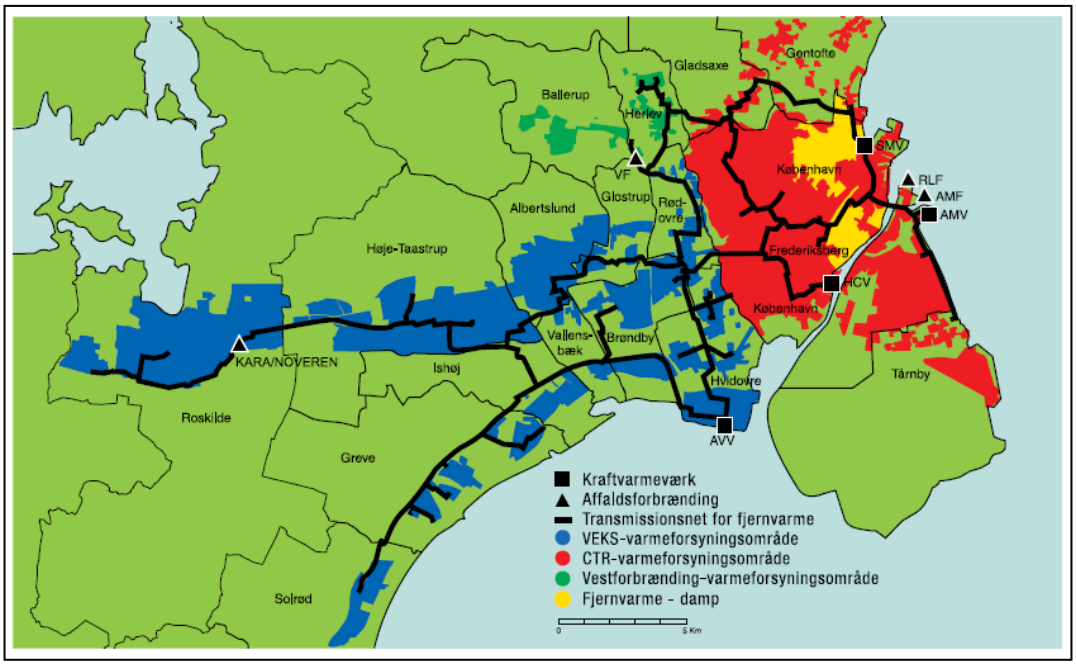
CENTER FOR STATISTIK

HEURISTIK TIL LØSNING AF VEHICLE ROUTING PROBLEMS

KANDIDATAFHANDLING

ERHVERVSØKONOMI & MATEMATIK
3. JUNI 2011



SKREVET AF:	KENNETH KNUDSEN & MAMONA TAHIR
VEJLEDER:	HANS KEIDING
STED:	COPENHAGEN BUSINESS SCHOOL
ANTAL NORMALSIDER:	108 (EKSKL. FORMLER, BILAG OG ILLUSTRATIONER)

Kenneth Knudsen **Mamona Tahir**

Executive summary

The Vehicle Routing Problem (VRP) is one of the most important and challenging problems in the field of Operations Research. It was first presented by Dantzig and Ramser in 1959, as a problem of designing a least-cost set of routes for vehicles in such a way, that all customers are visited once, and vehicle capacities are adhered to. The importance of VRP is due to the fact that it yields significant economic benefits, which is why researchers have given more and more attention to the various extensions of the VRP that arise from real life applications. Typically such extensions consider additional and more complicated constraints, such as time windows.

In this thesis, we have chosen to focus on approximate solution techniques, which not only provide high quality solutions, but within an acceptable computational time. Such methods are especially necessary in large VRP instances with hundreds or even thousands of both customers and constraints. Classical heuristics is a category, consisting of methods, which relatively quickly converges to a solution. It suffers, however, from the fact that the search often results in local minima, which is why the field of metaheuristics currently dominates the heuristics arena, since it considers inferior solutions as access points to more promising regions of the solution space.

At the time of this writing, the proposed metaheuristics for most of the VRP extensions is rather limited in the literature. However, we have nevertheless presented some of the currently best metaheuristics for several of such extensions.

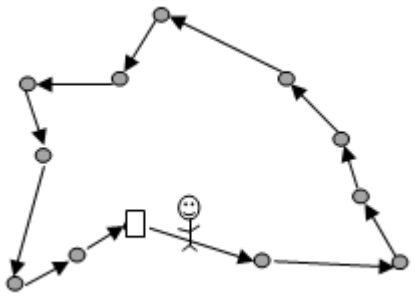
At the end of this paper, we have applied metaheuristics for an atypical real life instance of vehicle routing, which is based on the district heating network in Copenhagen. A solution is presented, and even though several of the special characteristics of the problem have been simplified and reduced, there is a large similarity between this solution and the existing network.

Indhold

Kapitel 1 – Introduktion

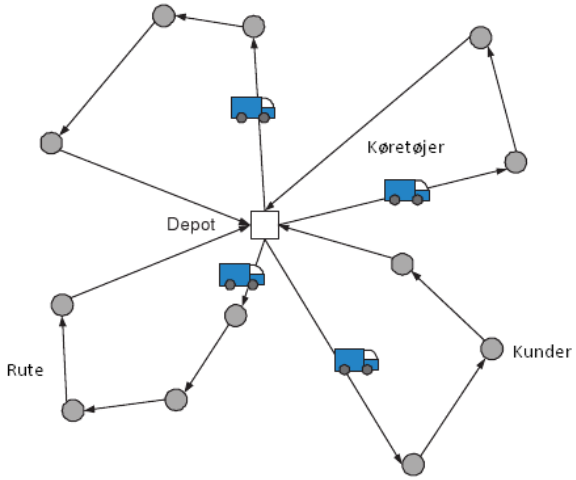
1.1 Motivation

”The Vehicle Routing Problem” (VRP) er selv i dag et af de mest udbredte og studerede emner. Det er en udvidelse til det allerede fra 1930'erne kendte kombinatoriske optimeringsproblem, "Travelling Salesman Problem" (TSP). I et TSP er der en mængde af lokaliteter (byer) og et antal veje der forbinder dem, hvor der til hver vej er tilknyttet en afstand. Problemet går ud på at bestemme den korteste rute gennem alle byer under betingelse af, at den rejsende sælger starter og slutter turen i samme by og at de andre byer besøges nøjagtig én gang. Figur 1 er et eksempel på et TSP.



Figur 1: Dette er et eksempel på et TSP, hvor hver by skal besøges netop én gang. Firkanten betegner udgangspunktet, cirklerne betegner byerne, mens pilene angiver den rejsende sælgers rute.

Dette problem er senere blevet generaliseret og udvidet til et ”Multiple Traveling Salesman Problem ” (*m*-TSP), hvor *m* sælgere dækker et givet antal byer, og hver by må kun besøges af én sælger. Hver sælger starter og slutter i samme by, som refereres som et depot. Denne nye idé har i sin tid været grundlaget for VRP, hvilket kan ses for værende et *m*-TSP med kundeefterspørgsler samt en kapacitet for hver sælger. Et eksempel på et VRP kan ses herunder.



Figur 2: Figuren viser et eksempel på et VRP, hvor 16 kunder serviceres af 4 køretøjer. Min Wen (2010).

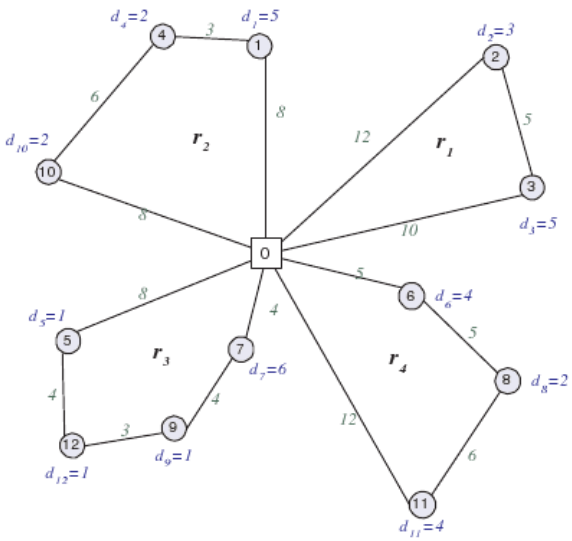
Det er muligt at se et VRP som en kombination af et TSP og et BPP (”Bin Packing Problem”). Problemet går kort og konkret ud på at undersøge, hvor mange kasser der mindst skal bruges, for at alle tingene kan pakkes ned. BPP kan altså angive en nedre grænse for, hvor mange køretøjer med en kapacitet Q, der mindst skal anvendes i et VRP med et bestemt antal kunder (der hver især har en efterspørgsel). VRP’s relation til de to kombinatoriske optimeringsproblemer er således relativ enkel; antal køretøjer samt hvilke kunder de tildeles løses ved et BPP, mens rækkefølgen de besøges i findes ved at løse det klassiske TSP for hvert køretøj. I kombinatorisk kompleksitetsteori er både TSP og BPP et NP-hårdt problem, som er et af de sværeste problemer, hvorfor VRP også tilhører klassen af NP-hårde problemer. Relativt små problemer kan nemt løses, men når antallet af noder stiger, bevæger man sig ud i et langt mere komplekst problem. Vi vil i dette afsnit ikke gå i dybden med NP-hårde problemer og beregningskompleksitet, men vender tilbage til det i et senere afsnit.

VRP har sin oprindelse i 1959, hvor Dantzig og Ramser for første gang introducerede VRP i deres artikel "The Truck Dispatching Problem”. Artiklen beskriver det grundlæggende VRP, dvs. et såkaldt ”Capacitated Vehicle Routing Problem” (CVRP), som er et af de simpleste VRP. Siden da har verden set tusindvis af artikler, som beskriver forskellige varianter af VRP og forsøger at give et billede af, hvordan man kan løse det. Hasle og Kloster definerer det klassiske CVRP som følgende:

”A number of identical vehicles with a given capacity are located at a central depot. They are available for servicing a set of customer orders, (all deliveries, or, alternatively, all pickups). Each customer order has a specific location and size. Travel costs between all locations are given. The goal is to design a least-cost set of routes for vehicles in such a way that all customers are visited once and vehicle capacities are adhered to.”, Hasle og Kloster (2007).

Herunder illustreres definitionen grafisk, for netop at vise princippet i et CVRP. Der er fire køretøjer til rådighed, hver med en kapacitet på 10. Kundeefterspørgsler samt rejseudgifter for hver kant er angivet i

figuren. En mulig rute er givet ved $r_1 = (0,2,3,0)$, hvor den respektive omkostning og leveret mængde er hhv. 27 og 8. En mulig løsning er givet ved $x = \{r_1, r_2, r_3, r_4\}$ med en totalomkostning på 103.



Figur 3: Her ses et eksempel på løsningen af et CVRP. Min Wen (2010)

Til dagligt tænker man ikke over, hvor stor en del af vores omverden egentlig kan beskrives som et VRP, selvom vi ofte selv er et direkte link i processen. Der findes en lang række praktiske problemstillinger, såsom transport af handikappede, skraldindsamling og distribution af dagligvarer, som alle tilhører klassen af traditionelle problemer. Der findes naturligvis også utraditionelle VRP-problemstillinger, som ikke følger ovenstående billede, men dog alligevel ejer visse VRP karakteristika. Disse vil vi komme nærmere ind på i næste kapitel.

Der er i praksis væsentlige økonomiske fordele at hente ved at optimere sine ruteplanlægningsproblemer. Derfor ser man løsninngen af VRP som nøglen til effektiv styring af transport og "supply chain"-koordinering. Det er dog ikke kun det økonomiske perspektiv, der gør VRP så attraktivt i praksis. Serviceniveauet er en anden side af sagen, som også kan tages højde for ved bearbejdning af problemet. Andre mål i praksis kan tænkes at være minimering af kundernes ventetid og tilbud om forskellige typer service (afhentning, levering eller begge).

Forskningen indenfor VRP har ikke kun betydning for den pågældende virksomhed og dets kunder, men anvendelsen heraf er også en fordel på det globale plan. Hermed menes reduceringen af CO_2-udledning som en ekstra gevinst ved en effektiv ruteplanlægning.

1.2 Formål og omfang

Med fokus på løsning af VRP og dets mest almindelige udvidelser, vil vi kort beskrive VRP og dets varianter samt redegøre grundigt for både heuristiske og metaheuristiske løsningsmetoder. Dernæst vil vi præsentere et virkelighedsnært VRP gennem Københavns fjernvarmenetværk, beskrive det som en VRP-variant samt udarbejde en algoritme til dets løsning.

Som følge af dette fokus og formål, har vi valgt kun at beskæftige os med de mest almindelige udvidelser til VRP, og det gælder alle dele af denne afhandling. Det drejer sig om følgende varianter af det grundlæggende problem; VRPTW, VRPPDTW, VRPBTW og OVRPTW.

Vi har også lagt mindre vægt på dele af rapporten omhandlende eksakte algoritmer, da disse benyttes forholdsvist begrænset i dag. I stedet forekommer metaheuristik at være sagen, når det kommer til løsning af udvidelser og større problemer.

Området for metaheuristikker har imidlertid vist sig omfattende i mere end én forstand. Der findes utrolig mange typer af metaheuristikker, der kan benyttes til at løse VRP, men kun få specifikke algoritmer skiller sig ud som forholdsvis succesfulde helt op til den dag i dag. Vi har derfor valgt først at beskrive de forskellige metaheuristikker for VRP på et generelt plan. Baseret på disse, udvælger og beskriver vi herefter specifikke algoritmer, rettet mod løsningen af bestemte udvidelser. Også her har vi valgt at begrænse os til et lille antal løsningsmetoder for hver udvidelse.

1.3 Rapportens opbygning

I kapitel 2 af rapporten gennemgår vi både den matematiske notation og fortolkningen af det grundlæggende VRP samt en række af de mest almindeligt forekommende udvidelser i praksis. Kapitel 3 og 4 udgør en kort overgang til heuristikkerne ved hhv. at introducere begrebet ”beregningskompleksitet” samt de mest almindelige eksakte algoritmer. I Kapitel 5 redegøres for nogle tidlige heuristikker for hhv. CVRP og VRPTW, mens kapitel 6 introducerer metaheuristik for VRP på et generelt plan og tager dernæst fat i de mest succesfulde metaheuristikker for VRPTW og de øvrige udvidelser, beskrevet i kapitel 2. Kapitel 7 tager udgangspunkt i et virkelighedsnært eksempel på et VRP, præsenterer en algoritme og løser problemet. Kapitel 8 udgør konklusionen på denne afhandling.

Kapitel 2 – The Vehicle Routing Problem

2.1 Det grundlæggende "Capacitated Vehicle Routing Problem"

Det grundlæggende problem indenfor VRP er et *"Capacitated Vehicle Routing Problem" (CVRP)*. Modellen for CVRP har følgende parametre:

c_{ij}: Omkostning forbundet med at køre fra kunde i til j.

K: Antal køretøjer.

$r(S)$: Mindste antal køretøjer til servicering af alle kunder i delmængden S, dvs. den optimale værdi af BPP tilknyttet kundemængden S.

n: Antal kunder i alt.

Q: Køretøjernes kapacitet.

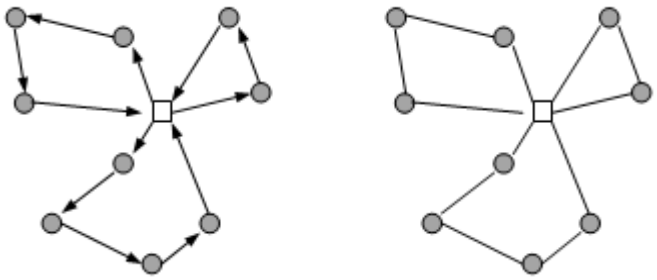
d_i: Kunde i's efterspørgsel.

hvor c_{ij} er en kontinuert, ikke-negativ variabel og de resterende er ikke-negative heltal. Samtlige parametre er deterministiske.

CVRP indbefatter et centralt depot med indeks $i = 0$ og en homogen flåde af K køretøjer med en kapacitet Q, der tilsammen servicerer kunderne $i = 1,...,n$. Kunderne har en deterministisk efterspørgsel og hver af dem må kun tilknyttes én rute og dermed kun serviceres af ét køretøj. Målet er at minimere den totale transportomkostning, givet at hvert køretøj starter og slutter i depotet og kapacitetsbegrænsningen er overholdt.

Den matematiske model kan beskrives som en graf: Lad $G = (V, E)$ være en komplet graf[1], hvor mængden af noder $V = \{0,1,...,n\}$ svarer til at $i = 1,...,n$ er kunderne og $i = 0$ er depotet. Mængden af kanter, $E = \{(i, j) : i, j \in V, i \neq j\}$, består af alle mulige forbindelser mellem noderne og hver kant $(i, j) \in E$ har en omkostning c_{ij} tilknyttet. Det er ikke tilladt at benytte løkker, (i,i), hvilket betyder at gå fra en node til samme node. Hvad angår grafen, kan denne opdeles i to varianter, enten som en orienteret graf eller som en uorienteret graf, nedenstående figur illustrerer de to tilfælde for en ikke-komplet graf. Figuren til venstre er det orienterede tilfælde, idet det eksplicit ses hvilken retning man skal køre. I modsat fald har man en uorienteret graf (figuren til højre).

[1] En komplet graf er en betegnelse for en simpel graf, hvor alle par af knuder er forbundet med en kant.



Figur 4: Eksempel på en orienteret og en uorienteret graf for en VRP-løsning.

Hvis G er en uorienteret graf, er der tale om et symmetrisk CVRP (SCVRP), idet omkostningsmatricen c er symmetrisk. Omvendt vil en orienteret graf lede til et asymmetrisk CVRP (ACVRP).

I praktiske situationer vil omkostningsmatricen oftest tilfredsstille trekantsuligheden.

(2.0) $$c_{ik} + c_{kj} \geq c_{ij} \quad , \forall\, i,j,k \in V$$

Den logiske forklaring er, at det ikke er optimalt at afvige fra en direkte forbindelse mellem to punkter i og j, ved at indsætte en node k mellem dem. Denne betingelse er ikke en del af den matematiske formulering af problemet, men i visse algoritmer for CVRP er trekantsuligheden et krav. I sådanne tilfælde, hvis det oprindelige problem ikke opfylder trekantsuligheden, kan et tilsvarende problem opnås på en umiddelbar måde ved at tilføje en passende stor positiv omkostning M til hver kant. Dog kan denne form for operation producere meget dårlige løsninger i forhold til de oprindelige omkostninger, da vi således kommer frem til en ikke-optimal løsning.

Lad os fortsætte den matematiske notation. Som sagt tidligere har hver kunde i en deterministisk efterspørgsel på d_i som skal leveres (eller afhentes), hvorimod depotet antager en fiktiv efterspørgsel, $d_0 = 0$. For en given mængde af kunder $S \subseteq V \setminus \{0\}$, skal der gælde at $d(S) = \sum_{i \in S} d_i$, betegner den totale efterspørgsel i den pågældende delmængde.

Sammenholdes dette til køretøjerne, må det nødvendigvis antages, at $d_i \leq Q$ for hvert $i = 1,...,n$, idet vi ved at de K identiske køretøjer, som udgår fra depotet, har en kapacitet på Q. Hvert køretøj får tildelt højst én rute og der antages at K ikke er mindre end $K_{\min}$, hvor $K_{\min}$ betegner det mindste antal af køretøjer der skal til for at få alle kunder serviceret[2]. I samme dur som tidligere er der igen tale om en delmængde $S \subseteq V \setminus \{0\}$, hvor størrelsen $r(S)$ betegner det mindste antal køretøjer der kræves til serviceringen af alle kunder i S. En vigtig bemærkning i denne sammenhæng er, at $r(V \setminus \{0\}) = K_{\min}$.

[2] $K_{\min}$ bestemmes ved at løse BPP for det tilhørende CVRP. Selv for hundredvis af elementer kan BPP give en optimal løsning på trods af NP-hårdheden.

Den binære beslutningsvariabel x_{ij} antager værdien 1, hvis kanten $(i,j) \in E$ tilhører den optimale løsning, og 0 ellers.

2.1.1 Matematisk formulering af CVRP

I det følgende opstilles en matematisk formulering af CVRP. Der er i litteraturen bl.a. nævnt tre forskellige typer af matematiske formuleringer, der hver især har sine fordele og ulemper. Typen "Vehicle Flow Formulations" benytter heltalsvariable tilknyttet hver kant på grafen. Disse variables funktion er blot at indikere hvorvidt et køretøj kører ad en bestemt kant. I "Commodity Flow Formulations" forbindes yderligere heltalsvariable til kanterne, der repræsenterer det flow af varer der er undervejs i ruten. "Set Partitioning Problems" opstiller et sæt binære variable for hver mulig rute, hvilket resulterer i et eksponentielt antal beslutningsvariable i forhold til antal kunder, men også gode muligheder for at tilpasse specifikke brugervalgte begrænsninger. I forbindelse med CVRP benyttes oftest den førstnævnte formulering, hvorfor vi i det følgende vælger at tage udgangspunkt heri.
Nedenstående formulering er gældende for et asymmetrisk problem, men kan uden besvær tilpasses det symmetriske. Forskellen er blot, at det symmetriske problem er et specialtilfælde af det asymmetriske med et noget mere begrænset mulighedsområde.

(2.1) $$\min \sum_{(i,j)\in E} c_{ij} x_{ij}$$

Ubb.

(2.2) $$\sum_{i \in V \setminus \{j\}} x_{ij} = 1 \qquad , \forall j \in V \setminus \{0\}$$

(2.3) $$\sum_{j \in V \setminus \{i\}} x_{ij} = 1 \qquad , \forall i \in V \setminus \{0\}$$

(2.4) $$\sum_{i \in V \setminus \{0\}} x_{i0} = K$$

(2.5) $$\sum_{j \in V \setminus \{0\}} x_{0j} = K$$

(2.6) $$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \qquad , \forall S \subseteq V \setminus \{0\}, S \neq \varnothing$$

(2.7) $$x_{ij} \in \{0,1\} \qquad , \forall (i,j) \in E$$

Objektfunktionen (2.1) beskriver kort at der er tale om minimering af de globale transportomkostninger. Betingelsen (2.2) og (2.3) kaldes hhv. ind- og udgangsbetingelserne, som sikrer at der til hver node i kun vælges én indgående og én udgående kant. Tilsvarende sikrer (2.4) og (2.5), at antallet af ruter, som hhv. forlader og vender tilbage til depotet, er lig antallet af køretøjer. Den næstsidste bibetingelse kendes ved navnet "Capacity-cut-constraints (CCC) og har til formål at undgå subtours i løsningen. En subtour er en rute der udgør en ring uden kontakt til hverken de andre ruter eller depotet. (2.6) lægger i den forbindelse både en kapacitetsbegrænsning på køretøjerne og sikrer at hvert muligt sæt af kunder er forbundet til depotet.

2.1.2 Omskrivning af "Capacity-cut-constraints"

Imidlertid er de ovenstående CCC-bibetingelser ikke særlig praktisk anvendelige i forhold til en løsningsmodel, hvorfor man typisk omskriver formuleringen til noget lineært. Dette kan gøres på forskellige måder, men langt den simpleste vil være at opstille en tre-indeks model og indføre hhv. en belastningsvariabel, B_i^k , der betegner belastningen på køretøj *k* efter node *i*, samt en positiv mængde, der skal leveres til kunde *i*, q_i (denne er oftest lig den efterspurgte mængde). På denne vis behøves der ikke yderligere data for at behandle problemet, idet belastningen er bestemt på baggrund af den leverede mængde alene. Når vi antager at $q_0 = 0$, vil modellen således kunne omskrives til følgende:

(2.8) $$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ij}^k$$

Ubb.

(2.9) $$\sum_{k \in K} \sum_{j \in V \setminus \{i\}} x_{ij}^k = 1 \quad , \forall i \in V \setminus \{0\}$$

(2.10) $$\sum_{i \in V \setminus \{h\}} x_{ih}^k - \sum_{j \in V \setminus \{h\}} x_{hj}^k = 0 \quad , , \forall h \in V \setminus \{0\}, k \in K$$

(2.11) $$\sum_{i \in V \setminus \{0\}} x_{ij}^k = 1 \quad , \forall k \in K$$

(2.12) $$\sum_{j \in V \setminus \{0\}} x_{ij}^k = 1 \quad , \forall k \in K$$

(2.13) $$\sum_{(i,j) \in E} q_i x_{ij}^k \leq Q \quad , \forall k \in K$$

(2.14) $$x_{ij}^k \left(B_i^k - q_j - B_j^k \right) = 0 \quad , \forall (i,j) \in E, k \in K$$

(2.15) $$0 \leq B_i^k \leq Q \quad , \forall i \in V, k \in K$$

(2.16) $$q_0 = 0 \quad , \forall i \in V, k \in K$$

(2.17) $$x_{ij} \in \{0,1\} \quad , \forall (i,j) \in E$$

Hvor (2.13) indeholder kapacitetsbegrænsningen, så indeholder (2.14)-(2.16) betingelsen om "forbundethed" ved at definere variablen B_i^k . Såfremt køretøj *k* kører mellem node *i* og *j*, skal belastningen efter service af *j* være lig belastningen efter *i* minus den mængde der bliver leveret til *j*. På denne måde vil subtours ikke kunne lade sig gøre, idet belastningen fortsat skal være positiv.

Betragt det tilfælde, hvor kunderne $i = 1,\ldots,n_s$ udgør en subtour. Ifølge (2.14) vil belastningen B_2^k efter service af kunde 2 være mindre end belastningen B_1^k efter service af kunde 1, da en positiv mængde q_2 er blevet leveret til kunde 2 af køretøj k. På samme vis vil $B_3^k < B_2^k$ og når køretøj k endelig kører mellem kunde n_s og 1, vil belastningen $B_1^k < B_{n_s}^k$. Men her springer kæden af. Såfremt $q_i > 0, \forall i \in \{1,\ldots,n_s\}$, vil dette medføre følgende:

(2.18) $$B_1^k > \ldots > B_{n_s}^k > B_1^k .$$

Dette kan ikke opfyldes, hvorfor subtours vil være elimineret fra problemet. Vi skal senere se, at der er andre måder at omskrive CCC-betingelserne på, som er særdeles praktiske når det kommer til udvidelserne af CVRP.

