrekte kostnadsfaktorer. Man kan identifisere de typene feilprioriteringer man ønsker å unngå og formelt utlede koeffisienter som umuliggjør disse eller man kan ta denne siste prosessen mer omtrentlig og teste ut om modellen oppfører seg som ønsket (tuning). Vi valgte den siste tilnærmingen her, og siden problemet er såpass oversiktlig, fungerte det som det skulle på første forsøk.

De problemene som oppsto i modelleringen av problemet kan i noen grad tilskrives et innebygd ønske om å holde problemkompleksiteten (antall variable) nede. Dermed ble et relativt enkelt problem formulert unødvendig knotete. Det å holde kompleksiteten nede i store problemer er helt essensielt, men det kan være en interessant lærdom at man bør tilstrebe å frigjøre seg fra dette ønsket der kompleksitet ikke er noe problem.

12.3 Prosess

Prosessen ble som vanlig preget av dårlig tid, og i dette tilfellet slo det uheldig ut. Problemstillingen var relativt velkjent, selv om det måtte noen avklaringer til. Undertegnede laget et første utkast til modell som virket tilforlatelig, og implementerte denne med en liten assosiert database for inngangsdata og resultater. Dette utkastet ble gitt videre til kolleger for testing og integrasjon i J-DARTS.

Imidlertid var det en feil i modellen, og det tok vesentlige testingsressurser å få fastslått dette. Mye av denne tiden kan nok føres som opplæring, og det kan derfor tenkes at det var vel anvendt tid, men fra prosjektet isolert sett virket det som en stor kostnad.

Etter dette ble modellen endret – den fikk enklere logikk men flere variabler. Den ble testet igjen, og fungerte nå som forutsatt. Den ble så integrert i D-FARM (behovsberegningsbiten av J-DARTS).

12.4 Ressursbruk

Prosjektet ble gjennomført over et lengre tidsrom. Den første varianten av modellen (med feil) ble laget i løpet av to dager. En tid senere ble den tatt opp igjen og testet/forsøkt tunet. I tilknytning til dette ble den remodellert og implementert, og til slutt integrert i J-DARTS. Ressursbruken var omtrent som følger:

- Initiell modellering ½ dagsverk
- Initiell implementering og funksjonstesting: 1 dagsverk
- Testing/tuning: 1-2 ukesverk

- Remodellering/re-implementering ½ dagsverk
- Generell frustrasjon noen dagsverk

Som nevnt kan trolig en del av testingen regnes som opplæring. Oppsettet omfatter ikke implementeringen av kall av modellen fra D-FARM.

12.5 Vurdering

Aerospace kvasifulfillment er et rent lineært problem. Som et lineært optimeringsproblem er det relativt lite og helt uproblematisk å løse. Lineærprogrammering er derfor åpenbart måten å tilnærme seg dette problemet. Med en korrekt tunet modell vet man at den løsningen som blir produsert oppfyller de optimalitetskravene man har formulert, og dette gir en helt annen trygghet rundt resultatet enn en konstruktiv tilnærming, der man ikke vet eksakt hva forholdet er mellom de kriteriene som er satt opp og den løsningen som konstrueres.

Dersom det hadde vært tid tilgjengelig til å gjennomføre dette prosjektet på en hensiktsmessig måte, ville det vært relativt uproblematisk å modellere, implementere og teste modellen på ett ukeverk. Selve beregningene utføres som en del av hovedberegningene for operative krav under DRR, og analysen kan vanskelig sees som et selvstendig arbeid som det kan regnes et tidsforbruk på.

Dette problemet utmerker seg ved å være enkelt beregningsmessig, men litt mer komplisert modelleringsmessig enn det ser ut til ved første øyekast. Det måtte også gås to runder på modelleringen før vi hadde en modell som fungerte. I en slik situasjon er det en stor fordel å kunne programmere i et språk som OPL, der man nærmest kan skrive modellen direkte inn som kode. Man får full uttelling for den hurtige prototype- og implementeringsfunksjonaliteten, mens konsekvensene på eksekveringstid er minimale. Det må sies at det er relativt lite av funksjonaliteten i OPL/C-PLEX som brukes i et rent lineært problem.

13 DRR FULFILLMENT

DRR er NATO's forsvarsplanlegging på mellomlang tidshorisont, typisk et halvt materiellomløp[4]. Analyseprosessen er organisert som hva man kan kalle kapabilitetsbasert planlegging med arv[5]. Det beregnes generiske kapabilitetskrav, og deretter ser man på måter å fylle disse behovene på. Så langt er det vel og bra, men siden planhorisonten er vesentlig mindre enn omløpstiden til en struktur, må man ta hensyn til eksisterende styrker og forsøke å møte styrkebehovene med dem. Disse styrkene har individuelle kapabiliteter og forskjellige

[4] Tidshorisonten på DRR var tidligere 8 år. Den har formelt blitt utvidet noe, men fokus forblir på mellomlang tidshorisont.

[5] Kapabilitetsbasert planlegging er en prosess hvor man først utleder kvantitative og kvalitative behov for abstrakte (generiske) kapabiliteter basert på strategisk målsetting og deretter ser på forskjellige måter å levere disse kapabilitetene (alternative strukturer). Dette står i motsetning til en prosess der man ser direkte på strukturer opp mot strategisk målsetting. Har man arv, må denne tas hensyn til i prosessen hvor det velges en tilnærming til å oppfylle behovene.

kapabiliteter avhengig av hvilken rolle de brukes i. Styrkene henger også sammen, slik at det er begrensninger på hvordan man kan stykke dem opp, smøre dem utover og blande dem. Dermed blir dette et meget stort heltallsproblem.

Prosjektet hadde meget stor synlighet, og i startfasen var det styrt av behovet for raskt å demonstrere gjennomførbarhet, samtidig som kompetanse måtte bygges opp i denne fasen.

13.1 Generell beskrivelse av basismodell

MIP-modellen er sterkt preget av hvilke aspekter det fremsto som viktig å få representert korrekt for å få aksept for å bruke en optimeringstilnærming. Den er også preget av å skulle håndtere mange delproblemer med forskjellige karakteristika. Uansett hvilket enkeltstående delproblem man tar utgangspunkt i, vil modellen kunne fremstå som unødig konkret, som ikke å ha utnyttet spesielle egenskaper ved delproblemet. I sum nødvendiggjorde imidlertid alle delproblemene denne eksplisitte modellen. Fremstillingen her tar bare for seg det mest sentrale enkeltelementet ved modellen – selve allokeringen og oppfyllingen av styrkebehov.

Fulfillment består i å matche to sider:

1. Kapabilitetskrav for å møte alliansens ”Level of Ambition” (LoA)
2. Styrker (militære enheter) med individuelle kapabiliteter

Resultatet av analysen er en pool av styrker (enheter) som er tilstrekkelig til å møte LoA

Sentrale Mengder og parametre

Alliansens ambisjonsnivå er uttrykt gjennom forskjellige typer av operasjoner som skal kunne gjennomføres samtidig og i varierende antall. Operasjonstypene er igjen representert gjennom konkrete scenarier som er med på å dimensjonere de totale styrkekravene. I fulfillment konstrueres de lovlige kombinasjonene av operasjoner eksplisitt til ”benchmark combinations”, og det er kravene i disse kombinasjonene av scenarier som skal oppfylles. De oppfylles ved at enheter deployeres til et scenario på et konkret tidspunkt innenfor en ”benchmark combination”, og forutsettes å bli i teateret en viss tid før de roterer tilbake til hjemlandet og erstattes av nye enheter. De enhetene som har forlatt teatret roteres tilbake en gitt tid senere.

