TEKST NR 405 2001

real life routing

- en strategi for et virkeligt vrp

et matematisk modelprojekt

udarbejdet af:

David Heiberg Backchi, Rasmus Brauner Godiksen,
Uffe Thomas Volmer Jankvist, Jogvan Martin Poulsen
og Neslihan Saglanmak

vejledt af:

Jørgen Larsen

TEKSTER fra

IMFUFA ROSKILDE UNIVERSITETSCENTER
INSTITUT FOR STUDIET AF MATEMATIK OG FYSIK SAMT DERES
FUNKTIONER I UNDERVISNING, FORSKNING OG ANVENDELSER

IMFUFA, RUC, Postbox 260, 4000 Roskilde, Danmark
Tlf. 4674 2263, Fax 4674 3020, e-mail: imfufa@ruc.dk
"Real Life Routing – en strategi for et virkeligt VRP"
Af: David Heiberg Backchi, Rasmus Brauner Godiksen,
Uffe Thomas Jankvist, Jogvan Martin Poulsen og
Neslihan Saglanmak
Vejleder: Jørgen Larsen
IMFUFA-tekst nr. 405/2001, 119 sider, ISSN 0106-6242

Abstract:

Dette projekt tager udgangspunkt i et virkeligt VRP-problem, nemlig FDBs. For deslige FDB-lignende problemstillinger foreslås udvidelser til en klassisk matematisk formulering af VRPTW, således at denne kan tage forbehold for relevante faktorer i et ruteplanlægningsproblem - såsom backhaul og køre- og hviletidsbestemmelser - som ikke er indbefattet i VRPTW. Ydermere vurderes hvilken klasse af algoritmer vi finder mest fordelagtig til implementering i et rutelægningssoftware til den givne type af problemer. De relaterede overvejelser bygger i høj grad på dataindsamling fra benchmarktests af de omhandlede algoritmer og i nogen grad på beregningskompleksitetsforhold. Afslutningsvis konkluderes på baggrund af en afvejning mellem løsningskvalitet og beregningstid, at det ved løsning af et FDB-lignende rutelægningsproblem ville være mest fordelagtigt at benytte tabu-søgning som søgealgoritme i et ruteplanlægningsprogram.

'you give me, you give me the sweetest taboo'
– Sade

Forord

Denne projektrapport beskriver ved en matematisk indgangsvinkel ruteplanlægning og diverse værktøjer i forbindelse hermed. Projektet indgår som en del af overbygningsuddannelsen i matematik ved IMFUFA, RUC. Projektet, som er et modelprojekt, hører under 1. og 2. modul.

Vi vil gerne takke vores vejleder lektor Jørgen Larsen, for et konstruktivt og engageret samarbejde. Desuden vil vi takke professor Jørgen Tind, KU, og professor Oli B.G.Madsen, DTU, for i begyndelsen af projektet at have ydet en stor hjælp til at belyse mulige angrebsvinkler.

Vi vil også gerne takke konsulent Anette Vainer, hos Carl Bro as, for at have hjulpet os med at få overblik over fagområdet, samt at have givet os en fornemmelse af hvordan ruteplanlægning foregår i den virkelige verden – udenfor den akademiske teoretiske spændetrøje.

Vi takker også Steen Thomsen, Jesper Kolvbech og Benny fra FDB, Albertslund, for en rundvisning på FDBs tørvarelager og en kort demonstration af deres ruteplanlægningsprogram.

Til sidst vil vi også gerne takke reklameafdelingen hos F.L.Smidth A/S for at have stillet ekspertise og tekniske hjælpemidler til rådighed i forbindelse med trykning af rapporten.

David Heiberg Backchi
Rasmus Brauner Godiksen
Uffe Thomas Volmer Jankvist
Jogvan Martin Poulsen
Neslihan Saglanmak

Roskilde, den 14. juni 2001

Rapporten er i forbindelse med dens udgivelse som IMFUFA-tekst blevet rettet for mindre fejl og mangler. Desuden er appendix B blevet tilføjet.

København, den 28. oktober 2001

Indhold

1 Optakt

1.1 Indledning

Distribution er en vigtig del af mange virksomheders aktivitet. Næsten alle virksomheder har behov for fysisk at flytte varer eller materialer fra et sted til et andet, og omkostningerne i forbindelse med dette kan udgøre en betydelig del af en virksomheds omsætning.

Så længe der i forbindelse med transporten ikke ligefrem ødelægges eller beskadiges varer, vil transporten ikke påvirke værdien af en virksomheds produkt. Alt hvad der spares på transportomkostningen er derfor ren fortjeneste for virksomheden. Man kan sige, at besparelsen 'går direkte ind på bundlinjen'.

Dette gælder naturligvis ikke for virksomheder, hvor selve distributionen er en del af produktet som f.eks. avisudbringning. Her vil besparelser nemt kunne gå ud over serviceniveauet og derfor svække produktet.

Med baggrund i ovenstående er det derfor ønskeligt at minimere transportomkostningerne så meget som muligt. Der vil dog altid være visse grænser for hvordan besparelserne kan foregå. Kapaciteten af distributionsnet, udgifter til brændstof og lønninger, vedligeholdelse af materiel, arbejdsregler o.lign. er med til at sætte nogle grænser for, hvordan der kan effektiviseres og hvor store besparelser, der kan opnås. Minimeringen af transportomkostningerne skal altså gøres under visse *begrænsninger* (betingelser).

I forbindelse med minimering af transportomkostningerne, vil det for opgaver af en vis størrelse ofte kunne betale sig at opbygge en matematisk model af transportsituationen. Modellen kan anses for en simplificering af virkeligheden, hvis formål er at give et mere gennemskueligt billede af den forelagte situation. Modellen skal beskrive de væsentligste omkostninger og begrænsninger situationen indeholder. Ved at minimere omkostningerne i modellen kan omkostningerne i den faktiske transportsituation sænkes.

Alt efter hvilken type transportsituation man har med at gøre, vil modellerne have forskellig opbygning. Da det vil være uoverkommeligt at dække alle typer af situationer, har vi været nødt til begrænse os. Dette har vi gjort ved at tage udgangspunkt i en virkelig transportsituation.

Vi har i projektforløbet haft kontakt med FDBs tørvarelager i Albertslund, som fornylig har indkøbt et af de kommercielle ruteplanlægningsværktøjer. Lageret distribuerer dagligvarer til deres butikker beliggende i det meste af Østdanmark.

Denne distribution har vi valgt som udgangspunkt for vores projekt. Transporten fra tørvarelageret til dagligvarebutikkerne sker med lastvogne, og projektet omhandler derfor ruteplanlægning i forbindelse med vejtransport, hvor en distributør skal forsyne et antal butikker med varer. Et sådant problem hører under kategorien *Vehicle Routing Problem*, eller blot VRP[1].

Løsningen af et VRP består i at træffe nogle valg; hvor hvilke vogne skal køre hen og hvornår, er eksempler på sådanne valg. Den optimale løsning ville kunne findes ved at afprøve alle mulige valg for en given situation, og efterfølgende vurdere hvilket, som var mest passende til formålet. I virkelige situationer vil antallet af valg ofte være så stort, at dette i praksis er umuligt. Selv med computere langt hurtigere end de som findes i dag, vil det tidsmæssigt ikke være muligt at afprøve alle valg i en virkelig VRP-situation.

Det er altså nødvendigt at bruge smartere metoder til at træffe valg end blot at afprøve samtlige muligheder. I løbet af de sidste 20-30 år er udviklingen af sådanne metoder tiltaget. Der er både blevet udviklet metoder, som garanterer at finde de bedste løsninger, og metoder som ikke gør. Indenfor hver af disse hovedkategorier er der adskillige underkategorier.

Mange af metoderne er udviklet til at løse simple VRP-situationer, dvs. situationer hvor der kun er få begrænsninger. Disse stemmer ikke godt overens med virkeligheden, da mange virkelige VRP-situationer har mange specielle hensyn, som skal tages. Der er dog indenfor de sidste år kommet mere fokus på virkelige VRP-situationer, og hvordan disse kan løses. Vi ønsker at lave en generel vurdering af hvilke metoder, der vil egne sig til optimering af virkelige VRP-situationer, som ligner FDBs.

1.2 Problemformulering

Hvordan kan en matematisk model for et virkeligt ruteplanlægningsproblem opstilles og løses?

1.3 Afgrænsning og uddybning

Virkelige VRP-situationer kan tage mange former. Der er f.eks. stor forskel på ruteplanlægningen i et kurér-firma og i et olieselskab. Vi bruger som nævnt situationen fra FDBs tørvarelager i Albertslund til at indsnævre problemfeltet, idet vi begrænser os til situationer, som ligner denne. Dvs. diverse former for planlægningsopgaver hvor varer skal transporteres fra et selskabs lager til dets kunder (i FDBs tilfælde deres egne butikker).

Vi beskæftiger os dog ikke udelukkende med FDBs specifikke situation, da det ville kræve et indgående kendskab til mange data omkring FDBs varedistribu-

[1] Principielt kan VRP dog også bruges til at beskrive distribution i forbindelse med andet end vejtransport.

tion. I stedet bruger vi FDB som udgangspunkt for at undersøge de typer af rutelægningsproblemer, der kan forekomme i situationer som minder om deres.

Udover at undersøge hvordan en matematisk model til beskrivelse af ruteplanlægning kan opstilles og løses, ønsker vi at give en vurdering af løsningsmetoderne.

For en virksomhed afhænger den *bedste* løsning af mange faktorer. Nogle af disse kunne være: Hvor store besparelser kan metoden give virksomheden, hvor besværlig er den at implementere i det eksisterende logistiksystem, hvor store er anskaffelsesomkostningerne og kræver metoden efteruddannelse af medarbejdere.

De fleste af den slags spørgsmål ligger uden for matematikerens kompetencesfære. Vi har derfor valgt, at beskæftige os med de besparelser metoden kan give, efter den er blevet implementeret, og medarbejdere har lært at betjene sig af den. Derudover indsnævrer vi os yderligere, idet vi kun ser på omkostninger forbundet med selve transporten og ikke besparelser som følge af administrative eller organisatoriske rationaliseringer.

En given ruteplanlægning vil resultere i en ruteplan bestående af et sæt ruter, som lastvognene skal køre. Man kan altså sige, at vi kun beskæftiger os med de omkostninger, som varierer fra én ruteplan til en anden.

1.4 Metode

I ruteplanlægning implementeres løsningsmetoder ofte i en computer for bl.a. at teste hvor gode løsninger de giver, og hvor hurtigt de er i stand til at finde dem. Da ingen i projektgruppen har særlige færdigheder indenfor datalogisk programmering, har vi desværre ikke haft mulighed for dette. Vi forsøger derimod at vurdere de forskellige modeller og løsningsmetoder ud fra de store mængder litteratur, der er skrevet om ruteplanlægning.

2 Modeller til ruteplanlægning

I dette projekt vil vi, som beskrevet i indledningen, beskæftige os med den del af *operationsanalysen*, der vedrører problemer i forbindelse med *varedistribution*. Mere specifikt vil vi undersøge, hvordan der med et antal vogne kan planlægges ruter, på en måde så ruterne bliver mest optimale. Hvad der menes med det 'mest optimale' afhænger af, hvad man ønsker at optimere. Et eksempel kan være at finde den ruteplan med det mindste benzinforbrug eller med den korteste afstand.

Overordnet er der altså tale om et *optimeringsproblem*. Et sådant drejer sig om at finde den *optimale løsning* til et problem, når *løsningsrummet* er givet. Løsningsrummet, som er mængden af alle løsninger, fremkommer ved de begrænsninger, der er forbundet med problemet. Det vil sige, at løsningsrummet for et ruteplanlægningsproblem indeholder alle de ruteplaner, der overholder begrænsningerne, inklusive den optimale ruteplan. Begrænsningerne kan for eksempel være, at vognene har en kapacitet, der ikke må overskrides, eller at hver rute skal indeholde mindst 3 butikker.

Før vi tager fat på, hvordan man i operationsanalysen behandler denne type af problemer, vil vi kort skitsere, hvordan man opstiller matematiske modeller, herunder optimeringsmodeller, samt give et billede af de matematiske begreber, der bruges i et ruteplanlægningsproblem. Man kan dele arbejdet med modellen op i to hovedpunkter, opstilling og behandling [Hermann and Niss, 1982].

Opstilling af model
Når en virkelig situation skal modelleres, er det vigtigt at *udvælge* og *idealisere* de elementer og koblinger, der er væsentlige for løsningen af situationen. Når de relevante elementer og koblinger fra den virkelige situation er valgt ud og simplificeret, kan problemet oversættes til matematiske objekter. Ofte vil der være flere muligheder for den matematiske repræsentation, men overvejelser over den videre proces kan medvirke til det endelige valg.

I praksis vil udvælgelse, idealisering og matematisk repræsentation ofte smelte sammen, så der ikke kan laves en skarp skelnen. For eksempel kan man ved udvælgelsen af elementer have en bestemt idealisering eller matematisk repræsentation i tankerne, som vil komme til udtryk allerede i denne første fase. En sådan kan være inspireret af erfaringer med lignende modeldannelsessituationer eller kan være et udtryk for visse forhåndsindstillinger.

Behandling af model

Behandlingen består i ved hjælp af matematikken at drage konklusioner om de i modellen indgående objekter, altså løse de matematiske problemer, der repræsenterer problemerne i virkeligheden. Til slut i behandlingsfasen skal de matematiske resultater oversættes til den virkelige situation og bedømmes.

Det er ikke altid muligt at løse de matematiske problemer og ofte vil det hænde, at resultaterne ikke viser sig tilfredsstillende. Dette kan give anledning til en yderligere simplificering eller tilnærmelse og en fornyet modeldannelse. Modelarbejdet bliver således en proces, hvor de forskellige faser påvirker hinanden.

I det efterfølgende vil vi konkretisere ovenstående med hensyn til ruteplanlægningssituationen.

Opstilling af ruteplanlægningsmodel

De vigtige elementer i et ruteplanlægningsproblem vil være kundernes placeringer i forhold til hinanden, antallet af vogne og deres egenskaber, specificering af varemængderne, vejforhold, ruternes tidsmæssige varighed m.m. Simplificeringen kunne bestå i at se bort fra vejsving, lyskryds o.lign. og i stedet beskrive vejene mellem de forskellige kunder som lige strækninger. En anden simplificering kunne være at angive vognene kun efter hvor meget vægt, der kan lastes på dem. Ofte vil man foretage den matematiske repræsentation ved at skitsere situationen med en *graf* (jf. afsnit 2.1). Som vi vil se, danner grafteorien en vigtig del af grundlaget for selve modelstrukturen for denne type af problemer.

Koblinger mellem de udvalgte elementer, dvs. begrænsningerne, kan være, at varemængden i en enkelt rute ikke må overstige vognens lastkapacitet, at en bestemt kunde altid skal være den førstbesøgte kunde, at der for nogle vejstrækninger er forbud for kørsel med lastvogn eller andet som distributøren finder relevant. De ruteplaner, hvor en begrænsning bliver overtrådt, vil være en *ulovlig* løsning, denne ruteplan vil altså ikke ligge i løsningsrummet. Den matematiske repræsentation af begrænsningerne kan f.eks. opskrives som uligheder eller ligninger.

Selve problemet går som nævnt ud på at optimere ruteplanlægningen. I mange tilfælde vil optimeringen bestå i at minimere den kørte afstand. Da vil den samlede afstand for en given løsning, x, kunne beskrives ved en funktion, $c(x)$, kaldet *objektfunktionen*. Objektfunktionen tillægger altså i dette tilfælde enhver løsning i løsningsrummet en værdi for den samlede afstand. I mange tilfælde er det den samlede omkostning, der skal minimeres, og her kaldes objektfunktionen også for *omkostningsfunktion*.

Problemet kan nu formuleres matematisk på følgende måde:

$$\text{Minimér} \quad c(x) \tag{2.1}$$

$$\text{med hensyn til} \quad g_i(x) \leq b_i \quad i = 1, \ldots, n \tag{2.2}$$

hvor x er en mulig løsning, c er *omkostningsfunktionen* og de n uligheder i (2.2) er begrænsningerne.

Vi har nu skitseret, hvordan man kan konkretisere virkeligheden. Vi har udvalgt de ting, vi vil forholde os til, og simplificeret dem. Med andre ord, så har vi opskrevet en matematisk formulering af den del af virkeligheden, vi vil modellere.

Behandling af ruteplanlægningsmodel
I denne fase skal den matematiske model behandles. Opgaven består altså i at finde den rute, der giver den mindste omkostning.

Matematisk formuleret kan man sige: Lad $\mathbf{X}$ være mængden af løsninger $x \in \mathbf{X}$ og $c : \mathbf{X} \rightarrow \mathbf{R}$ være en afbildning fra $\mathbf{X}$ ind i de reelle tal. Bestem da den løsning $x' \in \mathbf{X}$, hvor $c(x') \leq c(x)$ for ethvert $x \in \mathbf{X}$, altså find minimumspunktet for c.

I praksis vil der i mange optimeringsproblemer bruges matematisk programmering til løsning af disse. Ordet programmering skal ikke forstås i datalogisk betydning, men betegner den mulighed matematikken giver for at behandle et problem på en systematisk og præcis måde [Blomhøj et al., 1984].

I vores ruteplanlægningsmodel er løsningerne x en $n \times n$ matrix, hvor n er antallet af kunder. Elementerne i matricen x_{ij}, hvor $i, j = 1, 2, \ldots, n$, er såkaldte beslutningsvariable; de udtrykker hvilke ruter der køres i den givne løsning.

I de tilfælde hvor objektfunktionen er en lineær funktion, begrænsningerne er lineære uligheder og de variable tilhører de reelle tal er der tale om *lineær programmering* (LP). Objektfunktionen er en lineær funktion, når den opfylder

$$c(\alpha x + y) = \alpha c(x) + c(y) \qquad x, y \in \mathbf{X}, \alpha \in \mathbf{R} \tag{2.3}$$

Der er udviklet metoder til at løse LP-problemer, hvor den mest omtalte er *simplex metoden* [Foulds, 1984]. I nogle optimeringsproblemer giver løsningen kun mening hvis $x_{ij}, y_{ij} \in \mathbf{Z}$, hvor $\mathbf{Z}$ er mængden af hele tal. Her er der tale om *heltalsprogrammering* (IP).

Da vi i vores ruteplanlægningsmodel opererer med beslutningsvariable, som kun må antage værdierne 0 og 1, kan vi ikke betegne problemet som et LP-problem. Vores variable tilhører en endelig delmængde af $\mathbf{Z}$, nemlig $\{0, 1\}$, og kan betegnes som et *kombinatorisk* optimeringsproblem [Nemhauser and Wolsey, 1988]. I et kombinatorisk optimeringsproblem er løsningsrummet endeligt, det vil sige at der i vores tilfælde er et endeligt antal mulige løsninger til en ruteplan.

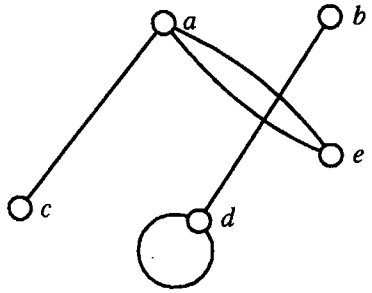
Der findes forskellige løsningsmetoder til kombinatoriske optimeringsproblemer, både nogle der garanterer at finde den optimale løsning og nogle der ikke gør. Vi vil ikke komme ind på dem her, da de behandles senere.

2.1 Et udvalg af netværks- og grafteori

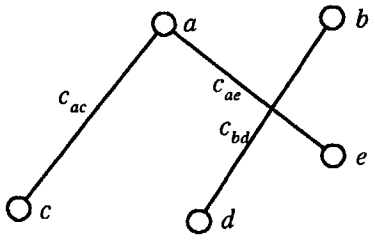
Lidt kendskab til netværks- og grafteori kan hjælpe på forståelsen af en matematisk formulering af VRP og TSP. Vi vil i det følgende kort introducere de grundlæggende dele af grafteorien, der bruges i rapporten. Vi har taget udgangspunkt i [Dolan and Aldous, 1995] og [Biggs, 1989].

En **graf** $G = (V, E)$ er et par bestående af en endelig mængde $V = \{1, 2, \ldots, n\}$ hvis elementer kaldes punkter, og en mængde E bestående af delmængder af V med to elementer, altså $V \supseteq \{i, j\} \in E$, herefter skrives de (i, j). Elementerne i E kaldes kanter (jf. figur 2.1).

For en **simpel graf** gælder, at der kun findes én kant mellem to givne punkter i og j, samt at der ikke findes nogen kanter fra i til i (figur 2.2).



Figur 2.1 En graf $G = (V, E)$ med fem punkter og fem kanter, hvor to er ens.



Figur 2.2 En simpel vægtet graf med fem punkter og tre kanter.

En graf kan vægtes ved at tildele enhver kant (i, j) en vægt c_{ij} (jf. figur 2.2). En **vægtet graf** skrives $G = (V, E, C)$, hvor $C \subseteq \mathbf{R}$.

En **komplet graf** er en graf, hvor ethvert par af punkter (i, j) med $i \neq j$ er forbundet med netop én kant[1] (jf. figur 2.3).

I nogle tilfælde kan det være nødvendigt at definere en retning på kanten, for eksempel hvis det kun er muligt at gå fra punkt i til j, men ikke fra j til i. Her benyttes en digraf (directed graph) i stedet for en graf.

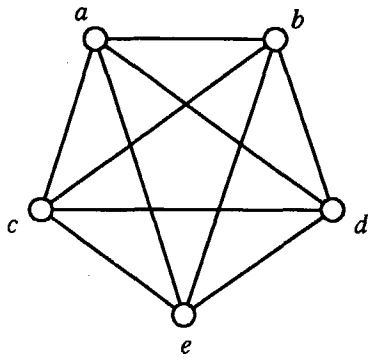
En **digraf** $D = (V, A)$ er en graf, hvor kanterne er retningsorienterede (jf. figur 2.5). Det vil sige $A = \{(i, j) \mid i, j \in V\}$ er mængden af ordnede par af punkterne tilhørende V.

Når punkter og/eller kanter i en graf eller en digraf har tilknyttet talværdier kalder man også det samlede system for et **netværk**.

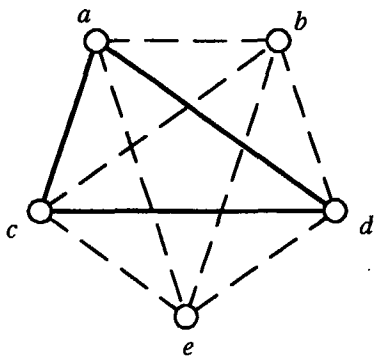
En **cykel** i en graf $G = (V, E)$ er en delmængde $V_k \subseteq V$ med $|V_k| = k$. Elementerne i V_k opskrives som en endelig følge $i_1, i_2, \ldots, i_{k+1}$, med $i_1 = i_{k+1}$, og de resterende elementer i V_k kun optrædende én gang, dvs. $i_j \neq i_l$ for alle

[1] Antallet af kanter i en komplet graf med n punkter er $\binom{n}{2} = \frac{n(n-1)}{2}$.

$j, l \in V_k \setminus \{1, k+1\}$. To på hinanden følgende punkter i følgen er forbundet med en kant, altså $E_k = \{(i_j, i_{j+1}) \mid 1 \leq j \leq k\}$. Cyklen består altså af k punkter og k kanter, hvor der til ethvert punkt knyttes to kanter (jf. figur 2.4). En cykel med k kanter kaldes også en **k-cykel**.



Figur 2.3 En komplet graf $G = (V, E)$ med fem punkter og 10 kanter.

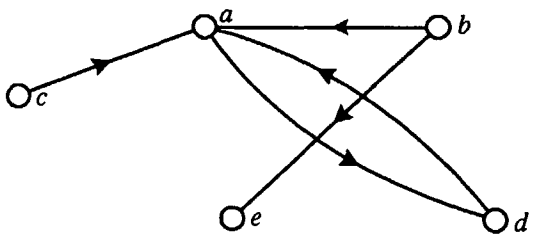


Figur 2.4 En 3-cykel i G.

En **hamiltonisk kreds** er en cykel af længde k i en graf med k punkter, det vil sige $V_k = V$ [Reinelt, 1994].

2.2 Den rejsende sælgers problem

Traveling Salesman Problem (TSP) består i, at en sælger skal besøge en række byer og derefter returnere til sit udgangspunkt. Givet et antal byer og et antal veje, som forbinder byerne, skal sælgeren finde den korteste rute rundt til alle byerne uden at besøge samme by mere end én gang [Lawler et al., 1985], [Dolan and Aldous, 1995].



Figur 2.5 En digraf $D = (V, A)$ med 5 punkter og 5 retningsorienterede kanter.

Betragtes TSP som et grafteoretisk problem, kan situationen beskrives ved en simpel, komplet og vægtet graf $G = (V, E, C)$, hvor $V = \{1, 2, \ldots, n\}$ er mængden af byer. $E = \{(i,j) \mid i,j \in V\}$ er mængden af alle uordnede par af elementer fra V, f.eks. repræsenterer parret (i,j) vejen fra den i'te til den j'te by. I og med at G er komplet, kan hver enkelt punkt parres med alle andre punkter, og da G også er simpel følger at $(i,j) = (j,i)$, med $i \neq j$. C er en $n \times n$ matrix hvor det c_{ij}'te element er vægten af kanten (i,j), som beskriver omkostningen af en rejse fra i til j [Dolan and Aldous, 1995]. Til hver kant knyttes altså en funktion[2] $c : E \to \mathbf{R}$ kaldet en *omkostningsfunktion*, som til hver kant $(i,j) \in E$ definerer en vægt c_{ij}. Den samlede vægt af en delmængde af kanter $F \subseteq E$ er defineret som

$$c(F) = \sum_{(i,j) \in F} c_{ij} \tag{2.4}$$

[Reinelt, 1994].

Man kan altså sige, at TSP består i at finde den *korteste* hamiltoniske kreds [Minieka, 1978].

Et TSP siges at være *symmetrisk*, hvis det for alle i og j gælder, at $c_{ij} = c_{ji}$. Matricen C er da en symmetrisk matrix. Et asymmetrisk TSP adskiller fra et symmetrisk ved at være beskrevet ved en digraf, altså en retningsorienteret graf. Til dette problem behøver matricen C ikke være symmetrisk.

En matematisk definition af et symmetrisk TSP kunne lyde:

Givet en simpel, komplet og vægtet graf $G = (V, E, C)$, er det ***symmetriske TSP at finde den korteste hamiltoniske kreds på G*** [Reinelt, 1994].

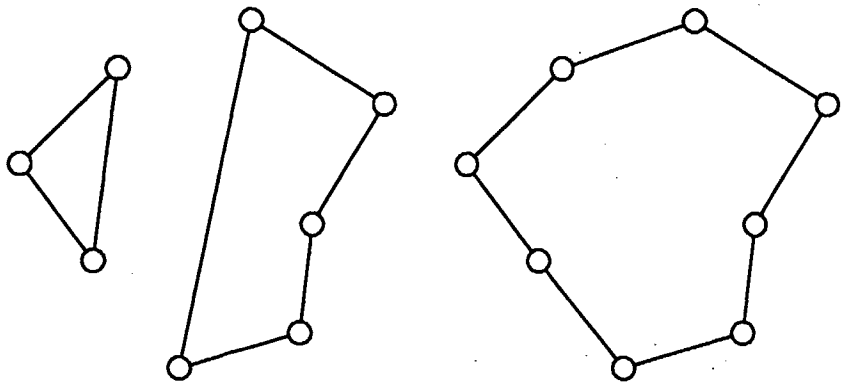
Hvis et symmetrisk TSP opfylder *trekantsuligheden*, altså

$$c_{ij} \leq c_{il} + c_{lj} \tag{2.5}$$

for alle punkter $i, j, l \in V$, kaldes det et *metrisk* TSP. I disse problemer svarer punkterne til punkter i et metrisk rum, og kanternes vægte er afstandene mellem disse givet ved den til rummet hørende metrik. F.eks. er et *euklidisk* TSP defineret ved en mængde af punkter i planen, hvor kanternes vægte er givet ved den euklidiske afstand mellem to punkter [Reinelt, 1994].

En matematisk model til TSP kan formuleres på adskillige måder. En af de i litteraturen hyppigt anvendte beskrivelser tager sit udgangspunkt i matematisk programmering. Til denne formulering introduceres en $n \times n$ beslutningsmatrix X, som beskriver den valgte rute. Pladserne i matricen kan antage værdierne 0 og 1 (diagonalen kan dog kun antage værdien 0), det vil sige at $x_{ij} \in \{0, 1\}$ for $i \neq j$ og $x_{ij} = 0$ for $i = j$. Hvis by j følger direkte efter by i på turen, er $x_{ij} = 1$, med $i \neq j$, ellers er $x_{ij} = 0$. Altså

[2] I praksis giver det kun mening at tillægge vægtene værdier som er endelige decimaltal, da kun disse kan repræsenteres i en computer.



Figur 2.6 Rute med og uden delcykler.

$$x_{ij} = \begin{cases} 1 & \text{hvis by } j \text{ følger direkte på by } i \\ 0 & \text{ellers} \end{cases} \tag{2.6}$$

Da hver by ankommes til og forlades netop én gang, vil der i hver søjle og hver række i X optræde et 1-tal og $(n-1)$ 0'er. Dette kan formuleres som følgende to betingelser: $\sum_{i \in V} x_{ij} = 1$ for alle $j \in V$ og $\sum_{j \in V} x_{ij} = 1$ for alle $i \in V$.

Disse betingelser alene udelukker ikke løsninger med delcykler, altså løsninger hvor en delmængde af punkterne udgør en cykel (jf. figur 2.6).

For at undgå disse løsninger opstilles endnu en betingelse.

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \qquad \forall S \subset V, 2 \leq |S| \leq n-2 \tag{2.7}$$

Denne sikrer at ingen delmængde $S \subset V$ af byerne i ruten med $2 \leq |S| \leq n-2$ er isoleret fra de resterende byer i V. Grundet betingelsen $x_{ii} = 0$ kan der ikke laves delcykler med kun èn by eller med $n-1$ byer.

En måde at løse TSP på er altså ved at minimere udtrykket

$$\sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij} \tag{2.8}$$

underlagt følgende betingelser

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \tag{2.9}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \tag{2.10}$$

$$x_{ij} \in \{0,1\} \tag{2.11}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, 2 \leq |S| \leq n-2 \tag{2.12}$$

(2.12) er en alternativ formulering af (2.7).

Denne formulering forbyder, at der i en delmængde $S \subset V$ er lige så mange kanter som der er punkter, hvilket der vil være, hvis der eksisterer en delcykel i ruten samtidig med, at betingelse (2.9) og (2.10) er opfyldt. Uanset hvilken af de to formuleringer man bruger, skal betingelsen undersøges for alle $S \subset V$ med $2 \leq |S| \leq n-2$. For en mængde V med n elementer vil S kunne vælges på $2^n - (2n+2)$ forskellige måder, da der er 2^n forskellige måder at udtage delmængder af V på, fratrukket delmængderne med $0, 1, (n-1)$ og n elementer. Dvs. at der til (2.7) og (2.12) knytter sig $2^n - (2n+2)$ bibetingelser, som skal være opfyldt [Nemhauser and Wolsey, 1988].

En tur som for et givet TSP er minimeret kaldes den *optimale* løsning. Dog er en sådan løsning i praksis ikke altid mulig at finde inden for en overskuelig tidsperiode. Derfor forsøger man ofte i stedet at finde en tilnærmelsesvis optimal løsning. Til bestemmelse af disse benyttes bestemte algoritmer, kaldet heuristikker, som beskrives indgående i kapitel 4.

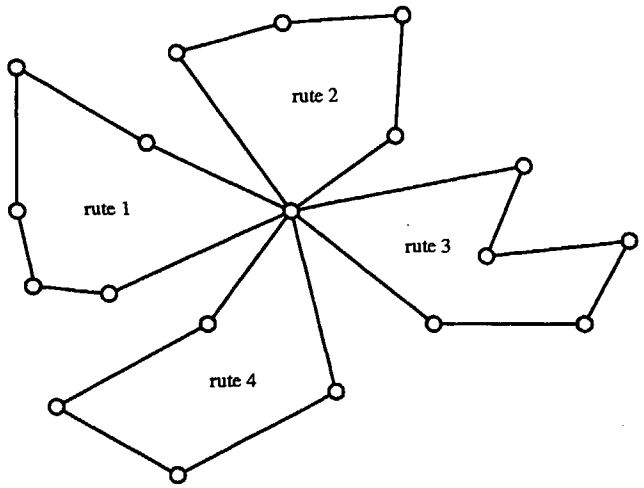
2.3 Ruteplanlægningsproblemet

Vehicle Routing Problem (VRP) er en udvidelse af TSP. Det klassiske VRP tager udgangspunkt i, at flere køretøjer (svarende til rejsende sælgere) skal levere varer til en række kunder (byer). Vognene starter alle i samme depot og skal efter endt tur vende tilbage hertil (jf. figur 2.7). Kunderne skal kun besøges én gang af én vogn. Ydermere er der en begrænsning på vognenes kapacitet, altså at de kun kan medbringe en begrænset mængde varer [Crainic and Laporte, 1998].

[Christofides, 1985a] giver følgende matematiske formulering af det klassiske ruteplanlægningsproblem. Lad $V = \{1, 2, \ldots, n\}$ være mængden af kunder og lad $i = 1$ referere til depotet. Mængden af vogne er givet ved $K = \{1, 2, \ldots, m\}$. En digraf forbinder nu kunder og depot. En kunde i har en ordrestørrelse d_i. Omkostningerne for at rejse fra i til j er givet ved c_{ij} i omkostningsmatricen C. Til enhver vogn knyttes kapaciteten Q.

I denne formulering knytter der sig en beslutningsmatrix X_k til hver enkelt vogn. Ligesom i TSP-formuleringen er her

$$x_{ijk} = \begin{cases} 1 & \text{hvis vogn } k \text{ besøger kunde } j \text{ direkte efter kunde } i \\ 0 & \text{ellers} \end{cases} \tag{2.13}$$



Figur 2.7 En mulig løsning til et VRP med 18 kunder, 4 varevogne og et depot.

I VRP-formuleringen benyttes yderligere en matrix Y, som bestemmer hvilke kunder der tildeles hvilke vogne. Her er

$$y_{ik} = \begin{cases} 1 & \text{hvis kunde } i \text{ besøges af vogn } k \\ 0 & \text{ellers} \end{cases} \tag{2.14}$$

I det følgende antages, at d_i ikke kan antage negative værdier, hvilket ville svare til at vognene skulle samle varer op hos kunderne. Ligeledes forudsættes, at der i C ikke indgår negative omkostninger. Det klassiske VRP er nu at minimere udtrykket

$$\sum_{i \in V} \sum_{j \in V} \sum_{k \in K} c_{ij} \cdot x_{ijk} \tag{2.15}$$

under følgende betingelser

$$\sum_{k \in K} y_{ik} = \begin{cases} 1, & i \in V \setminus \{1\} \\ m, & i \in \{1\} \end{cases} \tag{2.16}$$

hvor m er antallet af vogne (ruter). (2.16) sikrer at alle kunder er tildelt én og kun én vogn (på nær depotet, som besøges af alle varevogne).

$$\sum_{i \in V} d_i \cdot y_{ik} \leq Q_k \quad \forall k \in K \tag{2.17}$$

$$\sum_{i \in V} x_{ijk} = \sum_{j \in V} x_{ijk} = y_{ik} \quad \forall i, j \in V \quad \forall k \in K \tag{2.18}$$

(2.17) sikrer at en varevogn på en given tur ikke overskrider sin kapacitet, og (2.18) sikrer at en varevogn som besøger en kunde også forlader kunden igen.

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq V \setminus \{1\}, \forall k \in K \tag{2.19}$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in V, \forall k \in K \tag{2.20}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in V, \forall k \in K \tag{2.21}$$

(2.19) er den i TSP-afsnittet introducerede betingelse (2.12) om eliminering af delcykler. (2.20) og (2.21) er de ovenfor forklarede betingelser [Christofides, 1985a].