2.2 Udvidelser til problemet

Indtil videre har vi udelukkende betragtet det klassiske CVRP. Desværre passer langt de fleste af VRP i praksis ikke ind i disse meget forsimplede typer af problemer, idet vi i den enkelte branche oftest finder karakteristiske træk. Lad os kort redegøre for, hvilke typer af yderligere begrænsninger man kan forvente at se i sådanne praktiske problemer.

Depot: Der kan eksistere mere end ét depot, således at køretøjerne ikke nødvendigvis behøver at ende deres rute på samme depot, hvorfra de er kommet.

Køretøjer: Køretøjerne kan nemt være heterogene med forskellig kapacitet og indretning. F.eks. kan et køretøj have flere rum, hvert til sin type vare og med hver sin kapacitet. Der kan være tale om en asymmetrisk omkostningsmatrix, som betyder at det ikke koster det samme at transportere fra a til b, som fra b til a. De kan have andre specielle karakteristika, som betyder at bestemte kunder kun kan betjenes af en bestemt type køretøjer. Et køretøj kan bruges til mere end én rute og altså sendes ud igen efter at være vendt tilbage til depotet. Og så er der ikke mindst forskel på, hvorledes omkostningerne gøres op i variable (pr. afstand, pr. tidsenhed) og faste (pr. rute) omkostninger.

Chauffører: I dag finder vi mange krav fra bl.a. fagforeninger om arbejdstid, pauser, overtid osv. Mange af disse betingelser kan dog uden videre overføres til køretøjerne, og chaufførerne indgår således ikke direkte i problemet.

Kunder: Der kan nemt efterspørges mere end blot én vare, noget der komplicerer ruteplanlægningen yderligere. Derudover kan der være brug for ikke kun at levere en vare, men også opsamle varer hos andre eller selvsamme kunder. Der kan forekomme bestemte tidspunkter og/eller -intervaller for hver kunde, hvori kunden skal serviceres – der kan f.eks. være tale om kundens åbningstider. Nogle kunder kan det være tilladt at servicere adskillige gange med forskellige køretøjer. En virksomhed kan have særlige præferencer for nogle af deres kunder, f.eks. de der indgår som en væsentlig del af deres

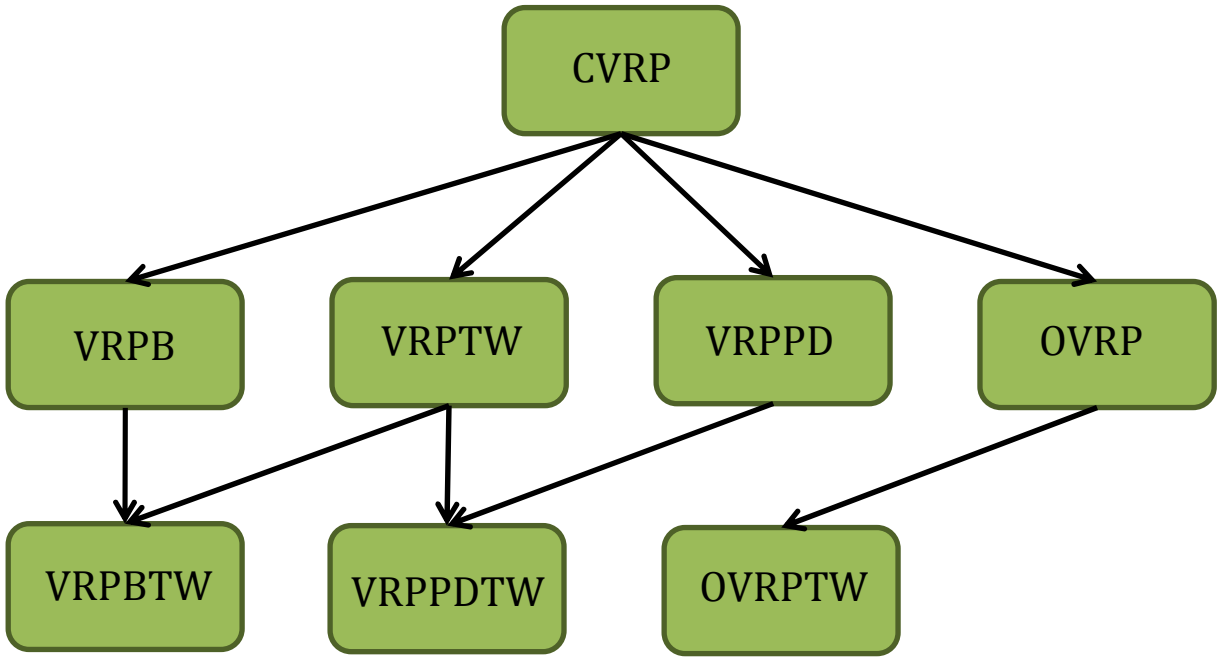
indtjeningsgrundlag. Der kan være varer eller personer der skal afhentes ét sted og leveres et andet. Kunder kan også have brug for service mere eller mindre jævnligt.

Objektfunktion: Normalt vil det være muligt at formulere problemet til et simpelt spørgsmål om at minimere den totale afstand mellem noderne i ruten. Dog kan man nemt forestille sig, at flere mål kan have interesse. Her kan som eksempel nævnes maksimering af kundetilfredshed, profit eller minimering af antal køretøjer, tidsforbrug, arbejdsbelastning, eller endda CO_2-emmisioner og andre miljøpåvirkninger. Hver af disse kan i sig selv være anledningen til problemet, men en virksomhed kan have adskillige interesser som skal tages højde for samtidig og evt. i prioriteret rækkefølge.

Rute: Udover de allerede nævnte udvidelser, vil vi i virkeligheden se meget forskelligartede problemer, noget der ændrer den måde ruten ser ud på. Tag som eksempel *Open VRP* (OVRP), hvor køretøjerne ikke er pålagt at vende tilbage til depotet, men hvis de gør, skal det gøres ved at besøge hver af kunderne i omvendt rækkefølge.

Usikkerhed: Et stort område inden for ruteplanlægning omfatter usikkerhed. Der kan være tale om stokastisk efterspørgsel og/eller placering af kunder; dvs. noget der ikke kendes med sikkerhed fra starten af, når køretøjerne sendes ud. I nogle tilfælde vil en sandsynlighedsfordeling være kendt, i andre ikke.

Alle disse mulige udvidelser og begrænsninger til det grundlæggende CVRP vil føre til forskellige stereotyper af problemer. Herunder kan ses et sæt af de mulige standardudvidelser, som indeholder én eller adskillige af ovenstående karakteristika og som vil blive beskrevet i de følgende afsnit.



Figur 5: Viser udvidelserne fra CVRP ved hjælp af pile. Ydermere har alle udvidelser en version der inkluderer tidsvinduer og det er disse vi vil beskæftige os med i særdeleshed.

Det skal igen bemærkes, at ovenstående er typiske udvidelser, og at der naturligvis kan forekomme betingelser fra flere af ovenstående varianter i det pågældende problem. Samtidig findes så mange andre typer af problemer udover de i figuren nævnte udvidelser, at det vil være formålsløst at forsøge at beskrive dem alle. Vi vil i de kommende afsnit kort beskrive de mest forekomne typer af problemer, herunder:

- The Vehicle Routing Problem with Time Windows (VRPTW)
- The Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW)
- The Vehicle Routing Problem with Backhauls and Time Windows (VRPBTW)
- The Open Vehicle Routing Problem with Time Windows (OVRPTW)

Vi vælger at indføre tidsvinduer for alle problemer beskrevet i dette kapitel, idet denne type begrænsninger forekommer i langt de fleste problemer i praksis. Det er forsøgt at holde notationen ren på tværs af udvidelserne, således at de forholdsvis overskueligt kan sammenlignes. Blandt andet derfor har vi i de følgende formuleringer valgt at minimere de totale transportomkostninger, og vi vil udelukkende beskæftige os med homogene køretøjer.

2.2.1 The Vehicle Routing Problem with Time Windows

VRPTW er én af de mest kendte udvidelser til det klassiske CVRP. Køretøjet skal begynde service af en kunde indenfor det angivne tidsvindue og blive der indtil service er afsluttet. Med hårde tidsvinduer er det ikke tilladt at ankomme senere end det specificerede interval, og hvis køretøjet kommer for tidligt, vil det være nødt til at vente til starten af tidsvinduet før service påbegyndes. Med bløde tidsvinduer vil det godt kunne lade sig gøre, men det sker med en omkostning eller straf. Vi vil i resten af denne afhandling antage, at vi har med hårde tidsvinduer at gøre, medmindre andet er specificeret.

Problemet kendes fra bl.a. bustransport eller diverse service-virksomheder, hvor en kunde bliver tildelt et tidsrum, hvor de skal være til stede og tage imod servicen (fx pakkepost, vand- og varmeaflæsning, hjemmepleje m.fl.).

Matematisk Formulering

Vi lader node 0 være depotet, hvorfra hvert af de homogene køretøjer $k \in K$ udgår og node $n+1$ være depotet hvor de kommer tilbage. Lad $G = (V, E)$ være en orienteret graf, hvor $V = \{0\} \cup C \cup \{n+1\}$ er mængden af samtlige noder inkl. depot, $C = \{1, \ldots, n\}$ er mængden af kunder og $E = \{(i, j) : i, j \in V, i \neq j\}$ er mængden af kanter.

Til hver kunde i er knyttet en efterspørgsel d_i samt en servicetid s_i, som betegner den tid det tager at servicere kunde i, og dermed hvor lang tid køretøjet må forblive på stedet. $[a_i, b_i]$ betegner tidsvinduet, hvori servicering af den i'te kunde skal være påbegyndt. Et køretøj er tilladt at ankomme til kunde i før tidsvinduet, men i så fald må service ikke påbegyndes før a_i. Dertil kommer at t_{ij} betegner rejsetiden mellem kunde i og kunde j. Per konstruktion sættes $d_0 = d_{n+1} = 0$ og $s_0 = s_{n+1} = 0$ for depotet. Q betegner køretøjernes kapacitet.

Lad x_{ij}^k være binære beslutningsvariable, der er lig 1 hvis køretøj k tilbagelægger afstanden $(i,j) \in E$ og 0 ellers. Der er knyttet omkostningerne c_{ij} til hver kant og ydermere defineres de kontinuerte tidsvariable w_i^k som k's start på service af kunde i. Vi skriver modellen op som følger:

(2.20) $$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ij}^k$$

Ubb.

(2.21) $$\sum_{k \in K} \sum_{j \in V \setminus \{i\}} x_{ij}^k = 1 \quad , \forall i \in C$$

(2.22) $$\sum_{i \in V \setminus \{h\}} x_{ih}^k - \sum_{j \in V \setminus \{h\}} x_{hj}^k = 0 \quad , \forall h \in C, k \in K$$

(2.23) $$\sum_{(i,j) \in E} d_i x_{ij}^k \leq Q \quad , \forall k \in K$$

(2.24) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k = 1 \quad , \forall k \in K$$

(2.25) $$\sum_{i \in V \setminus \{n+1\}} x_{i,n+1}^k = 1 \quad , \forall k \in K$$

(2.26) $$x_{ij}^k \in \{0,1\} \quad , \forall (i,j) \in E, k \in K$$

(2.27) $$x_{ij}^k \left(w_i^k + s_i + t_{ij} - w_j^k \right) \leq 0 \quad , \forall (i,j) \in E, k \in K$$

(2.28) $$a_i \leq w_i^k \leq b_i \quad , \forall i \in V, k \in K$$

(2.29) $$w_i^k \geq 0 \quad , \forall i \in V, k \in K$$

Objektfunktionen (2.20) minimerer de samlede transportomkostninger. (2.21) sikrer, at ét og kun ét køretøj betjener hver kunde i. Betingelsen (2.22) angiver, at hvis køretøj k ankommer til kunde h, skal det samme køretøj også forlade h. For hvert køretøj k, må mængden af servicerede kunder ifølge (2.23) ikke overskride køretøjets kapacitet. (2.24) og (2.25) dikterer, at hvert køretøj skal forlade hhv. ankomme til depotet præcis én gang. Med (2.27) må service af kunde j ikke påbegyndes før der er gået mindst $s_i + t_{ij}$ siden service af kunde i blev påbegyndt, dvs. service kan ikke startes før køretøjet er ankommet til kunden. Med ulighedstegnet er det dermed også tilladt, at service af j påbegyndes senere end ankomsten. Centralt i denne variant af VRP angiver (2.28), at service skal påbegyndes indenfor kundens tidsvindue, eller for depotet, dets "åbningstid".

Toth og Vigo (2002) beskriver, hvorledes (2.27) kan opskrives på lineær form ved:

(2.30) $$w_j^k \geq w_i^k + s_i + t_{ij} - M\left(1 - x_{ij}^k\right) \quad , \forall (i,j) \in E, k \in K$$

Hvor M er et tilstrækkeligt stort tal, således at bibetingelsen stadig er opfyldt (men ikke bindende) for de kanter, der ikke tilbagelægges af et køretøj. Denne omskrivning kan lette den beregningsmæssige løsning af problemet.

Som det bemærkes, undgår vi også CCC-betingelserne fra det oprindelige CVRP i dette problem. Vi har kapacitetsbegrænsningen i (2.23), og selv om vi ikke har en belastningsvariabel, der på den måde eliminerer subtours, så indgår her tidsvariable, w_i^k, der har samme effekt. Forbindelsen mellem alle noder skal være tidsmæssigt sammenhængende og af samme logiske ræsonnement som i forrige kapitel er det derfor klart, at (2.27) eliminerer subtours.

Dette vil være gældende for samtlige problemer med tidsvinduer, hvorfor de følgende problemer også vil opfylde kravet om "forbundethed".

2.2.2 The Vehicle Routing Problem with Pickup and Delivery and Time Windows

I VRPPDTW har vi nu, i stedet for et sæt kunder, et sæt ordrer om at transportere en mængde genstande fra ét sted til et andet. Det betyder rent praktisk, at afhentningen skal ske før leveringen og at de to noder, der repræsenterer hhv. afhentning og levering, skal indgå i samme rute, tilbagelagt af samme køretøj.

Ét af de bedste eksempler på denne type problemer er en taxi-central, som modtager en række opkald, ordrer om at transportere personer fra ét sted til et andet. Bemærk, her antager vi at for hver afhentningsnode er der én og kun én leveringsnode, det er altså ikke tilladt at personer der samles op samme sted bliver sat af forskellige steder.

Matematisk Formulering

Betragt problemet med et enkelt depot, der udsender et sæt køretøjer til at servicere n ordrer af "Pickup and Delivery"-typen, dvs. n afhentninger ved node $i = (1,\ldots,n)$ og n leveringer ved node $n+i = (n+1,\ldots,2n)$. Vi lader node 0 og $2n+1$ være hhv. depotet hvorfra hvert af køretøjerne $k \in K$ udgår og hvor de kommer tilbage. Lad $G = (V,E)$ være en orienteret graf, hvor $V = \{0\} \cup P \cup D \cup \{2n+1\}$ er mængden af samtlige noder inkl. depot, $P = \{1,\ldots,n\}$ er mængden af afhentningssteder og $D = \{n+1,\ldots,2n\}$ er de tilsvarende leveringssteder. $E = \{(i,j) : i,j \in V, i \neq j\}$ er mængden af kanter på grafen.

Til forskel fra VRPTW, specificerer d_i mængden der skal afhentes og leveres ved node i, hvorfor der gælder, at $d_i = -d_{n+i} > 0 \ (i = 1,\ldots,n)$. Depotet har ingen servicetid eller efterspørgsel og derfor sættes $s_0 = s_{2n+1} = 0$ og $d_0 = d_{2n+1} = 0$.

En ny variabel, B_i^k, betegner belastningen på k efter node i. Vi skriver modellen op som følger:

(2.31) $$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ij}^k$$

Ubb.

(2.32) $$\sum_{k \in K} \sum_{j \in V \setminus \{i\}} x_{ij}^k = 1 \qquad , \forall i \in P$$

(2.33) $$\sum_{i \in V \setminus \{h\}} x_{ih}^k - \sum_{j \in V \setminus \{h\}} x_{hj}^k = 0 \qquad , \forall h \in P \bigcup D, k \in K$$

(2.34) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k = 1 \qquad , \forall k \in K$$

(2.35) $$\sum_{i \in V \setminus \{2n+1\}} x_{i,2n+1}^k = 1 \qquad , \forall k \in K$$

(2.36) $$x_{ij}^k \in \{0,1\} \qquad , \forall (i,j) \in E, k \in K$$

(2.37) $$x_{ij}^k \left(w_i^k + s_i + t_{ij} - w_j^k \right) \leq 0 \qquad , \forall (i,j) \in E, k \in K$$

(2.38) $$a_i \leq w_i^k \leq b_i \qquad , \forall i \in V, k \in K$$

(2.39) $$w_i^k \geq 0 \qquad , \forall i \in V, k \in K$$

(2.40) $$x_{ij}^k \left(B_i^k + d_j - B_j^k \right) = 0 \qquad , \forall (i,j) \in E, k \in K$$

(2.41) $$d_i \leq B_i^k \leq Q \qquad , \forall i \in P, k \in K$$

(2.42) $$0 \leq B_i^k \leq Q + d_i \qquad , \forall i \in D, k \in K$$

(2.43) $$B_i^k = 0 \qquad , \forall i \in \{0, 2n+1\}, k \in K$$

(2.44) $$\sum_{j \in V \setminus \{i\}} x_{ij}^k - \sum_{j \in V \setminus \{i\}} x_{j,n+i}^k = 0 \qquad , \forall i \in P, k \in K$$

(2.45) $$w_i^k + s_i + t_{i,n+i}^k \leq w_{n+i}^k \qquad , \forall i \in P, k \in K$$

(2.32) sikrer, at ét og kun ét køretøj servicerer hvert afhentningssted og at samtlige afhentningssteder bliver serviceret. Betingelsen (2.44) angiver, at hvis køretøj k opsamler ved node i, skal samme køretøj også levere ved node $n+i$. Og da samme køretøj servicere $n+i$ som servicerede i, så er betingelsen (2.32) implicit også opfyldt for alle leveringssteder. Såfremt køretøj k kører mellem node i og j, skal belastningen efter service af j ifølge (2.40) være lig belastningen efter i inklusiv den mængde der

leveres til j. (2.41)-(2.42) indeholder kapacitetsbegrænsningen for hvert køretøj og ifølge (2.43) er belastningen på depotet ved start og slut lig 0. Sidstnævnte ses klart, idet hver opsamlet mængde skal leveres af samme køretøj inden det kan vende tilbage til depotet. Betingelse (2.45) tvinger køretøjet til at besøge afhentningsstedet i tidsmæssigt før leveringsstedet $n+i$.

2.2.3 The Vehicle Routing Problem with Backhauls and Time Windows

I VRPBTW skal varer fragtes fra et centralt depot rundt til en række kunder, der hver kan hhv. efterspørge en vare eller ønske en vare afhentet. Under bl.a. kapacitetsbegrænsninger og under hensyn til tidsvinduer for kunderne, gælder det nu om at minimere de totale transportomkostninger, samtidig med at al levering (linehaul) skal ske før afhentning (backhaul). Problemet er i grunden meget lig VRPPDTW, men her fyldes køretøjerne altså på depotet og leverer varerne til de kunder der efterspørger dem, før de sidst på deres rute kører rundt og samler varer op.

Denne grundlæggende antagelse, at al levering sker før afhentning, kommer af, at det ofte ikke er muligt (eller økonomisk fornuftigt) at omarrangere varerne i køretøjet ved hvert stop, hvilket nemt kan være nødvendigt, hvis det ikke er de samme varer der leveres, som dem der afhentes. Derfor leveres der først, således at der bliver gjort plads til de varer der senere skal læsses. Derudover ses levering til kunder oftest som en højere prioritet i praksis end afhentning.

I praksis finder vi denne type problemer indenfor produktionsvirksomheder, der sørger for både levering af produkter samt afhentning af råmaterialer fra leverandører. Det vil også være at finde i restaurationsbranchen og ved levering til større supermarkedskæder.

Matematisk Formulering

I stil med VRPPDTW deler vi kunderne i to mængder, linehaul $L=\{1,\ldots,n\}$ og backhaul $B=\{n+1,\ldots,n+m\}$; lad node 0 og $n+m+1$ være hhv. depotet, hvorfra hvert af køretøjerne $k\in K$ udgår og hvor de kommer tilbage. $G=(V,E)$ definerer en orienteret graf, hvor $V=\{0\}\cup L\cup B\cup\{n+m+1\}$ er mængden af samtlige noder inkl. depot og $E=\{(i,j):i,j\in V, i\neq j\}$ er mængden af kanter på grafen.

Hver kunde i har nu en ikke-negativ efterspørgsel d_i, der svarer til den mængde der hhv. ønskes leveret eller afhentet. Per konstruktion sættes $d_0=d_{n+m+1}=0$ og $s_0=s_{n+m+1}=0$. Modellen opstilles som følger:

(2.46) $$\min\sum_{k\in K}\sum_{(i,j)\in E}c_{ij}^k x_{ij}^k$$

Ubb.

(2.47) $$\sum_{k\in K}\sum_{j\in V\setminus\{i\}}x_{ij}^k=1 \qquad ,\ \forall i\in L\cup B$$

(2.48) $$\sum_{i \in V \setminus \{h\}} x_{ih}^k - \sum_{j \in V \setminus \{h\}} x_{hj}^k = 0 \quad , \forall h \in L \cup B, k \in K$$

(2.49) $$\sum_{(i,j) \in L} d_i x_{ij}^k \leq Q \quad , \forall k \in K$$

(2.50) $$\sum_{(i,j) \in B} d_i x_{ij}^k \leq Q \quad , \forall k \in K$$

(2.51) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k = 1 \quad , \forall k \in K$$

(2.52) $$\sum_{i \in V \setminus \{n+m+1\}} x_{i,n+m+1}^k = 1 \quad , \forall k \in K$$

(2.53) $$\sum_{i \in \{0\} \cup L} \sum_{j \in B \cup \{n+m+1\}} x_{ij}^k = 1 \quad , \forall k \in K$$

(2.54) $$x_{ij}^k \in \{0,1\} \quad , \forall (i,j) \in E, k \in K$$

(2.55) $$x_{ij}^k \left(w_i^k + s_i + t_{ij} - w_j^k \right) \leq 0 \quad , \forall (i,j) \in E, k \in K$$

(2.56) $$a_i \leq w_i^k \leq b_i \quad , \forall i \in V, k \in K$$

(2.57) $$w_i^k \geq 0 \quad , \forall i \in V, k \in K$$

Kapacitetsbegrænsningerne for hhv. linehaul- og backhaulnoder findes i (2.49)-(2.50), der med fordel kan skrives på denne form, idet køretøjet er tomt på kanten mellem overgangen fra linehaul til backhaul. (2.53) er den centrale bibetingelse i VRPBTW, idet den er tilstrækkelig til at sikre, at linehaulkunder serviceres før backhaulkunder.

2.2.4 The Open Vehicle Routing Problem with Time Windows

OVRPTW ligner på næsten alle områder VRPTW, men med den ekstra betingelse, at køretøjerne ikke er pålagt at vende tilbage til depotet efter endt service, men hvis de gør, skal det foregå ved at besøge de samme kunder i omvendt rækkefølge.

Problemet kan opdeles i følgende 3 typer:

1) *Kun levering:* Køretøjerne står for at levere varer til virksomhedens kunder og det er ikke nødvendigt at vende tilbage til depotet efter endt service.
2) *Kun afhentning:* Køretøjerne står udelukkende for afhentning af varer, der skal leveres til depotet og kan dermed starte ved noden for enden af ruten der går til depotet.
3) *Både afhentning og levering:* Efter levering til alle kunder på ruten, tager køretøjerne turen tilbage til depotet mens de afhenter varer hos samtlige kunder i omvendt rækkefølge, eller efter afhentning fra alle kunder på ruten, tager køretøjerne turen fra depotet med varer tilbage til kunderne i omvendt rækkefølge. Heraf ses, at OVRPTW rent faktisk er et specialtilfælde af VRPBTW, blot hvor returruten er fastlagt til de allerede besøgte kunder.

Rent grafisk ser problemet ud som et fra depotet udgående "Spanning Tree", hvor samtlige noder skal besøges og hver node højst er forbundet til 2 andre – en foregående og en efterfølgende. Der vil med andre ord ikke forekomme "ring-ruter", hvor køretøjerne både starter og slutter ved et depot – deraf navnet, et åbent VRP.

Denne type problemer findes i mange virksomheder i dag, idet det er blevet mere populært at outsource ikke-essentielle arbejdsopgaver, så der kan koncentreres om kerneaktiviteterne. Således finder vi virksomheder, der ikke selv står for levering af varer til deres kunder, enten fordi de ikke har køretøjer til at foretage levering, fordi de ikke har nok til at tilfredsstille samtlige kunder, eller fordi de ønsker at slippe for visse omkostninger forbundet ved at have en flåde af køretøjer, såsom reparation og vedligeholdelse. De eksterne køretøjer er dermed ikke pålagt at vende tilbage til depotet, men er fritstillet efter endt service og køretøjet er tømt for varer.

Erhverv hvor denne type problemer er stærkt forekommende: Offentlig transport som busser og tog, flytransport (ofte med simultan afhentning og levering samt tidsvinduer) og f.eks. levering af frokost til skoler.

Matematisk Formulering

Vi vil her benytte stort set samme notation som for VRPTW. Dog betegner $i = 0$ nu depotet for både ankomst og afgang, hvorfor $V = \{0\} \cup C, C = \{1, \ldots, n\}$. I de tilfælde hvor eksterne køretøjer rekvireres, vil det mange gange være nødvendigt at inddrage en fast omkostning i problemet, hvorfor parameteren f^k indføres og betegner den faste omkostning. y^k er en beslutningsvariabel der er lig 1 hvis køretøj k er aktiv og 0 ellers. Vi skriver modellen op som følger:

(2.58) $$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij}^k x_{ij}^k + \sum_{k \in K} f^k y^k$$

Ubb.