Alle kapabiliteter og kapabilitetskrav er uttrykt gjennom omkring 300 ”kapabilitetskategorier”. En kapabilitetskategori representerer en *type* kapabilitet, og *mengden* av denne kapabiliteten er uttrykt ved et tall. Mengdemålet på kapabilitet innenfor én og samme kapabilitetskategori er additivt. Det er også innbyrdes relasjoner mellom kapabilitetskategorier, slik at en kapabilitet innenfor en avansert kategori kan møte et behov innenfor en liknende men mindre avansert kategori. For eksempel kan langtrekkende fly av en viss type utføre oppgaver som ikke eksplisitt krever den ekstra rekkevidden. Disse avhengighetene er håndtert i formateringen forut for selve optimeringen.

Kravsiden består altså av et antall ”benchmark combinations”. Hver ”benchmark combination” består av et antall scenarier (planning situations) og hvert scenario har et antall tidssteg (faser) med en assosiert tid. For hver fase av en operasjon vil det være kapabilitetskrav for hver

kapabilitetskategori.

Ressurssiden består av en mengde av avdelinger (styrker). En avdeling kan ta forskjellige alternative roller og dermed få forskjellige kapabiliteter. Dette er representert gjennom instanser av avdelingen, og for hver enhet finnes det en mengde av instanser. En enhet er dermed en paraply over flere instanser, og de fleste egenskapene (for eksempel kapabiliteter og rotasjonsmønster) er knyttet til instansnivået. En instans av en enhet har kapabiliteter under en eller flere kapabilitetskategorier, og dette representerer kapabiliteter som avdelingen kan ha samtidig.

Variable:
Problemet over gir to viktige klasser av variable:

- Allokeringsvariable – for hver instans av hver enhet til hvert tidssteg av hvert scenario i hver benchmark combination. Variabelen beskriver hvorvidt denne konkrete allokeringen finner sted.
- Force-pool variable – én for hver enhet. Variabelen beskriver hvorvidt en avdeling er benyttet og altså må være i styrkepoolen eller ikke.

De fleste av disse variablene er boolske heltallsvariable, men dersom det vurderes at en konkret avdeling kan deployeres i småbiter, settes allokeringsvariablene for denne enheten til enten å kunne ta reelle verdier eller de kan ta et endelig antall verdier mellom 0 og 1. Force-pool variablene er alltid boolske. Det må nevnes at fulfillment modellen i sin helhet har vesentlig flere variabelklasser enn dette.

Objektfunksjon
Objektfunksjonen i den enkleste versjonen er summen av force-pool variablene vektet med prisen på hver enkelt enhet, som skal være minst mulig. Objektuttrykket er altså et minimumsuttrykk. I en mer komplett modell er objektfunksjonen hele kostnadsbildet som omfatter kostnaden ved å ha styrker tilgjengelig i poolen, kostnaden ved å bruke styrkene, kostnaden ved å ha høy beredskap på en avdeling, kostnaden ved å kvitte seg med en eksisterende enhet og i stedet bygge opp en ny, kostnaden ved ikke å oppfylle behov og så videre.

Beskrankninger
De tre første beskrankningene under er sentrale i grunnmodellen. To til er tatt med for illustrasjonsformål:

- **Fulfillment:** For hvert behov – indeksert ved benchmark, scenario, tidssteg og kapabilitetskategori – må summen av allokerte kapabiliteter som bidrar være større enn eller lik behovet. Også enheter som er allokert til et tidligere tidssteg men som fortsatt vil være i teateret i henhold til rotasjonsmønsteret vil bidra med kapabilitet.
- **Unik deployering:** For hver avdeling og hver benchmark skal summen av allokeringsvariablene over alle instanser av enheten og alle tidssteg, scenarier og faser innenfor benchmark kombinasjonen være mindre enn eller lik 1.
- **Forcepool:** For hver avdeling og for hver allokeringsvariabel assosiert til den avdelingen

skal force-poolvariabelen til avdelingen være større enn eller lik allokeringsvariabelen.

- **Beredskap (readiness):** For hver avdeling og tidssteg (og scenario og benchmark) der avdelingens klargjøringstid (beredskap) og reisetid er større enn tidskravet på tidssteget, skal allokeringsvariabelen være lik 0. (Variabel beredskap er en enkeltfaktor som bidrar sterkt til en kompleks modell).
- **Deployerbarhet:** Deployerbarhet er en avdelings evne til å bli deployert og brukt i operasjoner utenfor hjemlandet. For alle allokeringsvariable der styrkens hjemlokalisering og scenariets lokalisering ikke er den samme, skal allokeringsvariabelen være mindre enn eller lik styrkens deployerbarhetvariabel (boolsk variabel).

13.2 Utfordringer

Dette prosjektet var på grensen av det som kan håndteres med tilgjengelig teknologi, samtidig som modellen skulle tilfredsstille strenge krav fra mange brukere, og hadde meget høy synlighet. Tidsrammene og ressursrammene var størrelsesordener større enn for de to små prosjektene som er beskrevet kapittel 11og 12, men spesielt tidsrammene fremsto likevel som svært krevende.

Arbeidet ble påbegynt i Windows, og tidsrammene tillot aldri skifte av plattform. Windows hadde en absolutt grense på 2 GB minne per applikasjon inntil 64-bit versjonene kom i ferdig utgave. I praksis klappet applikasjonene gjerne sammen når forbrukt minne lå mellom 1,3 GB og 1,7 GB. Dette gav begrenset spillerom for store applikasjoner. Dermed kom alle problemene som er forbundet med en stor beregning inn for fullt. Antall variable er bare én av flere faktorer som gjør et problem vanskelig å løse, men det gir en indikasjon på problemstørrelse. Så vidt huskes lyktes det å håndtere delproblemer når de hadde rundt 250 000 heltallsvariable og en del flere kontinuerlige variable. Et delproblem med rundt 350 000 heltallsvariable hadde stor sannsynlighet for å krasje.

Det totale problemet måtte stykkes opp i bolker av benchmarks og i segmenter av kapabilitetskategorier for å la seg løse. Den optimale løsningen på hver av disse delproblemene er imidlertid i større eller mindre grad avhengig av de andre, og her var den konkrete oppdelingen i ”bolker og segmenter” viktig. Siden løsningen av hvert delproblem tar løsningen fra tidligere beregnede delproblemer som inngangsverdi, var også rekkefølgen på kjøring av de forskjellige delproblemene viktig.

Det var en stor utfordring å lage en modell som tok høyde for alle de forholdene som forskjellige brukere ønsket representert uten å miste kontroll på kompleksiteten. Det var mulig å få aksept for å forenkle en faktor dersom det kunne påvises at en fullstendig representasjon ville umuliggjøre hele prosjektet. Det var imidlertid lite gehør for mindre dramatiske argumenter. Hver gang en forbedring ga litt albuerom, glapp dette argumentet, og dermed endte problemet alltid med å vokse til man stanget inn i grensen for det håndterbare.