I næste kapitel vil komme en diskussion af, hvilke yderligere begrænsninger man kan lægge på det klassiske ruteplanlægningsproblem.

3 Modellering

I dette kapitel vil vi give nogle forslag til, hvordan FDB-lignende ruteplanlægningsproblemer kan modelleres. Vi vil ikke komme med en endelig formulering, men derimod angive en række virkelige aspekter og hvordan de kan inkorporeres i en eksisterende VRP-model.

3.1 Udvidelser og begrænsninger

Det klassiske VRP er, som beskrevet i kapitel 2, et problem med et depot, et antal kunder[1] og et antal vogne med samme kapacitet Q. Hver kunde skal besøges en og kun en gang. Problemet løses ved at minimere den afstand vognene skal køre. Det klassiske ruteplanlægningsproblem refereres ofte også til som *Capacitated Vehicle Routing Problem* (CVRP).

Det klassiske VRP vil ofte ikke være dækkende for virkelige ruteplanlægningsproblemer. I virkelige situationer kan der forekomme flere begrænsninger, som det er nødvendigt at tage højde for i den matematiske model. Der kan også være interesse i, at optimeringen for eksempel går på antallet af vogne i stedet for den kørte afstand. I [Madsen, 1997] gives en oversigt over nogle generaliseringer og udvidelser af VRP til brug i virkelige situationer. Nedenfor vil vi beskæftige os med nogle af disse samt flere, som vi mener kunne være relevante i FDB-lignende situationer.

Begrænsninger forbundet med lastvogne

1. Vognene kan have forskellige kapaciteter. Med kapacitet kan tænkes på både lastvægt og -volumen.
2. Visse vogne kan ikke transportere bestemte varer.
3. Nogle vogne kan ikke køre ad bestemte veje.
4. Visse vogne kan ikke køre til bestemte kunder.

I virkelige situationer kan vareudbringningen foregå med en vognpark, hvor vognene er forskellige. Disse forskellige vogne kan have forskellige kapaciteter, men også forskellige størrelser (højde, bredde m.m.). Der kan også være tale om,

[1] I dette kapitel vil vi veksle mellem kunder og butikker, som vi finder det passende.

at der kan hægtes trailere på nogle af vognene. I situationer, hvor der køres med tunge varer, skal man være mere opmærksom på at overholde en eventuel maximal lastvægt.

Vognens dimensioner kan medvirke til, at visse varer ikke kan køres i denne. Hvis varen kræver en bestemt behandling, kan den muligvis også kun køres i bestemte vogne. For eksempel skal frostvarer køres i en frostvogn, og kan ikke køres sammen med varer, som ikke tåler frost.

Hvis der mellem to kunder skal køres ad en vej med tunneller, viadukter eller lignende, kan det bevirke, at bestemte vogne (som er for brede/høje) ikke kan bruges til denne rute. Også ved butikkernes aflæsningssteder kan der forekomme begrænsninger, som udelukker brugen af bestemte vogne til disse butikker.

Tidsmæssige begrænsninger

5. Butikkerne skal modtage deres varer inden for et givet tidsinterval (*time window*) eller i et antal af tidsintervaller (*multiple time windows*).
6. Nogle ruter kan være så korte, at chaufføren på denne har mulighed for at køre en ekstra rute.
7. Chaufførerne kan være omfattet af køre- og hviletidsbestemmelser.
8. Der skal medregnes en servicetid ved kunderne.
9. Der kan være betydelig læssetid af vogne ved depotet samtidigt med, at der er et begrænset antal porte, hvorfra vognene læsses.
10. På visse tidspunkter af døgnet kan der være trafikforhold, som påvirker ruteplanlægningen.

I mange situationer vil tiden være en vigtig faktor i ruteplanlægningen, især hvis det er denne som ønskes minimeret.

Ofte har kunderne givne tidsintervaller, inden for hvilke de skal have leveret deres ordrer. Dette kan skyldes, at butikken ikke har mulighed for at modtage på alle tidspunkter af døgnet, men det kan også være en servicefaktor fra leverandørens side. Dette problem kaldes for *Vehicle Routing Problem with Time Windows* (VRPTW).

Man kan let forestille sig ruter, som er så korte, at en chauffør ikke vil have en hel arbejdsdag, hvis han nøjedes med at køre en sådan. I disse tilfælde vil det være en fordel, hvis modellen er i stand til at tildele den samme chauffør flere ruter.

Hvis en rute tager mere end et vist antal timer, er det nødvendigt at indsætte pauser til chaufføren for at overholde de eksisterende køre- og hviletidsbestemmelser. Dette vil selvfølgelig forlænge den samlede tid for ruten. Hvis ruterne er korte, og chaufføren derfor skal køre flere ruter, skal bestemmelserne stadig overholdes.

For at få en præcis angivelse af det samlede tidsforbrug, må der også medregnes en servicetid hos kunderne. Servicetiden kan for eksempel være, den tid det tager at læsse bestillingen af, og eventuelt køre varerne ind på butikkens lager. Servicetiden kan være forskellig for de forskellige butikker eller bestillinger.

Udover servicetiden hos kunderne tager det også tid at laste vognen ved depotet. Læsningstiden kan især være betydende, hvis der er flere vogne end der er porte til pålæsning. Her kan det være nødvendigt at udvikle en form for kø-styring.

Nogle kunder kan være meget besværlige at komme til i myldretider, og der skal derfor tages særligt hensyn til disse. Dette gælder især kunder, som befinder sig centrale steder i byer.

Yderligere begrænsninger og udvidelser

11. Der kan være mere end ét depot (*multi depot*).
12. Kunder kan få levering fra mere end én varevogn (*split delivery*).
13. Der kan samtidig med varelevering ske en afhentning af varer hos butikkerne (*pick-up and delivery/backhaul*).
14. Hver butik skal serviceres et givet antal gange i løbet af f.eks. en uge (*period routing*).
15. Butikkerne kan have en form for kontinuerlige bestillinger, hvor det er distributørens ansvar at sørge for at butikkernes varelager fyldes (*inventory routing*).

I det klassiske VRP findes kun et depot, hvorfra alle bestillinger køres ud. I virkeligheden kunne varerne være fordelt i flere depoter, eventuelt med forskellige varer i hvert depot. Ruteplanlægningen kunne da behandles som seperate problemer for hvert depot, eller som et samlet problem, hvor vognene kan hente varer fra flere depoter. Dette problem er bedre kendt som *Multi Depot Vehicle Routing Problem* (MDVRP).

En måde at minimere omkostningen på kunne være ved at fordele en ordre på flere vogne, så nogle kunder fik flere leveringer fra forskellige vogne. På denne måde har man bedre mulighed for at fylde vognens lastrum helt, og derved køre med færre vogne.

Udover vareleveringen kan der i ruten være steder, hvor der skal afhentes 'varer'. Dette kunne for eksempel være retur-flasker og brugte paller eller fejlleverancer og tilbagetrukne varer. Disse afhentningsvarer kan også indgå i ruteplanlægningen. Pick-up and delivery er hvis varerne kan samles op når som helst, backhauls er hvis alle leveringer skal være foretaget før en eventuel afhentning kan finde sted [Cordeau et al., 2000a].

Andre udvidelser af det klassiske VRP kunne være situationer, hvor kunderne skal have leveringer et givet antal gange i en periode (og ikke nødvendigvis

på bestemte dage), eller situationer hvor kunden har en form for kontinuerlig bestilling, så lageret altid er fyldt. I det første tilfælde kan distributøren planlægge ruterne for en hel periode ad gangen. Dette kaldes for *Periodic Vehicle Routing Problem*, PVRP. I det andet tilfælde udvides distributørens job til også at udarbejde en plan for bestillingerne.

Objektfunktionen
Objektfunktionen, som ønskes minimeret, skal ikke nødvendigvis være omkostningen. Fokus kan også ligge på at minimere antallet af varevogne eller maksimere serviceniveauet eller en kombination af disse. Serviceniveau er i sig selv ikke nogen kvantitativ størrelse og skal derfor på en eller anden måde tillægges værdier.

Hvis der ønskes at minimere antallet af vogne, vil det være oplagt at bruge *split delivery*. Her vil der nemlig være større mulighed for at fylde vognens lastrum helt. Dette kan medføre, at ordrerne kan fragtes af færre vogne, hvilket vil sige at der køres færre ruter.

Distributøren kan også vælge at optimere serviceniveauet. Service kan både ydes til kunderne og til chaufførerne. Det vil for eksempel være en service at levere ordrene indenfor et bestemt tidsinterval, eller for så vidt muligt have faste chauffører til bestemte kunder. Omvendt kan chaufføren have en fast vogn han kører i. Ydermere kan det betragtes som en service, at distributøren bruger *inventory routing*, og kunderne derved slipper for selv at indgive ordrer.

Den totale omkostning ved ruteplanen kan defineres ved tid, den kørte afstand, benzinforbruget eller noget helt fjerde. Hvis distributøren betaler et vognmandsfirma for at levere ordrene, og betalingen udregnes efter tid, må det være tiden der skal minimeres. Selv hvis der køres med egne vogne, vil tiden have en indflydelse, da der evt. skal betales timeløn til chaufførerne. Hvis der ønskes at minimere den totale afstand, der er kørt, skal der køres ad de veje der er kortest. Det vil ikke altid give den billigste rute. Det kan for eksempel være mere økonomisk at køre ad landevej end ad veje i byerne. Ofte vil det være en blanding af tid, benzinforbrug, vejlængde og andet som giver den virkelige omkostning.

En mere indgående diskussion af objektfunktionen og en metode til at bestemme denne på findes i afsnit 3.3.3.

3.2 FDBs situation

Selv om vi i rapporten bestræber os på ikke kun at dække FDBs problem, men derimod FDB-lignende tilfælde, vil vi her alligevel være lidt mere specifikke. Nedenstående beskriver FDBs situation og hvordan de løste denne, inden de investerede i en af de kommercielle programpakker til ruteplanlægning.

FDB har 3 tørvaredepoter til at dække vareleveringen i hele landet. Depotet, som er ansvarlig for vareleveringen til Sjælland, Lolland-Falster og Bornholm, er

placeret i Albertslund. Der er 20 porte i depotet, hvor pålæsningen af vognene foregår. Da depotet kun har med tørvarer at gøre, er der ingen problemer med hensyn til, at visse varer ikke kan køres sammen. FDB optæller deres leveringsmængde i paller, hvor de på en uge kører 8000-8500 paller ud. I forhold til varer leveret, afhentes i dag kun en meget lille mængde varer.

Indtil for nylig har FDB brugt et vognmandsfirma, med 60-70 vogne (som ikke alle er af samme type), til at klare deres ruteplanlægning. Vognene leverer kun varer fra dette depot, der er altså ikke tale om multidepoter. Derudover får hver butik kun levering fra én vogn, det vil sige, at der ikke er benyttet split delivery.

Tørvaredepotet i Albertslund har tilknyttet 313 butikker, som alle har fået tidsvinduer med en bredde på to til tre timer. Hver butik får i gennemsnit 2,4 leveringer på en uge; det præcise antal leveringer pr. uge afhænger dog af butikkens størrelse, art og placering. Dette svarer til at tørvaredepotet får gennemsnitlig 110 ordrer pr. dag. Af disse butikker er en stor del af dem såkaldte 'større' butikker, som har store ordrer. Dette medfører at ruteplanlægningen ofte indeholder vogne, som af kapacitetsmæssige årsager kun kører til én butik. For at dække alle ordrerne, kører FDB varer ud to til tre gange i døgnet, dog bliver alle ruter planlagt på en gang. Hver enkelt vogn kan derfor godt køre mere end én rute på en dag.

Bestillinger af varer og udbringelse af dem sker på en dag-til-dag basis. Når en butik indgiver en ordre, bliver den pågældende butik inddraget i ruteplanlægningen den følgende dag.

Chaufførerne, som er ansat hos vognmandsfirmaet, er omfattet af forskellige køre- og hviletidsregler. Selvom ruterne oftest er korte og ikke overskrider det maksimale antal tilladte timers kørsel, skal der køres flere ruter på en dag, og der skal derfor indlægges pauser til dem.

Inden FDB købte en VRP-programpakke, lagde de deres ruter 'i hånden'. Mere præcist foregik dette ved, at vognmanden fik de forskellige butikkers ordre og derefter manuelt lagde ruterne.

3.3 Opbygning af modellen

Ovenstående beskriver FDBs situation, når de ikke bruger en matematisk model til at løse deres ruteplanlægningsproblem. I dette afsnit vil vi diskutere hvorvidt de i afsnit 3.1 nævnte udvidelser og begrænsninger er relevante i FDB-lignende situationer og hvordan de eventuelt kan modelleres.

3.3.1 Matematisk formulering

I dette afsnit vil vi formulere en VRPTW-grundmodel samt komme med anvisninger til hvordan denne kan udvides til at omfatte flere af de i afsnit 3.1

nævnte udvidelser og begrænsninger.

Nødvendigheden af tidsstyring
Hvis et ruteplanlægningsværktøj skal bruges til decideret planlægning af vareudkørsler er det i FDB-lignende situationer oftest en bydende nødvendighed, at der er tidsstyring. Det skyldes flere ting:

- Tidsvinduer:
  Hvis der ikke er mulighed for at levere varer på et vilkårligt tidspunkt er tidsvinduer nødvendige.

- Køreplaner:
  Skal ruteplanlægningsværktøjet bruges til at producere køreplaner, som de enkelte chauffører skal følge er det nødvendigt at kunne angive start- og sluttider, da alle chaufførerne pga. kapacitetsbegrænsninger ved depotet ikke er i stand til at starte samtidigt. Herudover skal det også være muligt at kunne angive pauser, hvis køre- og hviletidsbestemmelser skal overholdes.

- Flere ruter:
  Er der flere korte ruter, som med fordel kan køres af den samme chauffør er det nødvendigt at disse ikke bliver lagt, så de skal køres på samme tid.

Det er altså nødvendigt at have en model, som kan håndtere tidsaspektet. Som udgangspunkt kan bruges en VRPTW-model. VRPTW er, som tidligere nævnt, en udvidelse af CVRP. I VRPTW skal besøget hos kunden starte i et givet tidsinterval, og vognen skal forblive hos kunden under besøget. *Bløde* tidsvinduer kan overskrides for en vis pris, mens *hårde* tidsvinduer ikke tillader en vogn at ankomme til en kunde efter tidsintervallets ophør. I tilfældet med hårde tidsvinduer skal vognen, hvis den ankommer for tidligt, vente indtil tidsvinduet 'åbner'. I det følgende vil vi koncentrere os om VRPTW med hårde tidsvinduer.

Der findes mange forskellige formuleringer af VRPTW. Vi har valgt at tage udgangspunkt i én givet i [Cordeau et al., 2000a]. Denne formulering giver mulighed for udvidelser og ekstra begrænsninger, som vi vil se i et følgende afsnit.

VRPTW er netværket $D = (V, A)$, hvor $V = \{0, 1, \ldots, n, n+1\}$ er mængden af kunder samt depot, A er mængden af retningsorienterede veje og K er mængden af vogne. Depotet er repræsenteret ved *to* punkter; 0 og $n+1$. Desuden defineres mængden $N = \{1, 2, \ldots, n\} = V \setminus \{0, n+1\}$ som værende mængden af kunder. En nødvendig betingelse for at en rute i D er lovlig er, at den starter i 0 og ender i $n+1$. Tidsvinduer repræsenteres ved $[a_i, b_i]$, hvor a_i er det tidligst mulige ankomsttidspunkt hos kunde i og b_i det senest mulige. Også depotet får tildelt et vindue; $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, hvor E er den tidligst mulige afgang fra depotet og L er den senest mulige ankomst til depotet. Ordrestørrelser for kunde i betegnes med d_i og servicetiden med s_i. Ordrestørrelse og servicetid for depotet er defineret til at være nul.

$$d_0 = d_{n+1} = s_0 = s_{n+1} = 0 \tag{3.1}$$

Lovlige løsninger eksisterer kun hvis

$$E \leq \min_{i \in N}(b_i - t_{0i}) \tag{3.2}$$

$$L \geq \max_{i \in N}(a_i + s_i + t_{i(n+1)}) \tag{3.3}$$

hvor t_{ij} er rejsetiden fra kunde i til kunde j. Det tillades at en kant $(i,j) \in A$ kan elimineres, grundet tidsmæssige overvejelser hvis $a_i + s_i + t_{ij} > b_j$, eller grundet kapacitetsbegrænsninger $d_i + d_j > Q_k$, hvor Q_k er kapaciteten af vogn k. Ydermere skal nævnes, at når det tillades vogne at forblive i depotet, specielt i tilfældet hvor problemet består i at minimere antallet af benyttede vogne, skal kanten $(0, n+1)$ lægges til mængden af kanter, altså $A \cup \{0,(n+1)\}$.

I den matematiske formulering introduceres også de to typer variable: Beslutnings- og tidsvariable. Beslutningsvariable er defineret som

$$x_{ijk} = \begin{cases} 1 & \text{hvis vej } (i,j) \text{ køres af vogn } k \\ 0 & \text{ellers} \end{cases} \tag{3.4}$$

for $(i,j) \in A$ og $k \in K$.

Tidsvariable kaldes w_{ik} hvor $i \in V$ og $k \in K$. w_{ik} er ankomsten hos kunde i for vogn k. VRPTW beskrives nu som følgende minimeringsproblem

$$\sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{3.5}$$

hvor c_{ij} er omkostningen ved at køre fra kunde i til kunde j underlagt følgende betingelser

$$\sum_{k \in K} \sum_{j \in V \setminus \{i\}} x_{ijk} = 1 \qquad \forall i \in N \tag{3.6}$$

(3.6) sikrer at hver kunde bliver placeret i netop én rute, altså at ruterne ikke overlapper.

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall k \in K \tag{3.7}$$

$$\sum_{i \in N \setminus \{j\}} x_{ijk} - \sum_{i \in N \setminus \{j\}} x_{jik} = 0 \qquad \forall k \in K, \forall j \in N \tag{3.8}$$

$$\sum_{i \in N} x_{i(n+1)k} = 1 \qquad \forall k \in K \tag{3.9}$$

(3.7) sikrer at en vogn, når den forlader depotet, kun kan køre til én kunde og (3.9) at den kun kan ankomme til depotet fra én kunde. (3.8) sikrer at en vogn, som kommer til en kunde, også forlader denne igen.

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \qquad \forall k \in K, \forall i, j \in V \quad (3.10)$$

$$a_i \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \leq w_{ik} \leq b_i \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \qquad \forall k \in K, \forall i \in V \quad (3.11)$$

(3.10) er kun interessant når kunde j følger på kunde i (ellers er $x_{ijk} = 0$). Den sikrer at ankomsttiden w_{jk} hos j er større end eller lig summen af ankomsten w_{ik} hos i, servicetiden s_i hos i samt rejsetiden t_{ij} fra i til j. (3.11) sikrer at ankomsttidspunktet hos kunde i ligger inden for tidsvinduet, bemærk at denne betingelse tvinger $w_{ik} = 0$ når kunde i ikke besøges af vogn k.

$$E \leq w_{ik} \leq L \qquad \forall k \in K, \forall i \in \{0, n+1\} \quad (3.12)$$

$$\sum_{i \in N} \sum_{j \in V \setminus \{i\}} d_i x_{ijk} \leq Q_k \qquad \forall k \in K \quad (3.13)$$

(3.12) sikrer at afgangs- og ankomsttidspunktet hos depotet, og dermed hos alle kunderne, ligger indenfor depotets tidsvindue. (3.13) sørger for, at den samlede ordrestørrelse i en given rute ikke overstiger kapaciteten af en pågældende vogn.

3.3.2 Inkorporering af begrænsninger

I dette afsnit vil vi diskutere hvorvidt en matematisk model kan tage forbehold for de i afsnit 3.1 listede begrænsninger, som vi finder relevante.

Kapacitetsbegrænsning
At vognene kan have forskellige kapaciteter indgår allerede i den valgte matematiske formulering af VRPTW. At kapaciteten er knyttet til hver enkelt vogn k må i øvrigt siges at være oplagt, da det er urealistisk at forestille sig at en virksomhed har en vognpark af helt ens vogne.

FDB løser kapacitetsbegrænsninger ved udelukkende at regne i paller; en Europa-palle har en fast størrelse, og der kan være et vist antal paller i en bestemt vogn k. Mere generelt kan man sige, at en vare har to egenskaber; vægt og volumen. Man kan således opfatte en ordre fra en given kunde, som værende *to* ordrer, en omfattende den pågældende vares vægt og en anden dens volumen. Hvis der både er vægt og volumen knyttet til varerne, er det nødvendigt at have to begrænsninger af samme type som (3.13) på vognene.

Transportbegrænsninger
At nogle vogne ikke kan transportere bestemte varer, opfatter vi som værende irrelevant for FDB-lignende situationer. Argumentet for dette er at alle varer på FDBs lager i Albertslund er tørvarer, hvorfor de så udmærket kan transporteres i samme last. Situationer med denne slags begrænsninger opfatter vi ikke som FDB-lignende.

Vej- og kundebegrænsninger
Problemet med vejbegrænsninger er praktisk talt ikke eksisterende. Selvfølgelig vil nogle lastvogne f.eks. ikke kunne køre under Knippelsbro osv., men dette problem kan løses ved en ekstra begrænsning i shortest-path algoritmen i GIS-systemet (se afsnit 3.3.3) .

En begrænsning på bestemte kunder er derimod mere relevant, og en sådan kan indføres i modellen som værende blot en ekstra begrænsning knyttet til den matematiske formulering. Vi har modelleret denne begrænsning ved at indføre U_{ik} hvor

$$U_{ik} = \begin{cases} 1 & \text{hvis kunde } i \text{ er ulovlig for vogn } k \\ 0 & \text{ellers} \end{cases} \tag{3.14}$$

for $i \in N$ og $k \in K$.

Betingelserne opskrives på følgende måde

$$\sum_{i \in N} \sum_{j \in V \setminus \{i\}} x_{ijk} \cdot U_{ik} = 0 \quad \forall k \in K \tag{3.15}$$

Det ses at (3.15) kun er opfyldt for lovlige kunder. Thi tilbagelægges vejen fra kunde i til kunde j af vogn k, og er kunde i ulovlig for netop vogn k da vil summerne ovenfor være forskellige fra 0, hvorfor betingelserne ikke vil være opfyldt.

En anden og mere elegant måde at indføre kundebegrænsninger i den matematiske model på, vil være ved at udnytte de tidsvariable, som allerede er inkorporeret i modellen. Vi definerer U_{ik} på fuldstændig analog vis og husker at tidsvariablen $w_{ik} = 0$ hvis vogn k ikke ankommer til kunde i. Betingelsen lyder nu

$$\sum_{i \in N} U_{ik} \cdot w_{ik} = 0 \qquad \forall k \in K \tag{3.16}$$

Det ses, at vi igen undgår ulovlige kunder på vogn k's rute. (3.16) er opfyldt hvis vogn k ikke besøger kunde i, eller hvis kunde i er tilladt at besøge for vogn k.

Tidsvinduer

Begrænsningen med tidsvinduer, er netop en del af den VRPTW-model, som vi har taget udgangspunkt i (jf. afsnit 3.3.1).

Multiple time windows kan indføres ved at tilføje ekstra tidsintervaller til kunderne. Begrænsningen i ligning (3.11) skal så modificeres til at acceptere løsninger hvis en kunde bliver besøgt af en vogn i et og kun et af kundens tidsvinduer.

Flere ruter

Som modellen er formuleret i afsnit 3.3.1, har hver vogn kun fået tildelt én rute, forstået på den måde, at når vognen er færdig med ruten, bliver den ikke brugt mere. Det er oplagt, at det nogle gange kan være nødvendigt, at en enkelt chauffør kører flere korte ruter på en enkelt dag.

Vi kan i stedet for at give en vogn to ruter efter hinanden, formulere det som to vogne, hvor vogn nummer 2 først kan køre fra depotet, efter at vogn nummer 1 er vendt tilbage. De forskellige vogne skal dog kobles. Vi kan f.eks. sige, at en vogn k defineres som k_p, hvis der er tale om vognens p'te rute. Begrænsningen har vi modelleret som følger

$$w_{(n+1)k_p} + s_0 \leq w_{0k_{p+1}} \tag{3.17}$$

hvor $w_{(n+1)k_p}$ er ankomsttidspunktet for vognens p'te rute, s_0 er servicetiden hos depotet og $w_{0k_{p+1}}$ er afgangstidspunktet for vognens $(p+1)$-rute. Det bemærkes at denne begrænsning kræver servicetid hos depotet, hvilket er en udvidelse i forhold til den originale model.

Det vil være en fordel at bruge så få chauffører i løbet af en dag som muligt, da det vil være dårlig service overfor chaufførerne kun at give dem halve arbejdsdage. Antallet af chauffører kan minimeres ved at tilføje en straf til omkostningsfunktionen for hver ny vogn k som medtages i løsningen. k_1, k_2 og k_3 osv. skal ikke give anledning til hver deres straf men kun én.

Køre- og hviletidsbestemmelser

Køre- og hviletidsbestemmelser er særdeles vigtige at indbygge i modellen idet disse skal overholdes hvis ikke lovgivningen skal brydes. Køre og hviletidsbestemmelser overholdes ved, at chaufførerene holder pauser på ruten, så de ikke kommer til at køre i for lang tid ad gangen.

Vi mener at køre- og hvilebestemmelser i mange tilfælde kan være nødvendige at tage højde for. Hvis modellen benyttes af virksomheder hvor de enkelte kunders ordrer er mindre, og ruterne dermed bliver længere, kan det meget vel blive aktuelt at indlægge pauser. På denne baggrund mener vi, at en modellering skal være i stand til at kunne tage højde for pauser.

Vi har selv modelleret denne udvidelse, men har hentet inspiration til metoden fra [Cordeau et al., 2000a]. Metoden kan generaliseres til at indlægge vilkårligt

mange pauser, men vi vælger af hensyn til overskueligheden at vise den for en enkelt.

Ideen i metoden er, at der laves en kopi af alle kunderne, hvor kopierne er identiske med de originale kunder på alle måder. Mængden N_1 (de originale kunder) består af de kunder, som bliver besøgt før en eventuel pause og N_2 (kopierne) er de kunder, som bliver besøgt efter. Nu har vi $N^* = N_1 \bigcup N_2$. På tilsvarende måde defineres V_1, V_2 og $V^* = V_1 \bigcup V_2$.

For at undgå at de nu dobbelt så mange kunder alle bliver rutelagt, skal ligning (3.6) modificeres således at en original kunde og dennes kopi ikke begge bliver besøgt i samme løsning. Dette kan gøres således:

$$\sum_{k \in K} \left(\sum_{j \in V^* \setminus \{i_1\}} x_{i_1 jk} + \sum_{j \in V^* \setminus \{i_2\}} x_{i_2 jk} \right) = 1 \qquad \forall i_1 \in N_1, \forall i_2 \in N_2 \quad (3.18)$$

hvor i_1 svarer til den originale og i_2 svarer til kopien af en given kunde. Der skal desuden indføres ekstra begrænsninger, som sørger for, at der ikke arbejdes uden pause i længere tid end tilladt og at længden på pausen er af en vis størrelse. Den længste tilladte arbejdstid uden pauser kaldes T_p og pauselængden kaldes t_p.

For at medregne den tid det tager at afholde en pause, udvider vi (3.10) til tre separate begrænsninger.

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \qquad \forall k \in K, \forall i \in V_1, \forall j \in V_1 \quad (3.19)$$

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \qquad \forall k \in K, \forall i \in V_2, \forall j \in V_2 \quad (3.20)$$

$$x_{ijk}(w_{ik} + s_i + t_{ij} + t_p - w_{jk}) \leq 0 \qquad \forall k \in K, \forall i \in V_1, \forall j \in V_2 \quad (3.21)$$

(3.19) og (3.20) er magen til (3.10), og træder i kraft hvis både afgangskunde i og ankomstkunde j befinder sig i enten V_1 eller V_2. (3.21) sikrer at pausen bliver afholdt hvis chaufføren bevæger sig fra en kunde $i \in V_1$ til en kunde $j \in V_2$.

Dette er ikke nok. Vi har brug for en begrænsning, som sikrer, at chaufføren skifter fra V_1 til V_2 hvis og kun hvis pausen bliver afholdt. Til dette tilføjer vi to ekstra begrænsninger.

$$(w_{jk} + s_j - w_{0k}) \left(\sum_{i \in V^* \setminus \{j\}} x_{ijk} \right) \leq T_p \left(\sum_{i \in V^* \setminus \{j\}} x_{ijk} \right) \forall k \in K, \forall j \in V_1 \quad (3.22)$$

$$T_p \left(\sum_{i \in V^* \setminus \{j\}} x_{ijk} \right) < (w_{jk} + s_j - w_{0k}) \left(\sum_{i \in V^* \setminus \{j\}} x_{ijk} \right) \forall k \in K, \forall j \in V_2 \quad (3.23)$$

(3.22) sikrer at en chauffør ikke kan besøge en kunde $j \in V_1$ hvis den maksimalt tilladte uafbrudte arbejdstid herved overskrides. Hvis dette sker kan chaufføren kun køre til kunder $j \in V_2$. (3.23) sikrer at der ikke er mulighed for at køre til kunder $j \in V_2$, medmindre der har været behov for en pause.

Servicetid
Servicetid af kunderne er allerede med i modellen for VRPTW. Dette aspekt må iøvrigt anses for at være vigtigt for en eventuel tidsstyring. F.eks. kan man forestille sig, at det tager lige så lang tid at losse en vogn hos en pågældende kunde, som rejsetiden mellem to kunder. En oplagt udvidelse vil være, at gøre servicetiden vognafhængig således at s_i bliver udvidet til s_{ik}. En anden mulig udvidelse er at gøre servicetiden afhængig af ordrestørrelsen, dvs. s_{d_i}. Som vi har set kan det være nødvendigt at tilføje en servicetid ved depotet, hvis vogne skal køre flere ruter.

Begrænsninger knyttet til depotet
Problemet med at der ved depotet kan være en betydelig læssetid af vognene, og at der ydermere kun er et begrænset antal porte til pålæsning, må anses for at være relevant i vores tilfælde, da FDB f.eks. kun har 20 porte.

En måde at indføre dette, kunne være ved at give hver enkelt vogn k et seperat depot med eget tidsvindue $[E_k, L_k]$. Nu kan man så definere disse tidsvinduer på en sådan måde, at man undgår, at der på noget tidspunkt er flere vogne i samme port. Selve modelleringen af dette har vi ikke beskæftiget os med.

Trafikforhold
Det kan ske, at én rute er den hurtigste på et tidspunkt, men ikke er det på et andet. Hvis der skal tages højde for tidsafhængige trafikforhold er det nødvendigt, at kørselstiden mellem kunder bliver tidsafhængig, dvs. kørselstiden skal være af formen $t_{ij\tau}$, hvor τ er tiden. Hvis omkostningsfunktionen afhænger af tidsforbruget, skal omkostningsmatricen ligeledes udvides, altså $c_{ij\tau}$.

Multidepot
Modelleringen af MDVRP benyttes mest i tilfælde hvor en vogn k kan starte i et depot og ende i et andet. Dette er ikke en mulighed i vores tilfælde, hvorfor vi finder en eventuel modellering af denne VRP-variant irrelevant.

Split delivery
Split deliveries er en spændende mulig udvidelse. Vi har dog ikke fundet eller kunnet konstruere udvidelser, som kan bruges til vores VRPTW-formulering.

Pick-up and delivery / backhaul
Pick-up and delivery er problemet, hvor afhentning af varer på ruten sker samtidig med, at der leveres varer. Vi vil koncentrere os om backhaul, som er en

variant af dette problem. Her skal alle leveringer være foretaget, førend afhentningerne kan finde sted. Da FDBs lastvogne læsses bagfra og ikke fra siden, er denne variant den mest hensigtsmæssige. En vogn pakkes jo på en sådan måde, at de varer som skal losses først læsses sidst, hvilket forhindrer at varer kan samles op og leveres i valgfri rækkefølge. Modelleringen af backhaul, som præsenteres her, stammer fra [Cordeau et al., 2000a].

Til at modellere backhaul defineres lastvariablen l_{ik} for $i \in V$ og $k \in K$. Denne beskriver samlet afleveret last for vogn k efter endt besøg hos kunde i. Med denne kan kapacitetsbegrænsningen (3.13) omskrives, så formen af den er den samme som tidsbegrænsningen (3.10).

$$x_{ijk}(l_{ik} + d_j - l_{jk}) \leq 0 \qquad \forall k \in K, \forall i, j \in V \tag{3.24}$$

(3.24) sikrer, at lastvariablen hos j er større end eller lig med summen af lastvariablen hos i og ordren hos j. Denne er dog ikke nok til at modellere backhaul og der indføres derfor

$$d_i \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \leq l_{ik} \leq Q_k \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \qquad \forall k \in K, \forall i \in N \tag{3.25}$$

$$l_{0k} = 0 \qquad \forall k \in K \tag{3.26}$$

$$0 \leq l_{(n+1)k} \leq Q_k \qquad \forall k \in K \tag{3.27}$$

(3.25) sikrer at l_{ik} er større end d_i hvis der køres til kunde i og at l_{ik} ikke på noget tidspunkt overskrider kapaciteten af vognen. (3.26) og (3.27) sikrer, at lastvariablen ved afgang fra og ankomst til depotet er hhv. 0 og mindre end eller lig Q_k.

Herefter deles N op i to delmængder af kunder; N_D og N_P. Hvor N_D er de kunder som skal have leveret varer, og N_P de kunder som skal have afhentet varer. En kunde kan ikke både tilhøre N_D og N_P. Hvis en kunde både skal have leveret og afhentet varer, må den anses som to forskellige kunder. Fra nu af vil l_{ik} ikke kun betegne afleveret last men summen af afleveret og opsamlet last, begge regnet positivt. Nu skal (3.25) udskiftes med to versioner af den samme.

$$d_i \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \leq l_{ik} \leq Q_k \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \qquad \forall k \in K, \forall i \in N_D \tag{3.28}$$

$$(Q_k + d_i) \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \leq l_{ik} \leq 2Q_k \left(\sum_{j \in V \setminus \{i\}} x_{ijk} \right) \quad \forall k \in K, \forall i \in N_P \tag{3.29}$$

og (3.27) med

$$Q_k \leq l_{(n+1)k} \leq 2Q_k \qquad \forall k \in K \tag{3.30}$$

Af betingelserne (3.28), (3.29), (3.30) og (3.24) ses, at når al last til levering er afleveret, genskabes Q_k nye lastkapacitetsenheder til at tage sig af afhentningen.

Periodisk VRP og inventory routing
Hos FDB og formenligt også hos andre FDB-lignende virksomheder foregår bestillingen og udbringningen af varer som beskrevet i afsnit 3.2 på en dag-til-dag basis. Periodisk routing er altså ikke aktuelt.

Inventory routing er ligesom split deliveries en interessant udvidelse, som vi desværre ikke har kunnet formulere.

3.3.3 Omkostningsfunktionen

Vi vil i dette afsnit diskutere hvordan, vi mener, omkostningsfunktionen bør opbygges. Den simpleste af alle omkostningsfunktioner er givet ved den direkte afstand mellem to kunder. Hvis depotet og kunderne betragtes som værende punkter i planen hver med et tilhørende koordinatpar, er det let at plotte depotet og kundernes placering samt beregne deres indbyrdes euklidiske afstand. At kunne plotte byerne gør det lettere at fange eventuelle fejl i data-grundlaget for ruteplanlægningen, end hvis man blot har en omkostningsmatrix.

Da der sjældent findes direkte veje fra den ene kunde til den anden, er den direkte vej selvfølgelig en urealistisk målestok. En måde at gøre denne approksimation bedre på er ved at øge hver vejlængde med en vis faktor, f.eks. 15% (eller mere). Dette virker udmærket hvis vejnettet er nogenlunde homogent, men skaber problemer hvis nogle vogne skal passere områder, som ikke er vejbelagt (f.eks. vand eller øde landområder) [Hall and Partyka, 1997].