(2.59) $$\sum_{k \in K} \sum_{j \in V \setminus \{i\}} x_{ij}^k = 1 \qquad , \forall i \in C$$

(2.60) $$\sum_{i \in V \setminus \{h\}} x_{ih}^k - \sum_{j \in V \setminus \{h\}} x_{hj}^k = 0 \quad , \forall h \in C, k \in K$$

(2.61) $$\sum_{(i,j) \in E} d_i x_{ij}^k \leq Q \quad , \forall k \in K$$

(2.62) $$x_{ij}^k \left(w_i^k + s_i + t_{ij} - w_j^k \right) \leq 0 \quad , \forall (i,j) \in E, k \in K$$

(2.63) $$a_i \leq w_i^k \leq b_i \quad , \forall i \in V, k \in K$$

(2.64) $$x_{ij}^k \in \{0,1\} \quad , \forall (i,j) \in E, k \in K$$

(2.65) $$w_i^k \geq 0 \quad , \forall i \in V, k \in K$$

(2.66) $$y^k \in \{0,1\} \quad , k \in K$$

(2.67) $$x_{ij}^k \leq y^k \quad , \forall (i,j) \in E, k \in K$$

Indtil nu er ovenstående blot et standard VRPTW, hvor vi har fjernet bibetingelserne hvori depotet indgår. Følgende bibetingelser skal ydermere opfyldes, alt efter hvilken af de førnævnte typer af problemer man står overfor.

1) *Kun levering:*

(2.68) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k \leq 1 \quad , \forall k \in K$$

(2.69) $$\sum_{i \in V \setminus \{0\}} x_{i0}^k = 0 \quad , \forall k \in K$$

Ved levering forekommer kun afgange fra depotet og ikke ankomster. Bemærk, da ikke alle køretøjer nødvendigvis er aktive, vil (2.68) ikke altid være bindende.

2) *Kun afhentning:*

(2.70) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k = 0 \quad , \forall k \in K$$

(2.71) $$\sum_{i \in V \setminus \{0\}} x_{i0}^k \leq 1 \quad , \forall k \in K$$

Her ses det omvendte tilfælde, hvor kun ankomster til depotet er tilladt, mens de pågældende bibetingelser dog stadig ikke nødvendigvis er bindende.

3) *Både afhentning og levering:*

(2.72) $$\sum_{j \in V \setminus \{0\}} x_{0j}^k \leq 1 \qquad , \forall k \in K$$

(2.73) $$\sum_{i \in V \setminus \{0\}} x_{i0}^k \leq 1 \qquad , \forall k \in K$$

(2.74) $$x_{ij}^k = x_{ji}^k \qquad , \forall (i,j) \in E, k \in K$$

Fordi der forekommer både afhentning og levering, vil både ankomster og afgange til/fra depotet være mulige. Vi indfører dog den betingelse at såfremt k tilbagelægger strækningen i til j, skal samme køretøj også køre fra j til i. På den måde kommer køretøjet til at tage samme tur tilbage gennem punkterne efter endt levering hhv. afhentning.

2.2.5 Øvrige Udvidelser

Indtil nu har vi indført 4 varianter til det klassiske VRP, men der er naturligvis langt flere, idet verden er fyldt med problemer af speciel karakter. I dette afsnit vil vi kort redegøre for nogle af de vigtigste i rent praktiske sammenhænge, for at fuldstændiggøre billedet af en rig verden af VRP og ikke mindst størrelsen og omfanget af emnet.

Heterogent Vehicle Routing Problem

Vi har allerede berørt spørgsmålet om heterogene eller homogene køretøjer, men indtil videre har vi udelukkende haft med homogene at gøre. Dog er virkeligheden oftest ikke så simpel, idet der er mange årsager til, at en virksomhed vælger at holde en forholdsvis forskelligartet flåde af køretøjer. Som eksempel kan tages et internationalt rederi, der er aktiv indenfor fragt af både tank og tørlast. Her vil det være nødvendigt med en flåde af skibe, hvor mange er specielt indrettet til den type last der skal fragtes, det være sig olie eller kul mm. Der kan også være hensyn der skal tages til lovgivning i byområder og mere eller mindre god infrastruktur. Alle disse specielle hensyn betyder, at nogle kunder udelukkende kan serviceres af bestemte køretøjer, hvilket nødvendiggør, at formuleringen omskrives til at omfatte køretøjer af forskellig karakter. Mange af disse omskrivninger kan umiddelbart tilskrives følgende:

- Opdelingen i faste og variable omkostninger
- Forskellige omkostninger for hvert køretøj
- Begrænset antal køretøjer af hver type
- Kunders præferencer for bestemte køretøjer (prioritetshensyn)

Vi henviser til Min Wen (2010) for den matematiske formulering af disse hensyn.

Periodisk Vehicle Routing Problem

Indenfor mange brancher vil der i forbindelse med et produkt være regelmæssige eftersyn med hensyn på vedligeholdelse, hvorfor kunderne skal besøges med en vis frekvens. Alt efter kundens og produktets karakter kan dette ske med forskellige intervaller, hvilket betyder, set over en periode, at ruterne

varierer på dag-til-dag basis. Denne type problemer kaldes PVRP, hvor målet er at minimere de totale transportomkostninger over hele perioden, således at kundernes efterspørgsel og frekvenskrav tilfredsstilles. Typisk vil dette involvere opstillingen af en 4-indeks model, der, i forhold til de i dette afsnit indførte 3-indeks modeller, yderligere tilføjer et indeks over den ønskede tidsenhed.

For en dybere forståelse af problemet og den matematiske formulering, henviser vi til Min Wen (2010) og S. Coene, A. Arnout og F. C. R. Spieksma (2008).

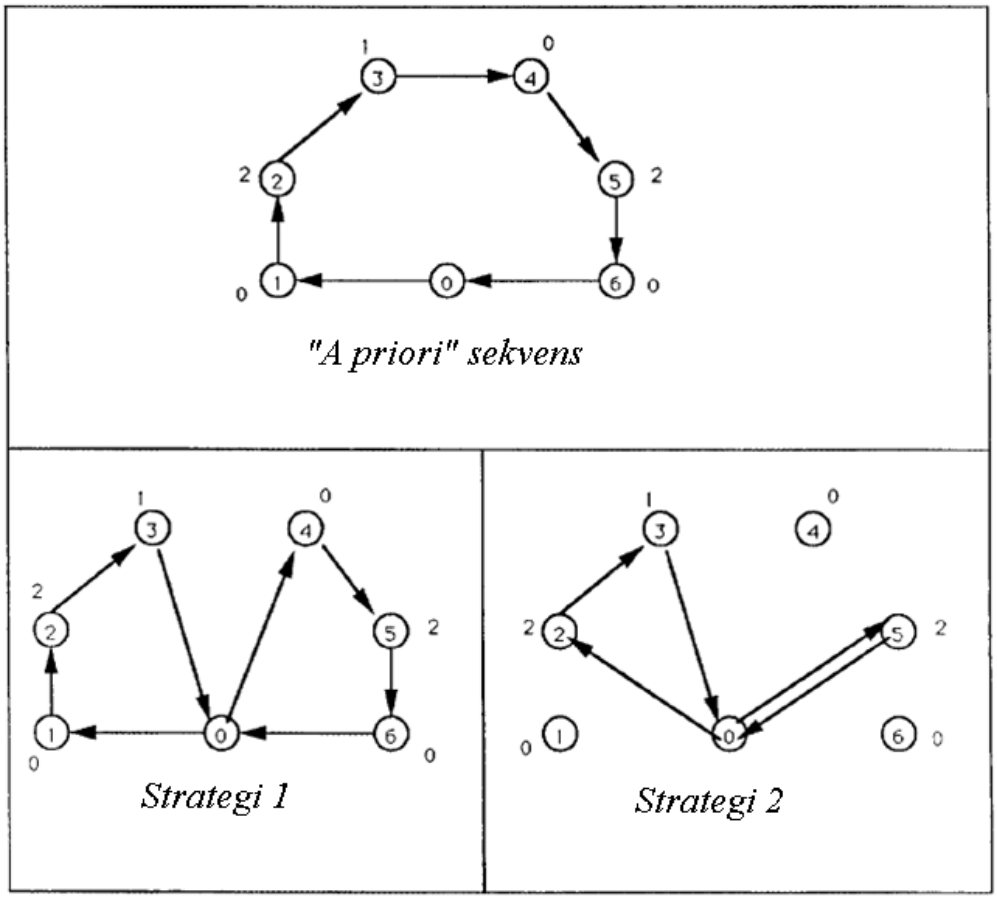
VRP med stokastisk efterspørgsel

En anden vigtig udvidelse, VRPSD, har at gøre med stokastisk efterspørgsel. Vi har indtil videre haft fuld information vedrørende kunders efterspørgsel (deterministisk efterspørgsel), men for mange virksomheder er efterspørgslen ukendt på det tidspunkt ruten optimeres. Typisk vil man dog kunne antage en sandsynlighedsfordeling.

En indlysende løsning vil være simpelthen at optimere ruten efterhånden som efterspørgslen bliver kendt. Men som D. J. Bertsimas (1991) skriver i deres artikel, er der adskillige ulemper ved denne metode: Typisk optimeres ruterne ikke dagligt, hvorfor det vil kræve ekstra ressourcer ved f.eks. daglig optimering. Desuden kan der være andre hensyn at tage, f.eks. til personlig og regelmæssig service. Og for det tredje er det ikke sikkert, at efterspørgslen kendes før kunden rent faktisk besøges.

Bertsimas foreslår derfor, at man fastlægger en *"a priori"* sekvens af kunder, som dermed besøges i denne rækkefølge uanset efterspørgslen. Afhængig af om information fra kunderne om deres behov bliver tilgængelig, kan to strategier nu anvendes: 1) En strategi hvor efterspørgslen ikke kendes før kunden rent faktisk besøges, hvorfor kunderne, der skal serviceres den pågældende dag, udvælges og besøges i den *"a priori"* fastlagte rute. 2) En strategi lig den i 1), men hvor efterspørgslen af en eller anden grund er blevet kendt forud for rutens begyndelse, hvorfor de kunder der ikke kræver service kan skippes fra den *"a priori"* fastlagte rute. Tilbage mangles nu kun en måde at fastlægge denne *"a priori"* sekvens af kunder på, for således at have en klar fremgangsmåde.

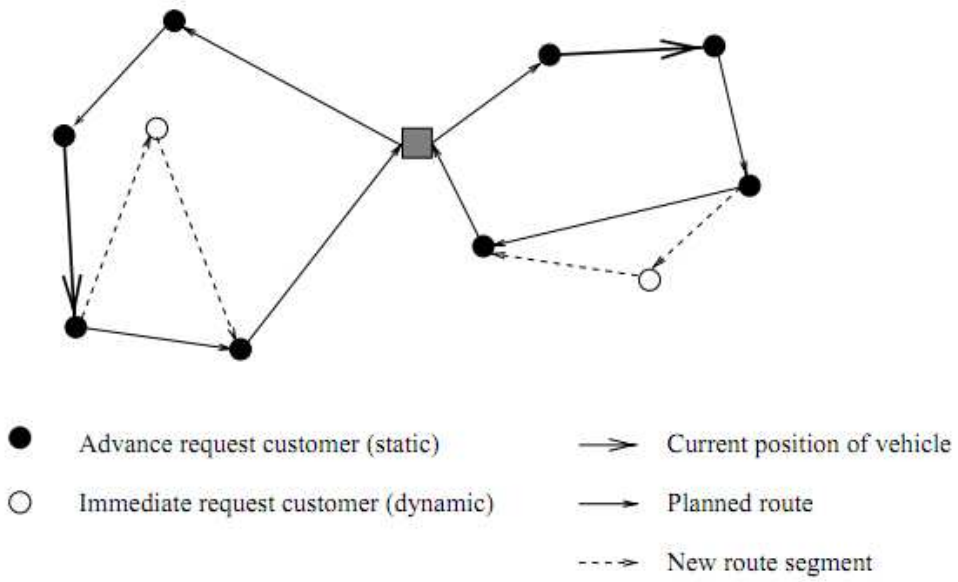
Såfremt der ønskes en dybere forståelse af problemet, henviser vi til Dimistris J. Bertsimas (1991).



Figur 6: Ovenstående illustrerer hhv. strategi 1 og 2 i forhold til en "a priori" fastlagt sekvens af kunder. Bertsimas (1991).

Dynamisk Vehicle Routing Problem

Meget lig VRPSD har vi i DVRP at gøre med usikkerhed i tidspunktet for ruteplanlægningen, men her angående identiteten (og placeringen) af kunderne. Denne type af problemer ses blandt andet ved udrykning af den ene eller anden art (politi, ambulance), eller i andre situationer, hvor en rute i forvejen er fastlagt og der derefter ankommer yderligere anmodninger fra andre kunder, som nu skal passes ind med de eksisterende (se nedenstående figur).



Figur 7: Allan Larsen (2000)

Her skal der tages højde for både den nuværende position af køretøjet, placeringen af kunden samt evt. tidsvinduer. Det er ikke sikkert der findes en mulig løsning, hvorfor kunden må serviceres en efterfølgende dag eller henvises til en anden distributør/omdeler.

Allan Larsen (2000) opstiller en række punkter, hvor de dynamiske problemer adskiller sig fra de statiske. Herunder nævnes kort de vigtigste:

- Hvor en tidsdimension oftest ikke er kritisk i det statiske tilfælde, er det nødvendigt i det dynamiske at det vides, hvor et bestemt køretøj er på et bestemt tidspunkt i forhold til en evt. kunde der skal passes ind i ruten, ikke mindst hvis der er tidsvinduer involveret.
- Problemet følges oftest med en række ukendte faktorer, hvor kun en sandsynlighedsfordeling er approksimeret, såsom stokastisk efterspørgsel.
- Da der sker løbende opdateringer, skal der være opdateringsmekanismer, som integrerer ny information i en mulig løsning.
- Ankomsten af nye kunder sætter også sine krav til beregningstid, idet der oftest ikke er tid til at vente timer på en mulig rute, optimal eller ej. Dette fokus på hurtige løsningsmetoder leder oftest til diverse forbedringsheuristikker (mere herom senere).
- Objektfunktionen er oftest i statiske problemer en minimering af de totale omkostninger, men idet mange parametre er ukendte, kan det være nødvendigt kun at medtage allerede kendte parametre eller anden tilpasning.
- Pga. det dynamiske aspekt vil det være sværere at passe kunder ind i en rute, hvorfor tidsvinduer oftest er bløde for at få en mulig løsning. Desuden er det klart at foretrække en evt. straf i forbindelse med overtrædelse af vinduet end helt at miste kunden.

Såfremt der ønskes en dybere forståelse af problemet, henviser vi til Allan Larsen (2000).

Multi-depot Vehicle Routing Problem

En vigtig og praktisk relevant generalisering af VRP er det såkaldte MDVRP. Givet adskillige depoter og en mængde kunder der skal servicers fra ét af disse, er problemet nu 1), at tildele hver kunde et depot og 2), at bestemme et sæt optimale ruter der minimerer de samlede transportomkostninger. Oftest er løsningsmetoder til denne type problemer netop baseret på en indledende cluster-fase, der grupperer kunder efter hvilket depot de skal serviceres af, efterfulgt af en konstruktionsfase der danner ruterne.

I et standard MDVRP er adskillige køretøjer placeret på adskillige depoter og køretøjerne returnerer til deres oprindelige depot efter at have besøgt deres tildelte kunder. Denne udvidelse kan dog benyttes i forbindelse med ethvert type VRP. Vi vil derfor heller ikke placere problemet i en specifik branche, da de kan dukke op i næsten alle typer problemer.

Kapitel 3 – Eksakte algoritmer

Inden vi går videre til de heuristiske algoritmer, følger her en kort redegørelse for state-of-the-art eksakte metoder, der, modsat de heuristiske, søger en optimal løsning. Disse kan til en vis grad benyttes i praksis, om end kun til løsning af problemer i mindre målestok. Af formuleringerne i kapitel 2 fremgår det klart, at hvad vi har med at gøre er et lineært heltalsproblem, mere specifikt med binære beslutningsvariable. Denne type problemer er som beskrevet i forrige kapitel oftest NP-hårde, hvorfor beregningstiderne på eksakte algoritmer hurtigt kan løbe op i ekstremer. Derfor søges det at skære kraftigt af på mulighedsområdet, for at spare både tid og maskinkraft.

Laporte (1991) foreslår en opdeling af de eksakte algoritmer i følgende typer:

- Søgetræsalgoritmer
- Dynamisk programmering
- Lineær heltalsprogrammering

Vi vil i det følgende gennemgå nogle af de mest fremtrædende eksakte algoritmer til løsning af VRP. Dynamisk programmering er dog et stort emne, som vi ikke vil beskæftige os med her.

3.1 Branch-and-Bound algoritmer

Af alle eksakte metoder er det konsensus i litteraturen[3], at specielt "Branch-and-Bound"-baserede algoritmer kan levere gode, brugbare løsninger til ikke blot VRP, men til lineære heltalsproblemer generelt. Metoden er en søgetræsalgoritme, hvor der, ved at indføre øvre og nedre grænseværdier, forsøges at afskære dele af løsningsrummet, således at en total gennemsøgning ikke er nødvendig.

Lad os betragte et minimeringsproblem. Algoritmen "brancher", eller forgrener sig, ved at dele et problem op i 2 eller flere subproblemer på følgende vis: Ved hvert trin i algoritmen vælges et problem S_i fra mængden af subproblemer S (initialt er kun det oprindelige problem indeholdt i S). Lad nu η_i være en nedre grænseværdi for S_i[4]; en almindelig måde at vælge sådan en grænseværdi på, er ved at løse en LP relaksering af det oprindelige problem. Når der løses en relaksering af det oprindelige problem, vil vi få en løsning der højst sandsynlig ikke er optimal, hvorfor dette bliver en naturlig ny øvre grænseværdi φ^* for fremtidige mulige løsninger, idet dårligere løsninger kan sorteres fra. φ^* kan i starten sættes til ∞ medmindre andet er oplyst. (1) Derfor, hvis $\eta_i > \varphi^*$, forkastes løsningen og grenen afskæres fra søgetræet. (2) Hvis der genereres en heltalsløsning, hvor $\eta_i < \varphi^*$, sættes $\varphi^* = \eta_i$. Såfremt $\exists S_i \in S$ for hvilke (1) gælder, kan disse trygt fjernes fra S. Såfremt S_i stadig er indeholdt i S,

[3] Se f.eks. P. Toth, D. Vigo (2002): "Branch-and-bound algorithms for the capacitated VRP", The Vehicle Routing Problem, chapter 2.

[4] For at specificere, der er tale om en nedre grænseværdi på objektfunktionen.

"branches" problemet til nye subproblemer der føjes til mængden S. Nu vælges et nyt S_i fra S og proceduren startes forfra. Algoritmen stopper når samtlige subproblemer er behandlet og kun ét element eksisterer i mængden af kandidatløsninger S, dvs. $|S| = 1$.

På denne vis kan vi nøjes med at benytte nogle af bibetingelserne i det oprindelige problem og tilføje resten om nødvendigt. Denne relaksering er den generelle tilgang indenfor branch-and-bound, men hvorledes en nedre grænseværdi bestemmes eller hvilke relakseringer der indføres, findes der mange forslag til, alt efter hvilken type formulering der tages udgangspunkt i. Lad os se på et par eksempler på LP relaksering.

3.1.1 Transportproblem

Den første og mest simple relaksering af CVRP er helt at se bort fra CCC-betingelserne (se formel (2.7)). Dermed transformeres problemet til et typisk transportproblem, som minimerer transportomkostningerne under betingelse af, at hver kunde besøges én gang og depotet besøges K gange. Der er dog en væsentlig risiko for, at løsningen ikke er mulig, idet vi her helt ser bort fra køretøjernes kapacitet og ikke mindst forbundethedskriteriet, som dermed muliggør subtours i løsningen.

3.1.2 Assignmentproblem

Det oprindelige CVRP kan også formuleres som et *m*-TSP, et TSP hvor en node besøges *m* gange, f.eks. depotet, som foreslået af Lenstra og Rinnooy Kan (1975). Herfra kan CCC'erne igen relakseres, hvorved problemet kan behandles som et assignmentproblem. Men dette vil have samme mangler som ovenstående transportproblem.

3.1.3 Lagrange relaksering

Lagrange relaksering er meget anvendt i heltalsproblemer. Her vælges den eller de bibetingelser ud som komplicerer problemet, flyttes op i objektfunktionen og ganges med en lagrangemultiplikator. Dermed vil bibetingelsen ikke længere være nær så streng, idet den blot tilføjer en straf til objektfunktionens værdi, såfremt den ikke opfyldes. Denne metode benyttes også for mange af udvidelserne til det oprindelige VRP, som f.eks. tidsvinduer.

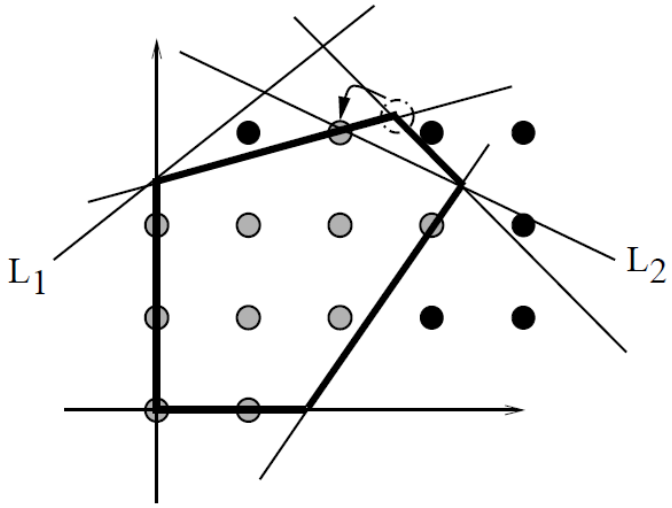
3.2 Branch-and-Cut algoritmer

En anden ofte anvendt metode er branch-and-cut, som er en variant af branch-and-bound og ligeledes en søgetræsalgoritme. Ved at basere metoden på en såkaldt "cutting plane"-algoritme, formår metoden at klare langt større problemer end branch-and-bound.

Lad P være mængden af CCC-betingelser til problemet, udvælg en delmængde $\hat{P}$ til at indgå i problemet og løs relakseringen. Er løsningen mulig i forhold til det oprindelige problem, er det også en optimal løsning. Såfremt én eller flere bibetingelser i P overtrædes (også kaldet "cutting planes"), tilføjes de til $\hat{P}$ og processen gentages. Ved at generere "cutting planes" til det oprindelige problem (og

evt. ved hver node i søgetræet), er det muligt at forbedre de nedre grænseværdier og dermed hurtigere nå frem til en løsning.

Et eksempel ses i nedenstående figur, hvor mulighedsområdet er indkranset med tykke linjer, de grå punkter er heltallige løsninger og den optimale LP-løsning er indkranset med en punkteret linje. Linjerne L_1 og L_2 repræsenterer begge gyldige uligheder[5]. Dette ses klart, idet ingen af dem udelukker nogle heltallige løsninger. L_2 er dog ikke blot gyldig, den er også en "cutting plane", idet den skærer en del af mulighedsområdet (og dermed søgetræet), og lige i dette tilfælde ydermere resulterer i en ny optimal løsning til problemet.



Figur 8: Jennifer L. Rich (1999)

3.3 Kolonnegenerering

Kolonnegenerering kan bruges til at løse alle typer LP-problemer, men bruges oftest hvor antallet af variable er ekstremt højt. Lad Ω være mængden af samtlige variable til det oprindelige problem. Der vælges i første trin af algoritmen kun en delmængde $\overline{\Omega} \subset \Omega$ af variablene i problemet (de resterende antages at have værdien 0), mens det sikres at der findes en optimal løsning til dette relakserede problem. Lad følgende være det relakserede problem, også kaldet "master"-problemet:

(3.1) $\min\left(c^T x\right)$

Ubb.

(3.2) $Ax \leq b\,,\ x \geq 0$

[5] En gyldig ulighed er en ulighed, der skærer dele af mulighedsområdet væk, som ikke indeholder mulige løsninger.

Andet trin består nu af at løse masterproblemet samt det duale problem. Lad $\overline{x}$ og $\overline{y}$ betegne hhv. løsningen til masterproblemet og det duale problem.

Såfremt en kolonne i, der ikke er med i masterproblemet, skal tilføjes, skal den have negative reducerede omkostninger[6], dvs. $c_i - A_i^T \overline{y} < 0$, hvor A_i^T er den i'te kolonne i A. På denne måde kan samtlige $\Omega \setminus \overline{\Omega}$ tjekkes, hvorvidt tilføjelsen af yderligere variable kan forbedre løsningen. Oftest tilføjes den kolonne med den mest negative reducerede omkostning, hvorfor vi i tredje trin kan opstille følgende subproblem, også kaldet "pricing"-problemet:

(3.3) $$\min\left(\left\{c_i - A_i^T \overline{y} : \forall i \in \Omega \setminus \overline{\Omega}\right\}\right)$$

Heraf returneres den mindste reducerede omkostning. Såfremt denne er negativ, tilføjes den pågældende kolonne, hvorefter vi fortsætter fra andet trin. Hvis samtlige mulige kolonnetilføjelser har ikke-negative reducerede omkostninger, vil vi ikke blot have en optimal løsning til det relakserede problem, men også til det oprindelige problem, og algoritmen standses.

For kort at opsummere består algoritmen af 3 trin:

1) Der udvælges en delmængde $\overline{\Omega} \subset \Omega$ af det oprindelige antal variable.
2) For $\overline{\Omega}$ løses et masterproblem samt dens duale problem.
3) Et subproblem løses; såfremt løsningen er ikke-negativ har vi en optimal løsning, ellers tilføjes den pågældende kolonne $\overline{\Omega}$ og der fortsættes fra 2).

På denne måde starter vi ud med et beskedent antal variable og yderligere variable tilføjes dynamisk ved hjælp af de reducerede omkostninger, men kun i det omfang de er nødvendige. Det kan bemærkes, at subproblemet oftest i VRP-sammenhænge er et SPP (*shortest path problem*).

Som nævnt i starten er kolonnegenerering rettet mod LP-problemer og ikke heltalsproblemer, hvorfor resultatet af kolonnegenerering oftest er en ikke-heltallig affære. Derfor kan kolonnegenerering ikke stå alene i forhold til VRP.