Forprosesseringen av informasjonen (oppbyggingen av det formelle problemet) omfattet mange

steg som besto i å knytte sammen data, for eksempel gjennom tabelloppslag (søk gjennom datalister). Med store datasett er det viktig å være nøye med hvordan denne sammenkoblingen gjennomføres: Hvis elementene i mengde A inneholder pekere til mengde B, og man ønsker å ende opp med en peker andre vegen, vil man for små mengder gjerne ta den trygge varianten og søke gjennom A for hvert element i B. Å gå gjennom A og plassere motsatte pekere etter hvert som de dukker opp krever noe mer kontroll, men mens den første operasjonen er kvadratisk i størrelsen på mengdene er den andre lineær. Med et par nøstede oppslag av denne typen kan retningen på oppslaget bety en endring fra et sekund til noen dager i prosesseringstid. Slike søk ligger gjerne implisitt i et programmeringsspråk – f eks OPL eller SQL i dette tilfellet. Å bygge opp en algoritme som klarte å bygge opp alle de nødvendige indeksmengdene på rimelig tid og med rimelig forbruk av minne tok mange iterasjoner.

Modellen hadde problemer med å velge optimalt eller endog tilnærmet optimalt når svært mange variable var linket sammen slik som beskrevet i kapittel 8.2. Forsøk med å legge prioriterte ”ceiling-variable” til å kontrollere disse variablene gav relativt gode resultater, men det vites ikke om dette ble implementert på alle områder i den ferdige modellen.

Prosessen som skulle støttes av modellen forutsatte at man i løpet av en ettermiddag og natt kunne sette opp en kjøring med justerte inngangsdata, kjøre optimeringen, kontrollere løsningen og forberede data for presentasjon. Optimeringen tok store deler av natten når den lyktes, og var i tillegg ustabil grunnet minnebegrensningene. Det var relativt krevende å få gjennomført denne prosessen, og det ble løst ganske mange verdensproblemer i sene nattetimer mens de mest trofaste hyrdene satt nattevakt over sin modell.

13.3 Prosess

Prosjektet tok til i desember 2001 etter at undertegnede begynte ved NC3A. Kjerneteamet omfattet også Sigurd Glærum. Prosjektet ble avsluttet med bruk ved fulfillment i januar 2003. Dette er en historie om et stresset utviklingsprosjekt, og her bør det tas med noen høydepunkter. Ved prosjektstart hadde vi liten eller ingen kunnskap om LP/MIP, eller andre konkrete optimeringsteknikker og verktøy, og vi måtte demonstrere gjennomførbarhet i løpet av et par måneder. Tilnærmingen ble mye styrt av hva som var lot seg gjøre til enhver tid, og hva vi hadde kunnskap om. En prinsipiell modell med kapabilitetskategorier og allokering av enheter til tidssteg ble formulert og vi så etter måter å implementere dette på.

Etter litt famling med ikke-lineær optimering og constraint programming (CP), viste en ansatt i ILOG sitt støtte- og salgsapparat oss hvordan et allokeringsproblem kan formuleres som en MIP. Resultatet var at vi anskaffet ILOG sine produkter C-PLEX og OPLStudio og gikk videre med MIP som tilnærming til problemet. Dette var hva man kaller et vellykket salgsfremstøt. Det var aldri anledning til noen større utprøving av alternativer, men i ettertid har de produktene vi valgte vist seg å være gode valg.

Det er mulig å uttrykke mange meningsløsheter i OPL, og prøve- og feilemetoden til å lære seg språket er ganske smertefullt. Etter flere ukers prøving og feiling og skrittvis utbygging hadde vi

en modell som begynte å løse reelle fulfillment problemer. Det var på dette tidspunktet vi oppdaget minnebegrensningen til Windows, for offisielt oppgav Microsoft at Win XP kunne håndtere 4 GB minne. Det tok nesten to timer sammenhengende på telefonen mot Microsofts supportapparat for å finne noen som ville bekrefte at grensen på 2 GB per applikasjon var uforandret.

Prototypen ble forbedret iterativt parallelt med at en ny implementering ble satt ut på kontrakt. Fra sommeren og utover høsten og vinteren ble både optimeringsverktøyet og databaseapplikasjonen som holdt alle dataene, utviklet eksternt. I parallell ble analysene av nasjonale enheter og behov i scenariene gjennomført internt. Når man hyrer andre enn seg selv til å gjøre en så avgjørende jobb, skal man vite at de er tilstrekkelig dedikerte til å føre oppgaven helt frem. Dette har vi skjønt nå. Den konsulenten som gjorde optimeringsjobben var i og for seg meget flink, men manglet i noen grad viljen til å sjekke og dobbeltsjekke at applikasjonen gjorde den jobben den skulle gjøre.

Da alle grunnlagsdata skulle være på plass i modellen tidlig i desember var det ingenting som fungerte. Det var alvorlige feil både i databaseapplikasjonen og i optimeringsmodellen. Prosessen fortsatte med heftig debugging helt til analysen var avsluttet. Sigurd må berømmes spesielt for å komme innom kontoret nesten daglig gjennom hele jule- og nyttårshelgen for å se til hvordan testkjøringene gikk.

Prosessen som skulle støttes besto i følgende:

- Generiske styrkebehov var beregnet, vurdert og akseptert av ”DRR Working Group” i løpet av høsten.
- Alle nasjonale enheter som kunne brukes ble vurdert i forhold til kvalitative og kvantitative kapabiliteter. Dette skjedde også i løpet av høsten og forvinteren og med støtte fra ”Force Planners” i SACLANT og SHAPE . Disse er senere reorganisert til Allied Command Transformation (ACT) og Allied Command Operations (ACO) slik at forsvarsplanlegging ligger under ACT.
- Dataene ble importert inn i en database (Defence Planning Database Tool)
- Fulfillment skulle gjennomføres i løpet av et ukelangt møte (fulfillment exercise) med ”Force Planners” med flere fulfillment iterasjoner.

Fulfillment exercise besto av følgende

- Deltagerne får seg hver dag forelagt resultatene fra et fulfillment
- De tar stilling til endringer de ønsker i forhold til dette med fokus på:
    - Avdelinger de ikke vil se i force-poolen
    - Avdelinger som de ønsker å se i force-poolen
    - Generelle preferanser for visse avdelingskategorier foran andre
- Innspillene implementeres i datafilene og legges inn i databasen
    - Preferanser som styrer koeffisientene i objektfunksjonen
    - Enheter merkes for at de ikke skal vurderes for fulfillment, og disse tas ut av enhetsmengden i modellen.
    - På områder der alle var fornøyd med løsningen ble det lagt en høy kostnad på

endringer fra dette

- Optimeringsapplikasjonen blir kjørt og overvåket av de mest trofaste.
- Neste morgen kontrolleres løsningen, resultatene eksporteres til regneark og klargjøres for presentasjon

I hver iterasjon ble informasjonen tatt inn og ut mellom databasen og regneark flere ganger.