Afstandsberegninger af *den direkte vej* bruges mindre og mindre pga. ovenfor nævnte upræcisheder og infleksibilitet. Metoden giver f.eks. ikke mulighed for at angive faktiske ruter på rigtige vejkort. I [Hall and Partyka, 2000] har 23 leverandører af ruteplanlægnings-software svaret på spørgsmål vedr. deres produkter. Ingen af dem brugte denne form for afstandsbestemmelse.

Det som bruges i de kommercielle pakker er såkaldte GIS (Geographic Information System). Et GIS er en stor database, som indeholder optegnelser af et helt vejnet (opbygget som et matematisk netværk). Vejkryds, rundkørsler, motorvejsfliet mv. angives som knuder og diverse veje angives som kanter i dette netværk. Systemet kan indeholde informationer om vejlængder, hastighedsbegrænsninger på veje, ensrettede gader og lignende. Ruten fra en kunde til en anden kan findes ved at løse en shortest-path algoritme [2] for netværket.

[2] En shortest-path algoritme er i stand til at finde den korteste vej mellem to punkter i et netværk. Denne algoritme har en beregningskompleksitet på $\mathcal{O}(n^2)$ [Dolan and Aldous, 1995]. Beregningskompleksiteter vil blive gennemgået mere indgående i kapitel 4.

Med hensyn til optimering af ruter er det mest interessante ved GIS måske, at et sådant system er i stand til at angive langt mere præcise afstande end den direkte. Kan rutelængderne angives mere præcist svarer modellen, hvori afstandene bruges, bedre overens med virkeligheden. Gode løsninger til modellen vil derfor have en større chance for at være gode løsninger til det faktiske problem.

Omkostningerne ved transport afhænger i virkeligheden ikke kun af, hvor langt vognene kører, men af benzinforbruget på de respektive veje. Kørsel på motorvej eller kørsel med mange stop medfører et større benzinforbrug pr. kilometer end jævn kørsel. En omkostningsfunktion, som kan tage højde for disse faktorer vil derfor være en bedre modellering end en 'ren' afstandsfunktion. Det er muligt at modellere sådanne hensyn i et GIS ved at give de forskellige kanter i netværket forskellige egenskaber, såsom 'motorvej' eller 'indre by'.

Ud over rutens længde og beskaffenhed har den tid det tager at køre den også stor indflydelse på omkostningerne. Dette skyldes at chaufførerne skal have løn for deres arbejde. Igen vil man få en forbedring af modelleringen, hvis der i omkostningsfunktionen kan tages højde for tidsforbrug.

Omkostningsfunktionen har ikke nødvendigvis indflydelse på hvordan løsningen af modellen foregår, men er vigtig hvis modellen skal ligne virkeligheden. Man må derfor argumentere for, at en omkostningsfunktion er så realistisk som mulig.

Problemer ved realistiske omkostningsfunktioner

En realistisk omkostningsfunktion giver som nævnt en bedre modellering end en simpel, men der kan være problemer forbundet med at benytte en sådan.

Hvis omkostningen ved at køre et bestemt vejstykke både afhænger af benzinforbruget, og tiden det tager at køre vejen, er det nødvendigt at udføre beregninger, som kan bestemme den faktiske omkostning ved at køre ad vejen. Disse beregninger kan ikke blot udføres en enkelt gang for hele GIS-netværket. Det skyldes, at de indgående parametre ændrer sig med tiden. Lønninger og benzinpriser er ikke en statisk størrelse. Vejnettet ændrer sig pga. vejarbejde, trafikreguleringer og lignende.

Men der er mere end dette, der komplicerer situationen. Omkostninger som benzinforbrug varierer alt efter hvilken type vogn, der er tale om, og der skal måske udføres beregninger for enhver vogntype.

Jo flere parametre der tages højde for, jo mere øges risikoen for fejl. Det kunne f.eks. være at en landevej fejlagtigt markeres som 'indre by' eller at en motorvej fik påhæftet mærkatet 'ensrettet'. Sådanne fejl er utrolig svære at finde for en ruteplanlægger da sådanne oplysninger kan ligge skjult i bagvedliggende databaser. Fejl kan selvfølgelig fanges hvis ruteplanlægningen på et tidspunkt ser meget mærkelig ud, og det undersøges hvorfor programmet ikke har valgt en oplagt bedre rute.

Omkostningsfunktionen i vores tilfælde

Vores formål med projektet er at forsøge at finde den løsning til en FDB-lignende

situation, som giver den laveste værdi af omkostningsfunktionen. En omkostningsfunktion baseret på et GIS-system giver selvfølgelig de bedste løsninger. Indflydelsen ved at bruge et GIS-system, er ikke noget vi har haft mulighed for at undersøge, men virker oplagt. Derudover ser vi det som en klar indikation, at alle kommercielle pakker med ruteplanlægningssoftware understøttes af et eller flere GIS-systemer.

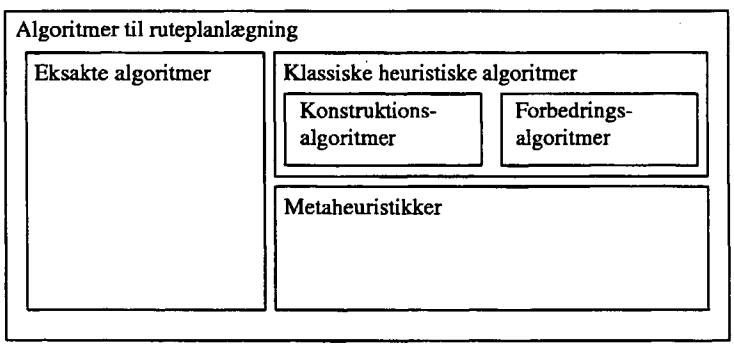
Vi er ikke klar over om omkostningsfunktioner, der tager højde for en blanding af afstand, tids- og benzinforbrug, anvendes i praksis, men GIS-systemerne må kunne udvides, så det bliver muligt.

4 Algoritmer til ruteplanlægning

Efter at et VRP-tilfælde er blevet formuleret, skal det løses, hvilket vil sige, at der skal dannes nogle ruter, som opfylder den opstillede models krav. Med et problem menes en overordnet VRP-situation, dvs. et VRP med nogle givne begænsninger. Et problemtilfælde er, hvor problemets specifikke begrænsningsværdier, antallet af kunder/vogne og placering af kunderne er angivet.

Der findes en lang række algoritmer der kan løse VRP-tilfælde. Vi vil i dette kapitel forsøge at give et overblik over disse, samt vurdere hvilken type af algoritmer, som vil være velegnet til løsning af virkelighedstro problemer som vores. Vi vil selvfølgelig primært beskæftige os med VRP-algoritmer, men vil også medtage TSP-algoritmer, da disse er nært beslægtede. Desuden anvendes visse TSP-algoritmer også i forbindelse med løsning af VRP.

Der skelnes overordnet mellem eksakte algoritmer, og approksimative, også kaldet heuristiske algoritmer. De eksakte finder den absolut bedste løsning [1], men bruger meget tid på at finde den. De heuristiske garanterer ikke at finde den bedste, men kan komme med en løsning inden for overkommelig tid. I dette kapitel vil vi beskæftige os med disse to typer af algoritmer. Gennemgangen af heuristikker vil være delt op i hhv. de klassiske heuristikker og meta-heuristikkerne, som er en form for udvidelse af de klassiske.



Figur 4.1 Overordnet struktur af algoritmer til løsning af VRP.

[1] Vi tillader os nu, og i det efterfølgende at sige 'den bedste løsning', selv om der udemærket kan findes mere end én optimal løsning til et problemtilfælde.

4.1 Eksakte algoritmer

De eksakte algoritmer er karakteriseret ved, at de altid finder den optimale løsning, også kaldet den *eksakte* løsning.

Den simpleste eksakte algoritme går i al sin enkelthed ud på at lave en liste over samtlige mulige ruter for derefter at udvælge den bedste. Denne metode kaldes *enumeration*, og garanterer selvfølgelig at finde den bedste løsning. Ulempen ved denne metode er, at hvis problemet bliver tilstrækkelig stort, vil der være ufatteligt mange ruter. Har vi et TSP med n byer for $n \geq 3$, vil antallet af mulige ruter blive $^{(n-1)!}/_{2}$.

Grunden til dette er følgende: Da løsningen til et TSP udgør en hamiltonisk kreds, er det lige meget hvilken byer den rejsende sælger starter (og slutter) i. 'Startbyen' kan altså vælges frit. Fra startbyen kan han vælge at rejse til en af de $(n-1)$ andre byer. Dernæst kan han vælge $(n-2)$ osv. indtil han har besøgt alle byerne og vender tilbage. Dette giver i alt $(n-1)!$ mulige ruter. Der vil dog være to af alle ruter, da der for hver rute vil eksistere en identisk, hvor byerne blot er besøgt i den omvendte rækkefølge. Derfor bliver antallet af forskellige ruter $^{(n-1)!}/_{2}$.

Lad os se på hvad ovenstående betyder i praksis. Hvis vi har et TSP med 15, byer bliver der i alt omkring $4,4 \cdot 10^{10}$ ruter. Har vi derimod 20 byer, får vi $6,1 \cdot 10^{16}$ og 21 byer giver $1,2 \cdot 10^{18}$ ruter. Hvis vi har en computer, der kan opskrive en milliard løsninger i sekundet, ville det for 15 byer tage 44 sekunder at opskrive samtlige ruter. For 20 byer ville det tage 2 år, og for 21 byer 38 år. Her kan enhver se, at 'brute force' enumeration ikke er en holdbar metode i praksis.

Det at antallet af mulige løsninger i et kombinatorisk problem vokser voldsomt som funktion af problemstørrelsen, er hvad man kalder for den *kombinatoriske eksplosion*.

Der findes eksakte algoritmer, som ikke er så 'dumme', at de skal liste samtlige ruter op for at finde den eksakte løsning. Den tid, det tager for dem at finde løsningen, vokser dog stadigt voldsomt, når antallet af byer vokser.

Et eksempel på en eksakt algoritme er *branch and bound*. Denne fungerer, kort skitseret ved succesivt at dele problemer op i mindre og mindre underproblemer. Således beregnes nedre grænser (korteste ruter) på objektfunktionen for hvert underproblem, ved at relaksere begrænsningerne for underproblemet. Er det relakserede underproblems nedre grænse ikke lavere end det ikke-relakseredes, vil underproblemet (og dermed deres underproblemer) kunne udelades som løsningsgrundlag for en optimal løsning, og løsningsrummet bliver derfor mindre. For en grundigere gennemgang af branch and bound og andre eksakte algoritmer se [Dolan and Aldous, 1995], [Balas and Toth, 1985] og [Madsen, 1997].

4.1.1 Beregningskompleksitet

Vi vil her introducere begrebet beregningskompleksitet, som er et redskab til at formalisere hvad der menes med, at en algoritme er hurtig eller langsom. Med dette redskab kan vi bl.a. konkretisere hvad der menes med, at beregningstiden for eksakte algoritmer vokser voldsomt, når antallet af byer vokser.

Beregningstid
Beregningstiden for algoritmer, som bruges til kombinatoriske problemer, øges næsten altid hvis 'størrelsen' af problemet øges. Med størrelsen af et problem forstås størrelsen af en eller anden karakteristisk parameter n. For TSP og VRP er n antallet af byer eller kunder.

Tiden, det tager en algoritme at løse et problem af en given størrelse, kan variere meget fra problemtilfælde til problemtilfælde. Det vil derfor være umuligt at angive en fast sammenhæng mellem beregningstid og n. Det der i stedet gøres er at forsøge at angive en ***værste tilfælde*** beregningstid, således at man har en øvre grænse, for tiden det tager at løse problemet. Denne 'værste tilfælde' funktion af n kaldes for ***tidskompleksitetsfunktionen.***

Den faktiske tid, det tager en algoritme at løse et problemtilfælde, er meget svær at afgøre. Beregningstiden afhænger bl.a. af den benyttede computer og programmet algoritmen er implementeret i. En hurtig computer vil selvfølgelig løse et problemtilfælde hurtigere end en langsom. Det vil derfor ikke være særlig generelt at angive beregningstid for en algoritme i f.eks. sekunder. Derfor angives i stedet hvordan beregningstiden for en algoritme udvikler sig som funktion af n. Øges en algoritmes beregningstid f.eks. med en faktor fire hvis størrelsen af problemet øges med en faktor to, vil dette gøre sig gældende for alle computere.

Beregningskompleksiteten defineres på følgende måde:

$f(n)$ er en algoritmes 'værste tilfælde' beregningstid for problemtilfælde af størrelse n. Hvis der findes en funktion $g(n) : \mathbf{N} \to \mathbf{R}$ og en konstant $c > 0$ således at

$$f(n) \leq c \cdot g(n)$$

for alle $n \geq 1$, siger vi at algoritmens tidskompleksitet er af størrelsesorden $g(n)$, eller mere kompakt at kompleksiteten er $\mathcal{O}(g(n))$ [Dolan and Aldous, 1995].

Denne definitionen retter sig mod beregningstiden, men det kan tænkes, at man også ønsker at skelne mellem forskellige algoritmers brug af arbejdshukommelse eller lagerplads. Man kan definere kompleksiteten for disse ting på samme måde, som det her er gjort for beregningstiden. I langt de fleste tilfælde er det dog beregningstiden, man er interesseret i at vurdere [Dolan and Aldous, 1995].

Ovenstående definition retter sig mod eksakte algoritmer, altså algoritmer hvor man kan spørge, hvor lang tid de er om at finde ***løsningen.*** Dette kan man ikke spørge om i tilfældet med heuristikker, da de ikke kan garantere at finde de bedste løsninger. Man kan dog stadig spørge hvor lang tid en given heuristik

kompleksitet	problemstørrelse (n)			
	10	25	50	100
$\mathcal{O}(n^3)$	$1,00 \cdot 10^{-6}$ s	$15,6 \cdot 10^{-6}$ s	$125 \cdot 10^{-6}$ s	$1,00 \cdot 10^{-3}$ s
$\mathcal{O}(2^n)$	$1,02 \cdot 10^{-6}$ s	$33,6 \cdot 10^{-3}$ s	313 timer	$40,2 \cdot 10^{12}$ år

Figur 4.2 Beregningstid af en algoritme med $\mathcal{O}(n^3)$ og en algoritme med $\mathcal{O}(2^n)$.

er om at behandle[2] problemtilfælde af en given størrelse, og dette kan gøres på samme måde som med eksakte algorimer.

Beregningskompleksitet kan bruges til at skelne mellem 'gode' og 'dårlige' algoritmer. Kan et givent problem løses med flere forskellige algoritmer, kan en kompleksitetsanalyse bruges til at vurdere hvilken af dem, som er hurtigst.

Man inddeler på baggrund af beregningskompleksitet algoritmer i to grupper – de polynomielle og de non polynomielle, også kaldet de eksponentielle. Polynomiel og eksponentiel henviser til hvilken størrelsesorden algoritmernes tidskompleksitet er af – altså om $g(n)$ i ovenstående definition er et polynomium eller en eksponentialfunktion[3]. Der er afgørende forskel på beregningstiden for en algoritme med polynomiel kompleksitet og en algoritme med eksponentiel kompleksitet, hvilket vi vil illustrere med et lille eksempel.

Vi forestiller os en algoritme med kompleksitet $\mathcal{O}(n^3)$ og en med $\mathcal{O}(2^n)$. Derudover forestiller vi os, at c i ovenstående definitionen er 1, og at tiden bliver angivet i 'beregningsskridt'. Algoritmerne køres på en computer, som er i stand til at udføre 1 mia. beregningsskridt i sekundet. I skemaet på figur 4.2 er angivet, hvor lang tid det ville tage at køre algoritmen, hvor problemstørrelsen n er hhv. 10, 25, 50 og 100.

Det ses af figur 4.2, at selvom de to algoritmer er nogenlunde lige hurtige om at løse et problem for $n = 10$, vokser beregningstiden for algoritmen med eksponentiel kompleksitet enormt sammenlignet med den polynomielle. Det er på grund af dette, at man generelt betragter polynomielle algoritmer for effektive og eksponentielle som uhåndterlige.

Inddelingen i polynomielle og eksponentielle algoritmer er sådan set lidt for simpel. Den tager ikke højde for, at polynomierne kan være af høj grad eller at eksponentialfunktionerne kan have små koefficienter. Der tages heller ikke højde for konstanten c, som indgår i definitionen af beregningskompleksiteten. I nogle problemtilfælde kan eksponentielle algoritmer derfor være hurtigere end nogle polynomielle. Inddelingen er dog fornuftig set i lyset af, at enhver eksponentialfunktion på et tidspunkt, for tilstrækkeligt stort n, vil få større funktionsværdi end ethvert polynomium. Dertil kommer, at kompleksiteten af de fleste polynomielle algoritmer er $\mathcal{O}(n^2)$ eller $\mathcal{O}(n^3)$ [Dolan and Aldous, 1995].

[2] Altså, hvor lang tid er heuristikken om at finde en (ikke-nødvendigvis optimal) løsning til problemet i værste tilfælde.

[3] Eksponentiel tidskompleksitet betegner ligeledes funktioner som er endnu hurtigere voksende (for $n \to \infty$), eksempelvis $n!$.

Eksakte VRP-algoritmer og beregningstid
Som nævnt ovenfor vokser beregningstiden af eksakte VRP-algoritmer voldsomt, når antallet af kunder øges. Mere præcist har alle eksakte algoritmer for VRP eksponentiel kompleksitet. For eksempel har *branch and bound* algoritmen beregningskompleksiteten $\mathcal{O}(2^n)$ [Dolan and Aldous, 1995], dog vil den i praksis præstere langt bedre.

4.1.2 Klassifikation af problemer

Kombinatoriske problemer kan inddeles efter hvilken kompleksitet den bedste algoritme, som kan løse dem eksakt, har. Beregningskompleksiteten kan altså bruges til at skelne mellem lette og svære problemer. Lette problemer er dem som kan løses med en algoritme med polynomiel kompleksitet, svære er dem, som ikke kan. De lettere problemer siges at tilhøre *P-klassen*, polynomieltidsproblemer, og de sværere *NP-klassen*, *non deterministiske polynomiel* problemer. Ethvert problem som kan løses i polynomiumstid kan selvfølgelig også løses i eksponentieltid, altså P $\subseteq$ NP.

Det generelle VRP tilhører NP-klassen, ja faktisk tilhører det en delmængde af NP kaldet *NP-komplet-klassen*[4]. Det der adskiller NP- fra NP-komplet-problemer er, at det er påvist, at hvis der til ét problem tilhørende sidstnævnte kunne findes en polynomieltidsalgoritme, så ville man kunne løse alle NP-problemer i polynomieltid (og dermed ville P være lig med NP). Man har i årtier, uden held, prøvet at finde en polynomieltidsalgoritme til et NP-komplet-problem. Der er idag, på baggrund af dette mangel på 'held' konvention for, at betragte P som en ægte delmængde af NP (P $\subset$ NP), dog uden direkte at kunne bevise at P $\neq$ NP. Det ser med andre ord ikke ud til, at det er muligt at finde eksakte algoritmer, som kan løse VRP og andre NP-problemer i polynomiel tid[Dolan and Aldous, 1995],[Reinelt, 1994], [Johnson and Papadimitriou, 1985].

4.2 Klassiske heuristikker

Heuristik kommer fra det græske *heuriskein*, som betyder at finde eller at opdage. I operationsanalysen bruges udtrykket idag indforstået som en fællesbetegnelse for en metode, hvor man på spidsfindig vis ved en eller anden algoritme søger en god, men ikke nødvendigvis optimal løsning.

En definition på heuristikker er givet i [Reeves, 1993]:

> *A heuristic is a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is.*

[4]Egentligt er komplet-betegnelsen, forbeholdt beslutningsproblemer (ja/nej-problemer) hvorimod betegnelsen *NP-hard* bruges om tilsvarende optimeringsproblemer.

Som det ses, garanterer heuristikker hverken lovlige (mht. begrænsningerne) eller optimale løsninger. Endvidere er det ikke altid muligt for en heuristik at afgøre, hvor langt fra optimum en given løsning er. Heuristikker har altså nogle alvorlige svagheder i forhold til eksakte algoritmer. De har dog også deres styrker, hvilket er hastighed og fleksibilitet, og oftest implicerer de ikke et alvorligt tab i kvalitet af løsning.

Der findes til løsning af TSP og VRP overordnet to slags heuristikker, nemlig *konstruktions-* og *forbedringsheuristikker*. I ruteplanlægningsregi vil dette sige algoritmer, som hhv. danner en rute og algoritmer, der ud fra en initialrute forsøger at forbedre ruten. I dette afsnit vil vi beskrive nogle af de klassiske konstruktion- og forbedringsheuristikker. Disse gennemgås i deres originale form uden tidsvinduer.

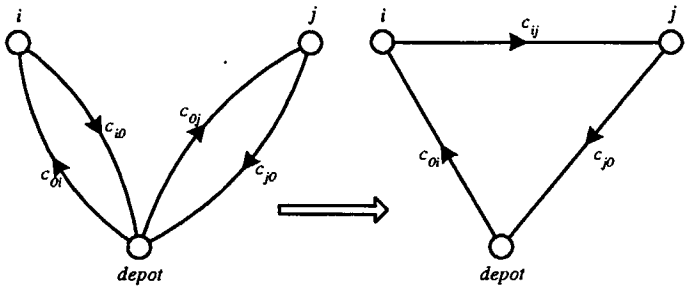
4.2.1 Konstruktionsheuristikker

Karakteristisk for konstruktionsheuristisker er, at en rute bygges successivt efter et eller andet konstruktionskriterie og at dele af ruten, der allerede er konstrueret, forbliver uforandrede gennem resten af den konstruerende algoritme [Reeves, 1993].

Savings-heuristik
Clark og Wright's *savings-algoritme* er måske den mest kendte heuristik for VRP-problemer. Den er beregnet til problemer, hvor også antallet af vogne er en variabel, og findes i to grundskabeloner; den parallelle version og den sekventielle version. Principperne bag savings-algoritmen er følgende:

1. Dan en løsning hvor hver kunde bliver besøgt af en dedikeret vogn, dvs. vognen besøger kun denne kunde.
2. Udregn besparelserne $s_{ij} = c_{0i} + c_{j0} - c_{ij}$, hvor 0 er depotet. Dette svarer til besparelsen ved at lade en vogn køre ad ruten $(0, i, j, 0)$ i stedet for at lade en vogn køre $(0, i, 0)$ og en anden $(0, j, 0)$. Princippet kan ses på figur 4.3.
3. Ordn besparelserne i en liste i aftagende orden.
4. Startende fra øverste ikke-undersøgte s_{ij} på listen gør følgende:
   a. Hvis kanten (i, j) er lovlig – overholder problemets begrænsninger, herunder at hver kunde kun må besøges af én vogn – tilføj da kanten (i, j), og slet $(i, 0)$ og $(0, j)$ fra løsningen.
   b. Prøv næste s_{ij} på listen og gå tilbage til 4a. indtil ikke flere (lovlige) kanter kan dannes.

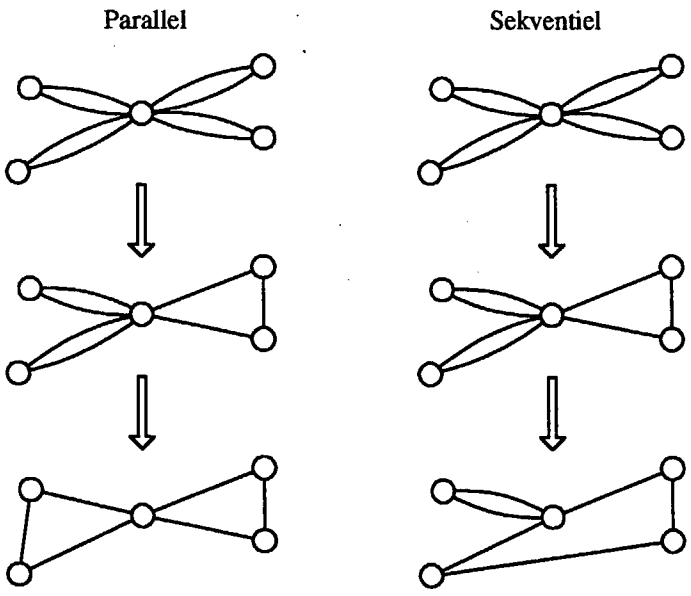
Den parallelle version, som her beskrevet, bygger altså ruterne for de forskellige vogne op samtidigt. Den sekventielle version afviger fra den parallelle ved, at



Figur 4.3 Princippet i savingsalgoritmen.

den udvider ruten $(0, i, j, 0)$ til den ikke længere kan udvides, før den lægger ruten for en ny vogn. Dette kan ses på figur 4.4.

Dette er den originale savings-algoritme opfundet i 1964, og hvorom det er konstateret, at parallelversionen finder bedre løsninger end den sekventielle [Laporte et al., 2000]. Savingsheuristikken har tidskompleksitet $\mathcal{O}(n^2 \log_2 n)$. Sidenhen er mange modificerede definitioner af besparelserne (s_{ij}) kommet til veje. Disse har primært haft til hensigt at nedsætte beregningstiden og hukommelseskravene [Christofides, 1985b],[Petersen and Surland, 1999].



Figur 4.4 Forskellen mellem den paralelle og den sekventielle version af savings-algoritmen.

Sweep-algoritme

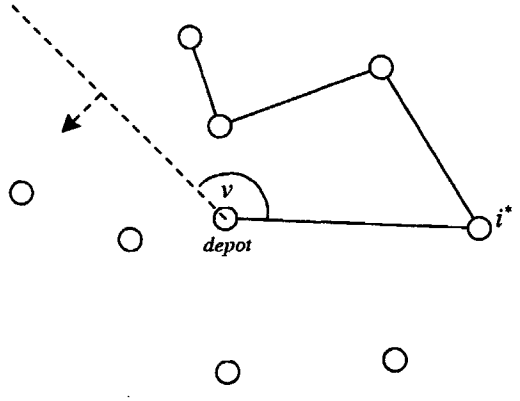
Sweep er en to-fase-algoritme, hvor man først inddeler kunderne i klynger, alle indeholdende depotet, og så løser et seperat TSP for hver klynge. Algoritmen kan kun bruges til VRP-tilfælde, hvor kunderne er placeret i et todimensionalt plan. Dette udgør dog intet synderligt problem, da langt de fleste tilfælde opfylder dette.

Algoritmen fungerer således:

Repræsentér alle kunderne på polarkoordinatform (r_i, v_i) med depotet i $r_0 = 0$ og en vilkårlig reference-kunde $i*$ med $v_{i*} = 0$. Ordn kunderne sådan at $v_1 \leq \ldots \leq v_i \leq \ldots \leq v_n$.

Fase 1

1. Vælg en ikke-benyttet vogn k.
2. Startende fra ikke-rutelagte kunde i med mindste v_i, inkludér kunder $i+1, i+2, \ldots$ i ruten indtil kapacitetsbegrænsningen (eller en anden begrænsning) for køretøj k er nået. Dette er illustreret på figur 4.5



Figur 4.5 Princippet i sweepalgoritmen.

3. Hvis alle kunder er blevet 'sweepet' eller alle vogne er brugt, gå da til Fase 2, ellers gå tilbage til 1.

Fase 2

4. Løs TSP, enten eksakt eller vha. heuristikker, for hver klynge af kunder tildelt de forskellige vogne.

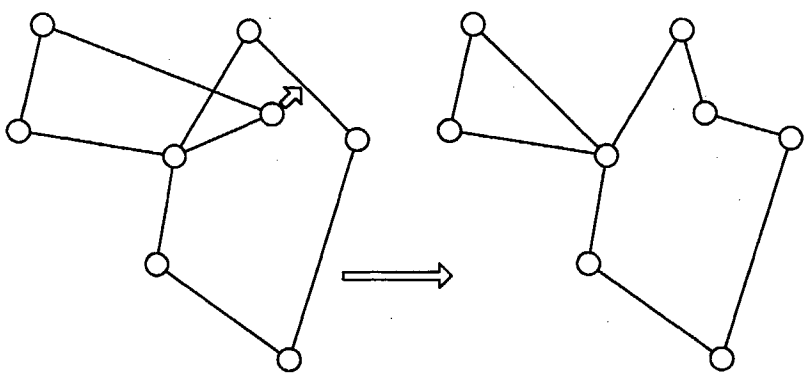
Bemærk i øvrigt at forskellige valg af reference-kunde $i*$ giver forskellige ruter.

4.2.2 Forbedringsalgoritmer

Der er udviklet elaborerede udgaver af de originale udgaver af savingsalgoritmen og andre konstruktionsalgoritmer, og disse er en smule hurtigere og giver måske lidt bedre løsninger end de originale. Rent intuitivt må man dog forvente, at forbedringsheuristikker giver bedre løsninger end konstruktionsheuristikker. Dette er som udgangspunkt også tilfældet, da forbedringsheuristikker som regel virker, efter at en konstruktionsheuristik har dannet en rute.

De fleste forbedringsalgoritmer er såkaldte nabosøgningsalgoritmer, *NS-algoritmer*. For at kunne forstå hvordan sådanne algoritmer fungerer må vi først introducere begreberne træk og naboområde.

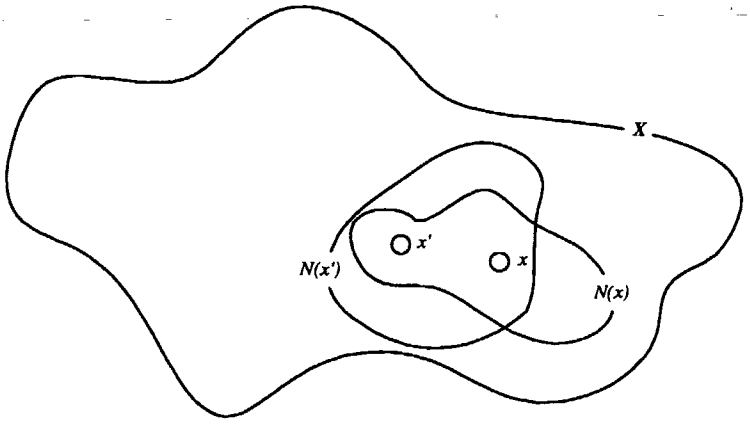
Hvis mængden af lovlige løsninger til problemtilfældet kaldes $\mathbf{X}$, da har hver løsning $x \in \mathbf{X}$ en tilhørende mængde $N(x) \subseteq \mathbf{X}$, som kaldes *naboområdet* til x. Naboområdet er defineret som alle de løsninger $x' \in \mathbf{X}$, som kan nås fra x ved en operation kaldet for et *træk*. Et træk er en operation, som kan lave en løsning om til en anden. I VRP sammenhæng kan det f.eks. være ved at 'flytte' en kunde, så den bliver betjent af en anden vogn end før. Et sådant træk kan ses på figur 4.6. $N(x)$ givet ved dette træk er så alle de løsninger x', som kan dannes ud fra x ved at lade en hvilken som helst af kunderne blive betjent af en ny vogn.



Figur 4.6 Et træk hvor en kunde flyttes fra én rute til en anden.

$N(x)$ kan opfattes som værende en lille del af hele $\mathbf{X}$. Mange træk er symmetriske, hvilket vil sige, at hvis man ved at benytte dette kan komme fra x til x', kan man også bruge det til at komme fra x' til x. At trækket er symmetrisk medfører at $x' \in N(x)$ og $x \in N(x')$. Dette er illustreret på figur 4.7.

En NS-algoritme gennemløber et antal iterationer. For hver iteration udføres et træk, som resulterer i naboløsning $x' \in N(x)$. Måden naboløsningen vælges på varierer fra algoritme til algoritme. Princippet for hvordan en NS-algoritme virker kan skitseres således:



Figur 4.7 Naboområdet $N(x)$ til en løsning x givet ved et symmetrisk træk.

Princippet i NS-algoritmer

1. Start

   Vælg en løsning, $x_{nu} \in \mathbf{X}$, dannet af en konstruktionsalgoritme. Sæt $x_{bedst} = x_{nu}$ og definér og udregn omkostningsværdien cost $= c(x_{nu})$.

2. Udvælgelse af nabo-løsning

   Vælg en naboløsning $x_{nabo} \in N(x_{nu})$ ud fra et udvælgelseskriterie. Hvis det anvendte udvalgskriterie ikke kan opfyldes for nogen af løsningerne fra $N(x_{nu})$, eller hvis andre anvendte stopkriterier (så som et begrænset antal af iterationer) er opfyldt, standser søgningen og løsningen x_{nu} er den endelige løsning.

3. Opdatering

   Hvis der findes en naboløsning x_{nabo} som opfylder udvælgelseskriterier sættes $x_{nu} = x_{nabo}$ og cost $= c(x_{nabo})$. Proceduren gentages fra skridt 2.

Det simpleste udvælgelseskriterie er den såkaldte *descent*-metode. Her kan skridt 2 ovenfor præciseres til følgende:

2. Udvælgelse af naboløsning for descent-metoden

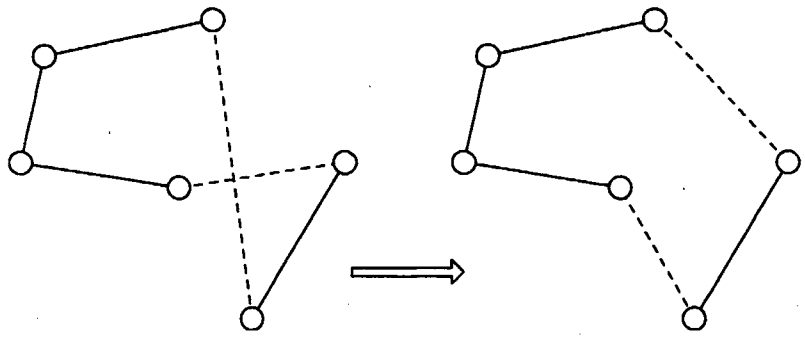
   Vælg den naboløsning $x_{nabo} \in N(x)$, som har den laveste omkostning $c(x_{nabo})$. Stop hvis der ikke findes et x_{nabo} således at $c(x_{nabo}) < c(x_{nu})$.

Descent-metoden går altså ud på for hver iteration at udføre det træk, som giver den største forbedring mht. objektfunktionen, og kaldes derfor også tit for en *greedy* algoritme.

Nabosøgning til VRP

NS-algoritmer for et VRP arbejder enten på hver rute for sig eller på flere ruter samtidigt. De heuristikker, der hører under første kategori, adskiller sig ikke fra forbedringsalgoritmer til et TSP. Den anden kategori af heuristikker udnytter derimod 'fler-rute' dimensionen i et VRP.

En af de klassiske NS-algoritmer til TSP er Lin's λ-*opt* algoritme [Reinelt, 1994]. Trækket i denne går ud på at fjerne λ kanter i ruten og så sætte de λ brudstykker af turen sammen på nye måder. Findes en gunstigere kombination af segmenterne – det være sig enten den først konstaterede eller den bedste – så vælges denne. Dette træk kan ses på figur 4.8.



Figur 4.8 Et træk i en 2-opt algoritme.

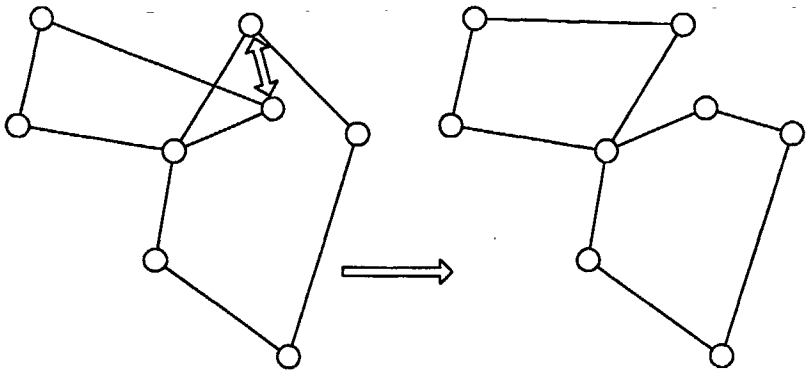
En rute siges at være λ-optimal, hvis det ikke er muligt at finde en kortere rute ved at udskifte λ kanter med λ andre kanter [Helsgaun, 1999]. At kontrollere λ-optimalitet for et træk har beregningskompleksiteten $\mathcal{O}(n^\lambda)$ [5] [Helsgaun, 1999]. λ-algoritmer findes som regel for $\lambda = 2, 3$ eller 4. Der er også udviklet mere avancerede algoritmer hvor værdien af λ varierer dynamisk gennem iterationer i algoritmen.