3.4 Branch-and-Price algoritmer

Branch-and-price er én af de algoritmer, der kan bruges i samarbejde med kolonnegenerering for at løse heltalsproblemer. Branch-and-price er meget lig branch-and-cut algoritmen, forskellen er at vi i stedet for rækkegenerering (tilføjelse af bibetingelser), sætter lid til kolonnegenerering.

Algoritmen er som følger: Problemet relakseres og løses ved kolonnegenerering; såfremt der eksisterer profitable reducerede omkostninger, tilføjes de pågældende variable til problemet og det løses igen. Når der ikke længere findes profitable reducerede omkostninger for nogen variable har vi en optimal løsning

[6] Bemærk, hvorvidt de reducerede omkostninger skal være positive eller negative er et definitionsspørgsmål. Der kan med fordel tænkes, at gevinsten ved at tilføje variablen til problemet skal være positiv.

til det relakserede problem. Hvis vi endnu ikke har en heltallig løsning, branches der på beslutningsvariablene og algoritmen starter forfra med at løse subproblemerne ved kolonnegenerering. Kolonnegenerering sker således ved hver node i søgetræet.

Kapitel 4 – Beregningskompleksitet

Beregningskompleksiteten af en algoritme er et redskab til at bestemme, hvor lang tid det teoretisk vil tage at løse et problem. Med dette redskab har man påvist, at beregningstiden for specielt eksakte algoritmer vokser ekstremt, når antal kunder stiger.

4.1 Beregningstid

Den tid, en algoritme bruger om at løse et problem af en given størrelse, afhænger af, hvilket problem man har med at gøre. Derfor er det ikke muligt at angive en fast sammenhæng mellem beregningstiden og størrelsen af en karakteristisk parameter n (som f.eks. antal kunder er for VRP). Det man kan forsøge, er at angive en værst mulig beregningstid som øvre grænse for tiden det tager at løse problemet.

Det er dog svært at afgøre den faktiske beregningstid, idet den bl.a. er afhængig af hvilken type computer der anvendes samt hvilket program algoritmen er implementeret i. En hurtig computer vil naturligvis løse et problem hurtigere end en langsom, af den grund er det ikke bekvemt at angive beregningstiden i faste tidsenheder, som sekunder eller minutter. Her vil en mere fornuftig løsning være at angive, hvordan beregningstiden udvikler sig som funktion af n for en given algoritme. Et eksempel kan være følgende; øges beregningstiden for en algoritme med en faktor 3, hvis størrelsen af problemet øges med en faktor 1, vil dette gælde uanset hvilken hastighed computeren har.

Beregningskompleksiteten kan defineres således:

$f(n)$ er en algoritmes værst mulige beregningstid for et problemtilfælde af størrelsen n. Hvis der findes en funktion $g(n): N \to \Re$ og en konstant $c > 0$, således at

(4.1) $$f(n) \leq c \cdot g(n) \quad , \forall n \geq 1,$$

siger vi, at algoritmens beregningskompleksitet er af størrelsesorden $g(n)$, eller mere kompakt, at kompleksiteten er $O(g(n))$.

For at forklare ovenstående definition, tages udgangspunkt i et VRP. Problemet løses med en algoritme, der har en beregningstid som funktion af n, $f(n)$[7]. På dette tidspunkt vil man gerne afgøre, om det pågældende problem har en polynomial beregningstid eller ikke. Hvis funktionen $f(n)$ er mindre end et polynomium $g(n)$ gange en konstant, er $f(n)$ et polynomium og dermed et håndterligt problem. Hvad angår konstanten c, kan denne f.eks. være et mål for en computers hastighed.

[7] $f(n)$ kan være en hvilken som helst funktion, enten er det et polynomium eller noget andet. Hvis der er tale om et polynomium siges en algoritme at være polynomial.

Definitionen er rettet mod eksakte algoritmer, hvor det store spørgsmål er, hvor lang tid de er om at finde den optimale løsning. Dette spørgsmål kan dog ikke stilles mht. heuristikker, idet løsningen ikke nødvendigvis er optimal. Det man kan undersøge i forbindelse med heuristikker, er, hvor længe en given heuristik er om at behandle et problem af en bestemt størrelse, hvilket foregår på samme måde som de eksakte algoritmer.

Skal man skelne mellem "gode" og "dårlige" algoritmer, kan beregningskompleksiteten anvendes. Hvis et givent problem kan løses med flere forskellige algoritmer, kan en kompleksitetsanalyse hjælpe en med at vurdere hvilken af dem er hurtigst.

På baggrund af beregningskompleksiteten kan man inddele algoritmerne i to grupper, de polynomielle og de ikke-polynomielle. De to typer refererer til, hvilken størrelsesorden algoritmens beregningskompleksitet har, dvs. om $g(n)$ er et polynomium eller f.eks. en eksponentialfunktion. Størrelsesordenen er en betydningsfuld faktor, da der er væsentlig forskel på beregningstiden for en algoritme med polynomiel og eksponentiel kompleksitet.

En lille ulempe med inddelingen er, at den er for simpel, fordi den bl.a. ikke tager højde for, at polynomierne kan være af højere grad eller at eksponentialfunktionerne kan have små eller store koefficienter. Ydermere tages der heller ikke højde for konstanten c i ovenstående ligning. I nogle tilfælde er det muligt, at en eksponentiel algoritme er hurtigere end en polynomiel. Inddelingen er dog i bund og grund fornuftig nok, da en eksponentialfunktion på et tidspunkt vil opnå en større funktionsværdi end ethvert polynomium, for et tilstrækkelig stort n.

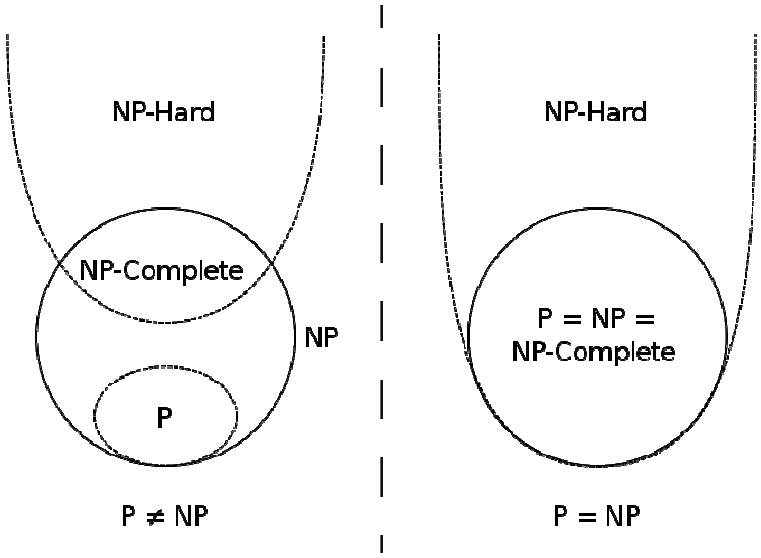
4.2 Klassificering af problemer

De kombinatoriske problemer kan inddeles efter hvilken kompleksitet den bedste eksakte algoritme har. Man kan igen benytte beregningskompleksiteten til at skelne mellem lette og svære problemer. De lette (håndterlige) problemer løses med en algoritme med polynomiel kompleksitet og tilhører klassen P. Derimod tilhører de svære problemer klassen NP (ikke-deterministiske polynomielle problemer)[8]. Det er klart at $\mathrm{P} \subseteq \mathrm{NP}$ og betyder, at ethvert problem som kan løses i polynomieltid, naturligvis også kan løses i eksponentiel tid.

Det grundlæggende VRP tilhører NP-klassen og kaldes i teorien for værende NP-hård. Et problem kaldes NP-hård, hvis det er mindst lige så svært som det sværeste problem i NP. Ved dette forstås et matematisk problem, hvor det ikke vides om der findes nogen hurtig løsningsmetode i praksis. Mere specifikt tilhører VRP en bestemt klasse af NP-hårde problemer der kaldes NP-komplet[9].

[8] Der er en del problemer som vi ikke kan vise er polynomiale, hvorfor NP-klassen er indført. Hvis man har løsningen for et komplekst problem med en eksponential kompleksitet, kan man verificere løsningen i polynomieltid.

[9] NP-komplet problemer er dem der indbyrdes kan omkodes til hinanden, polynomialt. Dvs., hvis et problem polynomielt omkodes til et andet, og det andet på tilsvarende vis kan omkodes til de forgående, har man to sammenhængende problemer. Men om denne sammenhæng er polynomial, vides endnu ikke.



Figur 9: Her ses to diagrammer der viser P, NP, NP-komplet og NP-hard mængderne og deres sammenhæng. Kilde: http://en.wikipedia.org/wiki/NP-complete.

Selvom NP-komplette problemer er en delmængde af NP-klassen, kendes endnu ingen polynomialtidsalgortime til løsning af disse. Det vides, at problemerne findes i NP, men om de også er i P, er stadig et åbent spørgsmål, som matematikere indtil videre forgæves har forsøgt at påvise. Hvis det lykkes at finde én polynomielalgoritme til blot ét NP-komplet problem, så vil det være muligt at løse alle NP-problemer i polynomieltid og $P = \mathrm{NP}$.

For at lette beregningskompleksiteten benytter man i stedet søgebaserede algoritmer – heuristikker – som er hvad næste kapitel omhandler.

Kapitel 5 – Klassisk heuristik

5.1 Klassisk heuristik og metaheuristik

En heuristik er kort fortalt en approksimationsalgoritme, som sigter mod at finde en god, mulig løsning (ikke nødvendigvis optimal) på kort tid. Man kategoriserer dem i to hovedklasser: Klassisk heuristik som hovedsageligt blev udviklet i perioden 1960-1990 og metaheuristik fra 1990[10], hvoraf det er førstnævnte klasse dette kapitel omhandler.

VRP tilhører de NP-hårde problemer, hvilket uheldigvis i praksis medfører at beregningstiden overstiger hvad man normalt vil mene er acceptabelt. Selvfølgelig spiller den teknologiske udvikling en betydelig rolle i løsningen af store VRP, men som det ser ud i dag, har vi brug for løsningsmetoder som tager væsentlig mindre tid. Hurtigere computere og bedre programmer til rådighed er ikke nok til at gøre eksakte algoritmer værd at foretrække for store problemer. Derfor må man ty til andre metoder som kan give næsten optimale eller endda optimale løsninger. Heuristiske løsningsmetoder løser VRP i polynomiel beregningstid, og kan derfor give løsninger inden for kort eller acceptabel beregningstid, men samtidig også af høj kvalitet (dog ikke nødvendigvis en optimal løsning). En heuristik kan betragtes som en søgeprocedure, som iterativt genererer og evaluerer løsningskandidater. Hvorvidt den reducerede nøjagtighed i disse metoder er acceptabel for at få en hurtigere behandling, afhænger naturligvis helt af, hvilket problem der skal løses og hvor lang tid der er til rådighed. Er der tale om et problem der skal løses og bruges i en daglig opdatering, kan det ikke blot være bekvemt, men en nødvendighed.

I langt de fleste tilfælde måler man heuristikker på kriterier som nøjagtighed og hastighed. Derudover er enkelthed og fleksibilitet også egenskaber som har stor betydning i praktiske situationer. At algoritmerne er gennemskuelige har betydning ikke mindst for ledelse/projektansvarlige, som ikke nødvendigvis er IT-uddannede. Hvis det ikke er gennemskueligt, er det nemt at lave fejl og svært at følge op på nye udfordringer. Fleksibiliteten giver helt klart også en vis overskuelighed, når det drejer sig om at benytte forholdsvis få antal løsningsmetoder, der fungerer på tværs af adskillige problemstillinger. Vælger man at arbejde med mange forskellige specialiserede metoder, vil det hurtigt blive meget uoverskueligt. Det er vigtigt for projektansvarlige at vide præcis hvad der foregår i sit team, hvilke parametre data er udtrukket på baggrund af og hvordan metoderne er implementeret, specielt hvis det skal sælges eller præsenteres for andre.

Det er således klart, at antallet af fordele ved at benytte heuristikker til at løse store problemer klart opvejer den tabte præcision. I det følgende vil vi gennemgå den klassiske heuristik, mens vi vender tilbage til metaheuristik i detaljer i kapitel 6.

[10] Toth og Vigo (2002) – The Vehicle Routing problem.

5.2 Klassisk heuristik

Algoritmerne i den klassiske heuristik undersøger et relativt begrænset område af mulighedsområdet, hvilket betyder, at undersøgelsens fokus rettes mod de steder, hvor man forventer at finde den potentielle optimale løsning. Dette kan virker lidt besynderligt, men algoritmerne er opstillet således, fordi en lav beregningstid er i højsædet og vi derfor vil undgå en total gennemgang af mulighedsområdet. En fordel ved disse algoritmer er, at de let kan udvides til at håndtere de mange specielle træk tilknyttet VRP-problemstillinger i praksis. Fleksibiliteten af algoritmerne gør dem populære og derfor stadig meget anvendte.

De klassiske heuristikker kan inddeles i tre kategorier: Konstruktiv heuristik, to-fase heuristik og forbedringsheuristik.

De konstruktive algoritmer er karakteriseret ved, at de gradvist opbygger en mulig løsning ved at vælge de omkostningsminimerende kanter. Algoritmerne har ingen indbygget forbedringsfase, hverken til slut eller undervejs i søgningen. To-fase heuristikker indeholder to faser; først dannes clusters af noder og dernæst konstrueres en rute for hver af disse. De to faser er logisk nok afhængige af hinanden og derfor er der mulighed for interaktion mellem dem. Man kan vælge at se problemet på to måder: Den første fremgangsmåde kaldes ”cluster-first, route-second” og den anden ”route-first, cluster-second”. I det første tilfælde organiseres noderne i mulige clusters, hvorefter der til hvert cluster konstrueres en rute. I modsætning hertil, dannes én rute med alle noder indeholdt, hvorefter den store rute opdeles i mindre mulige ruter. Hvorvidt den ene er bedre end den anden afhænger af problemet. Den sidste kategori, forbedringsheuristik, prøver at forbedre en allerede eksisterende mulig løsning ved at udveksle noder eller kanter indenfor eller mellem ruterne.

5.3 Heuristikker for CVRP

Den gængse litteratur beskriver adskillige heuristiske algoritmer for CVRP og derfor er der et stort udvalg. For at begrænse os til nogle få, har vi valgt at beskrive de mest anvendte algoritmer indenfor de 3 kategorier. Disse er Clarke and Wrights ”Savings”-algoritme, ”Sweep”-algoritmen samt en enkelt-rute og multi-rute forbedringsheuristik.

5.3.1 Clarke og Wrights ”savings”-algoritme

I det følgende beskrives Clarke og Wrights ”Savings”-algoritme (1964), som tilhører den konstruktive kategori. Denne algoritme er baseret på et besparelsesprincip og er nok den mest kendte heuristik for VRP. Metoden starter med n ruter, hver bestående af depotet og én kunde. Vi antager her et ubegrænset antal køretøjer, så denne initiale løsning er mulig, om end det næppe i praksis er således.. Herefter forsøger man at spare ved at sammensætte to små ruter $(0,...,i,0)$ og $(0,j,...,0)$ til en større $(0,...,i,j,...,0)$, såfremt dette er muligt, hvor i er den sidste kunde i en rute og j er den første i en anden rute. Besparelsen ved at servicere i og j lige efter hinanden defineres som følgende:

(5.1) $$s_{ij} = c_{i0} + c_{j0} - c_{ij}$$

Algoritmen er blevet benyttet til at løse både det symmetriske CVRP, som det oprindeligt var beregnet til, samt det asymmetriske, som foreslået af Vigo (1994). Metoden bliver dog væsentlig ringere når det bruges i sidstnævnte, selvom antallet af potentielle rutesammenslutninger bliver halveret.

Algoritmen findes i to grundskabeloner; den parallelle version og den sekventielle version. Principperne bag "Savings"-algoritmen er følgende:

Standard trin

1. Dan n enkel-kunde ruter $(0,i,0),\ \forall i = 1,...,n$.
2. Udregn besparelserne

   $$s_{ij} = c_{i0} + c_{j0} - c_{ij} \qquad \forall i, j = 1,...,n \text{ og } i \neq j$$

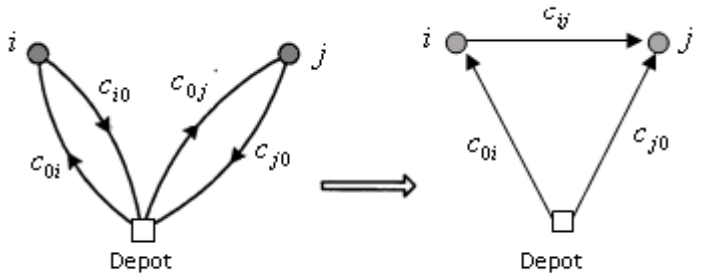
   hvor 0 svarer til depotet. Dette svarer til at man lader én vogn køre ruten $(0,i,j,0)$ i stedet for at lade en vogn køre $(0,i,0)$ og en anden $(0,j,0)$. (se princippet i figur 10)
3. Ordn besparelserne i en liste i aftagende rækkefølge.

Parallel version (bedst mulig sammenslutning)

4. Startende fra øverste ikke undersøgte s_{ij} på listen, udføres følgende:
   a. Givet besparelsen $s_{ij} > 0$, undersøg om der findes to ruter $(0,...,i,0)$ og $(0,j,...,0)$ som lovligt kan forbindes. Hvis dette er muligt, kombineres de to ruter ved at tilføje kanten (i,j), og sletter derimod $(0,j)$ og $(i,0)$ fra løsningen.
   b. Prøv næste s_{ij} på listen og gå tilbage til trin 4a indtil der ikke kan dannes flere kanter.

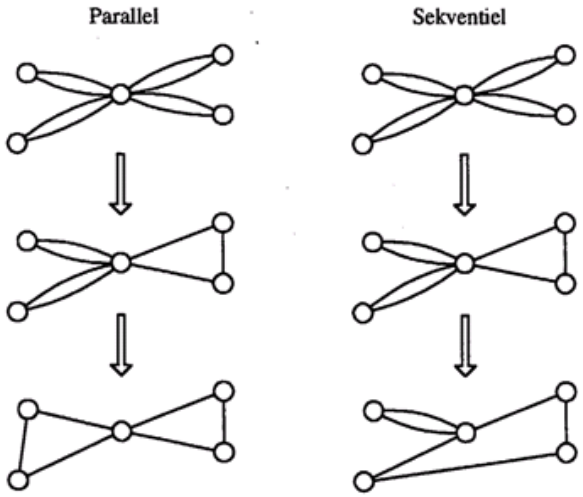
Sekventiel version (ruteudvidelse)

4. Betragt igen hver rute $(0,i,...,j,0)$
   a. Udregn i første omgang besparelserne s_{ki} eller s_{jl}, som lovligt kan bruges til at forbinde den nuværende rute med en anden rute $(0,k,0)$ eller $(0,l,0)$. Implementér forbindelsen og gentag denne operation, til den nuværende rute.
   b. Hvis der ikke findes flere sammenslutninger, betragt den næste rute og gentag 4a indtil der ikke er mulige rute sammenslutninger.



Figur 10: Princippet i Savingsalgoritmen

Den parallelle version bygger ruterne for de forskellige køretøjer op samtidig, hvorimod den sekventielle version afviger fra den parallelle ved, at den udvider ruten $(0,i,j,0)$ indtil den ikke længere kan udvides, før den begynder at lægge en rute for et nyt køretøj (se figur 11).



Figur 11: Figuren viser forskellen mellem de parallelle og den sekventielle version af "Savings" algoritmen. Backchi et al. (2001)

Ovenstående metode er den originale "savings"-algoritme fra 1964, om hvilken det er påvist af Toth, Christofides og Mingozzi (1979), at den parallelle version finder bedre løsninger end den sekventielle. Denne heuristik har en beregningskompleksitet på $O\left(n^2 \log(n)\right)$, som er mulig at reducere så længe der arbejdes med en passende datastruktur. Siden hen er der kommet mange modificerede definitioner af besparelserne, som primært har haft til formål at nedsætte beregningstiden og hukommelseskravene. Såfremt der ønskes en dybere forståelse af hukommelseskravene, henvises til Toth og Vigo (2002).

Clarke og Wrights algoritme ignorerer bl.a. køretøjets faste omkostninger og størrelsen på flåden. Man kan dog let tage højde for den faste omkostning f, ved at lægge denne konstant til hver c_{0j}

$(j = 1,...,n)$. Løsningen med et fast antal køretøjer opnås ved at gentage punkt 4 indtil det påkrævet antal ruter er nået, selv hvis besparelsen går hen og bliver negativ.

Algoritmen har været meget anvendt som grundlag i programmer til løsnings af CVRP, idet algoritmen let kan implementeres og har en forholdsvis kort beregningstid. Adskillige forfattere er kommet med modificerede versioner af "savings"–algoritmen, både som klar forbedring af den oprindelige algoritme for CVRP, men også som løsningsmetode for nogle af udvidelserne.

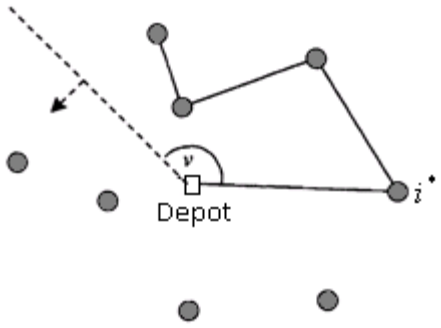
5.3.2 Gillett og Millers "sweep"-algoritme

Vi går nu videre til Gillett og Millers "sweep"-algoritme (1974), som er en to-fase algoritme, hvor man indledningsvis inddeler kunderne i nogle clusters, alle indeholdende depotet. Herefter løses et TSP for hver cluster. Algoritmen fungerer således:

Vi antager, at kunderne er placeret i planen med depotet i $(0,0)$. Lad hver kunde i's beliggenhed i planen være repræsenteret ved dennes polære koordinatsæt (θ_i, ρ_i) på baggrund af en vilkårlig referencekunde i^* med $\theta_{i^*} = 0$. Her angiver θ_i vinklen, som, når man bevæger sig i den retning fra depotet, kommer man til i. ρ_i angiver afstanden fra depotet til kunde i (bemærk, for depotet er $\rho_0 = 0$). For at gøre det lidt mere klart, skal vi have en referencekunde i^*, som angiver nulpunktet for vinklen θ_i, for dermed at kunne specificere kunde i's placering i planen Efterfølgende rangordner man alle kunderne i stigende orden efter vinkel, således at $\theta_1 \le ... \le \theta_i \le ... \le \theta_n$.

Fase 1

1. *(Ruteinitialisering).* Vælg et ubrugt køretøj k.
2. *(Rutekonstruktion).* Start ved den kunde i, som har den mindste θ_i og som endnu ikke er i ruteplanen. Inkludér derefter kunderne $i+1, i+2,...$ i ruten, indtil kapacitetsbegrænsningen (eller en anden begrænsning) for køretøj k er nået. Nedenstående figur illustrerer princippet.



Figur 12: Her ses princippet i "sweep"-algoritmen.

3. Hvis alle kunder er blevet "sweepet" eller alle køretøjer brugt, fortsættes til fase 2, ellers går man tilbage til trin 1.

Fase 2

4. *(Ruteoptimering).* Løs et TSP for hvert cluster af kunder tildelt de forskellige vogne. Enten løses problemet eksakt eller vha. heuristikker.

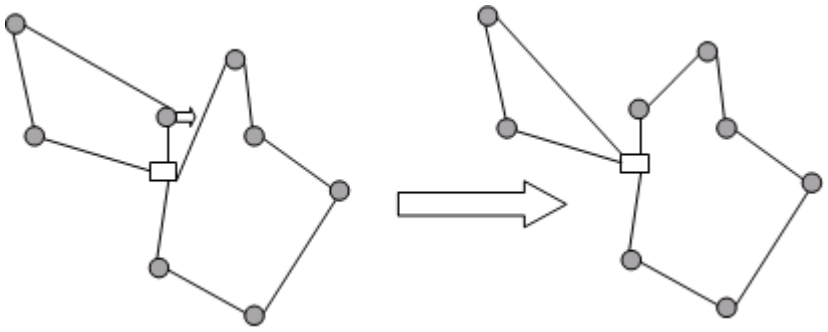
Det skal i øvrigt bemærkes, at forskellige valg af referencekunder i^* naturligvis medfører forskellige ruteplaner. Det er i nogle implementeringer også muligt at inkludere en efteroptimeringsfase, hvor noder udveksles mellem tilstødende clusters. At inkludere en sådan efteroptimeringsfase er helt klart et klogt valg, da man således forbedrer løsningen væsentligt og undgår en hel del dårlige træk.

Der er i litteraturen nævnt andre "sweeping"-algoritmer, som f.eks. Fisher og Jaikumar's algoritme. Forskellen fra ovenstående algoritme ligger i metoden, hvorpå clusters dannes. I stedet for at anvende en geometrisk metode til at forme clusters på (som ovenstående), vælger man i stedet at forme clusters ved at løse et generaliseret assignmentproblem. For nærmere detaljer se Toth og Vigo (2002).

5.3.3 λ-opt forbedringsalgoritme

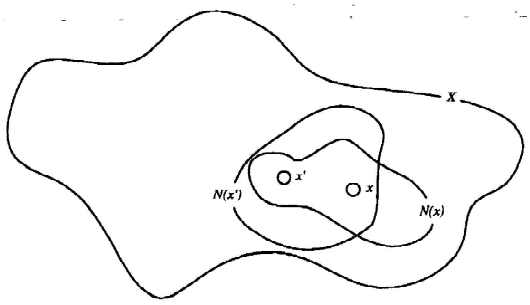
Den sidste kategori vi nu vil se på er en forbedringsheuristik. De fleste forbedringsalgortimer er såkaldte nabosøgningsalgoritmer (NS). I forbindelse med VRP starter metoden i en tilfældig brugbar løsning, hvorfra man søger efter forbedringer i løsningen ved f.eks. at bytte kunderne rundt til andre placeringer. Processen bliver ved indtil det ikke længere er muligt at forbedre den nuværende løsning. Vi starter ud med at introducere begreberne ”træk” og ”nabo-område”.