13.4 Ressursbruk

Inklusive kompetanseoppbygging, utvikling av prototyp og oppfølging av videre utvikling tok optimeringsapplikasjonen omkring to tredels årsverk internt. I tillegg kom ca et kvart årsverk hos ekstern utvikler. Totalt kostet altså prosjektet rett i underkant av et årsverk, men dette var nok et ganske langt årsverk. Gjennomføringen inkluderer ikke beregning av generiske behov og styrkekapasiteter som er inngangsverdier til optimeringen. Utvikling av databaseverktøyet kommer også i tillegg, og var ganske ressurskrevende. Hvis vi skjønnsmessig skal fordele de åtte månedsverkene som gikk med internt, kan dette være omtrent som følger:

- 2 månedsverk på kompetanseoppbygging
- 1 drøyt månedsverk på utvikling av prototyp
- 2 månedsverk på oppfølging av kontrakt og konsulent
- 3 månedsverk på debugging og kjøring

13.5 Vurdering

Prosjektet var stort og krevende, og slik det til slutt ble formulert var det på grensen av det vi kunne løse. Rammebetingelsene var at man for DRR 03 hadde tatt mål av seg til å gjennomføre et felles fulfillment for alle forsvarsgrener. Dette er en oppgave som er størrelsesordener mer kompleks enn fulfillment innenfor enkeltområder – modellen kan ikke utnytte forenklinger som er områdespesifikke. Totalt antall ”benchmark combinations” som måtte analyseres gikk opp en faktor mellom fem og ti, og antall enheter og behov innen hver kombinasjon økte en faktor tre. Dette krevde en helt ny tilnærming og nye verktøy, og de måtte være på plass i løpet av et drøyt år. Det er lett å se at utviklingen av et optimeringsverktøy var en risikabel prosess, men det er uvisst om det fantes et trygt alternativ til dette – altså en prosess som med sikkerhet ville ha møtt behovet innenfor tidsrammene.

Det er liten tvil om at problemet er linariserbart gjennom diskretisering, og modellen som ble brukt er en rimelig korrekt modell av beslutningsproblemet. Slik sett oppnådde man hovedhensikten med prosjektet: Modellen fant løsninger som oppfylte beskrankningene. Den fant dem innenfor rimelig tid, og den gav informasjon (i form av en øvre grense) om hvor langt løsningen kunne være fra et optimum. Videre støttet modellen den iterative tilnærmingen på en god måte ved at oppdaterte prioriteringer og ønsker fra beslutningstakerne enkelt kunne representeres i modellen.

Modellen i seg selv hadde altså høy kredibilitet, men den var svært kompleks, og gav derfor ikke nødvendigvis de best mulige løsningene. I 03 DRR var den også implementert i OPLStudio, og den var åpenbart en større modell enn det OPL er designet for å håndtere. Når

dette er sagt, må det også sies at modellering i OPLStudio – selv om det var å strekke denne programpakken vel langt – faktisk muliggjorde prosjektet. Bruk av en COTS-løsningsalgoritme som C-PLEX fungerte godt. Gitt forutsetningene er det lite trolig at prosjektet kunne vært gjennomført uten implementering i OPL, eller uten en god kommersiell løsningsalgoritme.

Prosjektforløpet var i stor grad drevet av at man fulgte eneste tilgjengelige veg heller enn av et valg av optimalt prosjektforløp, og dette var definitivt ubekvemt. Hadde det vært mer tid (og eventuelt ressurser) tilgjengelig, ville det vært gunstig å bruke mer tid på evaluering av alternative implementeringsformer, ha alle verktøy ferdig lenge før bruken skulle ta til og implementere og gjennomteste den applikasjonen som skulle brukes. En slik mulighet virket ikke å være til stede.

En alternativ tilnærming bør nevnes for å kunne vurdere hvor vellykket bruken av MIP var: Fulfillment problemet kunne trolig vært løst rent teknisk ved en konstruktiv regelbasert tilnærming. Allokeringene gjøres da mekanistisk ut fra et regelsett, og slik konstruerer man én konkret løsning som blir det endelige resultatet. DSTL har en applikasjon kalt ”Sabrina” som adresserer et liknende problem, og som trolig fungerer omtrent på denne måten. Eksekveringen av en slik modell er selvfølgelig veldig trygg – kompleksiteten er omtrent lineær i antall enheter, og derfor finner den en løsning. Regelbasert programmering er imidlertid ikke helt trivielt, i praksis trolig omtrent så komplisert som LP- eller MIP-implementering. Videre er oppbyggingen av en troverdig regelbase for et generisk allokeringsproblem en krevende oppgave. Et svært viktig poeng er at en slik tilnærming, hvor effektiv den enn er, ikke har det samme teoretiske fundamentet som en MIP-modell, og derfor ikke vil gi løsninger med samme kredibilitet. Så lenge det grunnleggende beslutningsproblemet er formulert gjennom et mål og ikke gjennom regler, er det trolig ingen alternative modelleringsformater som kunne ha vært implementert tilfresstilende innenfor rammene av prosjektet.

Som man vil se av fremstillingen her, var utviklingen av fulfillment optimereren en smertefull prosess for de involverte og også for resten av DRR. Slik har krevende analyseprosjekter alltid vært, og avanserte teknikker som gjør det mulig å gjennomføre mer på kort tid vil alltid føre til at problemstillingen i prosjektene blir enda mer krevende. Derfor er det grunn til å tro at krevende prosjekter også i fremtiden vil være like krevende. Vi vil neppe komme i en situasjon hvor teknologiske forbedringer resulterer i en uendret prosess som gjennomføres på en behagelig måte med gode marginer. Som en oppsummering var bruken av MIP til fulfillment en vellykket tilnærming, men man kan trekke noen viktige erfaringer:

- Selv om utvikling av en demoversjon av modellen går raskt, er det likevel krevende å implementere en ferdig versjon.
- Kostnaden ved kompetanseoverføring mellom analytiker og eksterne programmerere kan være stor og må ikke undervurderes.
- En modell har en sterk evne til å vokse inntil den åpenbart blir uhåndterbar – man bør gjøre hva man kan for å prøve å stoppe veksten før dette punktet, men dette kan være vanskelig.

14 ERFARINGER OG EGNETHET

De tre eksemplene som er beskrevet i rapporten representerer tre forskjellige situasjoner:

- Små egnede problemer – ingen utfordringer mht kompleksitet og ingen modellmessige problemer (Aerospace kvasifulfillment).
- Små problemer som ikke er spesielt egnet for MIP-modellering, men hvor dette likevel er en mulighet (luftrekognoseringsmodellen).
- Store prosjekter som er egnet for bruk av metoden men som tøyer grensen for det teknisk gjennomførbare mht kompleksitet (DRR fulfillment).

Som man ser mangler kombinasjonen stort problem og uegnet problem, men konklusjonen i det tilfellet ville nok vært gitt.

14.1 Små egnede problemer

For små egnede problemer, enten de er rent lineære problemer eller ”gode” MIPer, kan man generelt si at en LP/MIP-tilnærming er uovertruffen. Med ”god MIP” menes et problem der det er stor grad av samsvar mellom den optimale reelle løsningen og den optimale heltallsløsningen. Samsvaret trenger ikke bety at variabelverdiene er like, men at MIP-problemet og MIP-løsningen er en heltallsvariant av LP-problemet og LP-løsningen.

I en situasjon hvor det grunnleggende problemet er godt forstått – hvor man ikke trenger dra ut i felten for å finne ut hvordan det grunnleggende systemet oppfører seg – kan et lite og egnet problem typisk modelleres og implementeres i et høynivåverktøy i løpet av 2-3 dager. Dersom den samme beregningen senere skal gjennomføres svært mange ganger og eventuelt på mange forskjellige steder, kan man etterpå velge å implementere den på lavere nivå, primært for å spare lisenskostnader.