I stedet for at udskifte kanter i samme rute som λ-opt-metoder for TSP, er der designet andre nabosøgningsalgoritmer til VRP, hvori der muliggøres en interaktion mellem kunder i forskellige ruter. Trækkene, som defineres i disse algoritmer, sørger på en eller anden måde for, at en eller flere vogne i hver iteration bliver tildelt nye kunder. Nogle af de mest udbredte måder at gøre dette på er ved at flytte én kunde fra en rute til en anden, eller ved at bytte en kunde i en rute ud med en kunde i en anden (jf. figur 4.9). Man kan også definere træk ved at fjerne kanter i forskellige ruter, og så forbinde dem igen på tværs af de originale ruter (jf. figur 4.10). Som det ses af figur 4.9 og figur 4.10, kan en given løsning omdannes til en anden med forskellige træk. Dette betyder at naboområder for forskellige træk godt kan have fælles elementer.

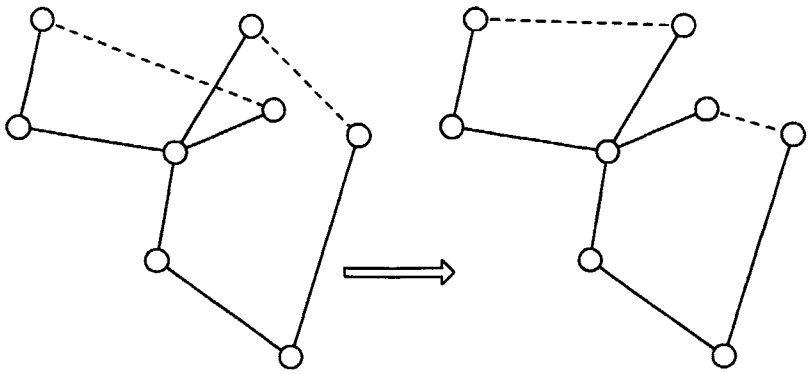
Lokale og globale optima

Nabosøgning starter med en initialløsning og man finder kun den bedste løsning

[5] Antal muligheder i et træk: $\binom{n}{\lambda} \sim n^\lambda$, hvis $\lambda \ll n$.



Figur 4.9 Et træk hvor to kunder i to forskellige ruter bliver ombyttet.



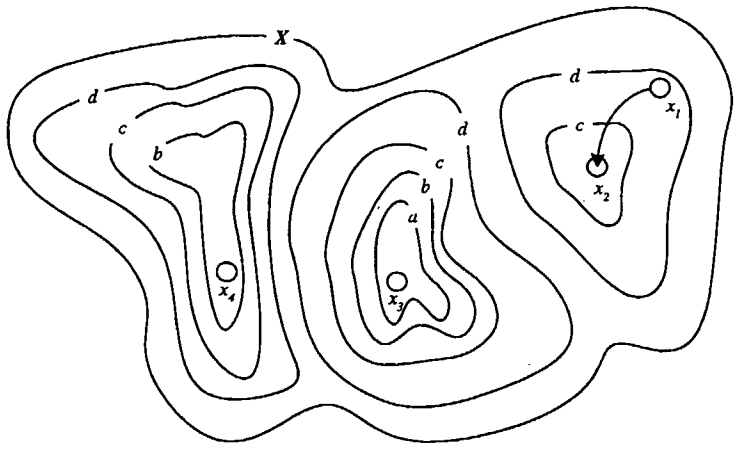
Figur 4.10 Et træk hvor kanter i forskellige ruter bliver slettet og enderne forbundet på nye måder.

indenfor et begrænset område af løsningsrummet, dvs. at løsningen kun kan garanteres at være et *lokalt optimum*. Dette er illustreret på figur 4.11.

Et karakteristikum ved mange optimeringsproblemer er, at der er én optimal løsning (eller nogle få), altså *globale optima*, men mange løsninger der kan være lokale optima. De lokale optima, som man finder, afhænger af hvordan NS-algoritmens træk er defineret. En løsning, som ved en algoritme vil være et lokalt optimum, vil altså ikke nødvendigvis være det for en algoritme, hvor et træk er defineret anderledes.

Man er gennem tiderne kommet med forskellige innovationer for at forøge chancen for, at det lokale optimum, en heuristik finder, også er et globalt optimum eller i hvert fald tættere på at være det. En tilgang til det er at ændre trækket, f.eks. ved at ændre en 2-optimal heuristik til en 3- eller sågar en 4-optimal algoritme. Et andet alternativ er at køre heuristikken flere gange fra forskellige initialløsninger. Herefter kan man så udpege det bedste af de forhåbentligt forskellige lokale optima.

En tredje metode er at tillade såkaldte *uphill-moves*, som giver mulighed for, at



Figur 4.11 Niveaukurver der illustrerer hvorledes en algoritme, som starter i x_1, vil bevæge sig ned i x_2, som er et lokalt men ikke et globalt optimum.

søgeheuristikken kan 'kravle' op af det lokale minimum og søge mod et lavere. Det nytter dog sjældent noget at tillade greedy algoritmer at lave uphill-moves, når de befinder sig i lokale minima. Ofte vil de nemlig vende tilbage til det lokale minimum i iterationen umiddelbart efter et uphill-move da det lokale optimum er den bedste løsning i naboområdet.

4.2.3 Tidsaspektet i klassiske heuristikker

De ovennævnte konstruktions- og forbedringsalgoritmer tager i deres grundlæggende form således ikke højde for tidsaspektet i ruteplanlægning. I [Solomon, 1987] er der beskrevet udvidelser af en række klassiske heuristikker, så de også er i stand til at løse VRPTW.

4.3 Metaheuristikker

En metaheuristik er som navnet antyder en form for 'over'-heuristik. Metaheuristikker bruger en eller flere klassiske heuristikker, som de via forskellige sindrige strategier kan styre, således at heuristikkerne bliver i stand til at opnå bedre resultater, end de kunne i deres originale form. Nogle metaheuristikker kan f.eks. styre en NS-algoritme ud af lokale minima og ind i nye områder af løsningsrummet, hvor de har mulighed for at finde bedre løsninger. Ligesom klassiske heuristikker kan metaheuristikker hverken garantere at finde den bedste løsning, eller angive hvor langt en løsning er fra optimum.

Vi vil i denne sektion først se på et eksempel på en metaheuristik for at illustrere, hvordan metaheuristikker kan sættes til at styre klassiske heuristikker og derefter kort nævne nogle af de andre typer af metaheuristikker, som der er blevet udviklet.

4.3.1 Simuleret udglødning

En af de tidligste og mest alsidige metaheuristikker er simuleret udglødning (SU) [Reeves, 1993]. Metoden er inspireret af simuleringer af fysiske systemer, hvor metoden ofte bruges til at simulere afkøling af et smeltet materiale. Her bestemmer simuleringen løbende systemets tilstand, mens det afkøles indtil tilstanden konvergerer til et 'steady state' dvs. frossent stadie.

Kernen i SU-simuleringer – for fysiske fænomener – er Boltzmann-fordelingen, som lyder, at sandsynligheden, for at et fysisk system skifter fra en tilstand med en given energi til en tilstand med højere energi, er givet ved:

$$p(\delta E) = \exp(-\delta E / k_b T) \tag{4.1}$$

hvor $p(\delta E)$ er sandsynligheden for at skifte tilstand med en resulterende energiforøgelse på δE, T er temperaturen og k_b er Boltzmanns konstant.

Princippet i simuleret udglødning er som følger: Først genereres en lille forskydning i systemets tilstand, som resulterer i en ændring af systemets energi (δE). Er energiændringen negativ, dvs. den nye tilstand har lavere energi, 'rykker' systemet over i det nye stadie. Er energien i systemet forøget, accepteres det nye stadie kun med en vis sandsynlighed. Sandsynligheden er givet ved Boltzmann-fordelingen. Denne proces udføres et på forhånd bestemt antal gange ved en given temperatur T, hvorefter temperaturen sænkes og proceduren gentages. Dette fortsætter indtil systemet/materialet fryser til et 'steady state'.

I 1980 blev det foreslået, at en sådan simulering kunne bruges til at søge efter mulige løsninger til optimeringsproblemer. Det er blevet vist at en SU-algoritmes løsninger konvergerer mod en optimalløsning. Det kan dog tage uendeligt lang tid at finde en sådan.[Reeves, 1993]. SU kan faktisk overføres til ethvert kombinatorisk optimeringsproblem. Der er som ved andre heuristikker selvfølgelig ingen garanti for, at den 'frosne' slutløsning er den optimale.

I SU til kombinatorisk optimering er der ikke tale om tilstande og energier men om mulige løsninger og omkostninger. Et skema over hvordan de fysiske begreber bliver oversat til kombinatorisk optimering kan ses i skemaet på figur 4.12.

'fysik' simulering	kombinatorisk optimering
tilstand	en mulig løsning
energi	omkostning c
ændring af tilstand	træk
temperatur	kontolparameter
frossent stadie	heuristisk løsning

Figur 4.12 Skema over begreber i simuleret udglødning i hhv. fysikken og kombinatorisk optimering.

Kernen i en SU-algoritme er en sædvanlig NS-algoritme. Følgende er en overordnet skitse af, hvordan et generelt kombinatorisk problem løses vha. af SU. Selv

om kombinatorisk optimering ikke har noget at gøre med temperaturer, kaldes kontrolparameteren dog for temperaturen.

Princippet i SU-algoritmer

1. Vælg initialløsning x_0 og sæt $x = x_0$.
2. Vælg initialtemperatur $T_0 > 0$, og sæt $T = T_0$.
3. Gentag følgende et antal gange
   a. Vælg et $x_{ny} \in N(x)$
   b. Udregn $\delta = c(x_{ny}) - c(x)$
   c. - Hvis $\delta < 0$ accepter x_{ny}, dvs. sæt $x = x_{ny}$.
      ellers
      - Generer et tilfældigt tal α mellem 0 og 1. Hvis $\alpha < \exp(-\delta/T)$ sæt $x = x_{ny}$, hvis ikke behold den gamle løsning, dvs. $x = x$.
4. Sænk temperaturen T og gentag 3. hvis ikke et eller andet stopkriterie er opfyldt.

Stopkriteriet kunne f.eks. være, at temperaturen havde nået en eller anden given værdi.

Det ser ved første øjekast måske ud til, at SU direkte og selvstændigt kan omsættes til en algoritme, som effektivt kan løse VRP-tilfælde – dette er ikke tilfældet. SU er kun en ydre skal ovenpå de klassiske konstruktion-/forbedringsheuristikker. F.eks har I. Osman en implementering af SU, der har vist sig effektiv [Petersen and Surland, 1999]. Denne finder en intitialløsning x_0 vha. af savingsalgoritmen og evt. en λ-opt mekanisme. Derudover bestemmes $N(x)$ i hver enkelt iteration også af en λ-opt mekanisme.

Fordelen ved en sådan overbygning til en klassisk NS-algoritme er, at den ikke (så let) bliver fanget i lokale minima. Der er altid en vis sandsynlighed for, at den kan slippe fri og opnå bedre løsninger.

Et problem ved SU er, at mange parametre skal stilles for at give en effektiv algoritme. Det drejer sig f.eks. om starttemperatur, antallet af iterationer ved hver temperatur og måden hvorpå temperaturen sænkes gennem algoritmen. Gode parameterværdier kan være meget svære at bestemme, og varierer fra problemtilfælde til problemtilfælde[6]. Vi vil ikke komme nærmere ind på muligheder og effekter ved at justere på de forskellige parametre, men blot påpege at disses optimale værdier oftest fastlægges ved empiriske analyser [Reeves, 1993].

[6] Og selvfølgelig fra problem til problem.

4.3.2 Andre typer metaheuristikker

Der er udviklet andre metaheuristikker, hvoraf mange er brugbare til at løse et VRP. Flere af dem benytter sig ligesom simuleret udglødning, af metoder udviklet inden for andre videnskaber, f.eks. biologien. Blandt disse kan nævnes *kunstige neurale netværk* og *genetiske algoritmer*[7]. Der er dog også udviklet meta-heuristikker, som ikke er inspireret af naturlige fænomener og processer. Dette gælder f.eks. for *tabu-søgning* (jf. kapitel 5), som ligesom SU er en overbygning til klassiske NS-algoritmer.

4.4 Hvilken type algoritme er bedst

I det følgende vil vi diskutere fordele og ulemper ved de forskellige algoritmer til VRP-løsning, og vurdere hvilken type vi finder mest egnet til løsning af FDB-lignende problemer.

4.4.1 Hvordan vurdereres algoritmer

Der er flere kriterier, som er vigtige, når det skal vurderes hvilke algoritmer, der egner sig bedst til en given type problemer. Vi vil her beskæftige os med de væsentligste af disse.

Beregningstid
Det er afgørende om en algoritme er i stand til at løse de aktuelle problemtilfælde inden for den tid en virksomhed har til rådighed. Skal en virksomhed ruteplanlægge en gang om dagen, er det på ingen måde muligt at anvende en algoritme, som er flere dage om at løse de aktuelle problemtilfælde. I det hele taget er det kritisk i praktiske situationer at benytte en algoritme, som tager meget lang tid at køre igennem, da en ændring af problemet kan nødvendiggøre genkørsler. Ændringer af problemet kan opstå, hvis der kommer en ekstra kunde til i sidste øjeblik, eller hvis der af en eller anden grund er begået en fejl i opstillingen af problemet.

Beregningstiden afhænger som tidligere beskrevet af mange parametre, såsom computertype og program hvori algoritmen er implementeret. Om en given algoritme kan løse et givent problem i løbet af en given tid er derfor meget svært af vurdere. Det vil i mange tilfælde først kunne afgøres endeligt, når en færdig implementering er til rådighed. Kender man beregningstiden, når en given implementering løser et problemtilfælde af en given størrelse, kan man dog få et indtryk af, hvor stor beregningstiden vil være for et problemtilfælde af en anden størrelse, hvis man kender beregningskompleksiteten for algoritmen.

[7]For den interesserede læser henvises til [Rayward-Smith et al., 1996].

Løsningskvalitet
Set fra et økonomisk synspunkt er forholdet mellem løsningskvaliteten og tidsforbrug afgørende for, hvorvidt en algoritme er god eller dårlig. Jo 'billigere' ruter en algoritme er i stand til at finde jo bedre.

Eksakte algoritmer finder selvfølgelig altid de bedste løsninger, men hvad med heuristikker? Det er meget besværligt at analysere kvaliteten af heuristiske algoritmer matematisk, især hvis modellen, som algoritmen søger at løse, har mange specielle begrænsninger [Helsgaun, 1999]. Derudover er resultaterne af sådanne analyser ikke altid særligt sigende. F.eks. kan det vises, at en bestemt TSP algoritmes løsninger altid ligger indenfor 50% af det globale optimum [Helsgaun, 1999]. I praksis er løsningerne langt bedre, og den matematiske analyse er derfor ikke særlig interessant i praktisk øjemed.

På grund af disse besværligheder vurderes heuristiske algoritmer som regel vha. bestemte testsæt. Testsættene indeholder et antal problemtilfælde, som skal løses. I VRP-situationer indeholder testsættene oplysninger om placering af depot og kunder, ordrestørrelse og tidsvinduer for kunder, kapacitet af vogner m.v.

Solomons testsæt
Det mest udbredte testsæt til VRPTW er foreslået af M. M. Solomon [Solomon, 1987], og indeholder 56 forskellige situationer. Da vi senere skal beskæftige os med disse tests, vil vi allerede her beskrive nogle af deres karakteristika.

Der er i testsættet tre serier, og måden, kunderne er placeret på, er forskellig i de tre. I R-serien er kunderne jævnt fordelt, i C-serien i klynger og i RC-serien en blanding af tilfældig fordeling og klynger. Samler man kunderne i klynger, får man en struktur, som bedre stemmer overens med en virkelig situation, da tætheden af kunder som regel er større i nærheden af byer[8]. Hver serie opdeles i to underserier, hvor den ene (kaldet hhv. R100, C100, RC100) har stramme, og den anden (kaldet hhv. R200, C200, RC200) har løse begrænsninger på rutelængde og kapacitet af vogne.

Desuden varierer også måden tidsvinduerne er arrangeret på, således at det er muligt at vurdere algoritmerne under forskellige grader af begrænsethed. Solomons testsæt bygger på et ældre testsæt udviklet af N. Christofides, A. Mingozzi og P. Toth [Christofides et al., 1979], som ikke har tidsvinduer [9]. Sættet består af 14 problemtilfælde med 50 til 199 kunder. Faktisk er Solomons R100-serie dannet ved at tage CMT-problemtilfælde nummer 8 og tilføje forskellige tidsvinduer. Ligeledes er C100-serien dannet ved at føje tidsvinduer til CMT 14. RC-serien er dannet som en blanding af disse.

Måden tidsvinduer er placeret på er vidt forskellig i R100-serien og C100-serien. I R100-serien er tidsvinduerne fordelt tilfældigt. I C100-serien blev problemet først løst med en klassisk heuristik, og tidsvinduerne derefter placeret så deres centrum var de tidspunkter, hvor kunder blev besøgt af en vogn [Solomon, 1987].

[8]Med en virkelig situation menes en FDB-lignende situation, hvor et depot dækker en hel region/landsdel og ikke f.eks. en by.
[9]Data for CMT-testsættet kan findes på http://mscmga.ms.ic.ac.uk/info.html

De fleste, som løser Solomons testsæt, optimerer mht. rutelængden [Cordeau et al., 2000b], [Rochat and Taillard, 1995], [Taillard et al., 1997] men der er også eksempler på tilfælde, hvor resultatet angives som samlet tidsforbrug (køretid + pauser) [Solomon, 1987]. I Solomons sæt er der dog en tæt knyttet sammenhæng mellem rutelængde og tidsforbrug, idet køretiden mellem to kunder antages at være proportional med afstanden.

Situationerne i Solomons tests har en række fællestræk. Disse er:[10]

- 100 kunder
- Et depot
- Ens vogne
- Ens servicetid ved kunderne
- Ingen 'avancerede' begrænsninger

Det ses af ovenstående, at sættet er ret simpelt, da situationerne f.eks. ikke har flere depoter, forskellige vogne, forskellige servicetider, adgangsbegrænsninger ved kunder, hviletider for chaufførerne el. lign.

Med andre ord stemmer Solomons testpakke ikke særligt godt overens med mange virkelige problemer. Man kan derfor ikke være helt sikker på, at en algoritme, som præsterer godt i testsættet, præsterer godt i virkeligheden.

Selv om Solomons testpakke er simpel, indeholder den dog de vigtigste træk ved virkelige VRPTW-tilfælde. Forskellige virkelige tilfælde kan have udvidelser i forhold til Solomons testsæt, men de har i hvert fald testsættets begrænsninger til fælles. En algoritme som yder godt i Solomon-testene, kan derfor formodes også at yde tilfredsstillende i mere avancerede tilfælde. Det kræver selvfølgelig, at algoritmen kan udvides, så den kan håndtere disse, uden at den på afgørende vis laves om.

Fleksibilitet

Skal en algoritme bruges til flere forskellige slags problemer, er det selvfølgelig vigtigt, at den kan håndtere dem. Fleksibilitet er som regel ofte det, at en algoritme er i stand til at håndtere mange forskellige begrænsninger, samtidig med at algoritmen relativt set præsterer godt. Bruges den så til et tilfælde uden mange begrænsninger, deaktiveres de dele af algoritmen, som ikke er nødvendige.

Skal algoritmen kun bruges til at løse et enkelt problem, er fleksibilitet ikke så vigtigt. Algoritmen skal blot være i stand til at løse det aktuelle slags problemtilfælde. Det vil dog altid være en fordel, at en algoritme er mere fleksibel, hvis den øgede fleksibilitet ikke går ud over løsningskvaliteten eller beregningstiden.

[10] Data for Solomon's testsæt kan findes på http://www.cs.strath.ac.uk/research/GreenTrip/

4.4.2 Eksakte algoritmer eller heuristikker

Som vi har set, er den naive tilgang til løsning af VRP-tilfælde, hvor alle mulige løsninger undersøges, yderst ineffektiv, og selv om det i teorien lader sig gøre, at løse et hvilket som helst problem, ville det i praksis være umuligt. Metoderne er dog blevet forbedret markant gennem de sidste 30 år og er i dag i stand til at løse problemer med over 100 kunder [Cordeau et al., 2000a],[Madsen, 1997]. På trods af dette er der dog stadig eksempler på problemtilfælde med ikke mere end 75 kunder, som til dato ikke er blevet løst eksakt [Crainic and Laporte, 1998]. Eksakte algoritmer kan altså have store problemer med at løse tilfældigt konstruerede problemtilfælde med omkring 100 kunder eller derover. Men det er netop den størrelse problemtilfælde, som vi arbejder med i dette projekt.

Eksakte algoritmer må, på grund af manglende evne til generelt at løse problemtilfælde med 100 kunder eller flere og deres lange beregningstid, betragtes som uegnede til løsning af vores model. Det ser ud til, at heuristikker i vores tilfælde er den eneste reelle løsnings-mulighed.

4.4.3 Heuristikker eller metaheuristikker

Det kan godt være, at man i vores tilfælde bliver nødt til at bruge heuristikker, men hvilken form for heuristik egner sig bedst? Det vil vi forsøge at redegøre for i dette afsnit.

Intuitivt må det formodes, at metaheuristikker er klassiske heuristikker overlegne, da de jo netop benytter klassiske heuristikker som de styrer med forskellige intelligente strategier. Dette er ifølge [Laporte et al., 2000] også tilfældet. I artiklen er de bedste klassiske heuristikker til VRP (uden tidsvinduer) blevet undersøgt. Ingen af dem kan konkurrere med de præsenterede metaheuristikker, som her er tabusøgning, når det gælder løsningskvalitet.

Grunden til at det er tabusøgning, som de klassiske heuristikker bliver sammenlignet med er, at det er tabusøgning, som har vist sig at være de metaheuristikker, som er i stand til at finde de billigste løsninger i VRP-tilfælde [Crainic and Laporte, 1998].

Et interessant faktum er, at de bedste løsninger klassiske heuristikker har fundet er næsten lige så gode, som de bedste løsninger tabusøgningsalgoritmer har fundet. Dette kan ses af sammenligningen mellem de bedste løsninger fundet af klassiske heuristikker (KH) og tabusøgningsalgoritmer (TS) i figur 4.13. Data stammer fra [Laporte et al., 2000] og testsættene stammer fra [Christofides et al., 1979].

Det ses, at de klassiske heuristikker i bedste fald giver lige så gode løsninger som tabusøgning. Dette kan umiddelbart virke mærkeligt, men figur 4.13 viser kun de absolut *bedste* løsninger fundet. I figur 4.14 er resultaterne fra udvalgte algoritmer sammenlignet. Vi har sammenlignet en enkelt tabusøgningsalgoritme (TS) med en enkelt genetisk algoritme (GA), en enkelt simuleret-

problem	KH	TS	forskel*
1	524,6	524,6	0,0
2	835,8	835,3	0,1
3	830,4	826,1	0,5
4	1038,5	1028,4	1,0
5	1321,3	1291,5	2,3
6	555,4	555,4	0,0
7	911,8	909,7	0,2
8	877,3	865,9	1,3
9	1176,5	1162,6	1,2
10	1418,3	1395,9	1,6
11	1043,4	1042,1	0,1
12	819,6	819,6	0,0
13	1548,3	1541,1	0,5
14	866,4	866,4	0,0
gennemsnit	983,4	976,0	0,8

Figur 4.13 Sammenligning af bedste resultater fundet med klassiske heuristikker (KH) og tabusøgning (TS). *Forskellen er KH's afvigelse fra TS i procent.

udglødningsalgoritme (SU) og en enkelt klassisk heuristik (KH). Her er forskellen mellem de forskellige algoritmer langt større end det sås i figur 4.13. Det kan godt være at klassiske algoritmers bedste resultater er næsten lige så gode som resultater opnået af metaheuristikker, men det ses at deres gennemsnitlige løsningskvalitet er dårligere. Desuden ses det at TS-algoritmen er de andre metaheuristikker overlegen.

Det koster selvsagt noget at gå fra klassiske heuristikker til metaheuristikker. Klassiske heuristikker er hurtigere end metaheuristikker. Kompleksiteten af metaheuristikker er dog ikke værre end de klassiske. F.eks. er der i [Crainic and Laporte, 1998] givet eksempler på to tabusøgealgoritmer, hvis beregningstid udvikler sig som hhv. $n^{2,025}$ og $n^{1,591}$.

På baggrund af undersøgelserne ovenfor anser vi altså tabusøgning som den mest velegnede algoritmetype til løsning af vores model.

problemserie	TS	GA	SU	KH
R100	1197,4	1296,8	1308,8	1436,7
		8,3%	9,3%	20,0%
R200	954,4	1117,6	1166,4	1402,4
		17,1%	22,2%	46,9%
C100	828,5	838,1	909,8	951,9
		1,2%	9,8%	14,9%
C200	590,3	590,0	684,1	692,7
		−0,1%	15,9%	17,3%
RC100	1369,5	1446,3	1473,9	1596,5
		5,6%	7,6%	16,6%
RC200	1139,8	1368,1	1401,5	1682,1
		20,0%	23,0%	47,6%

Figur 4.14 Sammenligning af resultater fundet med en tabusøgningsalgoritme (TS), en genetisk algoritme (GA), en simuleret-udglødningsalgoritme (SU) og en klassisk heuristik (KH). Procenttallene angiver afvigelse i forhold til TS-algoritmen. Tal stammer fra [Chiang and Russel, 1997].

5 Tabu-søgning

Tabu-søgning er som tidligere beskrevet den optimeringsmetode til ruteplanlægning som vi finder bedst. De bedste TS-algoritmer kan producere løsninger, som kommer meget tæt på de optimale [Cordeau et al., 2000b].

Mængden af TS-algoritmer foreslået i litteraturen er stor, men det kan som nævnt i afsnit 4.4.1 være svært at gennemskue hvilke algoritmer, som er mest velegnede i en given praktisk situation. Dette løses delvist ved, at lade de forskellige algoritmer regne på den samme samling problemtilfælde, f.eks. Solomons testpakke. Som nævnt er dette ikke tilstrækkeligt, da Solomons testpakke er simplere end de fleste 'real-world' tilfælde. Der er yderligere det problem, at de algoritmer, som bliver præsenteret i litteraturen, ofte er tilpasset netop den testpakke de er testet på, og derfor ikke nødvendigvis yder særligt godt på andre problemtilfælde [Rochat and Taillard, 1995], [Rochat and Semet, 1994], [Taillard et al., 1997].

Disse algoritmer er naturligvis interessante, fordi de ofte indeholder forskellige og ofte hidtil ukendte strategier til at løse problemer på snedig vis. Dette kunne være kreationer som 'selvtunende parametre' [Cordeau et al., 2000b] eller 'adaptiv hukommelse' [Rochat and Taillard, 1995], hvilke vil blive præsenteret senere. I dette afsnit vil vi derfor ikke fokusere på en enkelt TS-algoritme eller nogle få forskellige, men i stedet beskrive principperne bag TS og nogle af de interessante strategier, som kan bringes i anvendelse.

5.1 Grundlæggende principper i tabu-søgning

Vi vil her gennemgå de grundlæggende principper i en TS-algoritme. TS er ligesom simuleret udglødning en form for NS, der forsøger at komme udenom problemet med, at man ender i et ikke-nær-optimalt lokalt optimum.

Tabu-hukommelse (korttidshukommelse)
Da TS er en form for naboområdesøgning, virker en sådan ligeledes i iterationer. Det elegante i TS-algoritmer er måden, hvorpå de forsøger at undgå lokale optima. En TS-algoritme indeholder en liste over de nyligt besøgte løsninger, dvs. løsninger som algoritmen har valgt i de sidste m iterationer. Algoritmen er i en given iteration forment at vende tilbage til løsninger, som er på listen – man siger de er *tabu*. Da listen opdateres løbende er de tabubelagte løsninger forskellige fra iteration til iteration. Ved at forbyde de nyligt besøgte løsninger

kan det forhindres, at en algoritme vender tilbage til et lokalt optimum, som den lige har besøgt. Antallet af iterationer, hvor en besøgt løsning er tabu kaldes for *tabulængden.*

Ofte er det for besværligt at gemme alle detaljer ved de løsninger, som er tabu. Det der gøres i praksis er at gemme nogle parametre i forbindelse med løsningen. Sådanne parametre knytter sig ofte til, hvordan løsningen blev opnået, altså hvilket træk der blev udført for at danne den pågældende løsning.

Langtidshukommelse
TS-algoritmer indeholder ofte også en form for hukommelse, der rækker længere end blot tabu. En sådan langtidshukommelse kan være udformet på mange forskellige måder, men har altid til formål at øge algoritmens evne til finde gode løsninger. Vi vil i afsnit 5.2.2 og afsnit 5.3 komme med flere eksempler på, hvordan langtidshukommelse kan være udformet.

Aspirationskriterium
Det kan nogle gange være nødvendigt at muliggøre tabubelagte løsninger, hvis tabuet forhindrer algoritmen i at nå nye og bedre løsninger. Et aspirationskriterium kan udformes på adskillige måder, men vi nøjes her med at beskrive det mest udbredte. Kriteriet er som følger: En tabubelagt løsning bliver accepteret, hvis denne er bedre end den hidtil bedst opnåede. Dette kriterium er så oplagt, at det bruges af samtlige af de TS-algoritmer som vi har set beskrevet.

5.2 Et illustrativt eksempel

For at belyse hvorledes tabusøgning foregår, vil vi gennemgå et eksempel, som illustrerer grundprincipperne i en sådan. Til dette har vi designet et lille ruteplanlægningsproblem og programmeret en simpel TS-algoritme. Algoritmen har vi programmeret i MATLAB, i og med at det var det program vi var mest fortrolige med. Koden samt løsninger, opnået af algoritmen, kan ses som bilag bagerst i rapporten.

5.2.1 Problemtilfældet

Problemtilfældet er et klassisk VRP-problem med et depot og 10 kunder. Der er ingen tidsvinduer eller andre særlige begrænsninger. Placeringen af depotet og kunderne er givet ved kartesiske koordinater i planen, og omkostningen er givet ved den euklidiske afstand mellem disse. Data for problemtilfældet ses på figur 5.1.

Punkt 1 er depotet, og resten er kunder. Med disse placeringer bliver omkostningsmatricen som det ses på figur 5.2

Grafisk kan problemet illustreres som det ses på figur 5.3.

punkt	x-koordinat	y-koordinat	ordrestørrelse
1	0,0	0,0	0
2	−21,5	−4,0	20
3	−11,6	10,9	10
4	−5,2	−26,6	10
5	−2,0	−7,6	20
6	5,5	15,1	20
7	17,0	−28,2	20
8	20,2	17,3	30
9	20,7	−5,5	10
10	31,1	8,3	30
11	33,5	−12,0	10

Figur 5.1 Data for vores problemtilfælde.

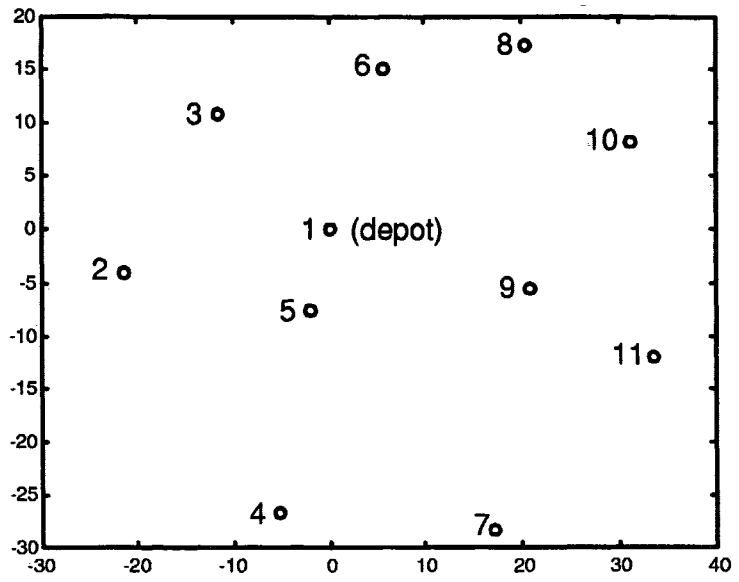
5.2.2 Algoritme

Algoritmen er kun i stand til at løse klassiske VRP-problemer, altså et problem hvis eneste begrænsning ligger i vognenes kapacitet. Kapaciteten i vores eksempel er desuden sat til at være den samme (80) for alle vognene.

Konstruktionsalgoritmen – sweep
Vores algoritme er udstyret med en konstruktionsalgoritme. Denne er en sweep-algoritme, som dog er lidt forsimplet. Traditionelle sweep-algoritmer optimerer de enkelte ruter efter, at kunderne er blevet tildelt en rute. Det gør vores ikke, da naboområdet i algoritmen muliggør optimering af de enkelte ruter, hvilket vi vil se nedenfor.

$$C = \begin{pmatrix} 0 & 21,9 & 15,9 & 27,1 & 7,9 & 16,1 & 32,9 & 26,6 & 21,4 & 32,2 & 35,6 \\ 21,9 & 0 & 17,9 & 27,9 & 19,8 & 33,1 & 45,5 & 46,8 & 42,2 & 54,0 & 55,6 \\ 15,9 & 17,9 & 0 & 38,0 & 20,8 & 17,6 & 48,4 & 32,4 & 36,2 & 42,8 & 50,6 \\ 27,1 & 27,9 & 38,0 & 0 & 19,3 & 43,1 & 22,3 & 50,7 & 33,4 & 50,4 & 41,4 \\ 7,9 & 19,8 & 20,8 & 19,3 & 0 & 23,9 & 28,0 & 33,4 & 22,8 & 36,8 & 35,8 \\ 16,1 & 33,1 & 17,6 & 43,1 & 23,9 & 0 & 44,8 & 14,9 & 25,6 & 26,5 & 39,0 \\ 32,9 & 45,5 & 48,4 & 22,3 & 28,0 & 44,8 & 0 & 45,6 & 23,0 & 39,1 & 23,1 \\ 26,6 & 46,8 & 32,4 & 50,7 & 33,4 & 14,9 & 45,6 & 0 & 22,8 & 14,1 & 32,2 \\ 21,4 & 42,2 & 36,2 & 33,4 & 22,8 & 25,6 & 23,0 & 22,8 & 0 & 17,3 & 14,4 \\ 32,2 & 54,0 & 42,8 & 50,4 & 36,8 & 26,5 & 39,1 & 14,1 & 17,3 & 0 & 20,4 \\ 35,6 & 55,6 & 50,6 & 41,4 & 35,8 & 39,0 & 23,1 & 32,2 & 14,4 & 20,4 & 0 \end{pmatrix}$$

Figur 5.2 Omkostningsmatricen for vores problemtilfælde.



Figur 5.3 Depotet og kundernes placering i planen.

Naboområdet

Naboområdet for vores algoritme er bestemt som værende alle løsninger, som kan nås ved følgende træk:

1. Fjern en kunde fra en rute og 'lap' ruten.
2. Indsæt kunden et andet sted. Dette kan enten være et andet sted i samme rute eller i en anden rute.