Hvis en mængde af mulige løsninger betegnes X, vil der til hver løsning $x \in X$ tilhøre en mængde $N(x) \subseteq X$, som kaldes nabo-området til x. Nabo-området kan defineres som alle de løsninger $x' \in X$, som kan nås fra x ved en operation kaldet et ”træk”, som laver en løsning om til en ny mulig løsning. I forbindelse med VRP betyder det, at en kunde bliver flyttet til et andet sted på ruten eller endda til en anden rute (se figur 13). Så på baggrund af disse træk kan man angive et nabo-område $N(x)$, som alle de løsninger x' dannet fra en forbedring af x.



Figur 13: Eksempel på et træk, hvor en kunde flyttes fra den ene rute til en anden.

Mange af disse træk er symmetriske, hvilket betyder at man kan komme fra x til x' og omvendt ved brug af samme træk ($x' \in N(x)$ og $x \in N(x')$). Nedenstående figur illustrerer situationen.



Figur 14: Nabo-området $N(x)$ til en løsning x, givet ved et symmetrisk træk. Backchi et al (2001).

Generel nabosøgning

Algoritmen gennemløber et antal iterationer, hvor der for hver iteration udføres et træk, som resulterer i en naboløsning $x' \in N(x)$. Hvordan man vælger naboløsningen afhænger helt af, hvilken algoritme man benytter. NS-algoritmens princip er som følger:

1. Start:
   Vælg en løsning[11] $x_{nu} \in X$. Sæt $x_{bedst} = x_{nu}$ og udregn den tilhørende $omkostning = c(x_{nu})$.
2. Find det bedste x' i nabo-området:
   Vælg en naboløsning $x_{nabo} \in N(x_{nu})$ ud fra et udvælgelseskriterium. Hvis det anvendte udvælgelseskriterium ikke kan opfyldes for nogen af løsningerne fra $N(x_{nu})$ eller et stopkriterium er opfyldt, standses søgningen og den endelige løsning x_{bedst} returneres.
3. Opdatering:
   Hvis der findes en naboløsning x_{nabo}, som opfylder det valgte udvælgelseskriterium sættes $x_{nu} = x_{nabo}$ og $omkostning = c(x_{nabo})$. Hvis $c(x_{nu}) < c(x_{bedst})$, sæt $x_{bedst} = x_{nu}$.
   Gentag proceduren fra punkt 2 indtil der ikke er flere forbedringer.

Ofte benyttes "Descent"-metoden, som er et af de simpleste udvælgelseskriterier. Ovenstående punkt 2 kan dermed beskrives således:

2. Find det bedste x' i nabo-området ved brug af "descent"-metoden:
   Vælg den naboløsning $x_{nabo} \in N(x_{nu})$, som giver den mindste omkostning $c(x_{nabo})$.
   Søgningen stoppes, hvis der ikke findes en nabo som opfylder $c(x_{nabo}) < c(x_{nu})$.

Formålet med denne metode er at udføre det træk i hver iteration, som giver den største forbedring mht. objektfunktionen. Algoritmen kendes også ved navnet "greedy"-algoritmen, som søger efter et globalt optimum i hver iteration.

Nabosøgning og VRP

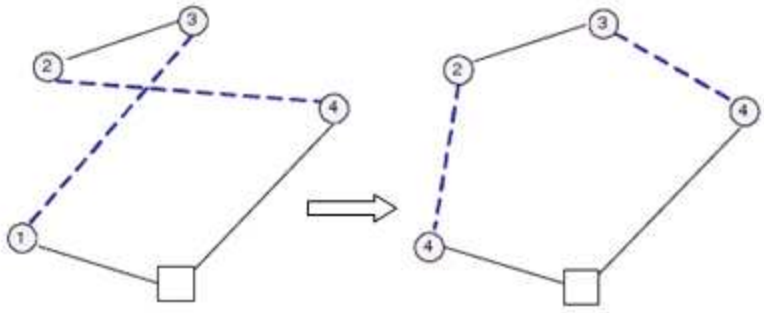
Forbedringsalgoritmen NS for VRP problemer opererer enten på hver enkelt rute separat eller på adskillige ruter samtidig. De heuristikker der anvendes i det første tilfælde, adskiller sig ikke fra

[11] Denne løsning kan f.eks. være udregnet på baggrund af konstruktionsalgoritmen.

forbedringsalgoritmer til et TSP. I det andet tilfælde udnytter heuristikkerne en multi-rute struktur for VRP.

Enkeltrute forbedring

Lin's $\lambda - opt$ -mekanisme er en klassisk NS-algoritme til TSP. Algoritmen går ud på at fjerne λ kanter fra en rute og derefter sætte de λ brudstykker sammen på andre mulige måder. Et typisk eksempel er følgende 2-opt: Erstat to forbindelser i ruten med to andre forbindelser på en sådan måde, at den nye rutelængde bliver kortere. Hvis der findes en profitabel kombination af brudstykkerne, evt. den første eller bedste kombination, implementeres denne. Proceduren gentages indtil der ikke er flere mulige forbedringer og dermed et lokalt optimum er nået. Et eksempel på et sådant træk kan ses herunder.



Figur 15: Et træk i en 2-opt algoritme. Kant (1,3) og (2,4) erstattes af kant (1,2) og (3,4). Min Wen (2010).

λ -opt-algoritmen er baseret på begrebet λ -optimalitet:

"En rute siges at være λ -optimal, hvis det ikke er muligt at opnå en kortere (billigere) rute ved at erstatte λ kanter med λ andre kanter", Keld Helsgaun (1998).

Fra denne definition er det oplagt, at enhver λ -optimal rute også er λ' -optimal for $1 \le \lambda' \le \lambda$, idet en λ' -opt er et specialtilfælde af en λ -opt, og de træk der foretages i en mindre opt implicit indgår i en større opt. Det er også let at se, at en rute som består af n kunder er optimal, hvis og kun hvis den er n -optimal.

Generelt kan det siges, jo større λ , des mere sandsynligt er det at få en optimal rute. Desværre vil antallet af operationer, der skal til for at teste alle λ -udvekslinger, stige hurtigt når antallet af kunder stiger. Normalt vil en test for λ -optimalitet for et træk have en beregningskompleksitet $O\left(n^{\lambda}\right)$. Der er ingen øvre grænse for antallet af λ -udvekslinger man kan foretage, men pga. beregningstiden er det typisk værdierne 2, 3 eller 4 der anvendes.

Det er noget af en ulempe, at λ skal specificeres på forhånd. Det er svært at vide hvilken λ -værdi der skal bruges for at opnå det bedste kompromis mellem beregningstid og kvaliteten af løsningen. Lin og Kernighan (1973) er kommet uden om denne ulempe ved at udvikle en algoritme, hvor værdien af λ

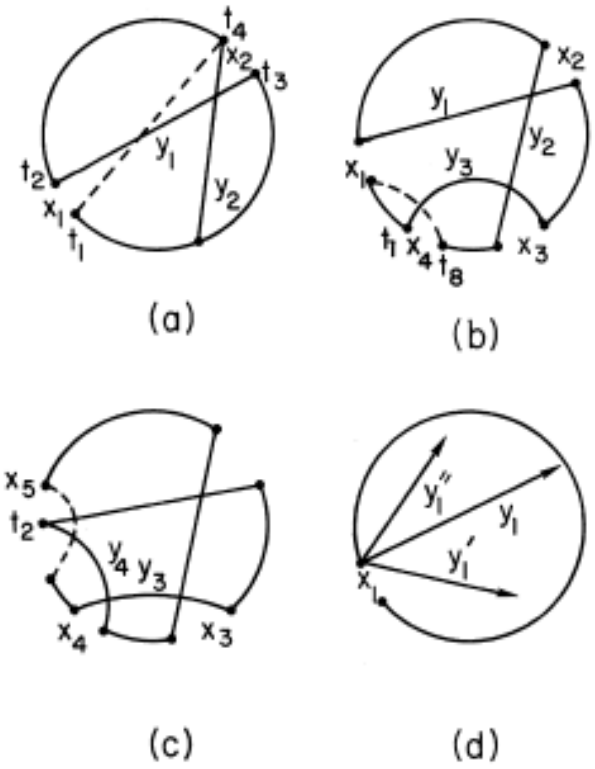
varieres dynamisk gennem iterationerne i algoritmen. På hver iterationstrin, undersøger algoritmen for stigende værdier af λ, hvorvidt en udveksling af λ kanter kan resultere i en bedre rute.

Da vi ikke ved hvilken λ-værdi der skal anvendes, virker det lidt absurd blot at fastsætte en værdi og derefter betragte alle ruterne. Derfor opstilles en algoritme herunder, der prøver at finde λ og $x_1,...,x_\lambda$ og $y_1,...,y_\lambda$ (kanterne der skal udveksles) på bedst mulig måde, trin for trin.

Lin og Kernighan (1973) præsenterer en algoritme på 7 trin som følger:

1. Generér en tilfældig initial rute T.
2. Sæt $G^* = 0$, hvor G^* er den hidtil bedste forbedring.
   Vælg en vilkårlig node t_1 og lad x_1 være en kant i ruten T som støder op til t_1.
   Sæt $i = 1$.
3. Fra x_1's andet slutpunkt t_2 vælges en kant y_1 til noden t_3 med en gevinst $g_1 > 0$. Hvis sådan et y_1 ikke findes, fortsættes til trin 6(d).
4. Sæt $i = i + 1$.
   Vælg x_i (som i øjeblikket forbinder node t_{2i-1} til node t_{2i}) og y_i som følgende:
   a) x_i vælges således, at hvis t_{2i} (sidste node) forbindes til t_1, så er den resulterende konfiguration en rute. (Dette er gennemførlighedskriteriet, som sikrer, at ruten kan lukkes ved blot at forbinde t_{2i} til t_1, for ethvert $i \geq 2$. Valget af y_{i-1} i trin 4e sikrer, at der altid vil være et sådant x_i).
   b) y_i er et tilgængeligt link ved slutpunktet t_{2i}, som deles med x_i, under hensyn til (c), (d) og (e). Hvis sådan et y_i ikke findes, forsættes til trin 5. (For at foretage en stor omkostningsreduktion i i'te trin, skal $|y_i|$[12] være lille og derfor vælges blandt nærmeste naboer).

[12] Giver god mening, da man ellers ikke ville bryde det forrige link x_i for at danne linket y_i.



Figur 16: Beskrivelse af algoritmens 4. trin. Den stiplede linje indikerer en mulig rute lukning. (a) et tilfælde hvor $i = 2$. (b) $i = 4$. (c) $i = 5$. (d) Backtracking i trin 6, flere valgmuligheder for y_1. Lin og Kernighan (1973).

c) For at sikre at x'er og y'er er disjunkte mængder, kan x_i ikke være et link der tidligere er blevet forbundet, og ligeledes må y_i ikke være et link der tidligere blev brudt.

d) $G_i = \sum_{1}^{i} g_i > 0$, den totale gevinst.

e) Såfremt (a) skal være mulig i $i+1$, skal det valgte y_i tillade bruddet af x_{i+1}[13].

f) Før y_i implementeres, tjekkes om rutelukningen som sker ved at forbinde t_{2i} til t_1 giver en gevinst der er bedre end den hidtil bedste. (Da gennemførlighedskriteriet er opfyldt for $i \geq 2$, kendes dette resultat).

Lad y_i^* være en kant der forbinder t_{2i} til t_1 og lad $g_i^* = \left|y_i^*\right| - \left|x_i\right|$.

Hvis $G_{i-1} + g_i^* > G^*$, sættes $G^* = G_{i-1} + g_i^*$ og $\lambda = i$.

(G^* er altid den bedste forbedring i T og derfor den standardværdi der sammenlignes med. Dette betyder, at $G^* \geq 0$ altid vil være monotont voksende[14]. Indekset λ

[13] Bryder man ikke kanten x_{i+1} er det ikke muligt at danne kanten y_i.

[14] Da lille g skal være strengt større end 0.

definerer antal elementer i x- og y-mængden, der skal udveksles for netop at opnå G^*).

5. Dannelsen af x_i og y_i i trin 2-4 stoppes, såfremt der ikke er flere kanter, x_i og y_i, som tilfredsstiller 4(c)-(e) eller $G_i \leq G^*$, som repræsenterer stopkriteriet.
   Hvis $G^* \geq 0$, tag udgangspunkt i den nye rute T' (med $f(T') = f(T) - G^*$[15]) og gentag hele proceduren fra trin 2, hvor T' nu er den initiale rute.
6. Hvis $G^* = 0$ foretages en begrænset backtracking således:
   a. Gentag trin 4 og 5, og vælg et y_2 fra en liste i stigende orden efter længde, så længe gevinstkriteriet $g_1 + g_2 > 0$ er tilfredsstillet. Har man fundet en forbedring, vendes tilbage til trin 2.
   b. Hvis alle valg af y_2 i trin 4(b) er opbrugt uden profit, vend tilbage til 4(a) og prøv et alternativt valg for x_2.
   c. Hvis ovenstående heller ikke giver forbedringer, udføres en yderligere backup til trin 3, hvor y_1'erne undersøges i en rækkefølge efter stigende længde.
   d. Hvis y_1'erne også er opbrugt uden profit, prøver man et alternativt x_1 i trin 2.
   e. Hvis dette ikke lykkes, vælges et nyt t_1, hvorefter trin 2 gentages.

Bemærk at backtracking kun udføres, hvis der ikke er nogen gevinst og kun for $i = 1$ og $i = 2$.

7. Proceduren stopper, når alle n for t_1 er undersøgt uden profit.

For at muliggøre denne algoritme er det nødvendigt at have nogle enkelte ting klargjort:

- Der er brug for en udvælgelsesregel, som angiver hurtigt og effektivt, hvilket par (x_i, y_i) der er mest fejlplaceret.
- Der er brug for en simpel funktion, som repræsenterer den totale profit for et forslået sæt af udvekslinger. Antag at, g_i angiver profitten eller gevinsten forbundet med udvekslingen af x_i og y_i, givet at $x_i, y_i, \ldots, x_{i-1}, y_{i-1}$ allerede er valgt. Gevinsten ved at udveksle $x_1, \ldots, x_i$ med $y_1, \ldots, y_i$ er $g_1 + \ldots + g_i$. Additiviteten betyder, at udvælgelsesprocessen ikke behøver at stoppe så snart $g_i < 0$, men derimod når $\sum_{i=1}^{i=k} g_i \leq 0$, hvilket hjælper med at undgå lokalt optimum.
- Hvis det skal være muligt at standse udvælgelsesprocessen for enhver værdi af λ og dermed udveksle $x_1, \ldots, x_i$ med $y_1, \ldots, y_i$, er et væsentligt spørgsmål i dette henseende om udvekslingen er mulig, dvs. om løsning tilfredsstiller et kriterium C.
- Der er selvfølgelig også behov for et stopkriterium, der angiver hvornår det ikke kan betale sig at kigge efter profitable udvekslinger, eller når marginalgevinsten begynder at falde.

[15] Hvor f angiver objektfunktionen

- Det sidste krav er, at de to mængder $(x_1,...,x_i)$ og $(y_1,...,y_i)$ er disjunkte. Har man flyttet et element, er det ikke tilladt at flytte det tilbage i samme iteration. Fordelene herved er, at programmeringsfejl undgås, beregningstiden reduceres samt gevinstfunktionen simplificeres.

Efter undersøgelsen af forslåede udvekslinger samt deres tilsvarende gevinster, vælges selvfølgelig den værdi af λ, for hvilken $\sum_1^k g_i$ er maksimal og ikke mindst mulig. Hvis denne sum er positiv, udveksles mængderne for derved at give en ny og bedre løsning. Hele processen gentages med den nye løsning som startpunkt. På et tidspunkt vil $\sum_1^k g_i \leq 0$, og der kan ikke længere foretages flere forbedringer (løsningen er et lokalt optimum).

Såfremt der ønskes en dybere forståelse af algoritmen, henviser vi til *Lin, Kernighan (1973).*

Multi-rute forbedring

I stedet for at udveksle kanter i samme rute som i $\lambda - opt$ for TSP, er der udviklet andre nabosøgningsalgoritmer specifikt til VRP, hvori der er mulighed for interaktion mellem kunder på tværs af ruterne. Trækkene i disse algoritmer sørger for, at én eller flere køretøjer i hver iteration får tildelt nye kunder.

Thompson og Psaraftis (1993) har i deres artikel beskrevet en "general b-cyclic, k-transfer scheme", hvor en "cirkulær" permutation[16] af b ruter overvejes og k kunder fra hver rute bliver flyttet til den næste rute i den cykliske permutation. Forfatterne viser, at bestemte kombinationer af de to parametre b og k giver interessante resultater.

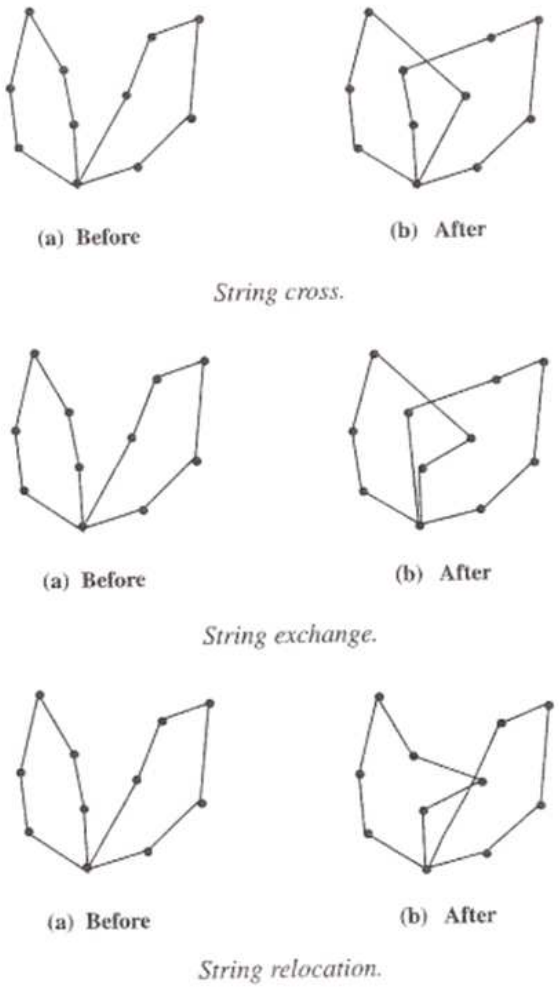
Et specialtilfælde af dette scheme er en 2-cyklisk udveksling, som i Van Breedam's (1994) forbedringsoperationer. Disse operationer er:

- String cross (SC): To strenge af noder udveksles ved at krydse to kanter fra to forskellige ruter.
- String exchange (SE): To strenge med maksimum k noder udveksles mellem to ruter.
- String relocation (SR): En streng med maksimum k noder flyttes fra én rute til en anden, typisk med $k=1$ eller $k=2$.
- String mix (SM): Det bedste træk mellem SE og SR vælges.

Man skal selvfølgelig vide, at de fire ovenstående træks performance helt afhænger af, om man ser på det klassiske problem eller en af udvidelserne. Med lidt varsomhed konkluderer Van Breedam, at det er bedst at starte søgningen fra en god løsning, idet initialløsningen påvirker både beregningstiden og

[16] En cirkulær permutation er en type permutation der ikke har noget udgangspunkt og intet slutpunkt. Det er et sæt af elementer, der har en ordre, men ingen reference. Det cirkler tilbage omkring sig selv og indeslutter.

kvaliteten af den endelige løsning. Han mener også, at de bedste løsninger opnås når SE træk udføres for $k = 2$. En illustration af de tre første træk kan ses i figur 9.



Figur 17: Eksempel på træk der foretager forbedringer på flere ruter samtidigt. Toth og Vigo (2002).

5.4 Heuristikker for VRPTW

En stor del af den klassiske heuristik til VRP med tidsvinduer er baseret på Solomon (1987), som var én af de første der præsenterede approksimative løsningsmetoder til denne type problemer. Ifølge Bräysy og Gendreau (2005) placerer Russel (1995) og Bräysy (2002) sig Pareto-optimalt i forhold til både nøjagtighed og beregningshastighed. I deres sammenligning af heuristikker er dog kun de medtaget, der er testet på det sæt af problemer som bl.a. også Solomon (1987) benyttede sig af.

Russell (1995) introducerer en forbedringsalgoritme i selve konstruktionsprocessen, en såkaldt hybrid heuristik, som, han konkluderer, dominerer den traditionelle 2-fase model; forbedring efter konstruktion. Bräysy (2002) bygger sin heuristik på Solomon (1987) og Russel (1995), hvori adskillige

heuristikker kobles sammen i en 3-fase model. Der indgår en konstruktionsfase, derefter en minimering af antal ruter og til sidst en minimering af den totale afstand ved en Or-opt forbedringsalgoritme.

Vi vil i det følgende redegøre for de oprindelige heuristikker præsenteret af Solomon (1987) og dernæst den mere avancerede fra Bräysy (2002).

5.4.1 Solomon (1987)

Op til Solomon's artikler fra 1986 og 1987 har næsten alle løsningsforslag til VRPTW været uden hensyn til tidsaspektet eller ved manuelt at justere løsningsforslaget mere eller mindre ad hoc. Solomon (1987) præsenterer 7 konstruktionsheuristikker, der mere eller mindre alle er omformuleringer af heuristikker til CVRP. Forud for disse er antaget, at køretøjerne er homogene og antallet kan variere frit i konstruktionen af ruterne.

Time feasibility

Inden præsentationen af heuristikkerne introduceres begrebet "time feasibility", der beskriver de nødvendige og tilstrækkelige betingelser for, at en kunde u kan indsættes mellem to kunder i_{p-1} og i_p, hvor $1 \le p \le m$, i en allerede eksisterende rute, $(i_0, i_1, \ldots, i_m)$, hvor $i_0 = i_m = 0$ svarer til depotet. Lad tidsvinduet for kunde i være $e_i \le b_i \le l_i$, hvor b_i er starttidspunktet for service af kunde i. Lad ydermere PF_{i_p} ("push forward") betegne den tidsmæssige virkning, som indsætningen af kunde u har haft på start af service af kunde i_p. Lad følgende være gældende:

(5.2) $$PF_{i_p} = b_{i_p}^{new} - b_{i_p} \ge 0 \text{ og } PF_{i_{r+1}} = \max\{0, PF_{i_r} - w_{i_{r+1}}\}, p \le r \le m-1,$$

hvor $b_{i_p}^{new}$ er starttidspunktet af service af kunde i_p efter indsætning af kunde u og $w_{i_{r+1}}$ er ventetiden ved kunde i_{r+1}. Fortolkningen af $PF_{i_{r+1}}$ ses klart, idet den tidsmæssige forskydning af service for i_p har direkte indflydelse på start af service for alle kunder efter i_p.

På lignende vis kan defineres en "push backward", dvs. indvirkningen på alle kunder forud for kunde i_{p-1}. For at en indsættelse skal kunne lade sig gøre, må der ikke være nogen ventetid for kunderne $1 \le r \le p-1$, og skal være en vis tid mellem start af service af en kunde og det tidligste servicetidspunkt, således at der er ekstra tid at tage af når en kunde indsættes. Det sidste er nødvendigt, netop fordi vi ikke har en ventetid at gøre godt med, ligesom for en "push forward". Ydermere må service af første kunde ikke skubbes længere tilbage end det tidligste tidspunkt et køretøj kan ankomme til kunden fra depotet. Pga. disse begrænsninger, vil en indsættelse, der vurderes på baggrund af en "push backward", oftere vil blive afvist end en "push forward". Dette begrænser antal tilfælde, hvor et "push backward" kan benyttes i et "time feasibility"-tjek. Et "push backward" vil være defineret forholdsvis simpelt:

(5.3) $$PB_{i_{p-1}} = b_{i_{p-1}} - b_{i_{p-1}}^{new} \ge 0 \text{ og } PB_{i_{r-2}} = PB_{i_r}, 1 \le r \le p-1,$$

Ifølge Solomon er en "push forward" langt at foretrække, idet vi på denne vis ved indsættelse af kunder reducerer ventetid.

Solomon tager udgangspunk i en "push forward" i følgende "time feasibility"-tjek:

Lemma 1.1[17]: De nødvendige og tilstrækkelige betingelser for "time feasibility" ved indsætning af kunde u mellem i_{p-1} og i_p, $1 \le p \le m$, i en allerede eksisterende rute, $(i_0, i_1, \ldots, i_m)$, hvor $i_0 = i_m = 0$, er

(5.4) $$b_u \le l_u \text{ og } b_{i_r} + PF_{i_r} \le l_{i_r}, p \le r \le m.$$

Såfremt ovenstående er gældende vil tidsvinduerne fortsat være overholdt for alle kunder efter indsætning af kunde u.

Lad os nu kort beskrive hver af de af Solomon præsenterede heuristikker for VRPTW.

Savings heuristik

En "savings"-algoritme, baseret på den klassiske algoritme af Clarke og Wright (1964), er den første Solomon omskriver til at inkludere tidsvinduer. I den parallelle version, bygges ruterne op simultant og besparelsen defineres som følger:

(5.5) $$sav_{ij} = d_{io} + d_{oj} - \mu d_{ij} \quad , \mu \ge 0.$$

Hvor d_{ij} svarer til afstanden mellem i og j og μ er en parameter der angiver vigtigheden af at hægte de to noder sammen der er tættest på hinanden. Er μ høj, vil de noder der ligger langt fra hinanden have langt mindre besparelser end dem, der ligger tæt på hinanden. Som i den oprindelige algoritme, er de eneste mulige rutesammenlægninger fra den sidste kunde (l) i den ene rute til den første kunde (f) i den anden rute. Besparelserne ordnes og ruterne der repræsenterer den største besparelse implementeres, såfremt der gælder at $sav_{ij} > 0$. Er dette ikke tilfældet, standser algoritmen.

Nu tager vi højde for at der eksisterer tidsvinduer, hvorfor nogle noder ikke kan hægtes sammen medmindre tidsvinduerne samtidig er opfyldt. Den måde savings-algoritmen tager højde for dette, er ved at tjekke for "time feasibility" ved hvert trin i algoritmen. I form af Lemma 1.1, vil $u = f$ skulle betragtes som det punkt der skal indsættes i en allerede eksisterende rute, bestående af den sammensatte rute med undladelse af kunde f. Dermed tjekkes hver af de kunder der følger efter f, hvorvidt f kan indsættes og dermed indgå i den samlede rute. Dette er både nødvendigt og tilstrækkeligt, idet alt forud for l (samt lig l) vil være opfyldt allerede.