Et lite problem i dag er ikke det samme som et lite problem for ti år siden. Et lineært problem med noen hundre tusen reelle variable eller et egnet MIP-problem med inntil tusen heltallsvariable er i dag et lite problem – et problem der den tekniske eksekveringen ikke tar lang tid og ikke er spesielt krevende.

Et egnet problem er per definisjon kjennetegnet ved at den lineære modellen er en riktig modell av hovedproblemet, selv om den mangler heltallsbeskrankningene. Formulering av en LP/MIP-modell vil derfor gi en god forståelse for problemet. Samtidig vil resultatet bli få høy kredibilitet sammenlignet med et resultat konstruert ut fra en regeldatabase eller liknende.

I implementeringen av denne typen problemer spiller høynivåverktøyene en spesiell rolle. Fordi implementeringen er enkel og ligger så nært den matematiske modellen, er det naturlig å se hele softwareutviklingen som en del av analysen. i OA prosjekter der softwareutviklingen utgjør en større del av arbeidsmengden og krever mer spesialisert programmerings kompetanse kan man velge en av to tilnærminger:

- Man kan identifisere en problemstilling, sette folk til å modellere fenomenet og nye

mennesker til å implementere modellen, før man har gjør analyser støttet av denne modellen. Dette gir høye kostnader i kompetanseoverføring og i å sikre konsistens mellom problem, modell, implementering og bruk av det ferdig implementerte verktøyet.

- Alternativt kan man la det samme personellet gjennomføre alle disse stegene. I så fall vil man gjerne ta betydelige opplæringskostnader for at kompetansen til det aktuelle personellet skal dekke hele dette spekteret. Man risikerer fortsatt at det ikke ligger nok spesialisert kompetanse bak hvert steg.

Det tradisjonelle unntaket har vært regnearkmodeller som gjerne har vært analytikerens egne modeller. Med et verktøy som OPLStudio, kan analytikeren selv implementere en modell for et egnet problem når hun har identifisert problemstillingen – på samme måte som i et regneark, men for vesentlig mer avanserte beregninger enn det som et regneark kan gjennomføre. Ifølge en ansatt i ILOG var dette den opprinnelige tanken bak OPLStudio – å gi en tilstrekkelig skolert beslutningstaker muligheten til å gjennomføre analysene selv på sitt eget kontor og når han trenger dem. Det er vel ikke sikkert dette er helt sant - prototyping for utviklere var trolig også en viktig del av motivasjonen bak programpakken.

14.2 Små men mindre egnede problemer

Generelt skal man alltid så langt mulig bruke den best egnede eller hensiktsmessige tilnærmingen til et problem. Hva som er mulig eller hensiktsmessig vil imidlertid bestemmes av hvor store ressurser som er til rådighet i form av kompetanse, tid, software og penger. Hvis ressurssituasjonen gjør det mulig å gjennomføre en analyse med en bestemt tilnærming, kan dette ut fra situasjonen være den best egnede tilnærmingen, selv om enkeltfaktorer som matematisk kompleksitet og eksekveringstid skulle tilsi en annen løsning. I mange analyseprosjekter er eksekveringstid en forsvinnende liten del av totalt tidsforbruk.

Luftrekognoseringsproblemet er en variant av det velkjente ”travelling salesman”-problemet. Implementeringen av luftrekognoseringsoptimereren er et eksempel på en vellykket bruk av MIP-modellering på et problem som kunne vært beregnet mer effektivt med forskjellige teknikker utviklet for ”travelling salesman”-problemer. En del av ressursbildet var at vi kjente verktøyene for høynivå implementering av MIP-problemer, problemet kunne modelleres korrekt på denne måten og gi faglig holdbare resultater. Selve eksekveringen ble imidlertid en rufsete affære – lite egnet for demonstrasjon for kunder eller prosjektråd. Enhver tilnærming som forutsatte opplæring i nye metoder ville sprengt ressursrammene med en størrelsesorden (- til opplæring uten veiledning i et nytt område må det normalt beregnes minst et månedsverk).

I hovedsak kan det antas at dersom man skal hente inn spesialkompetanse innenfor et modelleringsområde, så kan det være gunstig å se på beregningsmessig effektivitet og eleganse i eksekveringen. Dersom den eller de som forstår analyseproblemet kan gjennomføre modellering, implementering og analyse selv, så vil gevinsten ved å bruke kjent kunnskap langt overstige imperfeksjoner i eleganse og eksekveringstid så lenge tilnærmingen holder faglig mål. LP/MIP brukt på lineariserbare problemer har den styrken at det alltid gir faglig holdbare resultater eller i det minste gir veldig harde data på hvor holdbare resultatene er

(optimalitetsgap).

14.3 Store problemer

Uansett hvor langt teknologien tar oss, vil det være et behov eller ønske om å løse problemer som til enhver tid er i grenseland for hva som kan håndteres. Det er trolig ingen grense for dette: Økonomiske modeller og allokeringsmodeller er blant de viktigste LP/MIP-problemkategoriene, og disse problemene kjennetegnes ved at man ønsker å løse dem globalt og på høyt nivå samtidig som man ønsker optimalitet også på små delproblemer. Dette gjelder enten den detaljerte løsningen oppfattes som del av beslutningsproblemet eller man kun ønsker at den overordnede løsningen skal være konsistent med en optimalitetsantagelse lavt nivå.

Som eksempel vil en samfunnsøkonomisk modell minst måtte se på en nasjon som system, samtidig som de beslutningene som modelleres normalt tas på enkeltpersons- eller på enkelthusstandsnivå. Det er ikke spesielt mye vanskeligere å modellere et stort system enn et lite system, og siden data på enkelthusstander i stor grad finnes i offentlige systemer, ville det vært naturlig å modellere enkelte samfunnsmessige analyse- og beslutningsproblemer ned til veldig lavt nivå dersom det hadde vært mulig å løse et slikt problem. Dette er i liten grad mulig i dag, i hvert fall hvis vi snakker om diskret valg modeller, men ser vi noen teknologigenerasjoner frem i tid kan dette være problemer på grensen til det løsbare. Det er trolig at denne teknologiske muligheten vil bli brukt til å løse stadig større problemer like mye som den vil bli brukt til å løse eksisterende problemer mer komfortabelt.

For LP-modeller (rent lineære modeller uten heltallsbeskrankninger) finnes det få begrensninger. Hvis computeren kan håndtere problemet med hensyn til minne, så finner man en løsning i lineær tid (i antall variable), og denne løsningen er optimal. Hvis det finnes en korrekt og lineær modell av et problem så vil det omtrent aldri være noen annen tilnærming som kan konkurrere i håndterbar problemstørrelse. Problemene er knyttet til søk i heltallsproblemer. For store heltallsproblemer på grensen til det håndterbare vil løsningskvaliteten gå ned – man finner løsninger innenfor beskrankningene, men optimalitetsgapet kan være stort. (Optimalitetsgapet er den prosentvise differansen mellom objektfunksjonen til beste identifiserte heltallsløsning og beste relakserte løsning i de ikke-ekspanderte nodene i søketreet). Samtidig går søketid og minneforbruk opp mot grensen for det som er akseptabelt innenfor formålet med analysen.