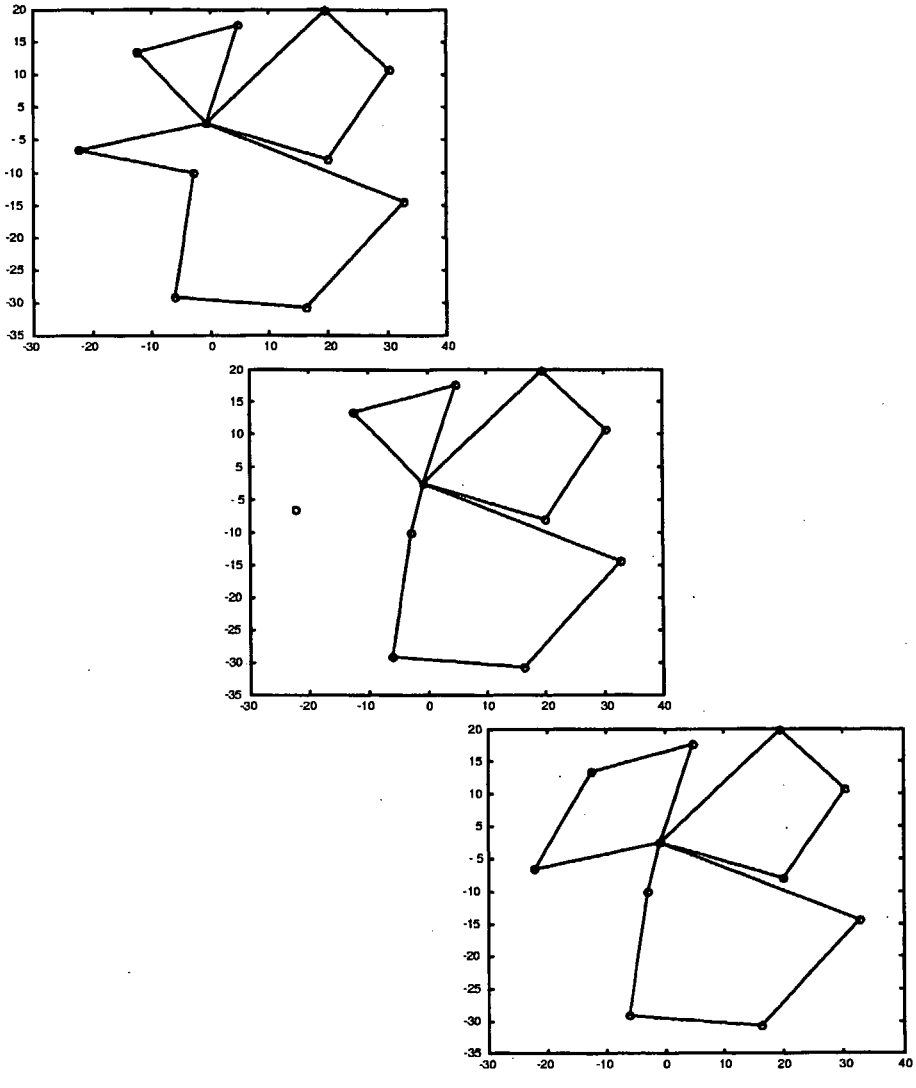
Et eksempel på et træk er vist på figur 5.4.

For hver iteration tager algoritmen en kunde ad gangen og beregner rutelængden for de løsninger, som kan nås ved at lave et træk med den pågældende kunde. I praksis udregnes besparelsen (som godt kan være negativ) ved at lave et givet træk, da dette er hurtigere. Træk som medfører overlast, betragtes som ulovlige og behandles ikke. Efter, at besparelsen ved alle lovlige træk er udregnet, vælges trækket med størst besparelse, og iterationen er færdig.

Trækket er valgt således, da det både kan bruges til at optimere de enkelte ruter og VRP-problemet som et hele. I mere avancerede algoritmer ville man nok vælge at benytte et træk, som ikke 'spilder kræfter' på at optimere de enkelte ruter, men kun virker ruter imellem. Efter hvert træk kunne der så benyttes en hurtig TSP-algoritme på de enkelte ruter. Dette ville give et mindre naboområde og dermed en hurtigere algoritme.

Kortidshukommelse

Algoritmen er udstyret med en konventionel tabu-korttidshukommelse. Tabu



Figur 5.4 Et træk udført af vores algoritme.

sættes på kunder, som lige er blevet flyttet, og forhindrer cykling mellem løsninger. På kort sigt forhindrer dette tabu faktisk ikke fuldstændigt, at der opstår cykling, hvilket bliver illustreret i det efterfølgende. På lidt længere sigt virker tabuet dog efter hensigten.

Tabulængden er fem iterationer, dvs. at en kunde som er blevet flyttet i en given iteration ikke kan flyttes i de fem følgende. Det virker måske voldsomt, da dette medfører, at kun halvdelen af kunderne i en given iteration må flyttes. Tabulængden på fem iterationer får dog algoritmen til at virke udmærket.

Langtidshukommelse
Algoritmen er udstyret med en straffunktion, som øger længden af en given løsning, dvs. lægges til omkostningsfunktionen, hvis denne løsning er opnået ved at flytte en kunde, som er blevet flyttet før. Straffen s er proportional med antallet af gange kunden er blevet flyttet:

$$s = k \cdot N_b \tag{5.1}$$

hvor N_b er antal gange kunde b er blevet flyttet, og k er en positiv proportionalitetskonstant. Straffunktionen har den effekt, at algoritmen ikke har mulighed for udelukkende at flytte enkelte af kunderne, men efterhånden må flytte dem allesammen.

k er blevet sat til omkring 3, da denne størrelse tvinger algoritmen til at flytte forskellige kunder uden dog at forhindre, at nogle kunder flyttes mere end andre. Denne værdi er bestemt ved en kalibrering, hvor vi systematisk har prøvet os frem med forskellige k-værdier.

Langtidshukommelsen nulstilles desuden, hver gang en ny bedste løsning er fundet, for på den måde at give algoritmen lov til at søge grundigt i denne løsnings naboområde.

Aspiration
Der er implementeret et sædvanligt aspirationskriterium, som annullerer tabu og straffunktion, hvis en løsning er bedre end den hidtil bedste løsning.

5.2.3 Løsning af eksemplet

Vi vil nu gennemgå algoritmens første iterationer, når den forsøger at løse det ovenfor opstillede problem. Vi håber på den måde at tydeliggøre, hvordan en tabu-søgning virker.

I det følgende vil der være et antal figurer. På disse vil der være en graf samt eventuelt et skema med tal. Skitsen viser ruten efter den pågældende iteration, og skemaet viser data for nogle af de træk, som algoritmen havde mulighed for at udføre i iterationen. Data for alle mulige træk er ikke medtaget, da dette ville fylde alt for meget – kun det bedste træk hørende til hver kunde er medtaget.

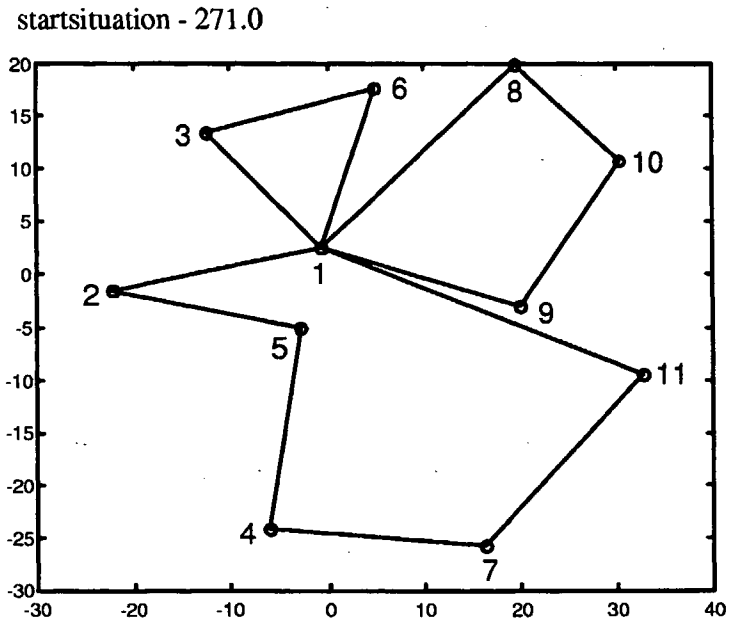
Et tabu-mærke på en given kunde vil blokere alle træk involverende denne kunde. Ligeledes vil straffunktionen straffe alle ruter dannet ved flytning af en given kunde lige meget. Således vil det valgte træk altid være en eller anden kundes bedste træk.

Skemaerne
Hver af rækkerne i et af skemaerne på figurerne angiver data for et givent træk.

- Første kolonne er rutelængden efter trækket.

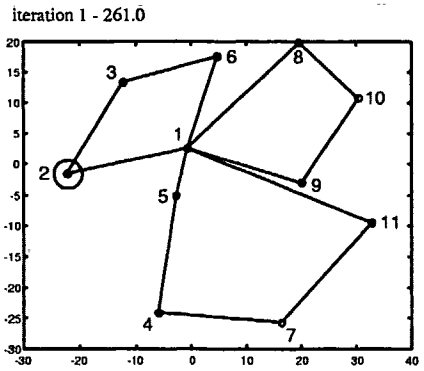
- Anden kolonne er straf-tillægget.
- Tredje kolonne er nummeret på den flyttede kunde.
- Fjerde og femte kolonne er numrene på kunderne den flyttede kunde sættes ind mellem.
- Sjette kolonne angiver den resterende tabutid for den flyttede kunde.

I de forskellige skemaer vil en * ud for rutelængden betyde, at det er dette træk som algoritmen har valgt i den pågældende iteration. Der vil desuden være sat en ring om kunden hørende til dette træk på figuren.

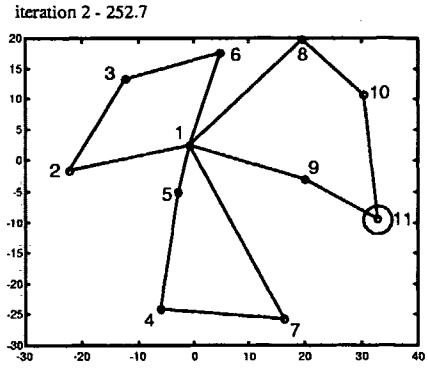


Figur 5.5 Startsituationen for løsningen af eksemplet.

Startsituation
Startsituationen er som nævnt lavet vha. en sweepalgoritme. Løsningsværdien, dvs. længden af ruten, er 271.0.



260.9578*	0	2.0000	3.0000	1.0000	0
270.9571	0	3.0000	1.0000	6.0000	0
284.7591	0	4.0000	2.0000	5.0000	0
265.5441	0	5.0000	1.0000	2.0000	0
270.9571	0	6.0000	3.0000	1.0000	0
287.4053	0	7.0000	11.0000	1.0000	0
282.0753	0	8.0000	9.0000	10.0000	0
282.0753	0	9.0000	8.0000	1.0000	0
282.0753	0	10.0000	8.0000	1.0000	0
262.6943	0	11.0000	9.0000	10.0000	0



265.5441	3.1110	2.0000	5.0000	4.0000	5
270.4710	0	3.0000	2.0000	1.0000	0
280.3384	0	4.0000	11.0000	1.0000	0
266.7543	0	5.0000	2.0000	1.0000	0
262.6495	0	6.0000	11.0000	1.0000	0
277.4060	0	7.0000	11.0000	1.0000	0
272.0761	0	8.0000	9.0000	10.0000	0
254.6377	0	9.0000	11.0000	1.0000	0
272.0761	0	10.0000	8.0000	1.0000	0
252.6950*	0	11.0000	9.0000	10.0000	0

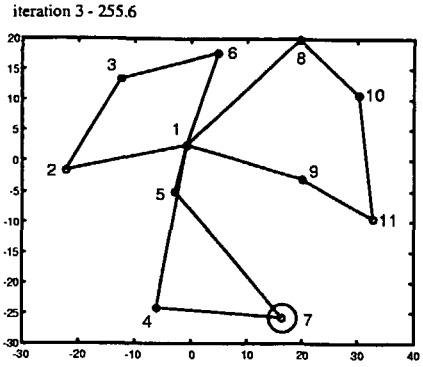
Figur 5.6 Iteration 1 og 2.

Iteration 1

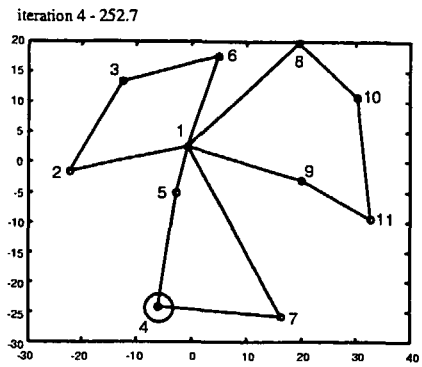
Iteration 1 er simpel, idet der ikke er nogen tabu-mærker eller straffe. Algoritmen vælger blot løsningen med lavest omkostning. I dette tilfælde opnås den laveste omkostning ved at flytte kunde 2 fra sin rute og sætte den ind i ruten bestående af depotet, kunde 3 og kunde 6 mellem depotet og kunde 3. Den nye omkostning er 261.0.

Iteration 2

I iteration 2 er der tabu på kunde 2 som følge af, at den blev flyttet i iteration 1. Desuden er der fra nu af en straf forbundet med ruter, som er opnået ved at flytte kunde 2. Tabuet og straffen får dog ikke betydning her i iteration 2, da den bedste løsning i iteration 2 opnås ved, at flytte kunde 11 og indsætte den mellem kunde 9 og 10. Dette giver en rutelængde på 252.7.



257.2813	0	2.0000	5.0000	4.0000	4
262.2082	0	3.0000	2.0000	1.0000	0
255.6274	0	4.0000	7.0000	1.0000	0
255.6274	0	5.0000	7.0000	1.0000	0
262.2082	0	6.0000	2.0000	1.0000	0
255.6274*	0	7.0000	5.0000	4.0000	0
269.5413	0	8.0000	1.0000	6.0000	0
263.6998	0	9.0000	11.0000	10.0000	0
270.0238	0	10.0000	8.0000	1.0000	0
260.9578	3.1110	11.0000	7.0000	1.0000	5



254.4172	0	2.0000	4.0000	1.0000	3
265.1406	0	3.0000	2.0000	1.0000	0
252.6950	0	4.0000	5.0000	7.0000	0
252.6950*	0	5.0000	4.0000	1.0000	0
265.1406	0	6.0000	2.0000	1.0000	0
252.6950	3.1110	7.0000	4.0000	1.0000	5
272.4737	0	8.0000	1.0000	6.0000	0
266.6322	0	9.0000	11.0000	10.0000	0
272.9562	0	10.0000	8.0000	1.0000	0
266.6322	3.1110	11.0000	1.0000	9.0000	4

Figur 5.7 Iteration 3 og 4.

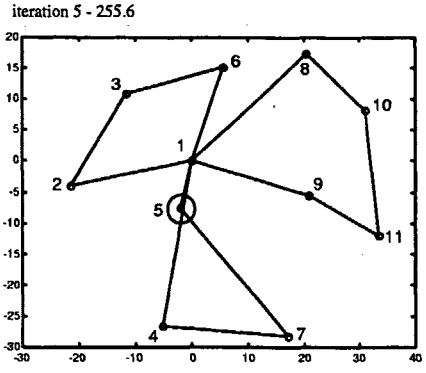
Iteration 3

I iteration 3 er der nu tabu på to kunder, nemlig kunde 2 og kunde 11. Derimod er der kun straf forbundet med kunde 11. Dette skyldes, at der blev opnået en ny bedste løsning i iteration 2 og straffene dermed blev nulstillet. I iteration 3 er der ingen forbedrende træk. En simpel descent-algoritme ville have stoppet her, men en TS-algoritme har lov at bevæge sig op fra lokale minima.

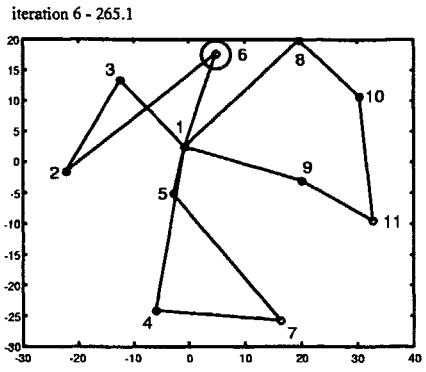
Algoritmen skal nu vælge den bedste af de løsninger, som ikke er tabubelagt. I iteration 3 er der tre træk, som fører til denne. Dette skyldes måden trækket er defineret på. Den bedste løsning kan opnås ved enten at flytte kunde 4, 5 eller 7. Værdien af løsningen er 255.6.

Iteration 4

Da trækket er symmetrisk, kan algoritmen bevæge sig frem og tilbage mellem to løsninger, hvis det ikke var fordi der var tabu-mærker. I iteration 3 kunne algoritmen komme til den samme løsning ved at flytte tre forskellige kunder. Der blev dog kun sat et tabu-mærke, hvilket betyder, at der stadig er to måder at vende tilbage til den tidligere løsning på. Det der sker i iteration 4 er netop tilbagevending, ved at flytte kunde 5, som ikke er tabubelagt. Denne cykling er ikke elegant og skyldes den noget alternative måde, hvorpå trækket er defineret. Det der gør trækket anderledes end sædvanlig er, at en by kan tages ud af en rute og sættes ind igen et andet sted i den samme rute.



257.2813	0	2.0000	4.0000	5.0000	2
262.2082	0	3.0000	2.0000	1.0000	0
255.6274*	0	4.0000	1.0000	7.0000	0
255.6274	3.1110	5.0000	1.0000	7.0000	5
262.2082	0	6.0000	2.0000	1.0000	0
255.6274	3.1110	7.0000	4.0000	5.0000	4
269.5413	0	8.0000	1.0000	6.0000	0
263.6998	0	9.0000	11.0000	10.0000	0
270.0238	0	10.0000	8.0000	1.0000	0
260.9578	3.1110	11.0000	1.0000	7.0000	3



254.4172	0	2.0000	1.0000	4.0000	1
265.1406	0	3.0000	2.0000	1.0000	0
252.6950	3.1110	4.0000	7.0000	5.0000	5
252.6950	3.1110	5.0000	1.0000	4.0000	4
265.1406*	0	6.0000	2.0000	1.0000	0
252.6950	3.1110	7.0000	1.0000	4.0000	3
272.4737	0	8.0000	1.0000	6.0000	0
266.6322	0	9.0000	11.0000	10.0000	0
272.9562	0	10.0000	8.0000	1.0000	0
266.6322	3.1110	11.0000	1.0000	9.0000	2

Figur 5.8 Iteration 5 og 6.

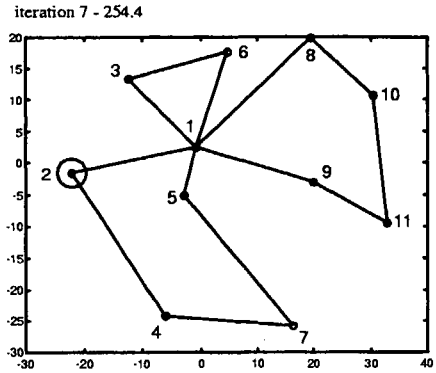
Iteration 5

Igen en tilbagevending til en tidligere løsning. Denne gang ved at flytte kunde 4.

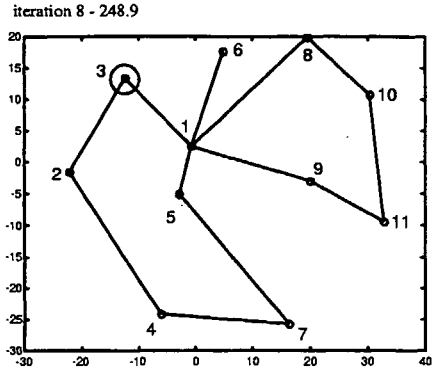
Iteration 6

De tre muligheder, som der var for at bevæge sig frem og tilbage mellem løsningerne i iteration 3, 4 og 5 er nu blevet tabu-belagt. Algoritmen er nu tvunget til at vælge et andet og dårligere træk, hvilket bliver at flytte kunde 6. Den nye løsning bliver 265.1.

Selv om algoritmen tillader cykling på kort sigt, forhindrer tabuet dog cykling på længere sigt. Hvis tabuet havde været så kort, at en cykling som set her alligevel blev mulig ville straffunktionen medføre, at algoritmen på længere sigt blev tvunget videre.



254.4172*	0	2.0000	1.0000	4.0000	0
255.6274	0	3.0000	2.0000	6.0000	0
262.2082	3.1110	4.0000	7.0000	5.0000	4
262.2082	3.1110	5.0000	1.0000	4.0000	3
255.6274	3.1110	6.0000	1.0000	3.0000	5
262.2082	3.1110	7.0000	1.0000	4.0000	2
281.9869	0	8.0000	6.0000	1.0000	0
276.1455	0	9.0000	11.0000	10.0000	0
282.4694	0	10.0000	8.0000	1.0000	0
276.1455	3.1110	11.0000	1.0000	9.0000	1



255.6274	3.1110	2.0000	1.0000	3.0000	5
248.8996*	0	3.0000	1.0000	2.0000	0
263.2697	3.1110	4.0000	7.0000	5.0000	3
257.2813	3.1110	5.0000	1.0000	2.0000	2
254.4172	3.1110	6.0000	1.0000	3.0000	4
263.2697	3.1110	7.0000	2.0000	4.0000	1
271.2636	0	8.0000	6.0000	1.0000	0
265.4221	0	9.0000	11.0000	10.0000	0
271.7460	0	10.0000	8.0000	1.0000	0
265.4221	3.1110	11.0000	1.0000	9.0000	0

Figur 5.9 Iteration 7 og 8.

Iteration 7

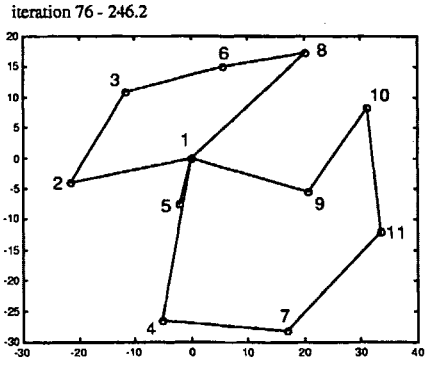
Det dårlige træk som algoritmen blev tvunget til at tage i iteration 6, giver i iteration 7 faktisk mulighed for at lave et træk, som giver en bedre løsning end den algoritmen forlod i iteration 6. Den nye løsning har værdien 254.4. Her kan man fornemme, hvorfor tabu-søgning er bedre end simpel naboområde-søgning.

Iteration 8

I iteration 8 finder algoritmen en ny bedste løsning. At det er muligt at finde denne løsning, er en konsekvens af, at algoritmen i iteration 6 blev tvunget til at tage et tilsyneladende dårligt træk. Dette træk ledte dog til et nyt område af løsningsmængden, hvilket muliggjorde fundet af en ny bedste løsning.

Vi har i iteration 1 til 8 ikke set effekten af langtidshukommelsen. Tabumærkerne alene har været i stand til at dirigere algoritmen rundt i områder af løsningsrummet, som ligger langt fra hinanden. Dette skyldes den relativt lange tabulængde.

For at illustrere langtidshukommelsen hopper vi frem til iteration 77. På dette tidspunkt har den bedste løsning haft værdien 241.2, som er opnået i iteration 28, som i øvrigt også er den bedste algoritmen finder overhovedet (jf. figur 5.11).



255.6805	15.5550	2.0000	6.0000	3.0000	4
255.6805	15.5550	3.0000	2.0000	1.0000	3
268.2463*	0	4.0000	1.0000	5.0000	0
246.1673	24.8880	5.0000	1.0000	1.0000	5
250.4714-	21.7770	6.0000	1.0000	8.0000	0
270.2306	0	7.0000	1.0000	4.0000	0
250.4714-	18.6660	8.0000	6.0000	3.0000	0
250.8520	21.7770	9.0000	11.0000	10.0000	2
250.8520	18.6660	10.0000	9.0000	1.0000	1
259.2484-	15.5550	11.0000	10.0000	9.0000	0

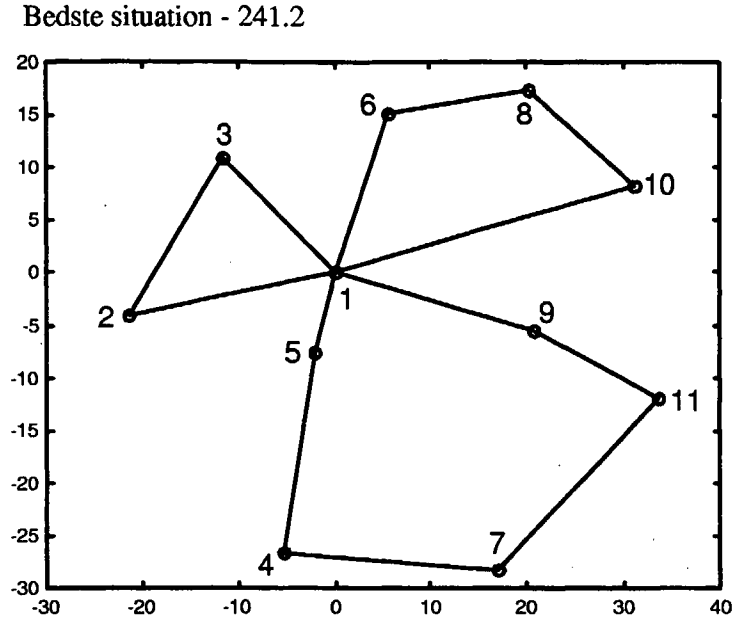
Figur 5.10 Iteration 76 og 77.

Iteration 77
Det ses at iteration 76 er god, men ikke den bedste. Derfor er straffene ikke nulstillet. I iteration 77 er der store straffe forbundet med at flytte mange af kunderne. Straffene medfører at det valgte skridt giver en rute, som er dårligere end tre af de andre ruter, der ikke er tabubelagt (markeret med -). Dette viser hvordan langtidshukommelsen gør algoritmen i stand til at 'slippe væk' fra 'dale' i løsningsrummet og søge nye veje.

5.3 Videregående TS-strategier

En simpel tabu-søgning vil oftest præstere utilfredsstillende. Det er nødvendigt at implementere strategier, som er i stand til at optimere tabu-søgningen, så bedre løsninger kan opnås. Følgende vil vi beskrive nogle af de mest almindelige eller de mest interessante optimeringsstrategier, som vi har inddelt på samme måde, som det ofte gøres i litteraturen [Rayward-Smith et al., 1996], [Reeves, 1993], [Hertz et al., 1994]. De fleste af strategierne bygger på styring af tabuhukommelsen, hvilket vi vil se.

I afsnittet har vi kun nævnt nogle af de strategier, som er præsenteret i litteraturen. Der findes mange flere, som vi ikke har beskæftiget os med. Formålet med dette afsnit er da heller ikke at komme med en komplet og systematisk gennemgang af strategier, men mere at komme med nogle interessante eksempler på sådanne. En mere systematisk og generel gennemgang af strategier kan findes i [Reeves, 1993], [Rayward-Smith et al., 1996], [Glover and Laguna, 1997].



Figur 5.11 Iteration 28 – den bedst fundne løsning.

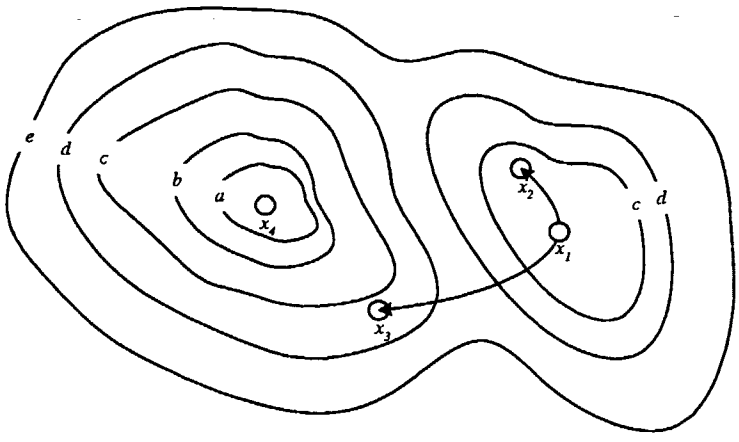
5.3.1 Spredning

Efter at en TS-algoritme har besøgt et lokalt minimum, er det vigtigt, at den søger videre i nye uudforskede områder af løsningsrummet. En strategi, som har dette formål, kaldes for *spredning*. Hvis spredningsstrategien virker efter hensigten, får den algoritmen til at søge videre i områder hvor der findes lokale optima, som er bedre end de som er fundet hidtil. Dette er illustreret på figur 5.12.

Frekvensbaseret tabu-hukommelse
Da TS helst skal udforske så store områder af løsningsrummet som muligt, vil det være fornuftigt at implementere strategier som straffer skridt, der fører til løsninger, som ligger i allerede besøgte områder af løsningsrummet. Dette kan gøres ved at indføre en straffunktion $s_f(x)$, som øger omkostningsfunktionen $c(x)$'s værdi for en given løsning x, hvis denne ligger i et i forvejen intensivt undersøgt område. Derved opnås en ny modificeret omkostningsfunktion $c_m(x)$:

$$c_m(x) = c(x) + s_f(x) \tag{5.2}$$

At en løsning ligger i et område, som tidligere er blevet undersøgt, vil sige, at den har noget tilfælles med tidligere løsninger. Det kunne f.eks. være, at visse vognes ruter er identiske med ruter i de tidligere fundne løsninger.



Figur 5.12 En del af løsningsmængden **X**, hvor der er indtegnet niveaukurver for omkostningsfunktionen; a er lavest og e er højest. Uden spredning ville algoritmen vælge løsningen x_2 og før eller siden vende tilbage til det lokale minimum x_1. Spredningen tvinger algoritmen til at vælge en anden løsning x_3, og får den derved til at søge videre i et nyt område hvor der findes et bedre lokalt minimum x_4.

Dette vil dog være upraktisk, da det bruger meget hukommelse, og man må derfor gøre noget andet. F.eks. kunne dette være at holde styr på de træk, som tidligere er blevet taget, og straffe nye træk, som minder om træk taget før. At to træk minder om hinanden kan f.eks. være, at det er den samme kunde, som bliver flyttet i de to træk. Ved at straffe sådanne træk nedsættes algoritmens muligheder for at bevæge sig tilbage til løsninger, som den har befundet sig i før [Reeves, 1993].

Hvis M er antallet af tidligere udførte træk, som minder om trækket der fører til løsningen x, kan straffunktionen $s_f(x)$, som tidligere beskrevet, udformes på følgende måde:

$$s_f(x) = \alpha \cdot M \tag{5.3}$$

hvor α er en positiv proportionalitetskonstant. Man kunne dog også implementere en sammenhæng mellem M og $s(x)$ som ikke er lineær.

Hvis $s(x)$ hele tiden blev lagt til $c(x)$ ville skridt, som fører til gode løsninger blive straffet hårdt, da disse naturligt vil optræde ofte. Derfor er det vigtigt, at $s(x)$ ikke bliver aktiveret hele tiden. Aktiveringen kunne f.eks. ske hver gang algoritmen nåede et lokalt minimum. Blev dette gjort, ville algoritmen have en tendens til at søge i retninger, som den ikke tidligere havde undersøgt, hver gang en bedre løsning ikke kan findes.

Flere initial-løsninger

Den enkleste og måske mest benyttede strategi til at sprede en TS-algoritmes

søgning er ved at lade den køre flere gange, hvor der startes fra forskellige initial-løsninger. Er initial-løsningerne på afgørende vis forskellige, vil dette resultere i afsøgning af områder af løsningsrummet, som ligger langt fra hinanden. Ved at sammenligne resultaterne fra flere kørsler, kan den bedste findes. Denne løsning er oplagt mindst lige så god som løsningen fundet, hvor algoritmen kun kører en gang.

Det kræver selvfølgelig mere computerkraft, hvis algoritmen skal køre flere gange. Den ekstra computerkraft kunne også bruges til at lade en enkelt kørsel af algoritmen gennemløbe flere iterationer, inden den stoppes. Har algoritmen indbygget gode spredningsstrategier, vil det nogle gange føre til bedre løsninger at øge antallet af iterationer i stedet for at køre flere gange. Dette er tilfældet i algoritmen udviklet af Cordeau, Laporte og Mercier [Cordeau et al., 2000b]. De finder, at deres algoritme finder bedre løsninger ved en gang at køre 10^5 iterationer end ved 10 gange at køre 10^4 iterationer.

Tilfældighed
En tredje måde, at sprede søgningen på, er ved at indføre tilfældigheder i algoritmen. Tilfældigheder medfører, at en TS-algoritme ikke opfører sig ens, selvom udgangspunktet er det samme. Tilfældige variationer af parametre kan indføres på mange forskellige måder, men en af de mest benyttede er variabel tabulængde. Det kan f.eks. gøres på en sådan måde, at tabulængden kan variere inden for et bestemt iterationsinterval $[t_{st}, t_{sl}]$ [Rochat and Semet, 1994].

Tilfældighed indgår også i adaptiv hukommelse, hvilket vil blive beskrevet nedenfor.

5.3.2 Intensivering

En anden vigtig type af strategier er *intensiveringsstrategier*, hvilket vil sige intens søgning i områder af løsningsrummet, hvor det formodes, at gode løsninger kan findes. Hvis der udelukkende bruges spredning, er der en risiko for, at algoritmen vil begynde at gå nye veje, inden et interessant område er grundigt undersøgt.

En måde at intensivere søgningen på er ved at nulstille straffunktioner (der kan sagtens være mange) ved fund af hidtil bedst fundne løsning. Dette giver algoritmen mulighed for at søge frit i nærheden af denne. Nulstilning af parametre tages ligeledes i brug i vor TS-implementering.

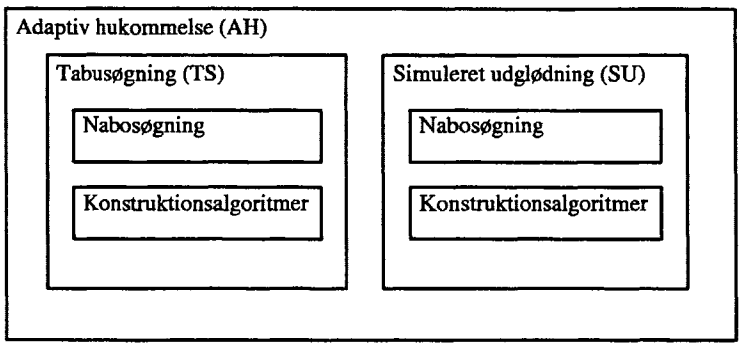
Nulstilning af parametre er en meget simpel intensiveringsstrategi. Nedenfor beskrives adaptiv hukommelse, som er en mere ingeniøs intensiveringsstrategi.

5.3.3 Adaptiv hukommelse

Af de koncepter til styring af TS-algoritmer, som er fremkommet indenfor de seneste år er *adaptiv hukommelse* (AH) ifølge [Laporte et al., 2000] et af de mest

interessante. Konceptet blev første gang præsenteret i [Rochat and Taillard, 1995].

Adaptiv hukommelse er ikke begrænset til TS, men er en strategi, som kan bruges på alle former for NS-algoritmer i forbindelse med VRP. AH er ikke en inkorporeret del af den pågældende algoritme, men er nærmere en måde at håndtere hele NS-algoritmer på. Fremover vil vi bruge udtrykket AH-algoritme for måden, hvorpå AH håndterer TS-algoritmer. Skematisk kan dette vises som på figur 5.13.



Figur 5.13 Strukturen i forbedringsheuristikker.

Initialisering

TS-algoritmen bruges først til at generere I løsninger til et VRP. For ikke at bruge for meget regnekraft på dette skal TS-algoritmen ikke udføre for mange iterationer i hver af de I gennemregninger. Ved at udregne I løsninger håber man på, at nogle af disse indeholder ruter, som er magen til ruter indeholdt i meget gode løsninger, såkaldte *elite-løsninger*. Dette er ikke urealistisk, da TS-algoritmen trods alt finder gode løsninger.