Denne heuristik har dog den ulempe, at det vil være muligt at sammenkæde to kunder med kort rumlig afstand til hinanden men langt rent tidsmæssigt, hvorfor disse vil udgøre én rute for sig selv, idet ingen anden rute kan indsættes før eller efter. Det vil føre til enorme ventetider og et større antal nødvendige

[17] Direkte oversat fra Solomon (1987).

køretøjer. Algoritmen kan forbedres ved at indføre en maksimal ventetid på kanten (l, f) i forbindelse med sammenføjningen af to ruter. Lad w_f^{new} være ventetiden ved kunde f og W en grænseværdi på ventetiden; (l, f) sammenkædes kun såfremt $w_f^{new} \leq W$.

En tidsorienteret, nærmeste-nabo heuristik

Denne heuristik er en sekventiel konstruktiv metode, som starter med at knytte den nærmeste kunde til depotet, og ved hver efterfølgende iteration søger efter den nærmeste kunde i forhold til den sidste kunde tilføjet ruten. Såfremt kunden opfylder "time feasibility", køretøjets ankomst til depotet samt kapacitetsbegrænsningen, kan kunden tilføjes for enden af ruten. Hvis ikke, startes en ny rute med den nu nærmeste kunde endnu ikke tilknyttet en rute.

Den nærmeste kunde vælges ud fra følgende: Lad t_{ij} være rejsetiden mellem kunde i og j, T_{ij} betegne tiden fra service af i er færdig og til service af j kan begynde samt v_{ij} være tiden fra hvornår køretøjet har mulighed for at ankomme til kunden til hvornår kunden senest skal serviceres. Nu gælder:

(5.6) $$T_{ij} = b_j - (b_i + s_i), \text{ og}$$

(5.7) $$v_{ij} = l_j - (b_i + s_i + t_{ij}).$$

Nærmeste kunde j fra i defineres som den mindste omkostning:

(5.8) $$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij},$$

hvor $\delta_1 + \delta_2 + \delta_3 = 1$ og $\delta_1, \delta_2, \delta_3 \geq 0$.

Indsættelsesheuristik

Nu generaliserer vi ovenstående nærmeste-nabo heuristik til ikke kun at tilføje kunder for enden af en rute, men til et hvilket som helst sted i en allerede eksisterende rute. På denne vis håber vi på at kunne mindske ventetiderne betydeligt.

Denne sekventielle konstruktive algoritme kan initialiseres på forskellig vis; 1) vi kan f.eks. starte med kunden længst fra depotet eller 2) kunden med den tidligste deadline. Så snart en rute er initialiseret følges følgende fremgangsmåde:

Lad $(i_0, i_1, \ldots, i_m)$ være den nuværende rute, hvor $i_0 = i_m = 0$. Bestem nu for hver kunde u, som ikke er tilknyttet en rute, et p, som betegner det bedst mulige sted mellem to kunder i_{p-1} og i_p at indsætte u. Problemet kan løses som følgende:

(5.9) $$c_1(i(u), u, j(u)) = \min\{c_1(i_{p-1}, u, i_p)\} \quad , \; p = 1, \ldots, m.$$

Idet indsættelsen af en kunde kan ændre start af service for de efterfølgende kunder, må "time feasibility" tjekkes for disse. Her kan lemma 1.1 med fordel benyttes.

Efter vi har valgt det bedste sted at indsætte en kunde, vælges nu den bedste kunde u at indsætte på baggrund af følgende funktion:

(5.10) $$c_2\left(i\left(u^*\right),u^*,j\left(u^*\right)\right)=\text{optimum}\left\{c_2\left(i(u),u,j(u)\right)\middle|\, u\,er\,mulig\,og\,ikke\,tilknyttet\,en\,rute\right\}$$

u tilføjes ruten og algoritmen starter med at vælge et nyt sted at indsætte en kunde. Såfremt der ikke kan findes et muligt u til indsættelse i den eksisterende rute, initialiseres en ny rute, medmindre samtlige kunder er tilknyttet en rute, hvorved algoritmen standses.

Solomon beskriver 3 mulige kombinationer af funktioner c_1 og c_2.

i) Lad c_{11} og c_{12} defineres som:

(5.11) $$c_{11}(i,u,j)=d_{iu}+d_{uj}-\mu d_{ij},\ \mu\geq 0$$

(5.12) $$c_{12}(i,u,j)=b_{j_u}-b_j$$

hvor b_{j_u} betegner start af service for kunde j såfremt u indsættes. Her beskriver c_{11} omkostningen ved at afvige fra den oprindelige rute for at besøge u og c_{12} den tidsmæssige forskydning af service (omkostning) for kunde j ved indsættelse af u. Nu defineres:

(5.13) $$c_1(i,u,j)=\alpha_1 c_{11}(i,u,j)+\alpha_2 c_{12}(i,u,j),\ \alpha_1+\alpha_2=1 \text{ og } \alpha_1,\alpha_2\geq 0,$$

(5.14) $$c_2(i,u,j)=\lambda d_{ou}-c_1(i,u,j),\ \lambda\geq 0.$$

hvor c_2 nu maksimeres for at finde det bedste u at indsætte. Det bedste sted at indsætte en kunde u, er således det som minimerer både den ekstra afstand og det ekstra tidsforbrug der kræves for at servicere u efter i og før j.

ii) c_1 defineres som i i), c_2 som følger:

(5.15) $$c_2(i,u,j)=\beta_1 R_d(u)+\beta_2 R_t(u),\ \beta_1+\beta_2=1 \text{ og } \beta_1\geq 0,\beta_2>0.$$

hvor $R_d(u)$ og $R_t(u)$ er hhv. den totale længde og det totale tidsforbrug af ruten, såfremt u indsættes, hvorfor c_2 nu skal minimeres.

ii) c_{11} og c_{12} defineres som i i); nu defineres et c_{13} som forskellen mellem hvornår u senest skal serviceres og hvornår han rent faktisk bliver serviceret, såfremt han indsættes det pågældende sted. Solomon betegner dette at tage højde for hvor meget u haster at få indsat. Altså er:

(5.16) $$c_{13}(i,u,j) = l_u - b_u,$$

hvorfor c_1 og c_2 nu bliver:

(5.17) $$c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j) + \alpha_3 c_{13}(i,u,j),\ \alpha_1 + \alpha_2 + \alpha_3 = 1 \text{ og } \alpha_1, \alpha_2, \alpha_3 \geq 0,$$

(5.18) $$c_2(i,u,j) = c_1(i,u,j).$$

hvor c_2 nu minimeres for at finde det optimale u^*.

En tidsorienteret sweep heuristik

Som i den oprindelige sweep-algoritme af Gillett og Miller (1974), inddeles denne i to faser, en "clustering"-fase og en optimeringsfase. I første fase deles kunderne ud på køretøjerne efter et sweep mod uret, og i anden fase benyttes indsættelsesalgoritmen (i) ovenfor til at optimere ruten for hvert køretøj. Fordi vi har med tidsvinduer at gøre, vil nogle kunder dog være udeladt af en rute efter optimeringsfasen. Disse kunder, ikke tilknyttet en rute, vil blive betragtet i endnu et sweep og efterfølgende optimering. Processen fortsættes indtil alle kunder er tilknyttet en rute.

Afsluttende kommentar

Af de 7 heuristikker Solomon (1987) introducerede, gav indsættelsesheuristikken (i) langt de bedste resultater over et bredt spektrum af problemer med tidsvinduer. Det vist sig dog også at heuristikken med fordel kunne bruges sammen med en sweep-algoritme for større problemer. Sweep-heuristikken leverede ikke blot bedre resultater for disse, men havde også en betydelig mindre beregningstid.

5.4.2 Bräysy (2002)

Bräysy foreslår en 3-fase algoritme, som starter med én af to konstruktive heuristikker til at danne en initial løsning. Her kan man med fordel tage udgangspunkt i den bedste løsning af de to til videre behandling. Herefter reduceres antallet af ruter ved en "ejection chain"-baseret tilgang, hvorefter proceduren gentages adskillige gange for alle valg af parameterværdier i de konstruktive heuristikker. Nu gemmes kun de løsninger med det mindste antal ruter efter optimering og kun disse bruges nu i den tredje fase af algoritmen. I denne fase ordnes først ruterne i den pågældende løsning, således at ruter med dårlige træk såsom lang ventetid og afstand i forhold til antal kunder tages først op til forbedring. En parameter β styrer prioriteten af disse to forhold, så jo større β des mere lægges vægt på afstanden. Dernæst fortsættes med en Or-opt forbedringsalgoritme til at optimere ruterne med hensyn til den totale afstand. Or-opt forbedringen går ud på at flytte først 3 på hinanden følgende kunder til andre mulige steder i ruten (muligvis med omvendt rækkefølge), og acceptere denne indsættelse såfremt det giver en bedre løsning. Når der ikke er flere forbedringer at finde, forsøges at flytte 2 på hinanden følgende kunder og til sidst flyttes enkeltkunder for at opnå en forbedring. Når tredje fase er udført for alle mulige løsninger returneres den bedste løsning. Vi kan opsummere denne tilgang ved følgende:

1) Benyt én af to konstruktive heuristikker.
2) Foretag ruteeliminering indtil det ikke er muligt at reducere antal ruter yderligere.
3) Benyt nye sæt af parameterværdier og gentag trin 1-3 indtil alle parameterværdier er undersøgt.
4) Gem løsninger med det mindste antal ruter.
5) Udfør på én af de valgte løsninger S_i først en ordning af ruterne og dernæst en Or-opt forbedring af disse. Såfremt løsningen er bedre end den bedste løsning S_b, sæt $S_b = S_i$.
6) Gentag trin 5 for alle i.
7) Returnér S_b.

I det følgende vil vi gå mere i dybden med de to foreslåede konstruktive heuristikker samt ruteelimineringsproceduren.

Hybrid konstruktiv heuristik

Denne sekventielle konstruktive heuristik bygger på Solomon (1987) og Russell (1995).

Først bestemmes kunden med den største afstand til depotet, og dernæst 3 kunder der er længst fra både depotet og hinanden. Disse 4 kunder udgør den primære kundemængde PK og vil grafisk danne et rektangel. En sekundær mængde SK dannes nu af de 4 kunder der geografisk ligger nærmest midtpunktet af hver kant i rektanglet. Disse i alt 8 kunder udgør mængden, hvorfra en kunde vælges til at initialisere den første rute. De efterfølgende ruter startes med en kunde fra mængden $I = T \cup PK \cup SK$, hvor T udgør de 30 % af kunderne der er placeret længst fra depotet. Udvælgelsen sker på baggrund af et sweep af mængden I med eller mod uret, med start i det sidst valgte seed. Såfremt alle disse kunder er tilknyttet en rute, vælges en kunde ikke tilknyttet en rute der er tættest på det sidst valgte seed.

Når det første seed er valgt, benyttes en indsættelsesalgoritme meget lig Solomon's (i). Til indsættelse i den partielle rute ledes der dog ikke blandt alle kunder, men kun de som ligger i nærheden af ruten, mere specifikt søges der blandt de kunder u der ligger indenfor afstanden $\overline{d}$ gange den maksimale afstand mellem to kunder i ruten, hvor $\overline{d}$ er en brugervalgt parameter. Det betyder at for hver ny kunde der indsættes i ruten, opdateres mængden af nærliggende kunder før en ny indsættelse foretages. Kunderne længst væk er dog af denne årsag de sværeste at indsætte i en rute, idet der oftest er et meget begrænset antal indsættelsesmuligheder for dem. Dermed kan vi risikere at vi til sidst vil være nødt til at lade disse indgå i ruter for sig, hvorfor det vil være fordelagtigt at indsætte dem forholdsvist tidligt i algoritmen. Dette kan gøres ved at trække afstanden fra depotet til kunde u fra i omkostningsfunktionen c_1, hvorved de kunder længst væk får større prioritet i algoritmen. Lad følgende omkostningsfunktioner være gældende:

(5.19) $$c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j) - \alpha_3 d_{0u},\ \alpha_1 + \alpha_2 = 1 \text{ og } \alpha_1, \alpha_2 \geq 0, \alpha_3 > 0,$$

(5.20) $$c_{11}(i,u,j) = d_{iu} + d_{uj} - d_{ij}$$

(5.21) $$c_{12}(i,u,j) = b_{j_u} - b_j$$

(5.22) $$c_2(i,u,j) = c_1(i,u,j).$$

Når det bedste sted for indsættelse er fundet, indsættes kunden med den laveste omkostning. For hver $\overline{k}$ antal kunder der indsættes i ruten, hvor $\overline{k}$ er en brugervalgt parameter, optimeres ruten med Or-opt forbedringer. Or-opt forbedringer er i grunden et specialtilfælde af 3-opt, som normalt har 8 forskellige genforbindelser efter fjernelsen af 3 kanter, men Or-opt har kun 2, idet det er påkrævet, at de fjernede kunder er i forlængelse af hinanden i ruten. Med andre ord fjernes 3 sammenhængende kanter fra én rute og forsøges indsat i en anden.

Der fortsættes med at indsætte kunder til den partielle rute, indtil det ikke længere er muligt at indsætte flere kunder, hvorefter en ny rute initialiseres som specificeret tidligere. Algoritmen standser når alle kunder er tilknyttet en rute.

Savingsheuristik

Nu tages udgangspunkt i savingsheuristikken af Clarke og Wright (1964). Som tidligere starter vi med n antal ruter, hvor der konstant sammenlægges ruter med de største besparelser i form af både afstand og ventetid, så længe både kapacitetsbegrænsningerne og tidsvinduerne er opfyldt. Lad besparelserne defineres som:

(5.23) $$sav_{ij} = TC_i + TC_j - TC_{ij},$$

hvor $TC_k = \alpha_1 TD_k + \alpha_2 TW_k$ er en vægtning af den totale afstand i rute k, TD_k, og den totale ventetid i rute k, TW_k. TC_{ij} er tilsvarende omkostningen efter sammenlægningen af rute i og j. For hver $\overline{m}$ kunder der føjes til den sammenlagte rute, foretages en O-opt forbedring af den sammenlagte rute før de nye besparelser beregnes, for på denne vis at reducere TC_{ij}.

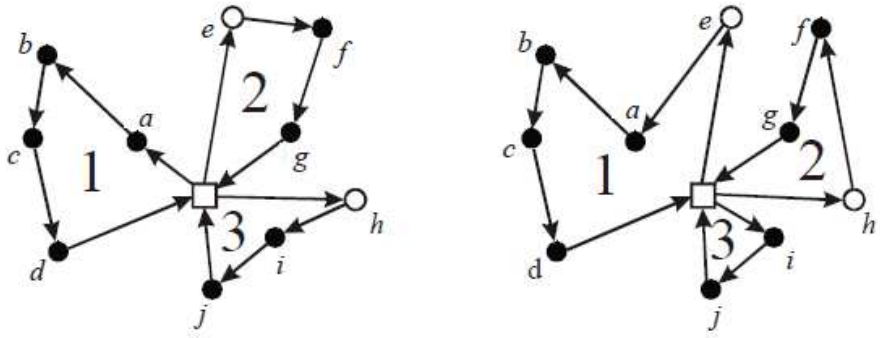
O-opt er en forbedringsalgoritme, hvor kunderne omfordeles inden for den valgte rute. Idéen går ud på at genopbygge en rute på baggrund af en indsættelsesheuristik som ovenfor, ved at initialisere ruten fra forskellige brugervalgte seed-kunder i stedet for én. De valgte seeds skal være geografisk spredt samt tilhøre den aktuelle rute. For hver seed genopbygges ruten med en hybrid konstruktiv heuristik ved at indsætte kunderne i stigende rækkefølge efter deres tidsvindues omfang. For hver $\overline{k}$ antal kunder der indsættes i ruten, optimeres den med Or-opt forbedringer. Den bedste rute udvælges til at indgå i en vurdering af besparelserne.

Savings-algoritmen fortsættes, indtil der ikke længere med fordel kan sammenlægges ruter.

Ruteeliminering

For hver løsning som de indledende konstruktive heuristikker giver os, forsøger vi at eliminere antallet af ruter mest muligt. Dette gøres ved en ”ejection chain”-baseret heuristik.

En ”ejection chain” i VRPTW-kontekst er en serie af flytninger af en enkelt kunde i_1 fra én rute r_1 til en anden r_2. Før kunden kan indsættes må der dog gøres plads til ham ved at fjerne (”eject”) en anden kunde i_2. Denne kunde skal nu finde en tredje rute $r_3 \neq r_1$, hvorfra endnu en kunde i_3 udskydes, osv. Kæden af disse udskydende kunder standses når der findes en ”ledig plads”, dvs. når kunde k kan indsættes uden nødvendigheden af at udskyde kunde $k+1$. Afhængig af problemets størrelse vil denne plads dog oftest være samme sted hvorfra den første kunde blev fjernet. Denne heuristik er temmelig tidskrævende, medmindre man begrænser antallet af mulige flytninger.



Figur 18: Betragt til venstre et VRP, hvor rute 3 forsøges elimineret. Kunde h forsøges derfor indsat i rute 2, men pga. tidsvinduer eller kapacitetsbegrænsninger er en sådan direkte indsættelse ikke mulig. I stedet undersøges, hvorvidt nogle kunde fra rute 2 kan indsættes i andre ruter (på nær rute 3). Antag at kunde e kan indsættes i rute 1 før kunde a, hvorefter der er blevet givet plads i rute 2 til kunde h. Disse indsættelser udgør en ”ejection chain”. Resultatet ses i højre figur. Bräysy (2003)

Målet er at eliminere en rute fra problemet ved at fjerne alle kunder via ”ejection chains” og på denne vis omplacere dem i andre ruter (se eksempel i figur 18). En kunde i fra ruten r_e til eliminering forsøges nu først at blive afsat direkte til en naborute, $r_n \in N(r_e)$, hvor $N(r_e)$ er mængden af alle naboruter til r_e. Såfremt dette ikke er muligt, udføres en procedure, hvor der sker en omfordeling af kunderne. Kunde i placeres i r_n, således at omkostningen i (5.19) minimeres. Såfremt ”time feasibility” ikke overholdes sker en omfordeling af kunderne i ruten i et forsøg på at gøre ruten mulig. Omfordelingen sker som en serie af flytninger, hvor en omplacering accepteres, såfremt den totale forsinkelse i ruten reduceres og man dermed kommer nærmere en mulig løsning. Forhåbentlig vil vi efter adskillige kundeflytninger indenfor ruten have en mulig rute.

Såfremt i ikke kan afsættes til nogen rute r_n, selv med en omfordeling af kunderne, annulleres alle ændringer og der tilføjes en ekstra kæde til vores ”ejection chain”, hvorefter det atter forsøges at få kunden afsat på ovenstående vis. Hvis dette heller ikke er muligt, tilføjes endnu et led, osv., indtil ét af vores stopkriterier er opfyldt eller kunden afsættes til en naborute. Såfremt det viser sig ikke at være muligt at afsætte alle kunder og dermed eliminere ruten helt, forsøges at hente så mange kunder fra naboruter til r_e. Hvis det ikke er muligt at fjerne den fra løsningen alligevel, så kan vi lige så godt udnytte den bedst muligt, hvorefter vi forsøger at fjerne en anden rute og så fremdeles. Hvis det på

noget tidspunkt lykkes at fjerne den sidste kunde i en rute, gemmes denne nye løsning og en ny rute forsøges elimineret.

Ved på denne vis at starte ud med et begrænset antal udskiftninger i kæden, forsøges det at reducere tidsforbruget ved algoritmen. For yderligere reduktioner indføres følgende:

- Den rute r_e vi vælger at forsøge at få elimineret er i første omgang den med det færreste antal tidsbegrænsede kunder, hvor en kunde er tidsbegrænset hvis dens tidsvindue er mindre end 50 % af depotets.
- Hvis vi har med et større problem at gøre, hvor det mindste antal kunder i en rute er større end 15, forsøges kun at eliminere ruten med det mindste antal kunder.
- Ved søgningen efter ruter til indsættelse af kunde i, startes med at undersøge naboruter. Afstanden mellem to ruter defineres som den mindste afstand mellem to kunder, én i hver rute. Ydermere, hvis afstanden er større end en brugervalgt parameter $\overline{d}$, vil ruten ikke betragtes som mulig for indsættelse. Lad derfor mængden af naboruter defineres som:

(5.24) $$N(r_e) = \left\{ r_j \in I^r \,\middle|\, \min_{c_k \in r_j, c_p \in r_e} \left\{ d(c_k, c_p) \leq \overline{d} \right\} \right\}.$$

  hvor I^r er mængden af samtlige ruter.
- Hvis indsættelse af i resulterer i en rute der er $\overline{l}$ gange længere, vil ruten heller ikke betragtes som mulig for indsættelse, hvor $\overline{l}$ er en brugervalgt parameter.
- Antal kæder i vores "ejection chain" og antal omfordelinger af kunder begrænses af maksimalt fastsatte værdier af brugeren, hhv. $\overline{c}$ og $\overline{n}$.
- Til at tjekke for "time feasibility" benyttes begreberne "push forward" og "push backward", som introduceret i Solomon (1987).

Vi kan opsummere algoritmen på følgende vis:

1) Vælg en rute til eliminering, r_e.
2) Lad der være 0 udskydninger i første sæt af "ejection chains".
3) Indsæt kunde u fra r_e i en naborute r_n ved "ejection chains".
    a. Hvis det lykkes, tag en ny kunde fra r_e og start fra trin 3.
    b. Var det den sidste kunde i ruten, gemmes løsningen og der startes fra trin 1.
4) Hvis ikke det lykkes, omfordel kunderne i r_n efter indsættelse af kunde u og forsøg at gøre indsættelsen mulig.
    a. Hvis det lykkes, udfør 3a-b.
5) Hvis ikke det lykkes, annullér da alle ændringer og tilføj en ekstra kæde til vores "ejection chain".
6) Udfør trin 3-5 indtil et stopkriterie er nået.

7) Er et stopkriterie nået, indsæt da så mange kunder som muligt i r_e. Vælg en anden rute til eliminering og start fra trin 2.
    a. Er der ikke flere mulige ruter at eliminere, annullér ændringerne og udskriv løsningen.

Kapitel 6 – Metaheuristik

Metaheuristik er en klasse af sofistikerede heuristikker, som ofte kombinerer konstruktions- og forbedringsheuristikker baseret på koncepter fra bl.a. biologien, fysikken, kemien og matematikken. Et eksempel herpå er "Genetic Algorithms" (GA), som er inspireret fra darwinistiske principper om naturlig udvælgelse, og "Simulated Annealing" (SA), som kommer fra udglødning af metaller.

Det der adskiller metaheuristikker fra den klassiske heuristik, er at vi på den ene eller anden måde accepterer en dårligere og evt. ikke-mulig løsning i et håb om at komme ud af lokale minima og over i andre lokale (eller endda globale) minima. Der findes mange måder hvorpå man kan undslippe et sådant lokalt minimum. Nogle metaheuristikker er baseret på en lokal søgning i samarbejde med en diversificeringsstrategi, andre gentager søgningen fra forskellige (evt. tilfældigt valgte) løsninger. På denne vis undersøges en langt større andel af løsningsområdet, men der er stadig fokus på det primære mål, at producere gode løsninger på kort tid. Derfor undersøges kun de mest lovende regioner i dybden, og først dernæst vurderes hvilken løsning er den bedste.

Metaheuristikker er mere effektive end de klassiske heuristikker og kvaliteten af resultaterne er langt bedre. Men denne forbedring kommer oftest på bekostning af en væsentlig længere beregningstid, om end stadig nær så lang som for eksakte algoritmer.

I det følgende gennemgår vi seks af de mest anvendte metaheuristikker for VRP: "Simulated Annealing" (SA), "Tabu Search" (TS), "Large Neighborhood Search" (LNS), "Genetic Algorithms" (GA), "Ant Systems" (AS) og "Neural Networks" (NN).

De tre første algoritmer, SA, TS og LNS, kendetegnes ved, at de udfører en lokal søgning ud fra en enkelt løsning ad gangen. Algoritmerne starter fra en initial løsning x_1 og bevæger sig i hver iteration t fra x_t til en løsning x_{t+1} i x_t's nabo-område $N(x_t)$. Den fundne løsning er i nogle tilfælde ringere end den tidligere, dvs. $f(x_{t+1}) > f(x_t)$, hvor $f(x)$ angiver omkostningen eller længden af løsningen x. Proceduren stopper i det øjeblik et på forhånd fastsat stopkriterium er nået.

GA undersøger i hvert trin en population af løsninger i stedet for blot en enkel løsning ad gangen. Hver population stammer fra den foregående ved at kombinere dennes bedste elementer og kassere de værste.

Den femte algoritme, AS, har en konstruktiv fremgangsmåde, hvor nye løsninger dannes i hver iteration baseret på informationen fra tidligere iterationer. Taillard et al. bemærkede at TS, GA og AS er metoder, der gemmer informationer om løsninger man støder på som søgningen skrider frem, og benytter disse til at opnå forbedrede løsninger.

NN er derimod en læringsmekanisme, hvor et sæt af vægte gradvist bliver justeret indtil en tilfredsstillende løsning er opnået. Hvert problem er forskelligt og har et specifikt karakteristisk træk,

derfor er det særdeles vigtigt at tilpasse reglerne som styrer søgningen efter det problem man sidder med.

I det følgende beskrives de seks nævnte algoritmer generelt for VRP, hvor fokus i første omgang vil være at beskrive principperne bag disse. Herefter tages fat i de algoritmer indenfor disse kategorier der har været de mest anvendte til løsning af VRPTW og dennes udvidelser.

6.1 Simuleret udglødning

Simuleret udglødning (SA, eng. simulated annealing) er en reference til den proces, hvor metal og andre normalt faste stoffer varmes op til høj temperatur, og derefter nedkøles langsomt for at opnå særlige egenskaber gennem krystallisering, såsom hårdhed og styrke. Temperaturen reduceres langsomt i en serie af trin, hvor temperaturen i hvert trin fastholdes indtil en ligevægt er nået. Sker nedkølingen for hurtigt, vil krystalliseringen være mindre vellykket og strukturen vil være løs og usammenhængende.