Dersom man har en tilstrekkelig hensiktsmessig MIP-modell (med godt samsvar mellom beskrankningene på det lineære problemet og heltallsproblemet), så vil et MIP-søk være den mest effektive søketilnærmingen. Den viktigste konkurrenten hva løsningstilnærming angår er ikke en alternativ søketilnærming, men en regelbasert konstruktiv metode – en metode som i stedet for å søke etter en optimal løsning, slavisk følger et regelsett for å konstruere én løsning, og som stopper når denne er konstruert. En slik tilnærming kan alltid følges – den vil ha meget liten kompleksitet – og løsningen blir så god som regelsettet man følger.

Like viktig som valg av tilnærming er valg av problemavgrensning. Skal man analysere et stort

problem er det veldig fristende å ta inn i optimeringen alle de detaljene som er en del av systemet, selv om de har begrenset innvirkning på løsningen. Hensikten er å øke kredibiliteten til modellen, og dette er helt legitimt. På et tidspunkt vil imidlertid den negative effekten på løsningskvaliteten (optimalitetsgapet) bli større enn den positive effekten på modellen, og da får man løsninger som ligger nærmere den reelt optimale løsningen ved en litt røffere modell enn med den mest korrekte modellen. Det er lett å fortsette å inkludere detaljer helt til den negative nettoeffekten blir veldig synlig, og da har man normalt passert optimal detaljeringsgrad godt og grundig. Har man mange interessenter i analysen som alle kjemper for sine detaljer kan det imidlertid være veldig vanskelig å avvise et detaljeringskrav uten å kunne vise til synlige negative konsekvenser, men da er man altså forbi den optimale detaljeringsgraden.

Dette ble grundig erfart i DRR fulfillment – etter at det først var gitt aksept for en forenklet representasjon, lyktes det å modellere problemet på en god måte hvor kompleksiteten for en periode ikke var et problem. Dette var kanskje en optimal detaljering. Innen prosjektet ble det da opplevd som umulig å hindre søkeproblemet fra å vokse inntil problemet igjen åpenbart var for detaljert og det ble vanskelig å nærme seg et optimum. (Det må sies at dette bare var ett av mange problemer innenfor fulfillment prosjektet, og langt fra den eneste årsaken til de problemene som oppsto.)

Med store søkeproblemer blir modellen og implementeringen viktig. Små modelleringsdetaljer kan få store konsekvenser for løsningstid, minnebruk og kvalitet på løsningen, og tilsvarende gjelder for implementeringen. Selv om man kan hoste opp en tidlig demo-prototyp i løpet av en dag eller to, må man gjerne beregne månedsverk i stedet for dagsverk på å få på plass en velfungerende modell.

Med store problemer blir totalkostnaden i form av tidsbruk så stor at det kan være hensiktsmessig å bruke dedikerte programmerere til implementering av modellen. Kostnaden ved overføring av kunnskap og problemforståelse blir ikke uforholdsmessig stor i forhold til totalkostnaden. Man må selvfølgelig fortsatt huske på at jobben ikke er gjort selv om man har hyrt inn noen til å gjøre den, og man må fremfor alt ta jobben med kunnskapsoverføring alvorlig.

Høynivå modelleringsverktøy, så som OPLStudio, er ikke laget med tanke på å håndtere store problemer på en effektiv måte. De er meget nyttig for utprøving og prototyping, men når modellen er etablert, vil det være regningssvarende å implementere den på et lavere nivå. Fulfillment i 03DRR viser imidlertid at det om nødvendig er mulig å kjøre veldig store modeller produsert i OPL.

14.4 Problemer som ikke er lineære

I alle problemene som er omtalt her er det virkelige problemet modellert slik man ønsket. Optimeringsmodellene er de opprinnelige modellene for problemet og ikke tilnærminger til dem. Når luftrekognoseringsproblemet omtales som lite egnet er det kun fordi løsningsmetoden er ineffektiv – LP-heuristikken hjelper oss ikke med å finne MIP-løsningen raskt.

I mange situasjoner kommer man over problemer som ikke er lineære, men hvor de aktuelle funksjonene ligger så nært lineære funksjoner at det er fristende å lage en lineær tilnærming til dem. Lineære *tilnærminger* til ikke-lineære problemer er et viktig område, og må nevnes. Punktene i dette delkapitlet gir kun generelle betraktninger om dette.

Så lenge man stoler på at alle forhold man ønsket å ta hensyn til er modellert korrekt i modellen, kan man bruke en optimering som en lukket beregning – en svart boks, og livet er ganske ubekymret. Resultatene blir like korrekte som inngangsverdiene. Hvis faktorer som kanskje har og kanskje ikke har vesentlig betydning er ukorrekt modellert, må man derimot vite veldig godt hva man gjør, og dette er langt mer slitsomt. Man må forstå om en ikke-linearitet som ikke er modellert ville hatt betydning for løsningen eller om den fører til valg med vesentlig sub-optimalitet. Lineariseringen vil gjerne være tilnærmet riktig veldig lokalt, og for å forstå betydningen av de tilnærmingene som er gjort, må man forstå løsningsrommet og den egentlige objektfunksjonens oppførsel på en helt annen måte enn man må dersom man stoler på modellen. Hvis man ikke vet om en ”falsk” linearisering har vært vesentlig for resultatet eller ikke, så gir som regel ikke resultatet så mye ny kunnskap, og derfor er verdien av optimeringen nært knyttet til denne dypere forståelsen for det reelle problemet.

Mange modeller, og deriblant en del LP/MIP-modeller har et vesentlig element av formell aggregering – de tar kunnskap på detaljert nivå (store mengder ensartet informasjon) og skal utlede kunnskap på overordnet nivå (små mengder informasjon) i henhold til kjente, formelle sammenhenger. Denne typen aggregering krever at et meget stort antall formelle beregninger kan gjennomføres på begrenset tid. Computermodeller er veldig gode til dette sammenlignet med mennesker. Dersom den forenklingen man ikke stoler på ligger i det lave nivået, og man skal vurdere resultatet på høyt nivå, er det som regel grunn til å være spesielt årvåken. Effekten av lavnivå ikke-lineariteter på et aggregert resultat er kompleks nok uten en optimering, og med en optimering på toppen er dette vanskelig å få oversikt over.

Hvis modellsvakheter gjør at LP/MIP-optimeringen i seg selv ikke gir tilstrekkelig kunnskap om hva som er den beste løsningen, er det to generelle veier å gå:

- Man kan avgrense den formaliserte LP/MIP-optimeringen til delproblemer som lar seg modellere tilstrekkelig godt, og velge en annen tilnærming til resten av problemet.
- Man kan beholde LP/MIP-tilnærmingen til hele problemet men skaffe flere fastpunkter for løsningen gjennom alternative tilnærminger.

En LP/MIP-tilnærming representerer visse faktorer veldig godt, og andre faktorer mindre godt. Andre tilnærminger vil ha komplementære egenskaper, og hvis de hver for seg (og eventuelt i kombinasjon) gir liknende resultater har man et mye sterkere resultat enn det hver av tilnærmingene hver for seg gir. Et viktig eksempel på dette er å kontrollere godheten (objektfunksjonen) av MIP-løsningen i det mer korrekte problemet og sammenligne denne med godheten til andre løsninger som er fremkommet på andre måter. På denne måten utnytter man LP/MIP-tilnærmingens styrke innen globale søk samtidig som man utnytter en annen

tilnærmings styrke når det gjelder spesielt troverdig evaluering av løsninger.