Ruterne bliver hver især associeret til en værdi svarende til omkostningsfunktionen $c(x)$, hvor x er den løsning ruten er en del af. Dette gøres, da gode løsninger oftere må formodes at have bedre ruter end dårlige løsninger.

Ruter som kun indeholder en enkelt kunde bliver frasorteret. Vi vil senere se, at AH-algoritmen sørger for, at sådanne ruter bliver indsat i løsningen, hvis det er nødvendigt. De resterende ruter indsættes i mængden T, som bliver organiseret som en liste. T sorteres således, at de ruter med lav værdi bliver placeret først i listen. Dette er blot en måde, hvorpå der kan holdes styr på T.

Intensivering og spredning

Efter listen T er dannet, skal ruter derfra sammensættes til nye løsninger. Dette sker ved at vælge ruter fra T på en sådan måde, at der er høj sandsynlighed for, at ruterne står øverst på listen – dvs. ruterne har tilhørt gode løsninger.

Ruterne udvælges succestivt således, at hvis en rute indeholdende en bestemt kunde er udvalgt, forhindres udvælgelsen af andre ruter indeholdende den samme

kunde.

Hvis der efter denne udvælgelse er kunder, som ikke er inkluderet i løsningen, kan disse ses som et selvstændigt og mindre VRP. Løsningen af dette mindre problem kan så blive lagt til de, af den foregående proces, udvalgte ruter. Der vil nu forefindes en ny løsning.

Den nye løsning er ikke nødvendigvis god endnu, men den indeholder forhåbentlig gode elementer, da udvælgelsen af ruter til den nye løsning er afhængig af, hvor gode løsninger de førhen var en del af. For at forbedre løsningen bruges TS-algoritmen på sædvanlig vis.

De ruter som indgår i løsningen x, fundet af TS-algoritmen, mærkes med værdien svarende til $c(x)$. Disse ruter indsættes nu i T, og T sorteres som før. For at undgå at T bliver for stor, begrænses den så $|T| = L$, hvor L er den maksimalt ønskede længde. Det gøres ved at fjerne de $|T| - L$ sidste elementer.

Hele processen gennemløbes et antal gange for på den måde at forbedre ruterne i T og skabe bedre og bedre løsninger.

Postoptimering

Som AH-algoritmen skrider frem, kommer listen T til at indeholde en større og større andel af ruter fra gode løsninger. Postoptimeringen udføres hver gang AH-algoritmen har udført et træk. Idéen med postoptimeringen er at undersøge, hvorvidt de i T fundne ruter tilsammen kan danne en god rute. Dette problem er et traditionelt *set covering-problem*[1], hvilket kan løses til optimalitet indenfor en overskuelig tidshorisont.

Grunden til, at denne postoptimering kan give forbedringer, er, at intensiverings- og spredningsprocessen ikke nødvendigvis udvælger den bedste kombination af ruter. Intensiverings- og spredningsprocessen er dog nødvendig, idet postoptimeringen ikke i sig selv er i stand til at producere nye ruter.

Oversigt over AH

Følgende er en oversigt over de enkelte skridt i AH-algoritmen:

Initialisering

1. Generér I forskellige løsninger med TS-algoritmen.
2. Markér hver rute t med værdien $c(x)$ svarende til løsningen x hvori t indgår.
3. Fjern ruter indholdende kun en kunde.
4. Indsæt resten af ruterne i T.

[1] Et set covering-problem drejer sig helt basalt om at finde den delmængde F af en mængde af delmængder N af M, som dækker hele mængden M, og giver laveste omkostning.

5. Sortér T efter deres værdier således, at ruter med lave værdier står først.

Intensivering og spredning

6. Dan to nye lister T' og S og sæt $T' = T$ og $S = \varnothing$
7. Sålænge $T' \neq \varnothing$, gentag følgende:
   (a) Vælg et t tilhørende T' således, at der er stor sandsynlighed for, at t har lav værdi.
   (b) Sæt $S = S \bigcup \{t\}$.
   (c) Fjern alle ruter fra T' som har en eller flere kunder tilfælles med t.
8. Løs VRP for de kunder ikke indeholdt i ruter i S.
9. Tilfør ruterne fra løsningen i 8. til S. Den samlede løsning kaldes S'.
10. Optimér S' med TS-algoritmen.
11. Markér ruterne i den optimerede løsning og indsæt dem i T på samme måde som i træk 2 til 5.
12. Hvis $|T| - L > 0$ fjern de $|T| - L$ sidste elementer i T
13. Udfør postoptimering vha. set covering på T.
14. Gentag 6.-13. et på forhånd fastsat antal gange.

AH-algoritmen er både spredning og intensivering i ét. Spredningen består i, at der fra start bliver dannet I forskellige løsninger. Desuden er det en form for spredning, at ruterne fra T bliver 'tilfældigt' udvalgt.

Intensivering består i, at der er større sandsynlighed for at inkludere gode ruter i den nye løsning. Desuden er der en kraftig intensivering i, at T hele tiden bliver opdateret med ruter fra bedre og bedre løsninger. I T er der intet i vejen for, at en bestemt tur kan indgå flere gange. Da meget gode ruter med stor sandsynlighed indgår i elite-løsninger, bliver T derfor med tiden domineret af gode ruter, hvoraf nogle af dem indgår flere gange.

Det koster selvfølgelig noget ekstra computerregnekraft at indføre denne strategi, idet TS-algoritmen aktiveres flere gange i løbet af processen. Dette medfører, at antallet af iterationer, TS-algoritmen udfører, her skal nedsættes i forhold til, hvis den stod alene. På trods af dette har AH-algoritmen vist sig at kunne producere meget gode løsninger. Da den første gang blev præsenteret, producerede den rekord-løsninger (bedste løsning eller lige så god som bedste løsning) til alle på nær 10 af Solomon-testene [Rochat and Taillard, 1995].

5.3.4 Andre strategier

I det følgende vil vi nævne andre strategier, som kan forbedre en TS-algoritme. Det er strategier, som ikke rigtigt karakteriseres som spredning eller intensivering, men som kan have stor betydning for opførslen af TS-algoritmen.

Relaksation af begrænsninger
Begrænsningerne på et givet VRP kan ofte være dem, der forhindrer en TS-algoritme i at slippe ud af et bestemt område af løsningsrummet. Hvis der er mange strenge begrænsninger, vil disse måske bevirke, at mange af de naboløsninger som algoritmen ellers ville vælge bliver ulovliggjort. Begrænsningerne kan komme til at virke som en for algoritmen uoverstigelig 'bjergkæde', som fanger den i en lille del af løsningsrummet.

En måde, at overkomme dette problem på, er, at indføre en 'relaksation' af begrænsningerne, hvilket betyder at lovliggøre brud på disse. Det giver selvfølgelig ikke mening blot at ophæve begrænsningerne, da det ville give ubrugelige løsninger. En fornuftigere fremgangsmåde ville være at tillade overskridelse af parametrene, men dog straffe sådanne. Det kunne gøres på samme måde som ved frekvensbaseret tabu-hukommelse, altså ved at indføre en straffunktion på omkostningsfunktionen:

$$c_m(x) = c(x) + s_r(x) \tag{5.4}$$

hvor $c_m(x)$ er modifikationen af omkostningsfunktionen $c(x)$ med straffunktionen $s_r(x)$. Straffunktionen kan (og gør det ofte) afhænge af, hvor meget begrænsningerne bliver overskredet [Cordeau et al., 2000b], [Rochat and Semet, 1994]. Det er ofte en lineær afhængighed, men man kan også vælge, at afhængigheden skal være polynomiel (af mere end første grad) eller eksponentiel, hvis man ønsker at straffe store overskridelser relativt hårdere end små.

Da overskridelser af begrænsninger bliver straffet, vil TS-algoritmen foretrække løsninger uden overskridelser. På den måde vil der som algoritmen kører fremkomme løsninger uden overskridelser af begrænsninger, selvom dette faktisk er tilladt. Ved at holde styr på de bedste løsninger, som ikke overskrider begrænsningerne, vil en sådan strategi stadig være i stand til at finde gode lovlige løsninger.

Automatisk tuning af parametre
Der indgår mange parametre i en TS-algoritme. Ting som tabulængde, straffunktioner, størrelse af T i AH-algoritmer m.v. kan antage mange forskellige værdier, og det er svært på forhånd at vide hvilke værdier af disse, der giver de bedste resultater.

Ofte vil der skulle udføres praktiske test-kørsler med algoritmerne for at finde de bedst egnede parametre [Rochat and Semet, 1994]. Dette er sådan set ikke

noget problem, da en given aftager af et ruteplanlægningsværktøj vil have VRP-tilfælde som minder om hinanden. Et bestemt sæt af parameterindstillinger, som virker godt på én af aftagerens tilfælde, vil derfor sandsynligvis også virke godt på et andet af tilfældene. En ruteplanlægger kan hjælpe en sådan aftager med at indstille hans algoritmes parametre.

Problemet opstår først, når der ønskes algoritmer, som virker universelt, altså en algoritme som virker på en lang række af VRP-situationer. Sådanne algoritmer er især vigtige i kommercielle ruteplanlægningsværktøjer.

En idé til at overkomme problemet med parameterindstillinger er foreslået i [Cordeau et al., 2000b]. De foreslår at indføre en simpel strategi, som kan 'tune' parametrene i straffunktioner knyttet til overskridelse af begrænsninger. I deres algoritme indgår der følgende modificerede omkostningsfunktion:

$$c_m(x) = c(x) + \alpha \cdot q(x) + \beta \cdot d(x) + \gamma \cdot w(x) \tag{5.5}$$

hvor $q(x)$ er den totale overskridelse af vognenes kapacitet, $d(x)$ den totale overskridelse af rutelængden (evt. køre- og hviletidsbestemmelser) og $w(x)$ er den totale overskridelse af kundernes tidsvinduer for løsningen x. α, β og γ er positive konstanter. Det er α, β og γ som dynamisk bliver tunet, hvilket sker på følgende måde: Lad $\delta > 0$ være et reelt tal. Hvis x overskrider begrænsningerne på vognenes kapacitet, bliver α ganget med $1 + \delta$, hvis ikke bliver α divideret med $1 + \delta$. Ligeledes gøres for køretid og tidsvinduer.

Denne strategi medfører, at det bliver sværere at overskride en bestemt begrænsning, hvis de foregående iterationer også har overskredet den. Det er altså lovligt at overskride en begrænsning, men det er ikke plausibelt i mange efterfølgende iterationer. Dette er altså en måde aktivt at gå ind og tvinge en algoritme til med mellemrum at vende tilbage til lovlige situationer.

6 Løsningsstrategi

I dette kapitel vil vi vurdere, hvor gode tabusøgningsalgoritmer er, og hvordan FDB-lignende VRP-problemer bør løses med disse.

6.1 Løsningskvalitet af TS-algoritmer

Vi vil her først sammenligne, hvor gode løsninger TS-algoritmer er i stand til at generere. Derudover vil vi undersøge, hvordan løsningskvaliteten afhænger af antallet af iterationer.

6.1.1 Sammenligning af tabusøgningsalgoritmer

For at sammenligne hvor gode forskellige TS-algoritmer er, bruges som tidligere nævnt empiriske tests. Vi vil i dette afsnit sammenligne løsningskvaliteten for forskellige TS-algoritmer, som er blevet præsenteret i litteraturen inden for de sidste ti år. Sammenligningen gøres på baggrund af Solomons testsæt, da alle algoritmerne er testet på dette. Vi sammenligner algoritmernes bedste løsninger med hinanden og med de optimale løsninger, hvor disse er til rådighed. Vi nøjes med at se på data for R-100, C-100 og RC-100 serierne, da 200 serierne har mange kunder pr. rute, hvilket FDB og lignende situationer ikke har.

De sammenlignede algoritmer samt deres bedste løsninger stammer fra [Rochat and Taillard, 1995], [Chiang and Russel, 1997], [Taillard et al., 1997] og [Cordeau et al., 2000b] og de optimale løsninger er fundet i [Cordeau et al., 2000a]. I figurerne i dette afsnit bruges forkortelserne RT, CR, T, C for de forskellige algoritmer. I figurerne er der, ud over de forskellige algoritmers bedste løsning af de forskellige problemer, angivet forskellen mellem disse og den optimale løsning. Både den mindste og den største forskel i forhold til optimalløsningen er angivet. Et '—' betyder, at problemtilfældet ikke er blevet løst til optimalitet.

R100-serien
Det ses på figur 6.1, at TS-algoritmers løsninger højst afviger 10% fra de optimale løsninger i R-100 serien, og at den bedste TS-algoritme aldrig afviger mere end ca. 5 %. Derudover ses, at afvigelsen i gennemsnit er på lidt over 3%. Afvigelsen mellem den bedste og den dårligste TS-algoritme er 1,1% i gennemsnit.

problem	RT	CR	T	C	optimal	afvigelse (%)
R101	1650,8	1651,0	1650,8	1650,8	1637,7	0,8 → 0,8
R102	1486,1	1498,1	1487,6	1488,1	1466,6	1,3 → 2,1
R103	1213,6	1299,2	1294,2	1299,8	1208,7	0,4 → 7,0
R104	982,0	986,0	982,7	984,0	—	—
R105	1377,1	1382,1	1377,1	1377,1	1355,3	1,6 → 1,9
R106	1252,0	1255,3	1259,7	1253,2	1234,6	1,4 → 2,0
R107	1159,9	1124,7	1126,7	1113,7	1064,6	4,4 → 8,2
R108	981,0	971,0	968,6	964,4	—	—
R109	1235,7	1213,2	1214,5	1194,6	1146,9	4,0 → 7,2
R110	1080,4	1174,5	1080,4	1125,0	1068,0	1,1 → 9,1
R111	1129,9	1119,5	1104,8	1108,9	1048,7	5,1 → 7,2
R112	953,6	970,9	964,0	957,0	—	—
gennemsnit*	1287,3	1301,9	1288,4	1290,1	1247,9	3,2 → 4,3

Figur 6.1 Resultater af TS-algoritmer på Solomons R100 serie. *Gennemsnittet er kun taget af de resultater, hvor en optimal løsning er fundet.

C100-serien

Som det ses af figur 6.2, afviger alle algoritmerne minimalt (0,2%) fra den optimale løsning i C100 serien. At løsningerne er så gode, skyldes givetvis måden tidsvinduerne i C-100 serien er blevet lagt på (jf. afsnit 4.4.1). Det er bemærkelsesværdigt, at alle TS-algoritmerne finder de samme løsninger i alle tilfælde, samtidig med at de løsninger de finder afviger fra de optimale.

problem	RT	CR	T	C	optimal	afvigelse (%)
C101	828,9	828,9	828,9	828,9	827,3	0,2
C102	828,9	828,9	828,9	828,9	827,3	0,2
C103	828,1	828,1	828,1	828,1	826,3	0,3
C104	824,8	824,8	824,8	824,8	822,9	0,2
C105	828,9	828,9	828,9	828,9	827,3	0,2
C106	828,9	828,9	828,9	828,9	827,3	0,2
C107	828,9	828,9	828,9	828,9	827,3	0,2
C108	828,9	828,9	828,9	828,9	827,3	0,2
C109	828,9	828,9	828,9	828,9	827,3	0,2
gennemsnit	828,4	828,4	828,4	828,4	826,7	0,2

Figur 6.2 Resultater af TS-algoritmer på Solomons C100 serie.

RC100-serien

På figur 6.3 ses, at algoritmerne i tilfældene RC101–103 afviger under 10% fra den optimale, som det var tilfældet i R-100 serien. RC105 har TS-algoritmerne svært ved at håndtere og finder løsninger, som afviger med 8,6 til 14,7%. Afvigelsen mellem den bedste og den dårligste algoritme er i denne serie 4,7% i

gennemsnit.

Tager man de bedste resultater opnået af TS-algoritmerne fra de tre skemaer, afviger disse i gennemsnit med 1,5% fra optimum.

problem	RT	CR	T	C	optimal	Afvigelse (%)
RC101	1623,6	1642,8	1696,9	1708,8	1619,8	1,4 → 5,5
RC102	1477,5	1541,0	1554,8	1558,1	1457,4	1,4 → 6,9
RC103	1262,0	1302,7	1264,3	1263,0	1258,0	0,3 → 3,6
RC104	1135,8	1155,1	1135,8	1135,5	—	—
RC105	1733,6	1735,8	1643,4	1644,4	1513,7	8,6 → 14,7
RC106	1384,9	1395,4	1448,3	1427,1	—	—
RC107	1231,0	1235,3	1230,5	1231,5	—	—
RC108	1170,7	1171,4	1139,8	1149,8	—	—
gennemsnit*	1524,2	1555,6	1538,9	1543,6	1462,2	2,9 → 7,6

Figur 6.3 Resultater af TS-algoritmer på Solomons RC100 serie. *Gennemsnittet er kun taget af de resultater, hvor en optimal løsning er fundet.

6.1.2 Beregningstid kontra løsningskvalitet

Resultaterne fundet i afsnit 6.1.1 svarer ikke helt til de afvigelser, som kan forventes ved brug af en algoritme for et virkeligt problemtilfælde. De fundne løsninger er de *bedste* som de forskellige algoritmer har kunnet finde overhovedet. En vilkårlig kørsel kan ikke forventes at være lige så god.

I [Cordeau et al., 2000b] er det blevet undersøgt, hvor meget dårligere løsningerne bliver, hvis antallet af iterationer sættes ned. I artiklen er de bedste resultater blevet fundet efter, at algoritmen har kørt i op til et døgn. Sættes kørselstiden ned til ca. fem minutter, forringer det dog højst løsningskvaliteten med 10%. I gennemsnit er forringelsen under 5%. Sammenlignes disse tal med resultaterne fra afsnit 6.1.1 kan man konkludere, at der findes TS-algoritmer som i løbet af kort tid kan finde løsninger, som i gennemsnit afviger mindre end 10% fra optimum.

6.2 Algoritmen

Vi vil her beskæftige os med, hvordan en tabusøgningsalgoritme skal udformes, hvis den skal bruges til løsning af FDB-lignende situationer. Vi har i afsnit 5.3 beskrevet forskellige strategier, som kan implementeres i TS-algoritmer, og vi vil her give vores vurdering af, hvordan nogle af disse med fordel kan bruges til løsning af FDB-lignende situationer.

De strategier vi har valgt at beskrive stammer fra algoritmer, som har fundet gode løsninger – i hvert fald på de testeksempler hvor de er afprøvet. Det ligger

nok over vores evne at vurdere, hvordan de enkelte strategier skal kombineres til en god algoritme, da dette kræver stor erfaring indenfor området. Desuden er det næsten umuligt at forudsige, hvor god en algoritme er, og kvaliteten af algoritmer bliver derfor som regel afgjort ved empiriske tests. I praksis viser det sig som det er set i afsnit 6.1.1, at flere avancerede TS-algoritmer er næsten lige gode.

6.2.1 Ekstra begrænsninger / udvidelser

Virkelige VRP-tilfælde har, som vi tidligere har været inde på, næsten altid udvidelser eller flere begrænsninger end de testeksempler, som de forskellige algoritmer bliver afprøvet på. Det må derfor være et krav, at en algoritme skal være i stand til at håndtere udvidelser og mange begrænsninger, hvis den skal kunne bruges effektivt i praksis. Vi vil i dette afsnit gennemgå omkostningsfunktionen samt de (relevante) begrænsninger, som er nævnt i afsnit 3.1. Disse ses her.

1. Trafikforhold.
2. Kapacitetsbegrænsninger.
3. Vej- og kundebegrænsninger.
4. Tidsvinduer.
5. Køre- og hviletidsbestemmelser.
6. Servicetid.
7. Begrænsninger knyttet til depotet.
8. Backhaul.
9. Flere ruter pr. vogn.

Omkostningsfunktionen, udtrykt ved omkostningsmatricen, har ikke nogen direkte indflydelse på selve løsningsmetoden. Algoritmen virker ens, hvad enten omkostningsfunktionen ser ud på den ene eller den anden måde. Elementerne i omkostningsmatricen vil som regel være udregnet på forhånd. Hvis algoritmen skulle kunne tage højde for f.eks. trafikforhold, kunne omkostningsmatricen dog udvides, så den var tidsafhængig, dvs. c_{ij} udvides til c_{ijt}, hvilket ikke ville kræve de helt store ændringer i algoritmerne.

Flere af de nævnte begrænsninger kan uden videre håndteres af de i litteraturen beskrevne algoritmer. Det skyldes, at begrænsningerne er inkluderet i standardformuleringer for VRPTW, som den f.eks. er præsenteret i afsnit 3.3.1. Dette gælder kapacitetsbegrænsninger, tidsvinduer og servicetid. Vægt- og volumenbegrænsninger er blot specialtilfælde af kapacitetsbegrænsninger, og kan af algoritmerne håndteres på samme måde som disse.

Algoritmerne kan på en simpel måde modificeres, så de kan håndtere vej- og kundebegrænsninger. Det kan gøres ved blot at kontrollere løsningerne for overskridelser af disse. Nedenfor vil vi dog se et eksempel på, hvordan håndteringen af disse begrænsninger kan gøres på en bedre måde.

At flere ruter kan køres af den samme vogn kan som nævnt i afsnit 3.3.2 håndteres ved at lave 'kopier' af hver enkelt vogn og sikre sig, at de originale vogne og deres kopiers arbejdstider ikke overlapper. En sådan løsning kræver dog, at algoritmerne kan angive separate starttider for de forskellige vogne. Er algoritmen i stand til dette kan problemet med et begrænset antal læsseporte i depotet også håndteres.

Vi vil vende tilbage til udvidelser i form af backhaul og håndtering af køre- og hviletidsbegrænsninger.

Begrænsninger har overordnet set den egenskab, at de gør et antal mulige løsninger ulovlige. Dette har bl.a. den effekt, at antallet af lovlige løsninger i naboområdet til en given løsning bliver mindre. Som beskrevet i afsnit 5.3.4 kan dette bevirke, at algoritmen bliver fanget i et lille område af løsningsrummet. Dette problem løses elegant af [Cordeau et al., 2000b] ved at tillade (straffede) overskridelser af begrænsningerne og indføre et system til automatisk at styre de indgående parametre. I [Cordeau et al., 2000b] bruges denne strategi til begrænsningerne på kapacitet, tidsvinduer og total rutelængde, men strategien kan uden videre overføres til andre former for begrænsning.

Benyttes denne strategi vil løsningerne i nogle af iterationerne være ulovlige. Jo flere begrænsninger strategien benyttes på, jo større risiko er der for, at løsningerne bliver ulovlige, og i værste fald finder algoritmen på intet tidspunkt lovlige løsninger. Det kan på denne baggrund være værd at overveje, hvorvidt man skal tillade slutløsninger, som overskrider visse begrænsninger. Dette vil vi vende tilbage til i diskussionen.

Selv om ovennævnte strategi kan bruges til mange forskellige begrænsninger, kan andre strategier også med fordel bruges. Et godt eksempel på dette er [Rochat and Semet, 1994], hvor et virkelighedstro ruteplanlægningsproblem bliver behandlet. I problemet kan alle vogne ikke besøge alle kunder, da nogle kunder er svære at få acces til. Algoritmen præsenteret i artiklen konstruerer en initialløsning, hvor disse kunder bliver rutelagt før de andre. Dette medfører, at de særlige vogne, som kan nå de svært tilgængelige kunder, primært bliver brugt til at besøge dem i stedet for andre kunder.

Udvidelser kræver også nogle gange specialbehandling. Dette gælder f.eks. hvis køre- og hviletidsbegrænsninger skal overholdes, da der her skal indlægges pauser. Sådanne begrænsninger behandles ligeledes i [Rochat and Semet, 1994]. Pauser indlægges her ved at tilføje ekstra kunder, som har tidsvinduer svarende til det tidsinterval, hvori pausen skal holdes. Der er ingen afstand og rejsetid mellem sådanne kunder og almindelige[1], og servicetiden hos dem svarer til pauselængden. I begrænsningerne indgår så, at alle vogne, hvis det er nødvendigt,

[1]Dvs. at pauserne ikke kan afholdes undervejs på rejsen fra kunde i til kunde j. I [Rochat and Semet, 1994] holdes pauserne umiddelbart efter et besøg hos en kunde.

skal besøge en sådan kunde, hvilket medfører at chaufføren holder den krævede pause.

Man kunne også benytte sig af vores tilgang, altså lave en kopi af alle kunder (jf. afsnit 3.3.2). Denne kan dog blive problematisk når man foretager træk, da der skal holdes styr på niveauerne.

Backhaul vil også kræve en eller anden form for udvidelse af de eksisterende algoritmer. Havde der været tale om pickup- and delivery, kunne ordrerne fra de enkelte kunder blot tillades at være negative, hvilket ville have indikeret en opsamling. Algoritmen skulle så holde styr på, at den samlede ordremængde på en given rute aldrig overskred bilens begrænsninger. Backhaul er noget mere besværlig, da det her kræves, at kunder med afhentning først må besøges, når alle leveringskunder i ruten er besøgt. Et træk udført af algoritmen kunne let forstyrre denne orden, da en vogn, som et givet sted på ruten ikke havde afleveret alle varer, blev sat til at køre til en by, hvor der skulle samles varer op.

6.2.2 Adaptiv hukommelse

Adaptiv hukommelse, som er beskrevet i afsnit 5.3.3 virker som en meget lovende strategi. Dette skyldes især, at den kan bruges på enhver form for nabosøgnings-algoritme inklusive simuleret udglødning og tabusøgnings-algoritmer. Lige meget hvilken form for TS-algoritme, som vælges til løsning af problemet, kan adaptiv hukommelse altid benyttes som en ydre skal.

Ser man på hvor gode resultater en AH-algoritme kan producere [Rochat and Taillard, 1995], virker strategien som en, der er værd at overveje at bruge.

7 Diskussion

I denne diskussion vil vi holde modellering og løsningsmetoder op mod hinanden, for at finde ud af hvordan FDB-lignende situationer håndteres bedst. Derudover vil vi diskutere konklusionerne draget gennem rapporten.

7.1 Model og løsningsmetoder

Måden modellen opstilles på afhænger af hvilken løsningsmetode, der ønskes anvendt. I vores modellering benytter vi en traditionel formulering inspireret af matematisk programmering. Det virker måske derfor overraskende, at vi i forbindelse med løsningen af modellen overhovedet ikke beskæftiger os med matematisk programmering. Dette skyldes, at heuristikker ikke nødvendigvis implementeres gennem disse traditionelle matematiske formuleringer.

Heuristiske algoritmer bygger på 'sund fornuft', og fungerer ved at træffe 'fornuftige' valg undervejs. Algoritmerne skal selvfølgelig på en klar og præcis måde 'have at vide', hvordan problemstrukturen ser ud, hvilke begrænsninger der skal overholdes, hvordan fornuftige valg træffes osv. Og da algoritmen bliver implementeret på en computer, skal dette selvfølgelig være på et for computeren forståeligt sprog. Dette sprog er ikke specielt velegnet, hvis problemet skal forklares til mennesker – det er matematikkens til gengæld.

Et bestemt VRP er det samme, ligegyldigt hvilket sprog det er formuleret på, så længe det er gjort korrekt. At vi vælger matematikkens skyldes, at vi er mennesker; ja sågar matematikere.

7.2 Modellens indflydelse på løsningskvaliteten

Forskellige aspekter har indflydelse på løsningskvaliteten. Disse vil vi diskutere her.

Tidsvinduer er en af de mest almindelige begrænsninger i ruteplanlægningsproblemer, men hvor stor betydning har de for kvaliteten af løsningen, og hvor vigtige er de for en virksomhed som FDB?

Vi vil her undersøge tidsvinduers indflydelse på rutelængden ud fra resultaterne opnået af forskellige algoritmer på en række tests. Som vist i afsnit 6.1.1 kan de bedste TS-algoritmer give resultater, som i gennemsnit afviger med 1,5% fra

problem	andel med t.v.	længde af t.v	bedste løsning	% over bedste
sol R101	100%	4,2%	1650,80	90,6
sol R102	75%	4,2%	1486,12	71,6
sol R103	50%	4,2%	1213,62	40,2
sol R104	25%	4,2%	982,01	13,4
sol R105	100%	12,5%	1377,11	59,0
sol R106	75%	12,5%	1252,03	44,6
sol R107	50%	12,5%	1113,69	28,6
sol R108	25%	12,5%	964,38	11,4
sol R109	100%	25,0%	1194,60	38,0
sol R110	100%	37,5%	1125,04	29,9
sol R111	100%	39,6%	1104,80	27,6
sol R112	100%	50,0%	953,63	10,1
CMT 8	0%	—	865,94	0,00

Figur 7.1 Tidsvinduers indflydelse på løsningskvaliteten.

optimum. Vi tillader os derfor at benytte de bedste resultater, TS-algoritmer har opnået, som en god approksimation for de optimale løsninger til et givet problemtilfælde. Grunden til at vi ikke bare benytter de optimale løsninger er, at vi gerne vil have et problemtilfælde fra [Christofides et al., 1979] med i undersøgelsen, og at vi ikke har kunnet finde den optimale løsning for dette.

Undersøgelsen af tidsvinduers betydning er mulig, da en række testproblemer kun afviger fra hinanden i kraft af forskellige tidsvinduer. I Solomons testsæt er de 56 forskellige problemer inddelt i tre serier, hvor byernes placering og ordrestørrelser er ens indenfor de enkelte serier, og kun tidsvinduerne varierer.

Tidsvinduerne i Solomons C-100 serie er i høj grad tilpasset ruterne. Da vi vil undersøge, hvordan tidsvinduer påvirker ruterne, vælger vi at fokusere på Solomons R100-serie.

På figur 7.1 er opstillet data for de relevante problemtilfælde. Første kolonne er navnet på problemtilfældet, anden kolonne er antallet af butikker med tidsvinduer, tredje kolonne er den gennemsnitlige bredde af tidsvinduer i forhold til depotets åbningstid, fjerde kolonne er længden af den bedst fundne rute for problemtilfældet og femte kolonne er afvigelsen fra den bedst fundne løsning.

Som det ses af skemaet, kan tidsvinduer have meget stor indflydelse på løsningskvaliteten, men der er selvfølgelig ingen garanti for, at disse tal er repræsentative. Det ses, at der er en klar sammenhæng mellem løsningskvalitet og hvor stramme tidsvinduerne er. En større andel af kunder med tidsvinduer eller mindre tidsvinduer giver en dårligere løsning.

I ovenstående tilfælde forringer tidsvinduer løsningerne med op til 90%. Selv når tidsvinduerne udgør halvdelen af depotets åbningstid, bliver løsningen 10% dårligere. Som vi konkluderede i afsnit 6.1.2, må en god TS-algoritmer forventes i gennemsnit at afvige mindre end 10% fra det optimale, selv om den kun kører

i kort tid. Det er på denne baggrund tydeligt, at tidsvinduer har mindst lige så stor indflydelse på løsningskvaliteten, som hvorvidt der bruges den ene eller den anden TS-algoritme.

Som det ses af figur 7.1, vil det kunne give en betydelig kortere rute, hvis begrænsninger i form af tidsvinduer relakseres, dvs. at gøre tidsvinduerne bredere og evt. færre. Da tidsvinduer ofte er en form for serviceparameter, er der her en klar konflikt mellem service og transportomkostning.

En anden servicerelateret diskussion er den, som drejer sig om muligheden for at tillade slutløsninger, som overskrider begrænsningerne (jf. afsnit 6.2). Her er det værd at overveje overskridelse af tidsvinduer som en mulighed. Det kan godt være, at det vil være ubelejligt for butikken, men en overskridelse vil som regel give løsninger med lavere omkostninger. En anden mulig overskridelse er at tillade at ruterne bliver lidt for lange. Dette vil selvfølgelig resultere i overarbejde for chaufførerne, som koster ekstra, men vil måske kunne give besparelser alligevel.

Der er andre begrænsninger, som på ingen måde kan overskrides. Her tænkes på begrænsninger som kapacitet af biler eller butikker som visse biler ikke kan komme til. Ej heller kan begrænsninger, som sikrer at en bil kan køre flere ruter på en enkelt dag, eller de som drejer sig om portbegrænsninger ved depotet, overskrides.

Hvor meget løsningerne kan forbedres ved at tillade overskridelser af nogle af begrænsningerne, kan kun afgøres ved praktiske forsøg med algoritmer, hvilket vi ikke har haft mulighed for.

Styring af tidsvinduer og overskridelse af begrænsninger drejer sig om service. For virksomheder, der leverer direkte til kunder, har service en høj prioritet. Kunderne i FDBs og lignendes tilfælde er virksomhedens egne butikker. Spørgsmålet er nu, i hvor høj grad en virksomhed skal yde service i forhold til sig selv.

Chaufører er f.eks. selv i stand til at læsse og losse paller ind og ud af deres lastvogn. Har de en nøgle til de butikker som besøges, er det sikkert ikke nødvendigt, at butikkens medarbejdere er til stede. Varerne skal selvfølgelig på et eller andet tidspunkt ud på hylderne i selve butikken, men selve leveringen kan i princippet foregå når som helst.

7.3 Opsummering af konklusioner

Her vil vi sammenfatte og diskutere de konklusioner, som er draget gennem rapporten.

I modelleringen af FDB-lignende situationer har vi tilstræbt at inkludere så mange relevante aspekter som muligt, men har desværre ikke kunne få alt med – split delivery og inventory routing har vi måttet udelade. Modellens form er,

som beskrevet tidligere i diskussionen valgt for at give en præcis model og ikke af hensyn til løsningsmetode.

Vi mener, at objektfunktionen bør være så avanceret som muligt. Jo bedre omkostningsfunktionen beskriver de virkelige udgifter, jo bedre beskriver modellen virkeligheden. Gode løsninger til modellen vil derfor også have stor sandsynlighed for at give gode løsninger i virkeligheden. Da en avanceret omkostningsfunktion ikke vil have nogen direkte indflydelse på algoritmen, ser vi det som oplagt at benytte en sådan.

Vi konkluderer, at tabusøgning er den mest velegnede metode til løsning af FDB-lignende problemer. Eksakte algoritmer er for langsomme, og metaheuristikkerne finder indenfor en overskuelig tid bedre løsninger end de klassiske heuristikker. Blandt metaheuristikkerne er tabu-søgning den metode, som finder de bedste løsninger.

Et emne som er værd at diskutere, når det gælder valg af løsningsmetode, er i hvor høj grad metoden tillader fleksibilitet, altså i hvor høj grad den er i stand til at håndtere forskellige udvidelser og begrænsninger. Vi mener, at det er muligt at udvide de TS-algoritmer, som er præsenteret i litteraturen på en sådan måde, at de tager højde for de udvidelser/ekstra begrænsninger, som vi har medtaget i modelleringen (jf. afsnit 6.2). Ydermere vil det sandsynligvis være sværere at udvikle eksakte løsningsmetoder, som på samme simple måde kan udvides, så de kan medtage disse ekstra begrænsninger [Reeves, 1993].

Hvis TS-algoritmer kan håndtere mere avancerede modeller end eksakte algoritmer, vil de have en stor fordel i forhold til disse, da modellen de løser er mere virkelighedtro. Lidt forenklet kan man sige, at en ikke-optimal løsning til en optimal model er bedre end en optimal løsning til en ikke-optimal model.

Ligemeget hvor avanceret en model er, vil den aldrig kunne beskrive virkeligheden fuldstændigt. Der vil altid være uforudsete hændelser eller specielle omstændigheder, som der til tider skal tages hensyn til. FDB har før implementeringen af et ruteplanlægningssystem lagt ruteplaner i 'hånden'. Et menneske vil i højere grad end en computer være i stand til at håndtere mærkelige uforudsigeligheder, men vil til gengæld have svært ved at gennemskue, hvor god en given rute er. Jo større planlægningsopgaven er og jo flere hensyn, der skal tages, dvs. jo mere avanceret modellen er, des sværere vil et menneske have ved at overskue situationen. Da situationer som FDBs er store og indeholder udvidelser og ekstra begrænsninger, er det vores klare opfattelse, at en god algoritme implementeret på en computer vil være et menneske overlegen i langt de fleste tilfælde.