6.1.1 Metoden som heuristik

Metoden blev introduceret i de tidlige 1980'ere af Kirkpatrick et al. (1983), men var tydeligvis inspireret af idéer fra Metropolis et al. (1953). Som heuristik har vi med en sandsynlighedsbaseret lokal søgning at gøre, hvor løsninger med lavere objektfunktionsværdi (i et minimeringsproblem) accepteres med det samme, mens dårligere løsninger accepteres med en sandsynlighed. Ved at have mulighed for at acceptere dårligere løsninger, undgår SA dermed at sidde fast i lokale minima.

Mere specifikt forløber algoritmen som følger: I hver iteration udvælges en løsning σ' fra et nabo-område $N(\sigma)$ omkring den aktuelle løsning σ. Såfremt objektfunktionsværdien $O(\sigma') < O(\sigma)$, accepteres σ' som den nye løsning uden videre. Er denne løsning tilmed bedre end den hidtil bedste løsning σ^*, opdateres sidstnævnte. Derimod, hvis $O(\sigma') \geq O(\sigma)$, accepteres σ' kun med en sandsynlighed $\exp\left(-\frac{O(\sigma')-O(\sigma)}{T_t}\right)$, der afhænger af forskellen i objektfunktionsværdi $O(\sigma')-O(\sigma)$ og en temperatur T_t. Accepteres den nye løsning σ' betragtes denne nu som den aktuelle løsning, ellers beholdes σ og der udvælges et nyt $\sigma' \in N(\sigma)$. Proceduren gentages indtil et stopkriterium er nået. Bemærk, at under hele algoritmen holdes styr på den bedste løsning, som så returneres når algoritmen stopper.

Temperaturen T_t er en parameter, der afhænger af den tid t (antal iterationer) som søgningen har kørt. Reglen der definerer temperaturen for alle t kaldes normalt en nedkølingsplan (eng. "cooling schedule". I starten er temperaturen høj, hvorfor dårligere løsninger accepteres ofte, vi kommer rundt i en stor del af løsningsrummet og vi danner en hel del forskellige alternative løsninger. Temperaturen holdes fast i korte perioder ad gangen og reduceres med jævne mellemrum med en konstant faktor. Senere vil temperaturen blive lav og vi vil mere eller mindre kun acceptere løsninger, der er en væsentlig forbedring til den aktuelle. På dette tidspunkt intensiverer vi søgningen efter gode løsninger for til sidst

at ende i et lokalt optimum. Ifølge Gendreau et al. (2008) er SA tilmed en af de få metaheuristikker, der faktisk asymptotisk konvergerer mod et globalt optimum.

Den initiale løsning er her ikke specificeret nærmere, men typisk vil denne være bestemt af en konstruktiv heuristik som f.eks. sweep eller savings. Hvorledes et nabo-område omkring den aktuelle løsning gennemsøges findes der også mange varianter af. Stopkriteriet er heller ikke udspecificeret, men kunne f.eks. bero på den totale beregningstid, antal iterationer, temperaturen, objektfunktionsværdien eller når der gennem et antal iterationer ikke er fundet en forbedring.

Følgende parametre bestemmes af brugeren:

- Starttemperaturen T_0.
- Periodelængden L, der angiver i hvor lang tid temperaturen holdes konstant inden næste temperaturtrin. Denne kaldes også "plateaulængden".
- Nedkølingsfaktoren α, der angiver størrelsen af temperaturfaldet ved hvert trin.
- Valg af stopkriterium.

Der findes adskillige succesfulde variationer af ovenstående algoritme, som dog typisk har at gøre med variationer over nedkølingsplanen. Her er nogle eksempler:

- Forlæng plateaulængden hver gang en ny bedste løsning findes. Denne variant bygger på idéen, at der findes temperaturer, der er mere hensigtsmæssige end andre til at finde gode løsninger, hvorfor vi med fordel kan beholde denne et længere stykke tid.
- Kortvarigt sænk temperaturen når en ny bedste løsning findes, for på den måde at intensivere søgningen omkring de bedste løsninger uden at give afkald på størrelsen af det løsningsrum der undersøges.
- Hæv temperaturen hvis der gennem et antal iterationer ikke er fundet en ny bedste løsning, for således at få søgningen ud af et område af løsningsrummet, der tilsyneladende ikke indeholder særlig gode løsninger.
- Algoritmen genstartes indtil flere gange efter endt kørsel, hvor temperaturen sættes lig starttemperaturen og den aktuelle løsning sættes til den sidst kendte aktuelle løsning eller den hidtil bedste løsning. På den måde spredes søgningen atter fra hvor den er endt, i det tilfælde at vi skulle være endt i et dårligt lokalt optimum.

6.1.2 Algoritmen i VRP-sammenhæng

Når det kommer til VRP er langt de mest succesfulde implementeringer af SA såkaldte hybrider. Her kan bl.a. nævnes Osman (1993), der benytter en række forbedringer af ovenstående algoritme: 1) Der vælges en initial løsning baseret på Clarke og Wrights "savings"-heuristik. 2) I søgningen efter en bedre løsning i et nabo-område omkring σ benyttes en λ-interchange forbederingsalgoritme, hvor to ruter p og q vælges sammen med et antal på hinanden følgende kunder i hver rute, S_p og S_q, hvor $|S_p| \leq \lambda, |S_q| \leq \lambda$. Nu byttes S_p og S_q i de to ruter, såfremt dette er muligt. Bemærk, at antal elementer i de to kundemængder endda kan være tomme, i hvilket tilfælde der blot vil være tale om at

placere en kunde fra én til en anden rute. Så snart en forbedrende løsning er lokaliseret standses forbedringsalgoritmen og denne accepteres som den nye løsning. Ellers vil det være nødvendigt at gennemsøge hele nabo-området og såfremt der ikke findes en bedre løsning, accepteres den bedste løsning i nabo-området med en sandsynlighed som specificeret tidligere. 3) Nedkølingsplanen fungerer på den måde, at så længe en ny aktuel løsning accepteres, reduceres temperaturen kontinuert, men så snart vi vælger at beholde den gamle løsning, enten halveres temperaturen eller vi antager temperaturen fra den hidtil bedste løsning, afhængig af hvilken der er størst.

Det skal nævnes, at den metaheuristik der normalt betegnes deterministisk udglødning (DA, eng. deterministic annealing) blot er SA uden det stokastiske element. Dvs. i stedet for at acceptere en dårligere løsning på baggrund af en sandsynlighed, vælges typisk en ny løsning på baggrund af afstanden fra den aktuelle løsning, målt enten i absolut eller procentvis afvigelse i objektfunktionsværdien.

6.2 Tabu Search

"Tabu Search" blev første gang introduceret af Fred Glover (1986) og har sidenhen været en af de mest anvendte og mest succesfulde metaheuristikker. Adskillige versioner er blevet foreslået i litteraturen og løser forskellige typer af optimeringsproblemer inden for kombinatorisk optimering. Disse metaheuristikker har trods deres forskellighed stadig visse ligheder, såsom "Adaptiv" hukommelse, der har til formål at styre søgningen ved brug af en tabuliste. Anvendelsen af TS er ikke kun begrænset til løsning af VRP problemer, men også problemstillinger inden for bl.a. planlægning, grafoptimering, teknologi m.fl. En liste over TS anvendelsesområder kan ses i Glover og Laguna (1997).

TS er i bund og grund en lokal søgnings-baseret algoritme, hvor den bedste løsning i nabo-området til den aktuelle løsning vælges som den nye løsning i hver iteration, også selvom det fører til en stigning i totalomkostningen.

TS er baseret på den forudsætning, at metoden skal inkorporere ”adaptiv hukommelse” (eng. ”adaptive memory”) og ”reaktiv udforskning” (eng. ”responsive exploration”) for netop at kunne kvalificere sig som intelligent. Egenskaben adaptiv hukommelse tillader implementering af en procedure, der er i stand til at undersøge løsningsområdet både økonomisk og effektivt. Da et valg styres af den information der er indsamlet undervejs i søgningen, er TS en kontrast til hukommelsesløse modeller som er afhængig af semi-tilfældige træk. Eksempler på hukommelsesløse metaheuristikker er bl.a. GA og SA. Man kan også se denne form for hukommelse i modsætning til branch-and-bound- strategier, som har en rigid hukommelsesstruktur, dvs. en fast branching-strategi.

Der lægges vægt på reaktiv udforskning i TS, hvad enten der er tale om en deterministisk eller stokastisk problemstilling, som stammer fra den antagelse, at et dårligt strategisk valg ofte giver flere informationer end et godt tilfældigt valg. Derfor giver denne form for undersøgelse grundlag for gradvist forbedrede strategier, der udnytter søgehistorikken. At TS benytter reaktiv udforskning betyder, at denne metaheuristik udnytter løsningers gode træk til at bestemme hvilke nye lovende regioner der skal undersøges.

I det følgende vil vi beskrive dels de to ovenstående begreber i flere detaljer samt TS metodologien generelt.

6.2.1 De grundlæggende principper

Som nabosøgning under klassisk heuristik, starter "Tabu search" med en initial løsning x_1, hvorefter søgningen fortsætter iterativt fra en løsning til en anden indtil et valgt slutkriterium er tilfredsstillet. Hver løsning x har et tilknyttet nabo-område $N(x) \subset X$, hvorfra en ny løsning $x' \in N(x)$ fremkommer på baggrund af et træk. Forskellen mellem den klassiske metode (”descent”- metoden) og TS er, at førstnævnte kun tillader træk som forbedrer den aktuelle objektfunktionsværdi, og stopper i det øjeblik der ikke er flere forbedringer. Den endelige løsning x vil dog typisk være et lokalt optimum og ikke et globalt optimum. TS derimod tillader træk der forringer den aktuelle objektfunktionsværdi, og vælger træk fra et modificeret[18] nabo-område $N^*(x)$. Dette nabo-område har fået sin helt specifikke komposition på baggrund af en korttids- og langtishukommelse som vi vil komme nærmere ind på senere.

Den iterative proces i TS inddeles ofte i to faser, som gennemløbes indtil et fastsat slutkriterium er tilfredsstillet. Inden den første fase starter, bestemmes en initial løsning ved en heuristik, dennes objektfunktionsværdi gemmes som den hidtil bedste og den fungerer nu som det første sammenligningsgrundlag. I første fase gennemsøges det definerede nabo-område, hvor den bedst mulige løsning vælges. Dette sker under nogle begrænsninger, som den adaptive hukommelse giver i form af en tabuliste. En tabuliste er en liste af løsninger eller attributter[19] af løsninger, som man ikke må vende tilbage til i den nære fremtid. Formålet er at få udforsket en større del af løsningsområdet og undgå de samme områder som allerede er blevet undersøgt for mulige løsningskandidater. Herefter kan man starte anden fase, hvor der vælges en ny startløsning blandt de attraktive løsninger, som den adaptive hukommelse har gemt. Herefter gentages første fase med en søgning af den nye startløsnings nabo-område.

Tabulisten anvendes til at styre søgningen, fordi den gemmer ulovlige træk som refereres for værende et tabu træk eller rettere sagt tabuaktive. En løsning som derefter indeholder et tabuaktivt element eller en bestemt kombination af disse attributter, er dem der bliver tabu. Denne liste spiller en væsentlig rolle, fordi den forhindrer at vi konstant vender tilbage til de samme løsninger og hjælper således søgningen med at flygte fra lokale minima Formålet opfyldes ved, at tabulisten modificerer nabo-området til den aktuelle løsning på baggrund af den viden, der løbende opsamles i den adaptive hukommelse.

En tabuliste kan have forskellige designs afhængig af, hvad der er passende for det pågældende problem. Den mest simple form angiver den nyligt besøgte løsning for værende tabu, hvilket betyder, at det nu bliver ulovligt at vende tilbage til denne løsning inden for de næste k iterationer, hvor k enten er fast eller vælges vilkårligt inden for et interval. I praktiske situationer er det ikke altid effektivt at gemme hele løsningen i tabulisten, fordi det forbruger meget plads og tid når dette skal anvendes på

[18] Et modificeret nabo-område kan enten være reduceret eller udvidet.
[19] En attribut kan bestå af noder eller kanter, som er tilføjet, slettet eller placeret et andet sted ved brug af et træk.

hver genereret løsning. I stedet vælger man at gemme en attribut for hver kant, der enten er slettet eller indsat i en rute.

En tabuliste behøver på ingen måder at være symmetrisk, hvorfor en tabustruktur kan designes til at behandle tilføjede eller slettede elementer forskelligt. Én mulighed er, at begge typer er tabuaktive i samme antal iterationer. En tabuaktiv status har forskellig betydning afhængig af, om en kant er slettet fra løsningen eller tilføjet. For en slettet/tilføjet kant, betyder tabuaktiv, at denne ikke må tilføjes/slettes fra løsningen i et antal iterationer, der definerer dennes tabulængde (dvs. k iterationer). Da der er mange kanter, som ikke indgår i løsningen, synes det rimeligt at implementere en tabustruktur, som beholder nyligt slettede kanter tabuaktive i længere tid end de nyligt tilføjede kanter.

I nogle tilfælde er det tilrådeligt at benytte sig af flere tabulister simultant. Et sådant tilfælde kunne være, hvis nabo-område-strukturen både består af tilføjet-, slettet- og swap [20]træk, hvorfor man med fordel kan holde en særskilt tabuliste til hver type træk.

Hvis længden af tabulisten er længere end antallet af attributter, vil søgningen gå i stå. Dette kan også være tilfældet, selvom listen ikke er så lang, eftersom det ikke altid er muligt at sammensætte alle kanter på en vilkårlig måde. Derudover kan et problem have diverse begrænsninger, hvilket kan være årsag til, at søgeprocessen stopper selv med en kortere tabuliste. En måde at komme uden om dette problem på, er ved at implementere en procedure, der sletter den ældste post i tabulisten. Det er også påvist, at en tabuliste af fast længde ikke altid forhindrer cykliske bevægelser, hvorfor nogle forfattere har forslået at variere tabulistens længde under søgeprocessen. En anden ulempe ved tabulisten er, at den forhindrer attraktive træk, selv når der ingen farer er for cykliske bevægelser. Det er derfor nødvendigt at bruge algoritmiske anordninger, også kaldet aspirationskriterier[21], som tillader annulleringen af tabuet fra listen. Det mest almindelige og simplest anvendte aspirationskriterium tillader et tabu træk, når det resulterer i en løsning med en objektfunktionsværdi bedre end den hidtil bedste. Der er foreslået aspirationskriterier langt mere komplicerede, men de anvendes dog sjældent, på trods af at de leverer bedre løsninger. For yderligere information om aspirationskriterier se Gendreau og Potvin (2005). Hovedreglen i et aspirationskriterium er: Hvis cyklisk tilbagevending ikke kan opstå, er det muligt at ignorere tabu'er. Herunder ses et eksempel på en tabuliste, hvor tilføjede kanter er tabuaktive i én iteration og slettede i tre iterationer. Listen er konstrueret på baggrund af et eksempel i Glover og Laguna (1997).

Iterationer	Antal iterationer attributten er tabuaktiv			Indsat attribut	Slettet attribut	Objektfunktions-værdi
	1	2	3			
1				(4,6)	(4,7)	47
2	(4,6)		(4,7)	(6,8)	(6,7)	57
3	(6,8)	(4,7)	(6,7)	(8,9)	(1,2)	63
4	(4,7), (8,9)	(6,7)	(1,2)	(4,7)	(1,4)	46

[20] Et ”swap”-træk er når to kunder byttes både inter- og intrarute.

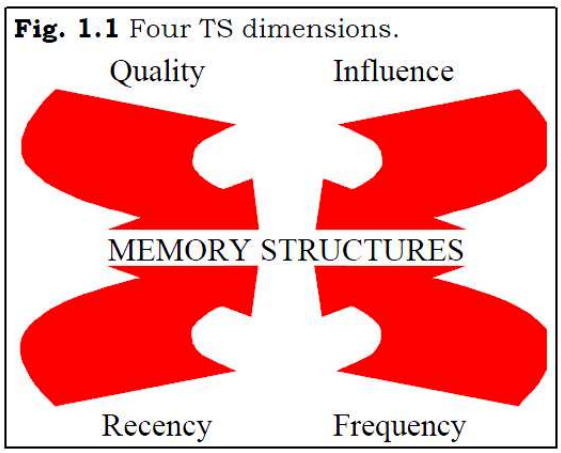
[21] Det er vigtigt at huske, at aspirationskriterier ikke tvinger et bestemt træk til at blive valgt, men blot stiller dette valg til rådighed eller alternativt ophæver evalueringsstraffen tilknyttet en bestemt tabu klassifikation.

Tabel 1: Her ses et eksempel på en tabuliste umiddelbart efter initialløsningen. Hvor f.eks. (4,6) angiver at kanten mellem kunde 4 og 6 tilføjes i løsningen. På tilsvarende vis slettes kanten mellem kunde 4 og 7. Disse træk fører til en objektfunktionsværdi på 47.

Hidtil har vi beskrevet tabulisten, som er en del af den tidligere nævnte adaptive hukommelse, som vi endnu ikke har beskrevet. Den adaptive hukommelse dækker nemlig over mere end blot en tabuliste.

6.2.2 Adaptiv hukommelse

Et af de vigtigste elementer i TS er tabulisten, som løbende opdateres på baggrund af den adaptive hukommelse. Hukommelsesstrukturen, dvs. den adaptive hukommelse, kan opdeles i fire dimensioner; korttidshukommelse, langtidshukommelse, kvalitet og indflydelse.



Figur 19: De fire dimensioner i TS. Glover og Laguna (1997).

Et sammenspil mellem disse fire dimensioner bevirker, at TS er i stand til at udføre en grundig undersøgelse af løsningsområdet. Hukommelsen kan både være eksplicit og attributiv. En eksplicit hukommelse gemmer hele løsningen i tabulisten og benytter disse til senere at udvide den lokale søgning. Alternativt, bruges en attributiv hukommelse, som typisk reducerer nabo-området.

Når en algoritme tager højde for flere dimensioner, medfører det oftest langt bedre resultater, om end heuristikken kompliceres noget. For at en heuristik kan klassificeres som TS, er det dog ikke nødvendigt at anvende alle fire dimensioner, men nok med en simpel tabuliste. Dimensionerne korttids- og langtidshukommelsen er begge redskaber til modificering af nabo-området, på hver sin måde. Derfor er det væsentlig at skelne mellem disse.

Korttidshukommelse

En vigtig egenskab i TS er korttidshukommelsen, som implementeres i tabulisten. Korttidshukommelsen er den mest almindelige hukommelsesstruktur og som navnet antyder, holder den styr på løsningers attributter, som har ændret sig i de seneste antal iterationer. For at udnytte denne form for hukommelse, er det nødvendigt at mærke nogle udvalgte attributter som tabu-aktive. Hermed forhindres tidligere besøgte løsninger at tilhøre nabo-området $N^*(x)$, som er en delmængde af $N(x)$.

Tabulisten bliver restriktiv som konsekvens af, at der benyttes attributter frem for hele løsninger, hvilket fører til, at en større del af løsningsområdet udelukkes. Som nævnt tidligere løses denne problematik ved at indføre et aspirationskriterium.

Ofte ser man, at der er mulighed for at finde en endnu bedre løsning i nærheden af en tidligere fundet god løsning. I så fald anvendes en hukommelse kendt som en ”kritisk hændelse” (eng. "critical event"). Dette er en samlet oversigt over lokale minima eller eliteløsninger fundet i løbet af søgningen. Da hele løsningen er relevant, gemmes denne i oversigten til det formål, at starte nye søgninger fra disse.

Langtidshukommelse

Langtidshukommelsen har fokus på det lange sigt og koncentrerer sig mere om de generelle tendenser i søgningen. Denne hukommelse komplementerer informationen leveret af korttidshukommelsen. Kort og præcist angiver langtidshukommelsen, hvor ofte et element i en løsning er brugt i løbet af søgningen. Den består af en transitionsfrekvens og en residensfrekvens.

Til hver attribut tilhører der en transitionsfrekvens, som angiver antallet af iterationer, hvor en attribut har ændret sig (dvs. er enten blevet tilføjet eller slettet fra løsningen). Hermed holdes øje med de attributter, der er mest aktive, dvs. en høj frekvens angiver, at en attribut ofte tilføjes og slettes fra løsningen. I dette tilfælde kan det være bekvemt at øge attributtens tabulængde. En længere tabulængde vil øge sandsynligheden for at undersøge uudforskede områder mere intensivt. Frekvensopdateringen forekommer efter et gennemsøgt nabo-område.

En attribut har også en residensfrekvens, som angiver antallet af iterationer, hvor en attribut er blevet i sin position (f.eks. tilhører den aktuelle løsning). Residensfrekvensen indikerer, hvorvidt en attribut er attraktiv eller ikke, hvorfor en attraktiv attribut vil have en høj residensfrekvens i områder præget af gode løsninger og en lav i områder med dårlige løsninger. På baggrund af dette kan man påvirke søgningen mod de mere attraktive områder ved at straffe hhv. belønne en attribut.

To andre strategier tilknyttet langtidshukommelsen er hhv. intensiverings- og diversificeringsstrategier. En intensiveringsstrategi sørger for at undersøge områder, hvor man forventer at finde løsninger af høj kvalitet, hvorimod en diversificeringsstrategi tilskynder søgeprocessen til at undersøge uudforskede regioner. Ofte benytter man også en diversificeringsstrategi i de områder, hvor løsningskvaliteten er dårlig. For at opnå en effektiv søgning, er det godt at veksle mellem de to søgningsstrategier, idet det giver mulighed for både at rette fokus mod potentielle områder og få dækket hele løsningsområdet.

Kvalitet og indflydelse

Kvalitet og indflydelse er dimensioner, der ikke så let kan indarbejdes i en given TS heuristik. Det kan virke lidt overflødigt at opdele dem i to separate dimensioner, idet kvalitet i en vis forstand kan bestragtes som en form for indflydelse.

Kvalitetsdimensionen sammenligner de forskellige lokale minima med det formål at identificere nogle bestemte karakteristika, der er fælles for gode løsninger. I praksis bliver kvaliteten en form for incitamentsbaseret læring, hvor en handling, der fører til gode eller dårlige løsninger, hhv. belønnes eller straffes. På denne måde sørger man for at pege søgningen i retning af de mere lovende regioner.

Indflydelse angiver hvilken effekt de trufne valg har haft på såvel kvaliteten som på strukturen af løsningen. Nogle træk fører til en drastisk ændring i løsningen, som kan være en hjælp til at flygte fra lokale minima. Dimensionen sørger også for at gemme informationen om, hvor langt man har bevæget sig fra den tidligere løsning.

6.2.3 Algoritmen i VRP-sammenhæng

Der er foreslået adskillige TS metoder til at løse VRP. Metoderne adskiller sig på måden nabo-området defineres for en løsning x. Man kan definere nabo-områderne ved brug af en "λ-interchange" mekanisme, for $\lambda = 2$. Vi mindes at denne inkluderer 2-opt træk, kundeomfordeling samt kundeudveksling mellem to ruter. Ved at udføre et træk på en VRP-løsning, undersøges der ikke blot ringere løsninger, men også ikke-mulige løsninger mht. kapacitetsbegrænsninger eller andre betingelser. Den sidstnævnte problematik behandles ved at benytte en modificeret objektfunktion, som består af et eller flere straf-led, afhængig af hvor mange bibetingelser der skal overholdes.

Sammenhængen mellem principperne i TS algoritmen og VRP kan i første omgang virke en smule forvirrende, men det hele falder på plads i det øjeblik termet "det definerede nabo-område" forstås. Lad os for enkelthedens skyld se på et træk, hvor to kunder udveksles mellem to ruter. Såfremt én af kunderne er tabuaktiv, er det ulovligt at udveksle denne med kunden fra den anden rute i et på forhånd specificeret antal iterationer. I det tilfælde hvor gennemførslen af udvekslingen fører til en bedre løsning end den hidtil bedste, er det tilladeligt at bryde tabuet (aspirationskriterium). Man fortsætter søgningen enten ved at intensivisere eller diversificere indtil et stopkriterium er opfyldt.

Inden vi afslutter afsnittet om TS, gives her et overblik over fremgangsmåden i Glover og Laguna (1997):

1. Find en initial løsning $x_0 \in X$, og sæt $x^{nu} = x^{bedste} = x_0$. Initialisér hukommelsen.
2. Kortsigtskomponent (Nabosøgningen)
   2.1. Hvis et slutkriterium (eks. efter et bestemt antal iterationer, ingen mulige forbedringstræk, ingen ændring i x^{bedste} efter mange iterationer i træk) er tilfredsstillet, gå til trin 3 ellers fortsæt til 2.2.
   2.2. Vælg det bedste $x^{næste} \in N\left(x^{nu}\right)$ således at $x^{næste}$ ikke er tabu eller at aspirationskriteriet er tilfredsstillet[22].
   2.3. Flyt fra x^{nu} til $x^{næste}$, dvs. sæt $x^{nu} = x^{næste}$
   2.4. Hvis x^{nu} er bedre end x^{bedste}, så sættes $x^{nu} = x^{bedste}$
   2.5. Opdatér kortsigtshukommelsen (tabu klassificeringen), langsigts hukommelsen og/eller critical event memory, gå tilbage til trin 2.1.
3. Langsigtskomponent (genstart)
   3.1. Hvis slutkriterium er tilfredsstillet, så stop ellers gå til 3.2.

[22] Med det bedste $x^{næste}$, menes det bedste træk og ikke den bedste objektfunktionsværdi sammenlignet med den hidtil bedste.

3.2. Ved brug af langsigtshukommelse og/eller ”critical event” hukommelse, find et nyt startpunkt x^{mu} og vend tilbage til trin 2.