15 OPPSUMMERING

Optimering er en samlebetegnelse på matematisk modellering og løsing av et beslutningsproblem, og som fundamental tilnærming ligger den derfor veldig nært kjernen i operasjonsanalyse. I sin alminnelighet er et optimeringsproblem komplisert å løse, og bare enkelte klasser av optimeringsproblemer kan løses eksakt på en effektiv måte. Lineære problemer står i en særstilling hva angår løsbarhet, og selv problemer med mange millioner beslutningsvariable kan løses eksakt i løpet av sekunder. Lineære problemer med heltallsbeskrankninger (MIPer) er egentlig vanlige søkeproblemer (tre-søk), men hvis de er tilstrekkelig godt formulert, vil løsningen av problemet uten heltallsbeskrankningene (den relakserte løsningen) være en ekstremt god heuristikk på søket. Dette setter MIP-problemer i særklasse blant diskrete søkeproblemer hva angår løsbarhet selv om de er størrelsesordener verre å løse enn rent lineære problemer.

Hensikten med denne rapporten har vært å gi en kort presentasjon av hva optimering er, og hvordan det kan brukes i operasjonsanalyseprosjekter, samt å formidle praktiske erfaringer i bruk av LP/MIP i analyseprosjekter. Mens mange vitenskapelige miljøer tar utgangspunkt i en konkret tilnærming og prøver å finne passende problemer til denne tilnærmingen, tar OA-miljøene mål av seg til alltid å starte med problemet og deretter se etter den mest egnede tilnærmingen til dette problemet. Denne rapporten har kun handlet om én måte å løse et problem på, men det er lagt vekt på å få frem erfaringer om hvor denne metoden bør anvendes og hvor den ikke er egnet. Det er gitt tre eksempler på relativt vellykkede prosjekter der MIP/LP-modellering er brukt.

Når man skal vurdere LP/MIP som tilnærming til et analyseproblem, må man først vurdere om problemet lar seg formulere korrekt som et lineært problem. Dersom de faktorene man er interessert i ikke lar seg representere korrekt lineært skal man være veldig forsiktig med å forsøke en lineær tilnærming basert på en overforenkling. En slik modell og løsning vil normalt ikke gi ny kunnskap om analyseproblemet.

På den annen side er det flere problemer enn man kanskje skulle tro som lar seg linearisere korrekt. Dette omfatter blant annet alle diskret valgproblemer og alle stykkevis lineære problemer i tillegg til de genuint lineære problemene.

Lineære problemer uten heltallsbeskrankninger kan løses utrolig effektivt, og LP er uten sammenligning den mest effektive måten å angripe et slikt problem. Samtidig vil forskjellige deler av løsningen gi en veldig dyp kunnskap om det problemet som analyseres. Med gode modelleringsverktøy kan et vanlig problem modelleres og implementeres i løpet av et par dagsverk.

MIP-problemer som ikke er veldig store kan normalt løses effektivt, og derfor er dette en

anbefalelsesverdig tilnærming. En viktig fordel ved en MIP-tilnærming sammenlignet med alternative tilnærminger til slike problemer er at optimering tar et ideelt utgangspunkt siden optimeringsproblemet er ekvivalent med beslutningsproblemet. Også med en MIP-tilnærming får man sterk informasjon om kvaliteten på den løsningen som er funnet også om den ikke er helt optimal.

Begrensede LP/MIP-problemer kan modelleres og implementeres effektivt med COTS programvare, og erfaringene her er basert på bruk av OPLStudio og C-PLEX. Typisk tid for modellering og implementering av begrensede problemer i et høynivåspråk ligger i størrelsesorden noen dagsverk. Dette representerer en stor besparelse i forhold til implementering på lavt nivå, og bør vurderes selv om kostnaden ved lisensiert programvare kan være betydelig.

Videre kan de korte implementeringstidene gjøre det hensiktsmessig å bruke en MIP-tilnærming til problemer hvor selve beregningen kan gjennomføres mye raskere med andre metoder. Dette er eksemplifisert med et travelling salesman problem. Her er det imidlertid viktig å skille mellom en engangsanalyse og en analyse som skal gjennomføres regelmessig. I sistnevnte tilfelle vil eksekveringstid og stabilitet i eksekveringen telle tyngre sammenlignet med utviklingstid og utviklingskostnad.

Mange reelle lineære problemer er meget store, deriblant mange økonomiske problemer og allokeringsproblemer. Derfor har også lineære optimeringsmodeller en tendens til å bli store. Eksemplet som er brukt her er NATO DRR fulfillment, som er en meget stor MIP. MIP er i forhold til andre optimeringstilnærminger veldig godt egnet til å håndtere store problemer, og dette er derfor en god analysetilnærming for slike problemene. Man bør imidlertid være klar over en del forhold ved slike problemer:

- Prosesseringstid og minnebruk er veldig avhengig av hvor god heuristikk LP-løsningene gir, og dermed må man legge mye mer arbeid i en god modell enn det man må med et mer håndterbart problem
- Når man nærmer seg grensen for hvor store problemer som kan håndteres, går kvaliteten på løsningen sterkt ned, og det er en god idé å prøve å stanse detaljering av modellen en stund før dette – dette gir en litt mindre eksakt modell, men altså en mye mer eksakt løsning på denne modellen. I et reelt prosjekt har det vist seg vanskelig å stoppe veksten til en modell i tide.
- Utviklingstider for et stort problem er gjerne måneder heller enn dager, og dette gjør det aktuelt å sette bort implementering. Overføring av problemforståelse til den som skal implementere kan være krevende og ta store ressurser siden det forutsetningsvis er snakk om et komplekst problem.

Store diskrete problemer som ikke lar seg linearisere på en hensiktsmessig måte – der det er dårlig samsvar mellom den lineære løsningen og heltallsløsningen – er ikke egnet for løsning med en MIP-tilnærming.

APPENDIKS

FORKORTELSER

AAR	Air to Air Refuelling
ACO	Allied Command Operations
ACT	Allied Command Transformation
AMPL	A Mathematical Programming Language
COTS	Commercial Off The Shelf
CAG	Component Area Group
CP	Constraint Programming
CPU	Central Processing Unit
D-FARM	DRR Force Allocation Rule Motor
DRR	Defence Requirements Review
FFI	Forsvarets Forskningsinstitutt
GB	Giga-byte
IP	Integer Programming
J-DARTS	Joint DRR Analysis and Requirements Tool Set
JAG	Joint Analysis Group
LoA	Level of Ambition
LP	Linear Programming
MIP	Mixed Integer Programming
MOPE	Measure of Policy Effectiveness
NATO	North Atlantic Treaty Organisation
NC3A	NATO Command & Control Consultation Agency
NPC	Non Polynomial Complete
OA	Operasjonsanalyse
OPL	Optimisation Programming Language
SACLANT	Supreme Allied Command AtLANTic
SHAPE	Supreme Headquarter Allied Powers Europe
SQL	Standard Query Language