I løbet af dette projekt er der ofte blevet argumenteret ud fra empiriske tests, hvor det oftest er Solomons testsæt, som er blevet benyttet. Dette har vi gjort på trods af, at vi i afsnit 4.4.1 har beskrevet, hvorledes Solomons testsæt er for simpelt i forhold til FDB-lignende situationer. Strengt taget kan vi derfor ikke være sikre på, at de konklusioner som vi drager også holder i FDB-lignende situationer.

Den primære grund til, at vi alligevel gør som vi gør er, at der ikke er andre muligheder. Hvis vi skulle være helt sikre på, at en given løsningsmetode ville kunne løse et givet problem så og så godt, skulle vi opstille problemet og implementere en algoritme til at løse det. Dette ville for det første ikke være særlig generelt og for det andet være meget besværligt. Desuden er projektets mål, at analysere en situation inden den forsøges løst.

Vi tror, at de konklusioner vi drager alligevel er troværdige, da Solomontestene trods alt har mange ligheder med virkelighedstro tilfælde. Det virker oplagt, at selvom testtilfældene er mere avancerede, vil TS-algoritmer stadig give bedre resultater end klassiske heuristikker. Derudover vil eksakte algoritmer nok ikke ligefrem blive meget hurtigere af at få mere komplicerede opgaver.

8 Konklusion

Opstilling
Vi har valgt at beskæftige os med den matematiske model for et virkeligt VRP. Som udgangspunkt vælger vi en standardformulering for VRPTW og foreslår yderligere udvidelser og begrænsninger. I og med at der i formuleringen for VRPTW allerede tages højde for kapacitet, tidsvinduer og servicetid, har vi selvfølgelig ikke tilføjet disse. Vi har derimod foreslået udvidelser i formuleringen som vedrører køre- og hviletidsbestemmelser, flere ruter pr. vogn pr. ruteplanlægning samt backhaul. Derudover har vi inkorporeret begrænsninger knyttet til adgangsmuligheder ved kunder, kapacitet af depotet samt trafikforhold.

Løsning
Vi har til løsning af vores virkelige VRP konkluderet, at tabu-søgning er den mest egnede strategi. Denne konklusion bygger på overvejelser angående løsningskvalitet kontra beregningstid. Eksakte algoritmer er selvfølgelig garanteret højere løsningskvalitet end heuristikker, men deres manglende evne til generelt at løse problemtilfælde med 100 kunder eller derover og deres lange beregningstid taget i betragtning, må disse betegnes som værende uegnede til løsning af vores model. Ydermere har vi valgt en meta-heuristik algoritme, da de er de heuristiske overlegne hvad angår løsningskvalitet. I og med at adaptiv-hukommelse er en meget lovende strategi, foreslår vi, at en sådan sættes til at styre tabu-søgningen.

9 Perspektivering

Implementering
Skulle vi lave en virkelig og grundig undersøgelse af vores forslag til løsning af et virkeligt (FDB-lignende) ruteplanlægningsproblem, må vi nok se i øjnene, at en implementering er essentiel. Dog ville noget sådant have et meget datalogisk tilsnit, og selvfølgelig kræve gode programmeringsfærdigheder. Testkørsler på konkrete problemtilfælde, samt sammenligninger af resultater fra testkørsler kørt med forskellige kombinationer af begrænsningerne, ville være grundlaget, for at vurdere ydelsen af implementeringen og dets parametre.

Under forløbet med udfærdigelsen af rapporten har vi haft kontakt til Anette Vainer, som er ansat i logistikafdelingen hos Carl Bro as. Hun fortalte blandt andet, at et af de vigtigste aspekter i at udfærdige et færdigt program til en kunde, netop var implementeringen. I praksis ved udvikling af et ruteplanlægningsprogram lægges vægten ikke så meget på, f.eks. valg af konkret TS-algoritme, men nærmere på den datalogiske implementering. Med dette menes at, da forskellige TS-algoritmer mht. løsningskvalitet præsterer næsten ens (og næsten optimalt), er der meget mere at hente ved at bruge smart implementerede hukommelsesbesparende datastrukturer, for derved at øge iterationshastigheden og derved tillade flere iterationer indenfor samme tidsrum.

Standard vs. design
Det store spørgsmål for en virksomhed, som overvejer at investere i et ruteplanlægningsværktøj, er hvilket de skal vælge. Skal de vælge en af de kommercielle standardprogrampakker så som *Roadshow*, som f.eks. Aage Hempel A/S udbyder, eller en af de, til firmaets behov, specielt designede programpakker, som f.eks. Carl Bro as tilbyder.

Det er klart, at hvis et firma henvendte sig et konsulentfrima med et ruteplanlægningsproblem, ville de ikke få en objektiv vurdering af dette. Konsulentfirmaet har jo også en forretning at pleje[1]. Så for firmaer som FDB er det at vælge et program ingen simpel sag. De bliver nødt til at sætte sig grundigt ind i deres egen problemstilling og hvilke mulige løsningsmetoder, der forefindes til denne. For store firmaer ville det eventuelt være værd at overveje, at ansætte en operationsanalytiker. Således ville man have en kompetence på området, som arbejdede for firmaets bedste, og dermed måske undgå at

[1] Det som firmaer i FDBs situation i virkeligheden har brug for, er endnu et konsulentfirma, helst et uvildigt, til at vejlede sig i hvilken løsning, altså hvilken anden ruteplanlægningsvirksomhed, de skal vælge.

købe 'katten i sækken'. Man kunne nemlig sagtens forestille at en potentiel programaftager uden indgående kendskab til VRP, ville lade sig imponere af smarte brugerflader. Hurtighed af programmet ville også kunne imponere, hvilket selvfølgelig er reelt nok, hvis da ikke bare den høje hurtighed dækker over at programpakken kun består af eksempelvis en primitiv savingsalgoritme. Vi har haft mulighed for at se et par testkørsler af Roadshow for en tilfældig dag på FDBs tørvaredepot. Beregningstiden var forbløffende kort (ca. 4 sek på en PIII-pc) og dette kunne tyde på at løsningen ikke var så nær-optimal som man kunne håbe. Dette kunne pege i retning af at, enten er algoritmen (for) simpel (måske en sweep-algoritme), eller også kører den med for få iterationer[2].

FDBs valg

Hos FDBs tørvarelager var bestemmelsen om at købe den kommercielle programpakke Roadshow fra Aage Hempel A/S taget på højere niveau i koncernen, så vi ved ikke hvilke overvejelser der ligger til grund for valget af netop denne. På Aage Hempel A/S hjemmeside[3] findes en liste over de virksomheder i Skandinavien, som har købt Roadshow. Her står bl.a. navne som Coca Cola Tapperierne A/S, Carlsberg A/S, Mejericentralen, Danske Bank (værditransport) m.fl. Dette kunne jo godt tænkes at have været et vægtigt argument i FDBs overvejelser.

Desværre har vi heller ikke fået lov til selv at afprøve Roadshow, så vi har kun et overfladisk kendskab til hvordan det egentligt fungerer. Ej heller har det været muligt at finde ud af hvilke algoritmer der gemmer sig i programmet, og om programmet overhovedet benytter sig af metaheuristikker. Roadshow ejes af det canadiske firma Descartes System Group og Aage Hempel A/S er skandinavisk forhandler af pakken. Ifølge Anette Vainer er Aage Hempel A/S selv dårligt nok klar over, hvad Roadshow indeholder, og det er efter sigende ikke ualmindeligt for forhandlerne af sådanne programpakker. Da FDB fik Roadshow foregik det ved at deres computer blev afhentet, og nogle uger senere leveret tilbage med programmet installeret. Altså virker det som om at praksisen og ikke mindst politikken omkring disse standardpakker er meget hemmelig, hvilket selvfølgelig er forståeligt, men set fra vores synspunkt lidt ærgeligt.

Eftersom prisen på de kommercielle pakker er adskillige hundrede tusinde kroner, har FDB forhåbentligt gjort sig nogle overvejelser inden de købte Roadshow.

FDBs alternativ

Et af FDBs alternativer til Roadshow ville have været at få udviklet et specialdesignet program. For et firma som Carl Bro as ville det, ifølge Anette Vainer, godt have kunnet betalt sig at udvikle et program for FDB, i og med at FDBs problem er så forholdsvist simpelt og derfor kunne fungere som fundament for Carl Bro as til fremtidige aftagere. Dette taget i betragtning ville Carl Bro as måske have været villige til give et afslag i prisen, selvfølgelig under forudsætning af, at FDB havde været de første i rækken. Ydermere

[2]Om det tager fire sekunder eller ét minut skulle man mene var underordnet for FDB.
[3]http://www.aagehempel.dk/logistik/logistik.htm

ville FDB antageligt have fået et bedre ruteplanlægningsprogram, i og med at dette jo ville være skræddersyet til netop dem. En ofte forekommende ulempe ved designpakker er at de i mangel på alsighed kan være besværlige at modificere. Eksempelvis kunne man forestille sig at det ikke var så ligetil at tilføje en ny kunde i ruteplanlægningen, og at man derfor skulle have fat i en højt betalt konsulent fra programlaverandøren til at foretage denne ændring. Standardprogrammer er nemmere at foretage sådanne ændringer i for lægfolk.

Kvalitet af standard- og designpakker

En af farerne ved at købe standardpakker er, sådan som vi ser det, at ikke alle parametre i programmet er specifikt fastsat til det pågældende problem. Dette problem kan dog delvist omgås hvis standardprogrammet har selvtunende parametre som f.eks. adaptiv hukommelse i afsnit 5.3.3. Os bekendt er det dog ikke alle parametre der kan implementeres som selvtunende, og alt andet lige vil til problemet kalibrerede parameterværdier, give bedre løsninger end algoritmer fra en standardpakke.

Litteratur

E. Balas and P. Toth. Branch and bound methods. In *The Traveling Salesman Problem*, Wiley-interscience Series in Discrete mathematics, chapter 10, pages 361–402. John Wiley & sons, New York, 1985.

N. L. Biggs. *Discrete Mathmatics*. Oxford Science Publications, 1989.

M. Blomhøj, K. Frisdahl, and F. M. Olsen. *Lineær Programmering*. FAG, 1984.

W.-C. Chiang and R. A. Russel. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4), Fall 1997.

N. Christofides. Vehicle routing. In *The Traveling Salesman Problem*. John Wiley & Sons, New York, 1985a.

N. Christofides. Vehicle routing. In *The Traveling Salesman Problem*, Wiley-interscience Series in Discrete mathematics, chapter 12, pages 431–448. John Wiley & sons, New York, 1985b.

N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, chapter 11, pages 315–338. John Wiley & Sons, 1979.

J.-F. Cordeau, G. Desaulnies, J. Desrosiers, M. M. Solomon, and F. Soumis. The vrp with time windows. *»not published«*, June 2000a.

J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *»not published« found on: http://www.crt.umontreal.ca/ cordeau/*, December 2000b.

T. G. Crainic and G. Laporte. *Fleet Management and Logistics*. Kluwer Academic Publishers, 1998.

A. Dolan and J. Aldous. *Networks and Algorithms*. John Wiley & Sons Ltd., 1995.

L. R. Foulds. *Combinatorial Optimization for Undergraduates*. Springer-Verlag, 1984.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

R. W. Hall and J. G. Partyka. On the road to efficiency. *ORMS today*, 24(3), June 1997.

R. W. Hall and J. G. Partyka. On the road to service. *ORMS today*, August 2000.

K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *Datalogiske Skrifter*, (nr. 86), 1999. Roskilde University, Department of Computer Science.

K. Hermann and M. Niss. *Beskæftigelsesmodellen i SMEC III.* Nyt Nordisk Forlag, København, 1982.

A. Hertz, E. Taillard, and D. de Werra. A tutorial on tabu search. *»Unpublished«*, 1994. Université de Montréal, Centre de Recherche sur les Transport, Montréal, Canada H3C 3J7 (Taillard) EPEL, Départment de Mathématiques, MA-Ecublens, CH-1015 Lausanne (Hertz & Werra).

D. S. Johnson and C. H. Papadimitriou. Computational complexity. In *The Traveling Salesman Problem*, Wiley-interscience Series in Discrete mathematics, chapter 3, pages 37–86. John Wiley & sons, New York, 1985.

G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, (7):285–300, 2000.

E. Lawler, J. Lenstra, A. R. Kan, and D. Shimoys. *The Traveling Salesman Problem.* Wiley-interscience Series in Discrete mathematics. John Wiley & Sons, New York, 1985.

O. B. G. Madsen. *Large-Scale Optimization and Vehicle Routing.* IMM, Department of Mathematical Modelling, The Technical University of Denmark, 1997.

E. Minieka. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, Inc., New York, 1978.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc., 1988.

N. B. Petersen and J. Surland. *Ruteplanlægning i oliebranchen.* Roskilde Universitetscenter, 1999.

V. Rayward-Smith, I. Osman, C. Reeves, and G. Smidth. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, New York, 1996.

C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, New York, 1993.

G. Reinelt. *The Traveling Salesman.* Lectures in Computer Notes. Springer-Verlag, Heildelberg, 1994.

Y. Rochat and F. Semet. A tabu search approach for delivering pet food and flour in switzerland. *Journal of the Operational Research Society*, (45): 1233–1246, 1994.

Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, (1):147–167, May 1995.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, (35):254–265, 1987.

E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, (31):170–186, 1997.

A Kode til TS-algoritme

```
%TABUSØGNING TIL VRP
%dette er et implementation af en simpel TS-algoritme
%programmet kan kun bruges til CVRP, idet tidsaspektet er udeladt

'----------------------*** INITIALISER ***------------------------------'

clear;                  %hukommelsen i matlab nulstilles
'Iterationer'           %antallet af iterationer TS-algoritmen skal køre fastsættes
N=1000
'Antal kunder'          %antallet af byer(kunde) sættes
n=100
'Kapasitet af biler' %den maksimale lastevne sættes (ens vogne)
loadkap=80

                              %tabu initialiseres:
bytabu=zeros(1,n+1);          %1xn matrix indeholdende tabu ved div. byer
frekvenstabu=zeros(1,n+1); %1xn matrix indeholdende antallet af straffe ved div. byer
V=[0;1];                      %blot en vektor
frekvenstabumodifier=1/40; %styrken af frekvenstabustraffen
tabutid=10;                   %tabutiden sættes

'Byernes ordre';
ORDRE=[0  0;            %første kolonne er bynummeret
       1 20             %anden kolonne er ordrestørrelsen
       2 10             %der kan sættes flere byer ind hvis 'n' ændres tilsvarende
       3 10
       4 20
       5 20
       6 20
       7 30
       8 10
       9 30
      10 10]

'Positionsmatricen';
P=[0   0.0   0.0        %første kolonne er bynummeret
   1 -21.5  -4.0        %anden kolonne er x-kordinatet
   2 -11.6  10.9        %tredje kolonne er y-kordinatet
   3  -5.2 -26.6
   4  -2.0  -7.6
   5   5.5  15.1
   6  17.0 -28.2
   7  20.2  17.3
   8  20.7  -5.5
   9  31.1   8.3
  10  33.5 -12.0];
```

```
                    %Initialiser omkostningsmatricen
C=zeros(n+1,n+1);
for i=1:n+1;        %værdierne over diagonalen i omkostningsmatiricen udregnes
   for j=i:n+1;
      C(i,j)=sqrt( (P(i,2)-P(j,2))^2 + (P(i,3)-P(j,3))^2 );
   end;
end;

'Omkostningsmatricen:';
C=C+C';             %nedre halvdel dannes som den øvre transponeret og lægges til den øvre

straffaktor=frekvenstabumodifier*sum(sum(C))/((n+1)^2-(n+1)); %frekvensstraf udregnes som
                                                              %den gennemsnitlige rutes omkostning
                                                              %ganget med 'frekvenstabumodifier'
                                                              %defineret tidligere

'----------------------*** Konstruktionsalgoritme (SWEEP) ***----------------------'

X0=zeros(n+1,n+1);  %beslutningsmatrix initialiseres

VINKEL=zeros(1,n);  %1xn matrix til at angive byernes vinkler i forhold til vandret

for a=1:n;
   VINKEL(1,a)=atan2( P(a+1,3)-P(1,3) , P(a+1,2)-P(1,2)); %vinklerne udregnes og indsættes
end;

by=0;                   %kontrolparameter - styrer indsættelsen af byer
load=0;                 %kontrolparameter - styrer kapaciteten af vognene

while 1;                    %kører så længe der er byer tilbage
   minimum=min(VINKEL);         %den mindste vinkel findes
   for b=1:n;                   %indsættelse af by i rute

      if minimum==VINKEL(1,b);      %hvis byen har den mindste vinkel forsøges den indsat
         load=load+ORDRE(b+1,2);    %bilens last udregnes

         if load>loadkap;           %hvis der er overskridelse af kapaciteten
            X0(by+1,1)=1;               %ruten føres tilbage til depotet
            X0(1,b+1)=1;                %en ny rute føres frem til den udvalgte by
            VINKEL(1,b)=9999;           %byens vinkel sættes stor så den ikke vælges
            load=ORDRE(b+1,2);
            by=b;
         else;                      %ellers
            X0(by+1,b+1)=1;             %ruten føres til den udvalgte by
            VINKEL(1,b)=9999;           %byens vinkel sættes stor så den ikke vælges
            by=b;
         end;
```

```
      end;
   end;
   if min(VINKEL)>10;  %hvis der ikke er flere byer tilbage afbrydes while-løkken
      X0(by+1,1)=1;
      break;
   end;
end;
                       %nu er startruten 'X0' dannet

'Startrute:';          %startruten plottes
Vx=[0;1;0];
Px=P*Vx;               %x-koordinater
Vy=[0;0;1];
Py=P*Vy;               %y-koordinater

hold off;
plot((Px),(Py),'o')    %byerne plottes som o'er på en figur
hold on;

for i=2:(n+1);         %hvis en vogn kører fra 'i' til 'j' tegnes en streg mellem dem
   for j=2:(n+1);
      if X0(i,j)==1;
         line([P(i,2),P(j,2)],[P(i,3),P(j,3)])
      end;
   end;
end;


'Start-omkostning'
cost=sum(sum(X0.*C))  %startomkostningen udregnes
BESTSIT=X0;           %bedste situation og bedste omkostning og bedste ittereation sættes
bestcost=cost;
bestiteration=0;

iteration=0;          %iterationstæller sættes


X1=X0;                %løsningen som ændres fra it. til it. X1 sættes

'---------------------*** Nu starter algoritmen for alvor ***---------------------'


%---------------------ITERATION FORBEREDES----------------------------------------

for ITERATION=1:N;    %algoritmen skal jo køre i nogle iterationer
   iteration=iteration+1;
```

```
X1(1,1)=1;          %der tilføjes en vogn, som kører fra depotet og til depotet
                    %denne rute kan udvides så flere vogne kan tilføjes løsningen

V=zeros(1+n,1);     %antallet af biler tælles
V(1,1)=1;
m=sum(X1*V);

                    %resultatmatrix med de forskellige situationers omkostninger, atributter og tabu initialiseres
RESULTAT=ones(n,5);
for a=1:n;
   RESULTAT(a,1)=9999;
   RESULTAT(a,2)=9999;
end;
%første kolonne er resultatet
%anden kolonne er tabu-straffen
%tredje kolonne er byen som blev flyttet
%fjerde og femte kolonne er hvilken rute, som blev splittet op

resultattaeller=1;   %tæller så resultaterne kan placeres rigtigt i RESULTAT


%-------------- LØSNINGEN TREVLES OP I ENKELTE RUTER OG LAST BEREGNES ---------------

TREVL=0;                %matrix med de optrevlede ruter
                        %hver række i matricen svarer til en rute
counti=1;               %kontrolparameter - styrer rækkenummeret i 'TREVL'
LOAD=0;                 %lasten i de forskellige vogne

for a=1:(n+1);
   countj=3;            %kontrolparameter - styrer kolonnenummert i 'TREVL'
   if X1(1,a)==1;       %hvis en vogn kører fra depotet og til by 'a' påbegyndes en ny rute
      r=a;
      TREVL(counti,1)=1;            %første by i ruten er depotet
      TREVL(counti,2)=r;            %anden by i ruten er 'r'
      LOAD(counti,1)=ORDRE(r,2);    %lasten er indtil videre ordren for by 'r'
      while r~=1;                   %sålænge vognen ikke vender tilbage til depotet
         V=zeros(1,(n+1));             %'V1' dannes som værende de veje, der går ud fra 'r'
         V(1,r)=1;
         V1=V*X1;
         for b=1:(n+1);             %vejene fra 'r' gennemgåes
            if V1(1,b)==1;             %hvis vognen kører ad vejen
               r=b;                       %vælges byen der køres til som de ny 'r'
               TREVL(counti,countj)=r;    %den nye by 'r' tilføres 'TREVL' matricen
               LOAD(counti,1)=LOAD(counti,1)+ORDRE(r,2); %den nye bys last lægges til
               countj=countj+1;
            end;
         end;
      end;
```

```
      counti=counti+1;           %vognen er vendt tilbage til depotet... næste rute
   end;
end;

TREVLSTR=size(TREVL);  %størrelsen på TREVL(og LOAD) udregnes
trevli=TREVLSTR(1,1);  %rækkestørrelse
trevlj=TREVLSTR(1,2);  %søjlestørrelse


for B=2:n+1;           %nu skal trækket til at tages og en by behandles ad gangen

   %----------------------- BYEN FJERNES-----------------------------------------------------

   %Det angives fra hvilken rute byen er fjernet og hvilket nummer på ruten den var
   [orgrutenummer,nummerpaaruten]=find(TREVL==B);

   %hvor meget man sparer ved at fjerne byen fra ruten udregnes
   orgafgangsby=TREVL(orgrutenummer,nummerpaaruten-1);
   organkomstsby=TREVL(orgrutenummer,nummerpaaruten+1);
   fjernspar=C(orgafgangsby,B)+C(B,organkomstsby)-C(organkomstsby,orgafgangsby);

   %byen fjernes fra løsningen og ruteoptrevlingen opdateres
   NYTREVL=TREVL;
   for j=nummerpaaruten:trevlj-1;
      NYTREVL(orgrutenummer,j)=NYTREVL(orgrutenummer,j+1); %alle byerne efter den fjernede
end;                                                       %flyttes en gang frem i rækken
   NYTREVL(orgrutenummer,trevlj)=0;                           %den sidste by i ruten er nu flyttet
                                                              %og hvor den var er der nu ingenting

   %---------- DET BEREGNES HVOR MEGET DET KOSTER AT INDSÆTTE BYEN I EN ANDEN RUTE------------

   %ordre for by B
   aktuelordre=ORDRE(B,2);

   %lasten i vognen, som besøgte byen før, sættes ned;
   LOAD1=LOAD;
   LOAD1(orgrutenummer,1)=LOAD(orgrutenummer,1)-aktuelordre;

   %last ved genindsættelse i alle ruter (incl. den hvor byen kom fra) vurderes
   NYLOAD=LOAD1+aktuelordre;

   %ruter uden overlast findes
   count=1;               %kontrolparameter
   TILLADTRUTE=0;         %matrix med tilladte ruter
   tilladteruter=0;       %antallet af tilladte ruter
   forbudteruter=0;       %antallet af forbudte ruter
   for a=1:trevli;
      if NYLOAD(a,1)<=loadkap;          %hvis lasten er acceptabel:
```

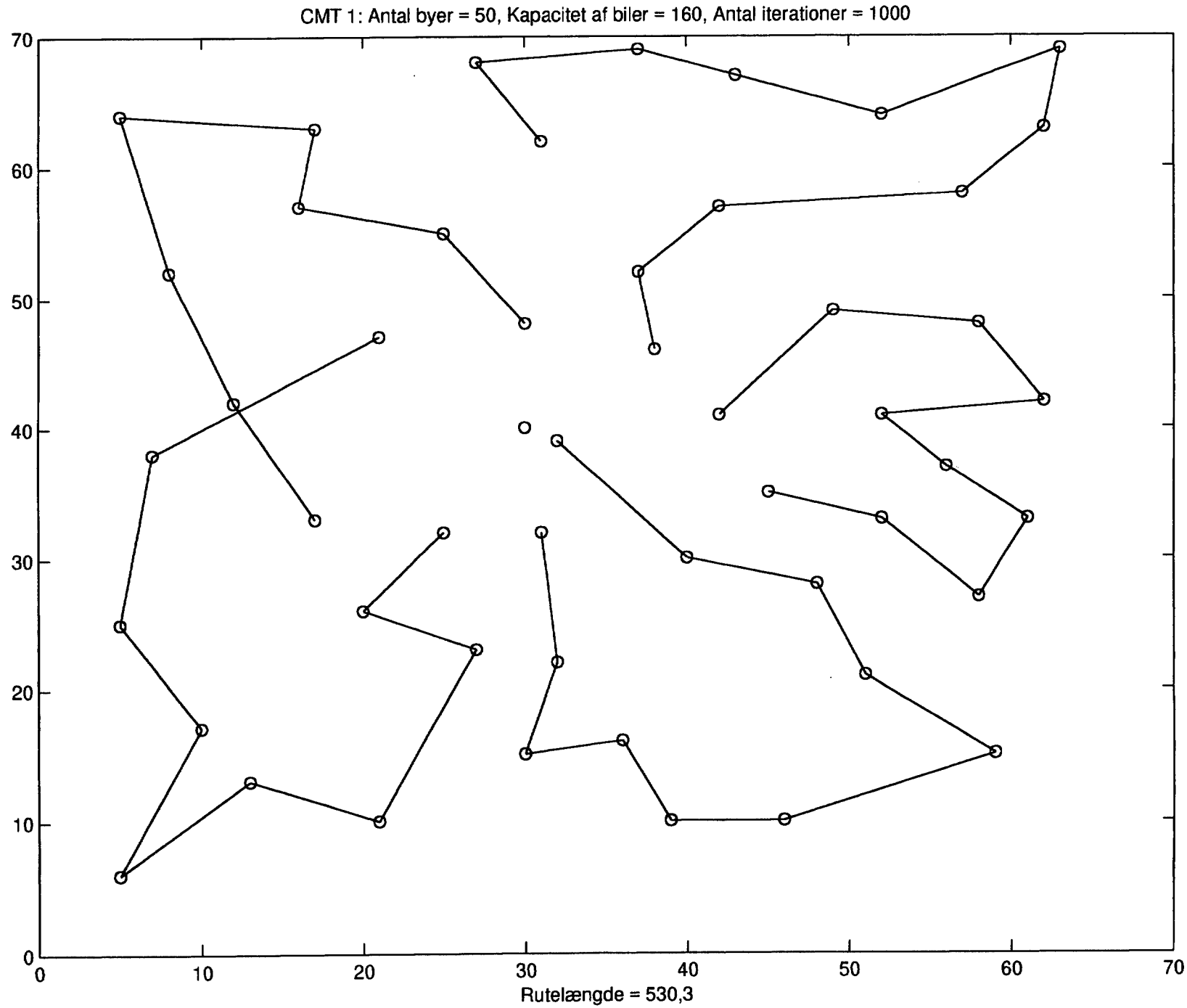
```
      TILLADTRUTE(count)=a;              %registres ruten
      tilladteruter=tilladteruter+1;     %og antallet af tilladte ruter tælles op
      count=count+1;
   else
      forbudteruter=forbudteruter+1;   %ellers tælles antallet af forbudte ruter op
   end;
end;

%antal mulige indsættelser udregnes
nuller=0;
for a=TILLADTRUTE;
   for b=1:trevlj;
      if NYTREVL(a,b)==0;    %antallet af nuller i 'NYTREVL' tælles...
         nuller=nuller+1;
      end;
   end;
end;
indsaettelser=((trevlj-1)*(tilladteruter)-nuller); %og trækkes fra antallet af elementerne i 'NYTREVL'
%den ekstra omkostning ved indsættelse af 'B' i forskellige ruter udregnes
EXTRACOST=9999*ones(4,indsaettelser); %matrix med ekstra omkostninger
                                %første kolonne er den ekstra omkostning
                                %anden kolonne er byen som er før den indsatte by på ruten
                                %tredje kolonne er byen som er efter den indsatte by på ruten
                                %fjerde kolonne er den indsatte by

count=1;                             %kontrolparameter
for i=TILLADTRUTE;                   %for hver tilladt rute
   for a=1:trevlj-1;                    %og for hver plads i ruten gøres følgende
      rutenummer=i;
      afgangsby=NYTREVL(i,a);              %afgangsby og
      ankomstsby=NYTREVL(i,a+1);           %ankomstby ved denne indsættelse registreres
      samme=(afgangsby==orgafgangsby)&(ankomstsby==organkomstsby)&(rutenummer==orgrutenummer);
      %'samme' er 1 hvis byen sættes ind samme sted som den blev fjernet 0 ellers
      if (ankomstsby~=0)&not(samme); %hvis ikke byen indsættes samme sted udregnes
         EXTRACOST(1,count)=C(afgangsby,B)+C(B,ankomstsby)-C(afgangsby,ankomstsby);
         EXTRACOST(2,count)=afgangsby;
         EXTRACOST(3,count)=ankomstsby;
         EXTRACOST(4,count)=B;
         count=count+1;
      end;
   end;
end;


%--------------- BYEN INDSÆTTES I DEN BEDSTE ANDEN RUTE OG RESULTATET NOTERES ---------------


%den mindste indsættelsesomkostning findes
```

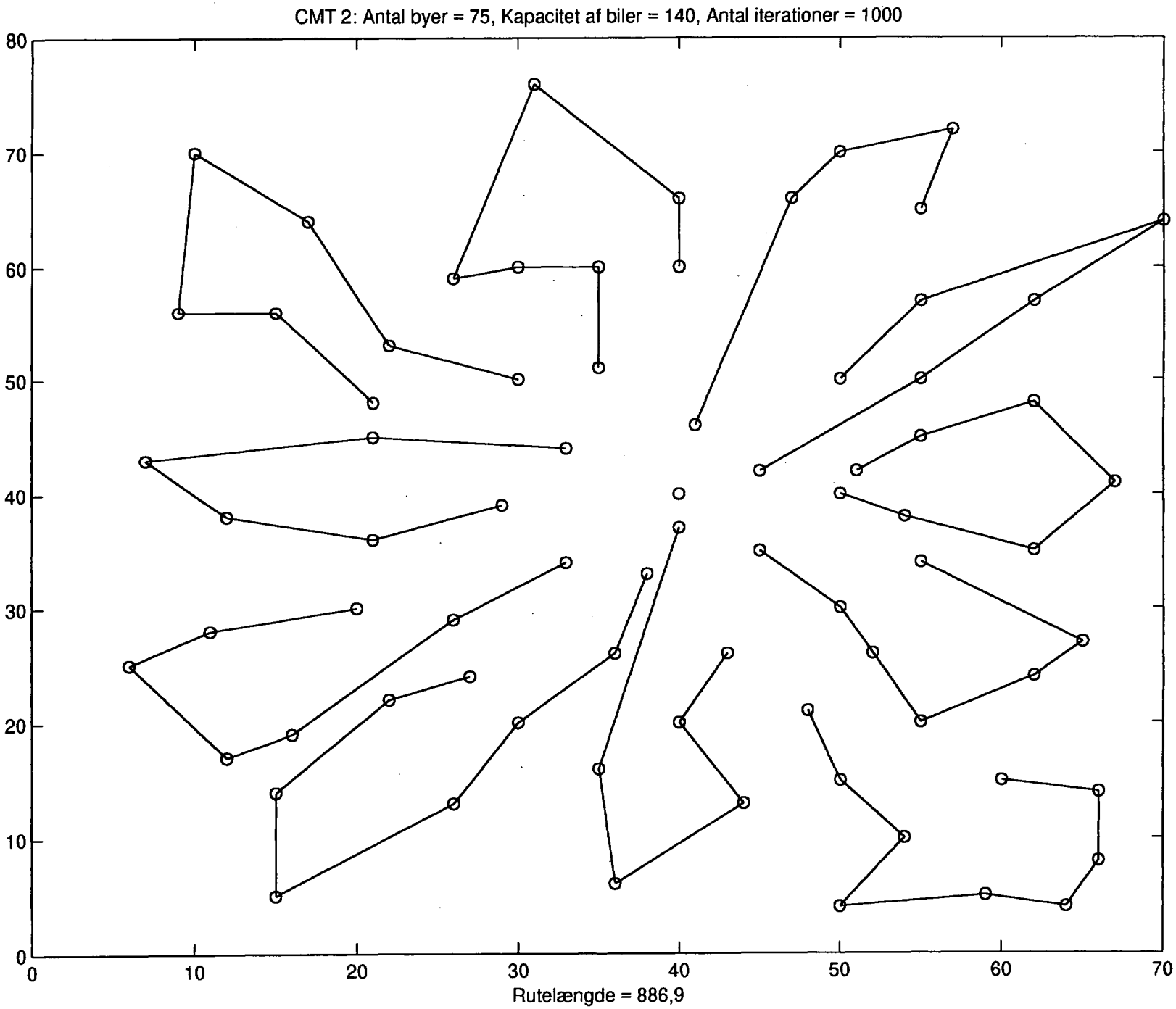
```
   V=[1 0 0 0];
   EXTRACOST2=V*EXTRACOST;            %vektor udlukkende indeholdende ekstra omkostninger dannes
   minimum=min(EXTRACOST2);           %og minimum af denne findes
   [VALG]=find(EXTRACOST2==minimum);  %de træk med denne omkostning vælges
   valgtindsaet=VALG(1,1);            %den første af dem vælges
   afgangsby1=EXTRACOST(2,valgtindsaet);   %afgangsby,
   ankomstsby1=EXTRACOST(3,valgtindsaet);  %ankomstsby og
   valgtby=EXTRACOST(4,valgtindsaet);      %den valgte by registreres

   RESULTAT(resultattaeller,1)=cost-fjernspar+minimum;              %resultatet udregnes
   RESULTAT(resultattaeller,2)=straffaktor*frekvenstabu(1,valgtby); %straffen udregnes
   RESULTAT(resultattaeller,3)=valgtby;
   RESULTAT(resultattaeller,4)=afgangsby1;
   RESULTAT(resultattaeller,5)=ankomstsby1;
   resultattaeller=resultattaeller+1;

end; %her slutter behandlingen af byerne


%--------------RESULTATER MODIFICERES PGA. TABU ------------------------


%løsninger som er tabu umuliggøres ved at straffen øges kraftigt
for e=1:n;
   if bytabu(1,RESULTAT(e,3))>0;
      RESULTAT(e,2)=9999;
   end;
end;


%----------------NÆSTE LØSNING FINDES---------------------------------


V1=zeros(5,1);
V2=zeros(5,1);
V1(1,1)=1;
V2(2,1)=1;

%den straffede VÆRDI findes
RUTEVAERDI=RESULTAT*V1+RESULTAT*V2;    %vektor ('RUTEVAERDI') med omkostninger plus straf dannes

%den laveste VÆRDI findes
lavestevaerdi=min(RUTEVAERDI);
for c=1:n;
   if lavestevaerdi==RUTEVAERDI(c,1);
      valgtrute=c;                     %trækket med lavest værdi vælges
   end;
end;
```