LUFTREKOGNOSERINGSMODELL – OPL KODE

```
struct directedsegment {
   string startnode;
   string endnode;
   float length;
};

struct nodedata {
   string name;
   float latitude;
   float longitude;
   int+ fuelpoint;
};



DBconnection inputandstuff ("odbc", "airreccedata//");


///  SETS
//
{directedsegment} SEGMENTS from DBread (inputandstuff, "select startnode,
endnode, length from segments;");
{nodedata} TEMPNODES from DBread (inputandstuff, "select node, latitude,
longitude, fuelpoint from nodes;");

{string} FUELPOINTS ={fp | <fp, l1, l2, i> in TEMPNODES : i=1};
{directedsegment} FUELPOINTSEGMENTS = {<c,c,0>|c in FUELPOINTS};

{directedsegment} DIRECTEDSEGMENTS = SEGMENTS union {<a,b,l>|<b,a,l> in
SEGMENTS} union FUELPOINTSEGMENTS;
{string} NODES = {a|<a,b,l> in DIRECTEDSEGMENTS};

int noofsegments = card(SEGMENTS);
int noofroutesteps = noofsegments+2;//minl(6, noofsegments+2);
{int} AIRFRAMES =1..noofsegments;
{int} ROUTESTEPS = 1..noofroutesteps;
{int} ROUTESTEPSLOW = 1..noofroutesteps-1;
{int} ROUTESTEPSHIGH = 2..noofroutesteps;

/// PARAMETERS

{float} templat[n in NODES]={la |<n, la, lo, fp> in TEMPNODES};
{float} templon[n in NODES]={lo |<n, la, lo, fp> in TEMPNODES};
//display templat;
float lat[n in NODES]=templat[n].first();
float lon[n in NODES]=templon[n].first();

//display noofsegments;
//display AIRFRAMES;
//display noofroutesteps;
//float distance[a in NODES, b in NODES] = 6367 *
acos(cos(lat[a]*cos(lat[b]*cos(lon[a]-lon[b] + sin(lat[a])*sin(lat[b]));
float coslat=...;
float distance[a in NODES, b in NODES] = 111.12 * sqrt((lon[a]-
lon[b])*(lon[a]-lon[b])*coslat*coslat+(lat[a]-lat[b])*(lat[a]-lat[b]));
//display distance;
```

//Merk at OPL ikke har trigonometriske funksjoner. Parameteren ’coslat’ er lest fra datafil.

```
float fuelrange[ac in AIRFRAMES] = ...;
```

```
///   VARIABLES
var int covers[AIRFRAMES, ROUTESTEPS, DIRECTEDSEGMENTS] in 0..1;
var int transit[AIRFRAMES, ROUTESTEPSLOW, NODES, NODES] in 0..1;
var int inuse[AIRFRAMES] in 0..1;
var int notcovered[SEGMENTS] in 0..1;
//var int coverwithinroute[AIRFRAMES, DIRECTEDSEGMENTS] in 0..1;

/// COSTPARAMETERS
float+ extraairframecost=10;
float+ costofnotcovered=20;
float+ earlyairframeselector=0.01;
float+ earlyroutestepselector=0.01;
float+ extratransitcost=0.2;



/// OBJECTIVE FUNCTION
minimize
( sum (ac in AIRFRAMES) inuse[ac]*extraairframecost
+ sum (s in SEGMENTS) notcovered[s]*costofnotcovered
+ sum (ac in AIRFRAMES) inuse[ac]*earlyairframeselector*ac
+ sum (ac in AIRFRAMES, rs in ROUTESTEPS, ds in DIRECTEDSEGMENTS) covers[ac,
rs, ds] * earlyroutestepselector*rs
+ sum (ac in AIRFRAMES, rsl in ROUTESTEPSLOW, a in NODES, b in NODES : b<>a)
transit[ac, rsl, a, b] * extratransitcost
)


///   CONSTRAINTS
subject to
{
// cover of all segments
forall (<a, b, l> in SEGMENTS)
   sum (ac in AIRFRAMES, rs in ROUTESTEPS) (covers[ac, rs, <a, b, l>] +
covers[ac, rs, <b, a, l>]) >= 1 - notcovered[<a, b, l>];

// fueling available at start and end of sortie
// at start:
forall (ac in AIRFRAMES)
  (sum (fps in FUELPOINTSEGMENTS) covers[ac, 1, fps] >= inuse[ac]);

// at end:
forall (ac in AIRFRAMES)
   sum (fps in FUELPOINTSEGMENTS) covers[ac, noofroutesteps, fps] >=
inuse[ac];

// flyin equals cover equals flyout - consistent route
// flyin:
forall (ac in AIRFRAMES, rs in ROUTESTEPSHIGH, a in NODES)
   sum (c in NODES) transit[ac, rs-1, c, a] = sum(<a, b, l> in
DIRECTEDSEGMENTS) covers[ac, rs, <a, b, l>];

// flyout:
forall (ac in AIRFRAMES, rs in ROUTESTEPSLOW,  b in NODES)
   sum (c in NODES) transit[ac, rs, b, c] = sum(<a, b, l> in
DIRECTEDSEGMENTS) covers[ac, rs, <a, b, l>];

// route wihtin range
```

```
forall (ac in AIRFRAMES)
   ( sum (rs in ROUTESTEPS, ds in DIRECTEDSEGMENTS) covers[ac, rs,
ds]*ds.length
   + sum (rsl in ROUTESTEPSLOW, a in NODES, b in NODES) transit[ac, rsl, a,
b]*distance[a, b])
   <= fuelrange[ac];


// sum (ac in AIRFRAMES, rsl in ROUTESTEPSLOW, a in NODES, b in NODES)
transit[ac, rsl, a, b]*distance[a, b]<= 1000;

// control of forcepool
forall (ac in AIRFRAMES, rs in ROUTESTEPS, ds in DIRECTEDSEGMENTS) covers[ac,
rs, ds]<=inuse[ac];

// unique deployment
forall (ac in AIRFRAMES, rs in ROUTESTEPS) sum (ds in DIRECTEDSEGMENTS)
covers[ac, rs, ds] <=1;

// redundant constraints
//forall (<a, b, l> in SEGMENTS)
//   sum (ac in AIRFRAMES, rs in ROUTESTEPS) (covers[ac, rs,<a, b,
l>]+covers[ac, rs,<b, a, l>])=1

};


{int} USEDAIRFRAMES = {ac | ac in AIRFRAMES : inuse[ac]>=1};

display USEDAIRFRAMES;
display notcovered;

{directedsegment} location[uaf in USEDAIRFRAMES, rs in ROUTESTEPS] = {ds | ds
in DIRECTEDSEGMENTS : covers[uaf, rs, ds]>=1};

display location;

{directedsegment} transitflights[uaf in USEDAIRFRAMES, rs in ROUTESTEPSLOW] =
{<a, b, distance[a, b]> | a, b in NODES : transit[uaf, rs, a, b]>=1};// :
((transit[uaf, rs, a, b]>=1) & (l=distance[a, b]))};

display transitflights;

 //*/
 //solve{};
```

Litteratur

(1) Frederick S Hiller, Gerald J Lieberman (1990): *Introduction to Operations Research, fifth edition*, McGraw-Hill international editions, Industrial Engineering Series, Singapore.

(2) ILOG homepage: www.ilog.com

(3) Grotmol Ø., Sukkestad J. A., Braathen S. (2001): Hotdog: a heuristic on the determination of optimum globally using the mathematical programming language AMPL, FFI/NOTAT-2001/03011.

(4) Braathen S., Grotmol Ø., Langsæter T. (2000): Modell for flerscenario strukturoptimering – modellbeskrivelse, FFI/RAPPORT-2000/04739.

(5) NATO C3 Agency: Fulfillment documentation (formal status unknown)