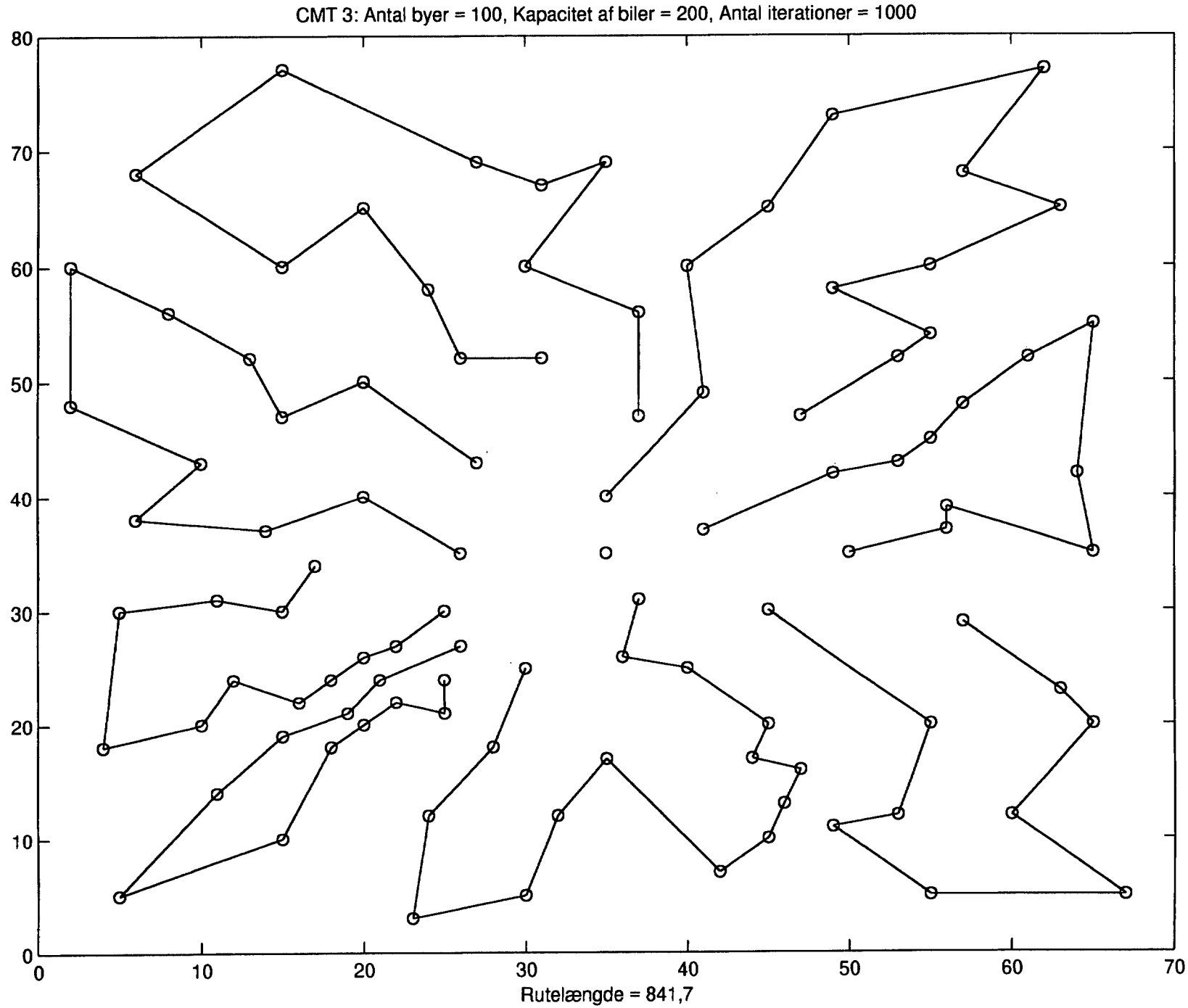
```
%de faktiske omkostningerne ved ruterne findes - det kan jo være, at der er en ny vinder!
RUTECOST=RESULTAT*V1;          %vektor ('RUTECOST') med de faktiske omkostninger

%den laveste omkostning findes og det kontrolleres hvorvidt det er den hidtil bedste
lavestecost=min(RUTECOST);
nybedst=0;
if (lavestecost+0.01)<bestcost;      %det undersøges hvorvidt en ny bedste løsning er opnået
                                     %de 0.01 lægges til for at sikre at den nye rute faktisk
                                     %er mindre og ikke blot vælges pga. afrundingsfejl
   nybedst=1;                        %det registreres at der er opnået en ny bedste løsning
   for d=1:n;
      if lavestecost==RUTECOST(d,1);
         valgtrute=d;                %trækket med en ny laveste omkostening vælges
      end;
   end;
end;

%----------------- RUTEN GENDANNES ----------------------

for i=1:(n+1);
   if X1(i,RESULTAT(valgtrute,3))==1;
      I=i;
      X1(i,RESULTAT(valgtrute,3))=0;   %den gamle vej til ruten fjernes
   end;
end;
for j=1:(n+1);
   if X1(RESULTAT(valgtrute,3),j)==1;
      J=j;
      X1(RESULTAT(valgtrute,3),j)=0;   %den gamle vej fra ruten fjernes
   end;
end;
X1(I,J)=1;                             %den gamle rute lappes


X1(RESULTAT(valgtrute,4),RESULTAT(valgtrute,5))=0; %ruten hvori byen indsættes klippes op
X1(RESULTAT(valgtrute,4),RESULTAT(valgtrute,3))=1; %den nye vej til ruten dannes
X1(RESULTAT(valgtrute,3),RESULTAT(valgtrute,5))=1; %den nye vej fra ruten dannes
cost=RESULTAT(valgtrute,1);                        %omkostningen registreres



if nybedst; %hvis der findes en ny bedste løsning
   '------------------***Ny bedste løsning***------------------'
   ITERATION
   BESTSIT=X1;
   by=RESULTAT(valgtrute,3)
   bestcost=cost
```

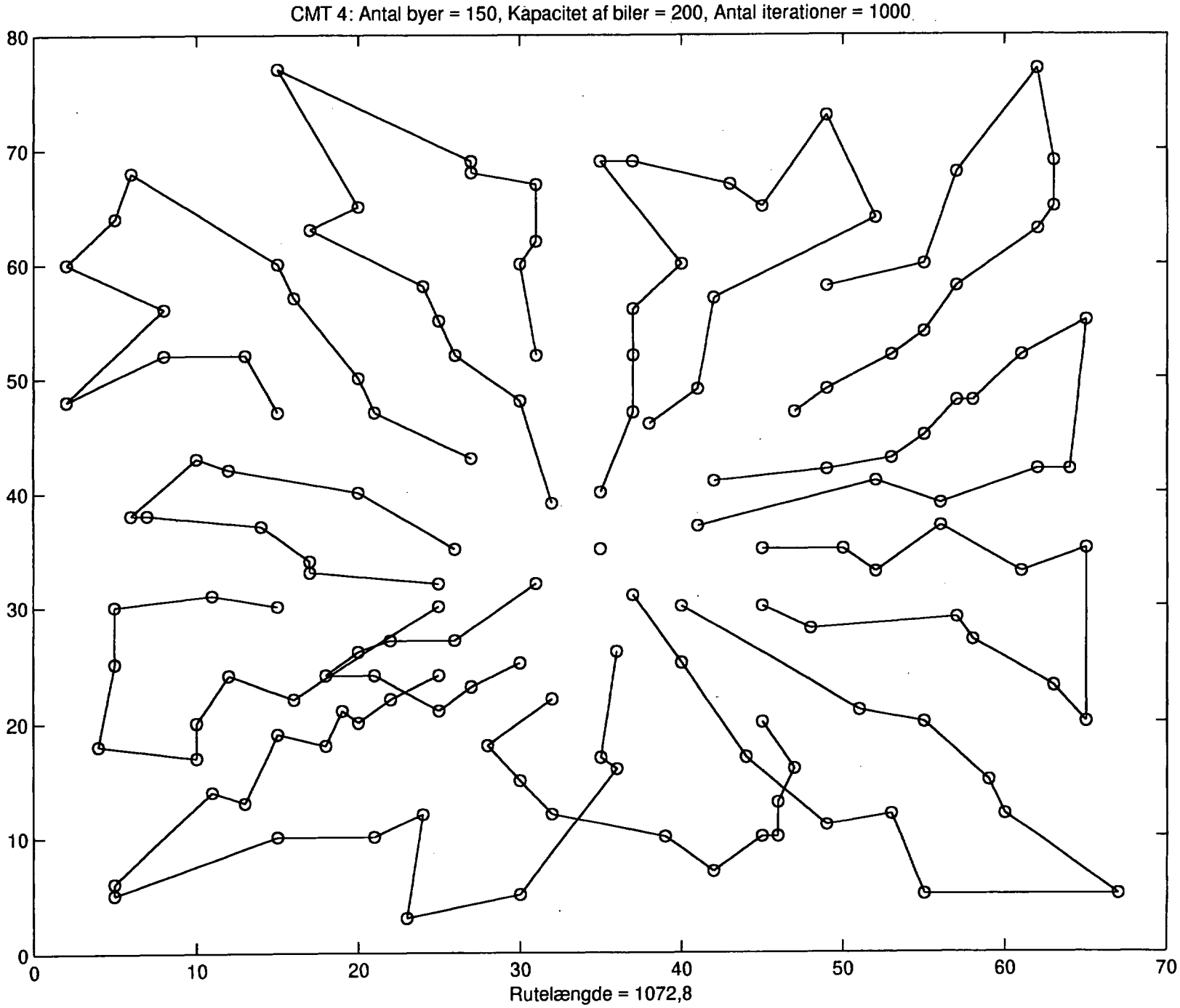
```
      bestiteration=ITERATION;
      frekvenstabu=zeros(1,n+1);                %frekvenstabu nulstilles
      frekvenstabu(1,RESULTAT(valgtrute,3))=1;  %frekvenstabu tælles op for den valgte by

   else       %hvis der ikke findes en ny bedste
      '-------------------*** Ny løsning ***-------------------'
      ITERATION
      by=RESULTAT(valgtrute,3)
      cost
      bestcost
      %frekvenstabu tælles op for den valgte by
      frekvenstabu(1,RESULTAT(valgtrute,3))=frekvenstabu(1,RESULTAT(valgtrute,3))+1;

   end

   bytabu(1,RESULTAT(valgtrute,3))=tabutid+1; %bytabu sættes

   for L=1:(n+1); %bytabu tælles en ned (for alle)
      bytabu(1,L)=max(bytabu(1,L)-1,0);
   end;
   bytabu;
   frekvenstabu;

end; %her slutter iterationerne


'*** DEN BEDSTE LØSNING FUNDET ***'
BESTSIT;
bestcost
bestiteration
bytabu;
frekvenstabu;

%-------------RUTEN PLOTTES-----------------

hold off;
plot(Px,Py,'o')

hold on;
for i=2:n+1;
   for j=2:n+1;
      if BESTSIT(i,j)==1;
         line(  [P(i,2),P(j,2)] , [P(i,3),P(j,3)] )
      end;
   end;
end;
```

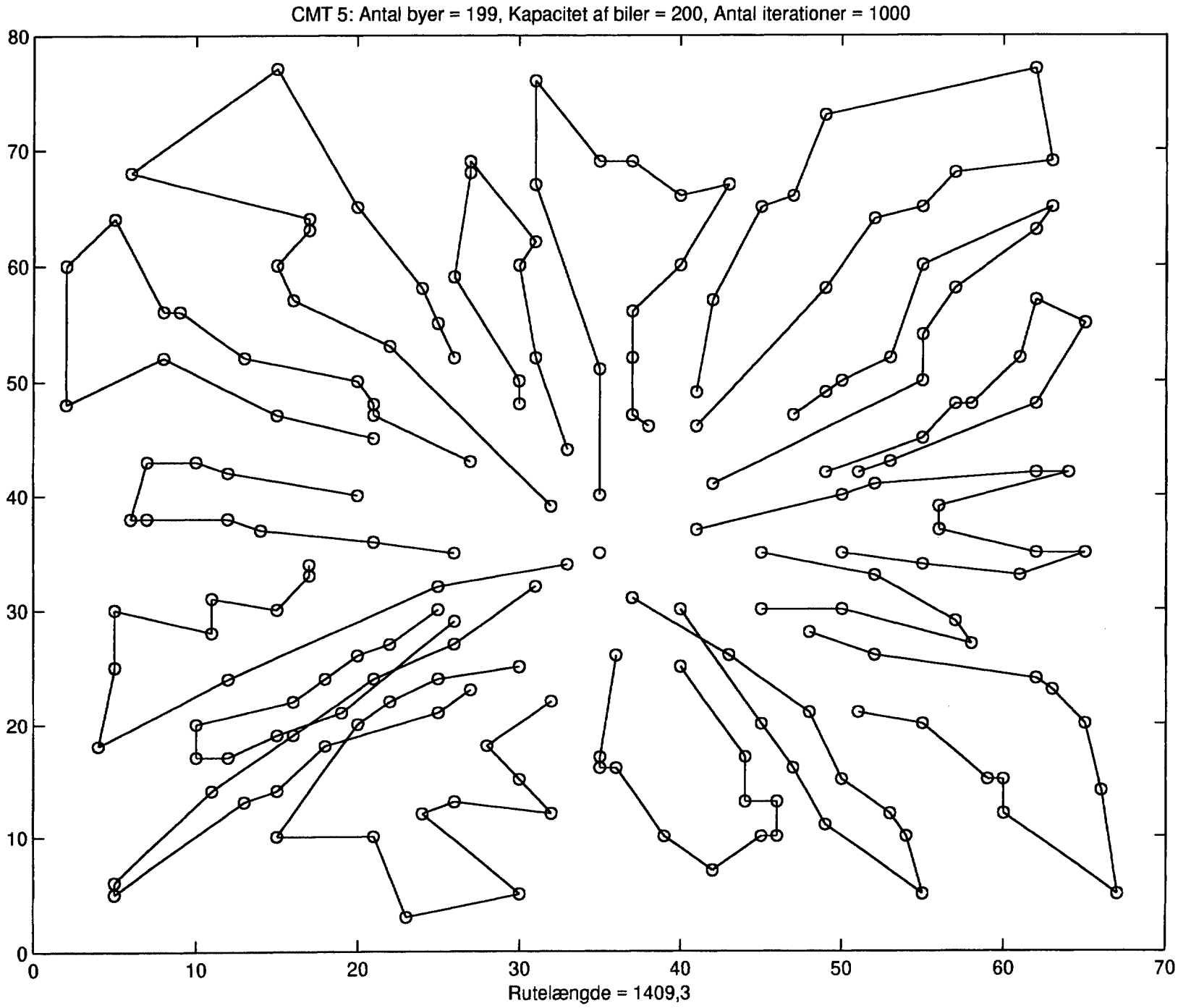
B Løsninger opnået af vores algoritme

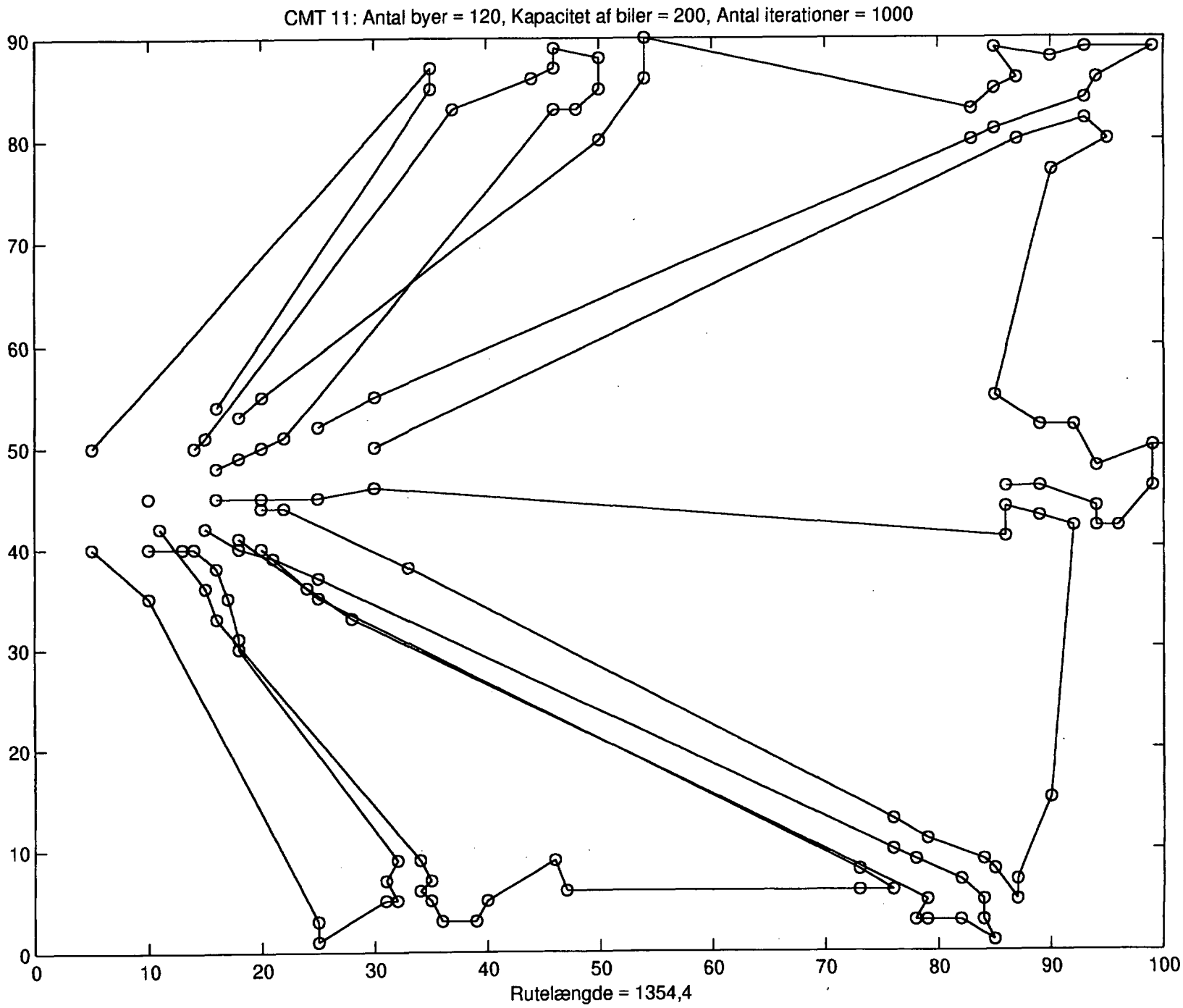

CMT 1: Antal byer = 50, Kapacitet af biler = 160, Antal iterationer = 1000
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Rutelængde = 530,3

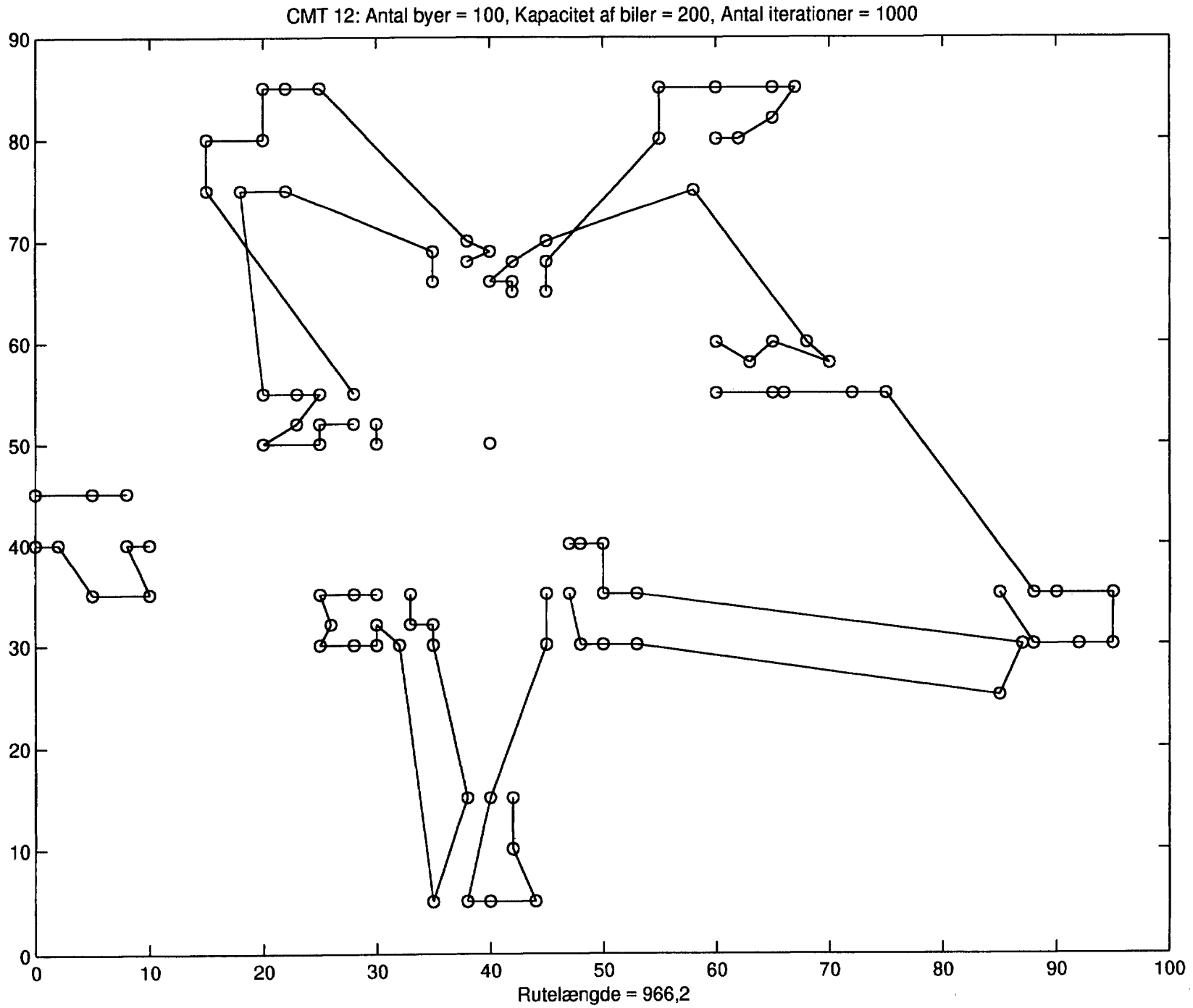

CMT 2: Antal byer = 75, Kapacitet af biler = 140, Antal iterationer = 1000
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Rutelængde = 886,9


CMT 3: Antal byer = 100, Kapacitet af biler = 200, Antal iterationer = 1000
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Rutelængde = 841,7


CMT 4: Antal byer = 150, Kapacitet af biler = 200, Antal iterationer = 1000
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Rutelængde = 1072,8


CMT 5: Antal byer = 199, Kapacitet af biler = 200, Antal iterationer = 1000
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Rutelængde = 1409,3


CMT 11: Antal byer = 120, Kapacitet af biler = 200, Antal iterationer = 1000
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Rutelængde = 1354,4


CMT 12: Antal byer = 100, Kapacitet af biler = 200, Antal iterationer = 1000
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Rutelængde = 966,2

Liste over tidligere udsendte tekster kan ses på IMFUFA's hjemmeside: http://mmf.ruc.dk eller rekvireres på sekretariatet, tlf. 46 74 22 63 eller e-mail: imfufa@ruc.dk.

332/97 ANOMAL SWELLING AF LIPIDE DOBBELTLAG
Specialerappoprt af: Stine Korremann
Vejleder: Dorthe Posselt

333/97 Biodiversity Matters
an extension of methods found in the literature on monetisation of biodiversity
by: Bernd Kuemmel

334/97 LIFE-CYCLE ANALYSIS OF THE TOTAL DANISH ENERGY SYSTEM
by: Bernd Kuemmel and Bent Sørensen

335/97 Dynamics of Amorphous Solids and Viscous Liquids
by: Jeppe C. Dyre

336/97 Problem-orientated Group Project Work at Roskilde University
by: Kathrine Legge

337/97 Verdensbankens globale befolkningsprognose
- et projekt om matematisk modellering
af: Jørn Chr. Bendtsen, Kurt Jensen, Per Pauli Petersen

338/97 Kvantisering af nanolederes elektriske ledningsevne
Første modul fysikprojekt
af: Søren Dam, Esben Danielsen, Martin Niss,
Esben Friis Pedersen, Frederik Resen Steenstrup
Vejleder: Tage Christensen

339/97 Defining Discipline
by: Wolfgang Coy

340/97 Prime ends revisited - a geometric point of view -
by: Carsten Lunde Petersen

341/97 Two chapters on the teaching, learning and assessment of geometry
by: Mogens Niss

342/97 A global clean fossil scenario DISCUSSION PAPER prepared by Bernd Kuemmel for the project LONG-TERM SCENARIOS FOR GLOBAL ENERGY DEMAND AND SUPPLY

343/97 IMPORT/EKSPORT-POLITIK SOM REDSKAB TIL OPTIMERET UDNYTTELSE AF EL PRODUCERET PÅ VE-ANLÆG
af: Peter Meibom, Torben Svendsen, Bent Sørensen

344/97 Puzzles and Siegel disks
by: Carsten Lunde-Petersen

345/98 Modeling the Arterial System with Reference to an Anesthesia Simulator
Ph.D. Thesis
by: Mette Sofie Olufsen

346/98 Klyngedannelse i en hulkatode-forstøvningsproces
af: Sebastian Horst
Vejledere: Jørn Borggren, NBI, Niels Boye Olsen

347/98 Verificering af Matematiske Modeller
- en analyse af Den Danske Eulerske Model
af: Jonas Blomqvist, Tom Pedersen, Karen Timmermann, Lisbet Øhlenschlæger
Vejleder: Bernhelm Booss-Bavnbek

348/98 Case study of the environmental permission procedure and the environmental impact assessment for power plants in Denmark
by: Stefan Krüger Nielsen
project leader: Bent Sørensen

349/98 Tre rapporter fra FAGMAT - et projekt om tal og faglig matematik i arbejdsmarkedsuddannelserne
af: Lena Lindenskov og Tine Wedege

350/98 OPGAVESAMLING - Bredde-Kursus i Fysik 1976 - 1998
Erstatter teksterne 3/78, 261/93 og 322/96

351/98 Aspects of the Nature and State of Research in Mathematics Education
by: Mogens Niss

352/98 The Herman-Swiatec Theorem with applications
by: Carsten Lunde Petersen

353/98 Problemløsning og modellering i en almendannende matematikundervisning
Specialerapport af: Per Gregersen og Tomas Højgaard Jensen

354/98 A Global Renewable Energy Scenario
by: Bent Sørensen and Peter Meibom

355/98 Convergence of rational rays in parameter spaces
by: Carsten Lunde Petersen and Gustav Ryd

356/98 Terrænmodellering
Analyse af en matematisk model til konstruktion af digitale terrænmodeller
Modelprojekt af: Thomas Frommelt, Hans Ravnkjær Larsen og Arnold Skimminge
Vejleder: Johnny Ottesen

357/98 Cayleys Problem
En historisk analyse af arbejdet med Cayleys problem fra 1870 til 1918
Et matematisk videnskabsfagsprojekt af: Rikke Degn, Bo Jakobsen, Bjarke K.W. Hansen, Jesper S. Hansen, Jesper Udesen, Peter C. Wulff
Vejleder: Jesper Larsen

358/98 Modeling of Feedback Mechanisms which Control the Heart Function in a View to an Implementation in Cardiovascular Models
Ph.D. Thesis by: Michael Danielsen

359/99 Long-Term Scenarios for Global Energy Demand and Supply
Four Global Greenhouse Mitigation Scenarios
by: Bent Sørensen (with contribution from Bernd Kuemmel and Peter Meibom)

360/99 SYMMETRI I FYSIK
En Meta-projektrapport af: Martin Niss, Bo Jakobsen & Tune Bjarke Bonné
Vejleder: Peder Voetmann Christiansen

361/99 Symplectic Functional Analysis and Spectral Invariants
by: Bernhelm Booß-Bavnbek, Kenro Furutani

362/99 Er matematik en naturvidenskab? - en udspænding af diskussionen
En videnskabsfagsprojekt-rapport af: Martin Niss
Vejleder: Mogens Nørgaard Olesen

363/99 EMERGENCE AND DOWNWARD CAUSATION
by: Donald T. Campell, Mark H. Bickhard, and Peder V. Christiansen

364/99 Illustrationens kraft - Visuel formidling af fysik
Integreret speciale i fysik og kommunikation
af Sebastian Horst
Vejledere: Karin Beyer, Søren Kjørup

365/99 To know - or not to know - mathematics, that is a question of context
by: Tine Wedege

366/99 LATEX FOR FORFATTERE - En introduktion til LATEX
og IMFUFA-LATEX
af: Jørgen Larsen

367/99 Boundary Reduction of Spectral Invariants and Unique Continuation Property
by: Bernhelm Booss-Bavnbek

368/99 Kvartvejsraport for projektet SCENARIER FOR SAMLET UDNYTTELSE AF BRINT SOM ENERGIBÆRER I DANMARKS FREMTIDIGE ENERGISYSTEM
Projektleder: Bent Sørensen

369/99 Dynamics of Complex Quadratic Correspondences
by: Jacob S. Jalving
Supervisor: Carsten Lunde Petersen

370/99 OPGAVESAMLING - Bredde-Kursus i Fysik 1976 - 1999
Eksamensopgaver fra perioden 1976 - 1999. Denne tekst erstatter tekst nr. 350/98

371/99 Bevisets stilling - beviser og bevisførelse i en gymnasial matematik undervisning
Et matematikspeciale af: Maria Hermannsson
Vejleder: Mogens Niss

372/99 En kontekstualiseret matematikhistorisk analyse af ikke-lineær programmering: Udviklingshistorie og multipel opdagelse
Ph.d.-afhandling af Tinne Hoff Kjeldsen

373/99 Criss-Cross Reduction of the Maslov Index and a Proof of the Yoshida-Nicolaescu Theorem
by: Bernhelm Booss-Bavnbek, Kenro Furutani and Nobukazu Otsuki

374/99 Det hydrauliske spring - Et eksperimentelt studie af polygoner og hastighedsprofiler
Specialeafhandling af: Anders Marcussen
Vejledere: Tomas Bohr, Clive Ellegaard, Bent C. Jørgensen

375/99 Begrundelser for Matematikundervisningen i den lærde skole hhv. gymnasiet 1884-1914
Historiespeciale af Henrik Andreassen, cand.mag. i Historie og Matematik

376/99 Universality of AC conduction in disordered solids
by: Jeppe C. Dyre, Thomas B. Schrøder

377/99 The Kuhn-Tucker Theorem in Nonlinear Programming: A Multiple Discovery?
by: Tinne Hoff Kjeldsen

378/00 Solar energy preprints:
1. Renewable energy sources and termal energy storage
2. Integration of photovoltaic cells into the global energy system
by: Bent Sørensen

379/00 EULERS DIFFERENTIALREGNING
Eulers indførelse af differentialregningen stillet over for den moderne
En tredjesemesters projektrapport på den naturvidenskabelige basisuddannelse
af: Uffe Thomas Volmer Jankvist, Rie Rose Møller Pedersen, Maja Bagge Pedersen
Vejleder: Jørgen Larsen

380/00 MATEMATISK MODELLERING AF HJERTEFUNKTIONEN
Isovolumetrisk ventrikulær kontraktion og udpumpning til det cardiovaskulære system
af: Gitte Andersen (3.moduls-rapport), Jakob Hilmer og Stine Weisbjerg (speciale)
Vejleder: Johnny Ottesen

381/00 Matematikviden og teknologiske kompetencer hos kortuddannede voksne
- Rekognosceringer og konstruktioner i grænselandet mellem matematikkens didaktik og forskning i voksenuddannelse
Ph. d.-afhandling af Tine Wedege

382/00 Den selvundvigende vandring
Et matematisk professionsprojekt
af: Martin Niss, Arnold Skimminge
Vejledere: Viggo Andreasen, John Villumsen

383/00 Beviser i matematik
af: Anne K.S.Jensen, Gitte M. Jensen, Jesper Thrane, Karen L.A.W. Wille, Peter Wulff
Vejleder: Mogens Niss

384/00 Hopping in Disordered Media: A Model Glass Former and A Hopping Model
Ph.D. thesis by: Thomas B. Schrøder
Supervisor: Jeppe C. Dyre

385/00 The Geometry of Cauchy Data Spaces
This report is dedicated to the memory of Jean Leray (1906-1998)
by: B. Booss-Bavnbek, K. Furutani, K. P. Wojciechowski

386/00 Neutrale mandatfordelingsmetoder – en illusion?
af: Hans Henrik Brok-Kristensen, Knud Dyrberg, Tove Oxager, Jens Sveistrup
Vejleder: Bernhelm Booss-Bavnbek

387/00 A History of the Minimax Theorem: von Neumann´s Conception of the Minimax Theorem - - a Journey Through Different Mathematical Contexts
by: Tinne Hoff Kjeldsen

388/00 Behandling af impuls ved kilder og dræn i C. S. Peskins 2D-hjertemodel
et 2. moduls matematik modelprojekt
af: Bo Jakobsen, Kristine Niss
Vejleder: Jesper Larsen

389/00 University mathematics based on problemoriented student projects: 25 years of experience with the Roskilde model
By: Mogens Niss
Do not ask what mathematics can do for modelling. Ask what modelling can do for mathematics!
by: Johnny Ottesen

390/01 SCENARIER FOR SAMLET UDNYTTELSE AF BRINT SOM ENERGIBÆRER I DANMARKS FREMTIDIGE ENERGISYSTEM Slutrapport, april 2001
Projektleder: Bent Sørensen
Projektdeltagere: DONG: Aksel Hauge Petersen, Celia Juhl, Elkraft System[#]: Thomas Engberg Pedersen[##], Hans Ravn, Charlotte Søndergren, Energi 2[#]: Peter Simonsen, RISØ Systemanalyseafd.: Kaj Jørgensen[*], Lars Henrik Nielsen, Helge V. Larsen, Poul Erik Morthorst, Lotte Schleisner, RUC: Finn Sørensen[**], Bent Sørensen
[#]Indtil 1/1-2000 Elkraft, [##] fra 1/5-2000 Cowi Consult
[*] Indtil 15/6-1999 DTU Bygninger & Energi, [**] fra 1/1-2001 Polypeptide Labs.
Projekt 1763/99-0001 under Energistyrelsens Brintprogram

391/01 Matematisk modelleringskompetence – et undervisningsforløb i gymnasiet
3. semesters Nat.Bas. projekt af: Jess Tolstrup Boye, Morten Bjørn-Mortensen, Sofie Inari Castella, Jan Lauridsen, Maria Gøtzsche, Ditte Mandøe Andreasen
Vejleder: Johnny Ottesen

392/01 "PHYSICS REVEALED" THE METHODS AND SUBJECT MATTER OF PHYSICS
an introduction to pedestrians (but not excluding cyclists)
PART III: PHYSICS IN PHILOSOPHICAL CONTEXT
by: Bent Sørensen.

393/01 Hilberts matematikfilosofi
Specialerapport af: Jesper Hasmark Andersen
Vejleder: Stig Andur Pedersen

394/01 "PHYSICS REVEALED" THE METHODS AND SUBJECT MATTER OF PHYSICS
an introduction to pedestrians (but not excluding cyclists)
PART II: PHYSICS PROPER
by: Bent Sørensen.

395/01 Menneskers forhold til matematik. Det har sine årsager!
Specialeafhandling af: Anita Stark, Agnete K. Ravnborg
Vejleder: Tine Wedege

396/01 2 bilag til tekst nr. 395: Menneskers forhold til matematik. Det har sine årsager!
Specialeafhandling af: Anita Stark, Agnete K. Ravnborg
Vejleder: Tine Wedege

397/01 En undersøgelse af solvents og kædelængdes betydning for anomal swelling i phospholipiddobbeltlag
2. modul fysikrapport af: Kristine Niss, Arnold Skimminge, Esben Thormann, Stine Timmermann
Vejleder: Dorthe Posselt

398/01 Kursusmateriale til "Lineære strukturer fra algebra og analyse" (E1)
Af: Mogens Brun Heefelt

399/01 Undergraduate Learning Difficulties and Mathematical Reasoning
Ph.D Thesis by: Johan Lithner
Supervisor: Mogens Niss

400/01 On Holomorphic Critical quasi circle maps
By: Carsten Lunde Petersen

401/01 Finite Type Arithmetic
Computable Existence Analysed by Modified Realisability and Functional Interpretation
Master's Thesis by: Klaus Frovin Jørgensen
Supervisors: Ulrich Kohlenbach, Stig Andur Pedersen and Anders Madsen

402/01 Matematisk modellering ved den naturvidenskabelige basisuddannelse
- udvikling af et kursus
Af: Morten Blomhøj, Tomas Højgaard Jensen, Tinne Hoff Kjeldsen og Johnny Ottesen