6.3 Large Neighborhood Search

Lokal søgning er i stand til at undersøge et stort antal af løsninger på kort tid, men evner kun at ændre løsningen en smule i hver iteration. Ifølge Røpke og Pisinger (2005) kan det således være svært at flytte fra én lovende region til en anden, når det pågældende problem har en hel del strenge bibetingelser. Én løsning er at lade søgningen besøge ikke-mulige løsninger ved at relaksere nogle af bibetingelserne. Røpke og Pisinger tager imidlertid en anden tilgang, hvor de i stedet for små standard træk benytter sig af store træk, der potentielt kan omarrangere 30-40 % af kunderne i en enkelt iteration. Dette bevirker at beregningstiden sættes voldsomt i vejret, idet gennemførslen og den efterfølgende evaluering tager længere tid sammenlignet med små træk.

Det er baggrunden for "Large Neighborhood Search" (LNS), som først blev introduceret af Shaw (1998). LNS forbedrer en initialløsning gradvist ved skiftevis at ”ødelægge” og ”reparere” løsningen, som reelt blot betyder, at vi fjerner kunder fra løsningen og dernæst genindsætter dem på en bedre måde. Ødelæggelses- og reparationsmetoderne definerer således nabo-området som mængden af løsninger, der kan nås med sådan en operation.

Ødelæggelsesmetoden er en vigtig del af LNS heuristikken. Det vigtigste valg der skal foretages når en ødelæggelsesmetode skal implementeres, er graden af ødelæggelsen. Hvis man vælger at ødelægge en lille del af løsningen, kan heuristikken have et problem med at udforske løsningssummet. Modsat, ødelægges en vældig del af løsningen, vil LNS næsten havne i en nærmest konstant re-optimering. Dette kan være både tidskrævende og resultere i en dårlig løsning, afhængig af hvorledes løsningen repareres. Shaw valgte en gradvis stigning i ødelæggelsesgraden, hvorimod Røpke og Pisinger valgte ødelæggelsesgraden tilfældigt i hver iteration, ved at vælge graden fra et specifikt interval afhængig af problemstørrelsen. Ødelæggelsesmetoden skal også vælges således, at hele løsningsrummet kan nås og ikke mindst de mest interessante områder, hvor man forventer at finde en global løsning.

Der kan også vælges forskellige typer af reparationsmetoder. Valget ligger f.eks. mellem en eksakt algoritme og en heuristik. Det er oplagt at en eksakt reparationsmetode vil være langsommere end en heuristik, men vil potentielt føre til kvalitetsløsninger på kun få iterationer. Ønsker man i stedet en høj grad af diversificering, er en eksakt reparationsmetode ikke attraktiv, eftersom det kan være svært at forlade et lokalt minimum, med mindre en stor del af løsningen ødelægges i hver iteration.

Herunder opsummeres algoritmen i et antal trin:

1. Bestem en mulig initial løsning x.
2. Sæt $x = x^b$.
3. Bestem $x^m = r(d(x))$.
4. Hvis x^m accepteres, sæt $x = x^m$.

5. Hvis $c(x^m) < c(x^b)$, sæt $x^b = x^m$.
6. Gentag trin 3-6 indtil et stopkriterium er mødt.
7. Returnér den bedste løsning.

x^b er den bedste løsning observeret under søgningen, x er den aktuelle løsning og x^m er en midlertidig løsning, som enten kan kasseres eller forfremmes til værende den nye aktuelle løsning. Funktionen $d(\cdot)$ er ødelæggelsesmetoden mens $r(\cdot)$ er reparationsmetoden. Dvs. $d(x)$ returnerer en kopi af x, som er delvist ødelagt, mens funktionen $r(\cdot)$ af den delvist ødelagte løsning returnerer en (repareret) mulig løsning.

Algoritmen initialiseres med en simpel konstruktionsheuristik, hvis løsning gemmes som den hidtil bedste løsning. I trin 3 anvendes først en ødelæggelsesmetode og derefter en reparationsmetode for at opnå en ny løsning x^m. Den nye løsning evaluerer, hvorvidt denne løsning bør sættes til den nye aktuelle løsning eller om den bør afvises. Accept-funktionen kan implementeres på forskellige måder, men det simpleste vil være kun at acceptere løsninger der er bedre. Trin 5 tjekker om den nye løsning er bedre end den hidtil bedste løsning, i så fald opdateres den bedste løsning. Algoritmen gentages fra trin 3, medmindre et stopkriterium er mødt, i hvilket tilfælde den bedste løsning returneres.

I Shaw's repræsentation af algoritmen tillader accept-metoden (trin 4) kun løsninger, der fører til en forbedring. Man har dog med tiden valgt at benytte andre former for acceptkriterier. Røpke og Pisinger (2005) har bl.a. benyttet acceptkriteriet fra simuleret udglødning. Ved implementeringen af dette acceptkriterium kan man se LNS heuristikken som en standard SA heuristik.

6.4 Genetiske algoritmer

Teorien om naturlig udvælgelse blev forslået af Charles Darwin i 1859. Ifølge teorien har individer med særdeles gode karakteristika en større sandsynlighed for at overleve, formere sig og dermed videregive egenskaber til deres afkom. Derimod vil individer med mindre gunstige egenskaber gradvist forsvinde fra populationen. I naturen gemmes den genetiske arv i kromosomerne, som er opbygget af gener. De tilhørende karakteristika for ethvert organisme kontrolleres af generne, der videregives til afkommet når organismer parrer. En gang i mellem kan der forekomme en mutation, som er årsag til en ændring i kromosomet. Grundet den naturlige udvælgelse sker der i gennemsnit en gradvis forbedring i populationen, idet antallet af individer med gunstige egenskaber stiger.

Idéen bag genetiske algoritmer (GA) er inspireret af Darwins teori og er oprindelig udviklet af John Holland (1975), mens den praktiske anvendelse blev demonstreret af De Jong (1975) og Goldberg (1989). Idéen bag GA er at modellere Darwins teori. Lad en mængde individer (eller løsninger) udgøre en population og reproduktion hhv. mutation af denne population ske ved en "crossover"-operator og en mutationsoperator. I efterligning af den naturlige udvælgelse vælges nogle forældreløsninger af en udvælgelsesprocedure, der herefter benyttes til reproduktion eller, med andre ord, generering af nye løsninger/afkom. I slutningen af hver iteration dannes en ny generation, som svarer til den tidligere population inklusiv nye afkom. Oftest ønskes en konstant population, hvorfor udvælgelsesprocessen

samtidig kasserer dårlige løsninger i forbindelse med tilføjelsen af nye. Kasseringen sker som en evaluering af løsningerne mht. en fitness-værdi, eftersom denne viser, hvilken af kandidaterne der er stærkest. Brugeren skal i denne forbindelse foretage en række valg inden algoritmen kan sættes i gang, disse inkluderer:

1. Repræsentation af løsningsrummet.
2. Dannelse af initialpopulation.
3. Definition af fitness-værdi.
4. Udvælgelseskriterium for forældreløsninger.
5. Reproduktionskriterium.
6. Mutationskriterium.

Vi vil i det følgende komme ind på hver af disse valg og eksempler på sådanne. Det er altså klart, at GA er adaptive læringsheuristikker, men adskiller sig fra andre heuristikker via følgende karakteristika:

- GA arbejder med kodede løsninger i stedet for løsningerne selv, hvorfor en god og effektiv repræsentation af løsningen er påkrævet.
- Søgningsproceduren starter fra en mængde løsninger i modsætning til de forrige metaheuristikker, der starter søgningen fra en enkel løsning og bevæger sig frem ved en gennemsøgning af et nabo-område omkring den aktuelle løsning. Den genetiske procedure reducerer dermed sandsynligheden for at ende i et lokalt minimum.

GA for VRP kan let tilpasses forskellige problemer og udvidelser, derfor findes mange forskellige versioner af algoritmen, afhængig af hvilket problem der skal løses. Der er f.eks. adskillige måder, hvorpå en population kan vedligeholdes samt adskillige operatorer der påvirker den.

6.4.1 Repræsentation af løsningsrummet

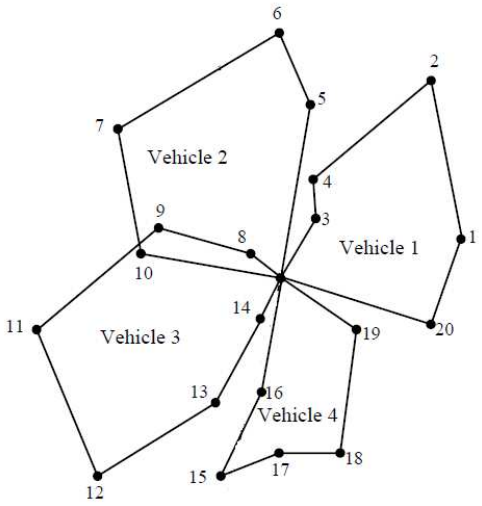
En god repræsentation af en VRP-løsning skal selvfølgelig angive antal køretøjer, hvilke kunder der er tildelt de enkelte køretøjer samt i hvilken rækkefølge kunderne serviceres. I GA ses ofte en repræsentation af løsningen, som en streng der indeholder binære variable, hvilket naturligvis ikke er passende for et VRP. Det er en let sag at specificere antallet af køretøjer samt hvilke kunder der tilhører dem, men problemer opstår når serviceringsrækkefølgen skal angives. Der er forskellige måder at repræsentere løsningen på, men vi har af hensyn til forståelsen valgt den mest simple, som dog ikke angiver hvilken rækkefølge kunderne serviceres i, se figuren herunder.

Kunde:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Køretøj:	1	1	1	1	2	2	2	3	3	2	3	3	3	3	4	4	4	4	4	1

Tabel 2: Et eksempel på en VRP-løsning, repræsenteret som et kromosom.

Ovenstående kromosom er en løsningskandidat, der består af fire køretøjer og 20 kunder. Repræsentationen specificerer for hver kunde det køretøj den er tildelt. Vi kan således sige, at givet n kunder og m køretøjer har kromosomet for en løsning form af en streng af længde n, og hver kunde har

én værdi i intervallet $[1;m]$. I modsætning til andre repræsentationer til VRP specificerer ovenstående ikke eksplicit, hvilken rute hvert køretøj skal følge. Men da man kender hvilke kunder der er tildelt de enkelte køretøjer, herunder den totale tilbagelagte afstand samt niveauer af enhver overtrædelse af begrænsningerne, er det en forholdsvis simpel sag at bestemme en sådan som et TSP. En sådan fuldstændig løsning muliggør bestemmelsen af en fitness-værdi, tilknyttet hver enkelt løsning i populationen. Den grafiske illustration af løsningen fra tabel 2 ses herunder:



Figur 20: Løsningen fra tabel 2 illustreret grafisk, efter der for hvert køretøj er løst et TSP. Baker og Ayechew (2003).

Der foretages to trin for at vedligeholde strukturen så vidt som muligt. Først sorteres kunderne og nummereres således at de i træk kommende kunder sandsynligvis bliver serviceret af det samme køretøj. Problemer hvor kunderne er tilfældig fordelt omkring depotet, sorteres efter stigende orden i polære vinkel. For det andet, forsøger man at nummere køretøjerne således at ethvert køretøj i, opererer i nogenlunde samme region for alle medlemmer i populationen. Herefter kan GA proceduren anvendes.

6.4.2 Dannelse af initialpopulation

Dannelsen af en initialpopulation er af stor betydning for udførelsen af GA, da den indeholder det meste af det "materiale" den endelige løsning er lavet af. En initialpopulation med rimelig strukturerede løsninger vil således forventes at udvikle sig til løsninger af høj kvalitet i relativt få antal generationer. Men en mulig ulempe er, at sådanne løsninger hurtigt kan blive dominerende, og en population vil mangle den mangfoldighed, der er nødvendig for at opnå næsten-optimale løsninger. Dette argumenterer mod at anvende for gode løsninger i starten.

To metoder er oftest blevet anvendt til dannelsen af en initialpopulation. Den første er baseret på Gillett og Miller's sweep algoritme og den anden på Fisher og Jaikumar's generaliserede assignmentproblem.

Forskere har også gjort fremskridt med parallel GA, som bruger flere populationer eller delpopulationer, der udvikler sig uafhængigt af hinanden Vi har dog valgt kun at beskæftige os med én population for at bevare enkelheden.

Populationsstørrelsen M har en betydelig effekt på GA's performance og påvirker beregningstiden samt den rate, hvormed GA konvergerer mod en endelig løsning. En for lille population kan give en dårlig performance, idet der ikke er variation nok i løsningerne. En større population giver derimod langt bedre performance og undgår en alt for tidlig konvergens, men forlænger beregningstiden betydeligt.

Man skal også overveje, om man vil arbejde med en konstant eller dynamisk population. Vi vælger at betragte det konstante tilfælde.

6.4.3 Definition af fitness-værdi

Løsningskvaliteten måles ved en fitnessværdi f_i, som f.eks. kan være objektfunktionsværdien for den pågældende VRP-løsning, men der eksisterer også mange andre definitioner. Med henblik på at udføre en naturlig udvælgelse, skal hvert individ evalueres i form af dennes fitnessværdi, hvilket muliggør sammenligning på tværs af løsningerne i populationen. Udvælgelse af individer til reproduktion og overlevelse har en afgørende indflydelse på effektiviteten af GA, idet en grådig udvælgelse fører til en alt for tidlig konvergens. Derfor er det af stor betydning hvilken evalueringsfunktion der vælges

Evalueringsfunktionen er normalt givet for en mulig løsning, f.eks. er den mest åbenbare fitnessværdi for et TSP eller CVRP den totale omkostning eller afstand. Drejer det sig om ikke-mulige løsninger findes andre muligheder. Man kan f.eks. afvise sådanne løsninger eller blot straffe dem. En afvisning af ikke-mulige løsninger vil naturligvis simplificere algoritmen, men oftest kan ikke-mulige løsninger lede til bedre løsninger i løbet af processen. Derimod kan inkluderes et straf-led, som f.eks. er afhængig af hvor meget kapacitetsbegrænsningen er overtrådt. I de første iterationer kan det være en fordel at tillade ikke-mulige løsninger, hvor straffen antages at være ubetydelig, hvorimod den er mere dominerende i slutningen og dermed tvinger den sidste løsning til at være mulig.

6.4.4 Udvælgelseskriterium for forældreløsninger

En udvælgelsesprocedure er nødvendigt for udvælgelse af individer til både reproduktion og overlevelse. Herunder beskrives kort tre metoder der efterligner den naturlige udvælgelse, hvor den stærkeste løsning har en større sandsynlighed for at producere nye løsninger og overleve end en ringere løsning.

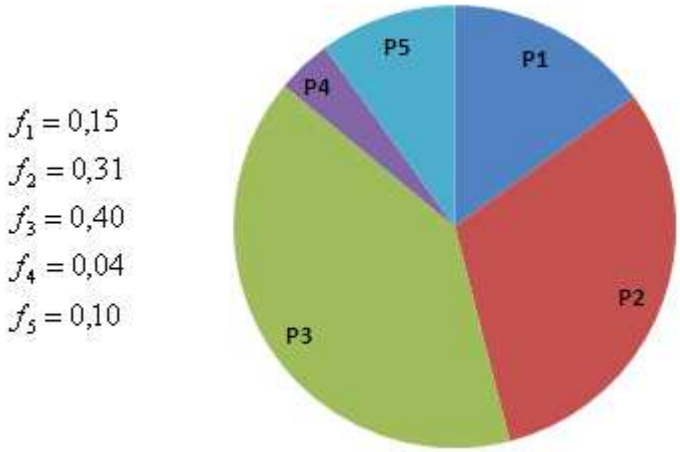
Udvælgelseskriteriet er imidlertid styret af et såkaldt selektivt pres. En stigning i det selektive pres mindsker populationsvariationen og lignende for et fald. Hvis populationen bliver for homogen, er mutation reelt den eneste operator, der dernæst kan variere populationen.

Det selektive pres er altså et mål for, hvor ofte de bedste løsninger vælges sammenlignet med de dårlige. Et stærkt selektivt pres betyder, at de bedste har en stor sandsynlighed for at blive valgt og dermed oftere, hvorimod de dårligere sjældent vælges. Har man et svagt selektivt pres, får de dårligere løsninger en større sandsynlighed for at blive valgt.

Der er altså tale om en balancering mellem på den ene side gode løsninger og en for hurtig konvergering, samt løsninger af varierende kvalitet men en langsom konvergering.

Roulette Wheel metoden

Det første eksempel på en udvælgelsesmetode kaldes ”Roulette Wheel”. Her er sandsynligheden for at vælge en løsning direkte proportional med dennes fitness-værdi, således at løsninger med en højere fitness-værdi, har en større sandsynlighed for at blive valgt. Et illustrativt eksempel kan ses herunder.



Figur 21: Roulette Wheel metoden.

Ovenstående figur viser en population bestående af fem løsninger. Fitness-værdien for hver løsning sættes i forhold til den totale fitness-værdi for samtlige løsninger, hvorved man får et sæt sandsynligheder, f_1 til f_5 .. Hvis man derfor forestiller sig en pointer mens hjulet drejes, vil pointeren hyppigst standse inden for P3, som har den største sandsynlighed.

Ulempen ved metoden er, at fitness-værdien benyttes direkte. Derfor vælges løsninger med en lav fitness-værdi ift. de andre meget sjældent.

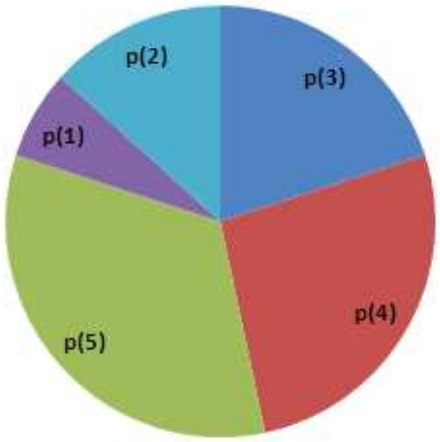
”Ranking”-metoden

I "Ranking"-metoden sorteres alle løsninger i stigende orden efter deres fitnessværdi. For at udvælge en løsning, associeres hver løsning i listen med en sandsynlighed, der er afhængig af denne rangordning og ikke fitness-værdien. Sandsynlighedsfunktionen defineres som følger:

(6.1) $$p(k) = \frac{2k}{M(M+1)},$$

hvor k betegner den k 'te løsning i rangordningen og M svarer til populationsstørrelsen. Den værste løsning svarer til $k = 1$ og den bedste svarer til $k = M$. Herunder illustreres hvordan sandsynlighedsfordelingen af selvsamme 5 løsninger, som i ”roulette wheel”-metoden, ser ud efter anvendelse af "ranking"-metoden.

$$f_1 = 0{,}15 \rightarrow k = 3:\ p(3) = 0{,}2$$
$$f_2 = 0{,}31 \rightarrow k = 4:\ p(4) = 0{,}266667$$
$$f_3 = 0{,}40 \rightarrow k = 5:\ p(5) = 0{,}333333$$
$$f_4 = 0{,}04 \rightarrow k = 1:\ p(1) = 0{,}066667$$
$$f_5 = 0{,}10 \rightarrow k = 2:\ p(2) = 0{,}133333$$



Figur 22: Her ses et eksempel på "ranking"-metoden. Vi har valgt at basere eksemplet på de samme fitness-værdier som i figur 3, for at illustrere forskellen i den andel de to metoder tildeler løsningerne. Bemærk, at de løsninger, der i figur 21 havde en minimal sandsynlighed, vil i "ranking"-metoden blive valgt lidt oftere på bekostning af dem, som før havde en høj sandsynlighed.

Fordelen ved denne metode er, at den forhindrer meget "fitte" løsninger i at opnå dominans fra starten, hvilket kan resultere i dårlige resultater som tidligere beskrevet.

Turneringsmetoden

I turneringsmetoden (eng. "tournament method") dannes i første omgang to delmængder bestående af S tilfældige løsninger fra populationen, hvor der mindst skal være to løsninger i hver delmængde, for derved at kunne udføre en sammenligning. I hver delmængde konkurrer løsningerne nu mod hinanden for at blive valgt som én af de to forældreløsninger Det selektive pres påvirker udvælgelsesmetoden i den forstand, at jo højere selektivt pres, des flere løsninger indgår i delmængderne og jo bedre er forældreløsningerne. Samme procedure kan anvendes for eliminering af løsninger i populationen.

6.4.5 Reproduktionskriterium

Et reproduktionskriterium defineres som en såkaldt "crossover", der kombinerer forældreløsninger på en bestemt måde og derigennem genererer to afkom (nye løsninger). De to afkom får dermed nogle egenskaber fra deres forældre og egenskaberne videregives til de næste generationer. Det er dog ikke muligt at producere nye egenskaber, hvorfor fremtidige generationers egenskaber altid vil stamme fra tidligere generationer.Der findes adskillige sådanne crossovers, vi vil dog i det følgende blot give et par eksempler.

"Simple random crossover"

Det simple crossover starter med to forældreløsninger, hvorefter et tilfældigt "cut" vælges, således at de to forældreløsninger deles op i to dele (se tabel 3). Den lodrette linje mellem kunde 7 og 8 repræsenterer "cuttet". Der bliver genereret to afkom O1 og O2, hvor O1 sammensættes af kunderne i P1 efter "cuttet" og kunderne i P2 før "cuttet", og omvendt for O2.

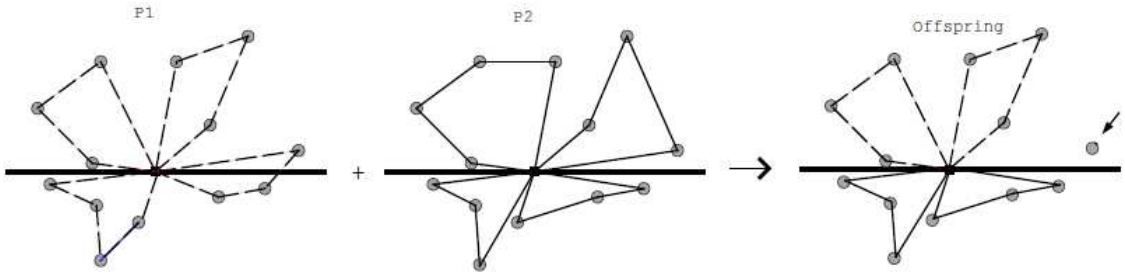
Kunder	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1:	1	1	1	1	2	2	2	3	3	2	3	3	3	3	4	4	4	4	4	1
P2:	1	1	2	2	1	2	2	2	2	3	3	3	3	3	4	4	4	4	1	4
Kunder	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
O1:	1	1	2	2	1	2	2	3	3	2	3	3	3	3	4	4	4	4	4	1
O2:	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	1	4

Tabel 3: Illustration af det simple crossover. Afkom O1 er genereret fra P1's højre side og P2's venstre side, mens O2 er sammensat af P1's venstre halvdel og P2's højre halvdel.

Bemærk, når et afkom dannes, tages der ikke højde for om kapacitetsbegrænsningen er overholdt for alle køretøjer. Den ene indsatte halvdel i afkommet kan have den ulempe, at den kun ser på første og/eller sidste kunde i en delrute, og dermed ikke nødvendigvis indeholder hele ruten. Såfremt P1 og P2 er gode løsninger og muligvis har næsten eller helt fyldte køretøjer, vil operatoren højst sandsynligt generere en ikke-mulig løsning. Eftersom ikke-mulige løsninger straffes bliver algoritmen ineffektiv, hvis den simple crossover ofte genererer ikke-mulige løsninger. For mange tilfældigheder kan gøre det svært at sprede gode karakteristika i populationen, ,men uden en vis grad af tilfældighed bliver variationen i populationen meget lille.

Andre crossovers

En anden crossover er en "horizontal line crossover". Metoden prøver at kombinere de to forældreløsninger, P1 og P2, mere ligeligt i afkommet. Der tegnes en horisontal linje igennem depotet som opdeler løsningen i to dele. Ruterne i P1, over den horisontale linje indsættes uberørt i afkommet sammen med ruterne i P2 under linjen. En illustration af dette ses herunder.



Figur 23: Figur: tekst En illustration af en "horizontal line crossover": Ruterne i P1 over linjen og ruterne i P2 under linjen indsættes uberørt i afkommet. Det skal bemærkes at der er en kunde i afkommet der ikke tilhører en rute. Bjarnadóttir (2004).

Som det ses sker det nogle gange at linjen skærer en rute, hvilket efterlader nogle kunder uden en tilknyttet rute. Problemet løses ved at benytte en heuristik, der tildeler disse kunder en rute og rekonstruerer denne.

Én af ulemperne ved metoden er, at den afhænger af, hvor depotet er placeret i forhold til kunderne. Hvis depotet er placeret relativt højt eller lavt i en graf, vil ruterne hovedsageligt komme fra én af forældreløsningerne. På samme vis, hvis kunderne ligger omkring linjen, vil de fleste ruter blive afskåret af linjen.

I en ”2-point”-crossover dannes 2 tilfældige "cuts" i kromosomet. I det ene afkom indsættes den del af P1, som er til venstre for første "cut" og højre for andet "cut", samt den del af P2, der er mellem første og andet "cut". Omvendt for det andet afkom.

6.4.6 Mutationskriterium

Mutationskriteriet har til formål at flygte fra lokale minima og anvendes på en enkel løsning ad gangen. Den foretager små tilfældige ændringer i løsningen, hvilket medfører en gradvis tilføjelse af nogle nye karakteristika i populationen, som et "crossover" ikke ville kunne give. Et eksempel er en såkaldt "simple random mutation", der muterer løsningen ved kun at flytte en enkelt kunde (se nedenstående tabel).

Simpel mutation																				
Kunder	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Afkom (før)	1	1	1	1	**2**	2	2	2	2	3	3	3	3	3	4	4	4	4	4	1
Afkom (efter)	1	1	1	1	**3**	2	2	2	2	3	3	3	3	3	4	4	4	4	4	1

Tabel 4: Et eksempel på en simpel mutation, hvor kunde 5 før mutationen blev serviceret af køretøj 2 og efter mutationen serviceres af køretøj 3.

I en mere effektiv mutation byttes to tilfældige kunder fra hver sin rute. I ovenstående eksempel svarer dette til, at kunde 2 og 15 efter mutationen vil tilhøre hhv. køretøj 4 i stedet for 1 og køretøj 1 i stedet for 4.

Det er vigtigt ikke at foretage for mange ændringer på én gang eller for ofte, fordi algoritmen mere eller mindre minder om en tilfældig søgning.

6.4.7 Opstilling af algoritmen

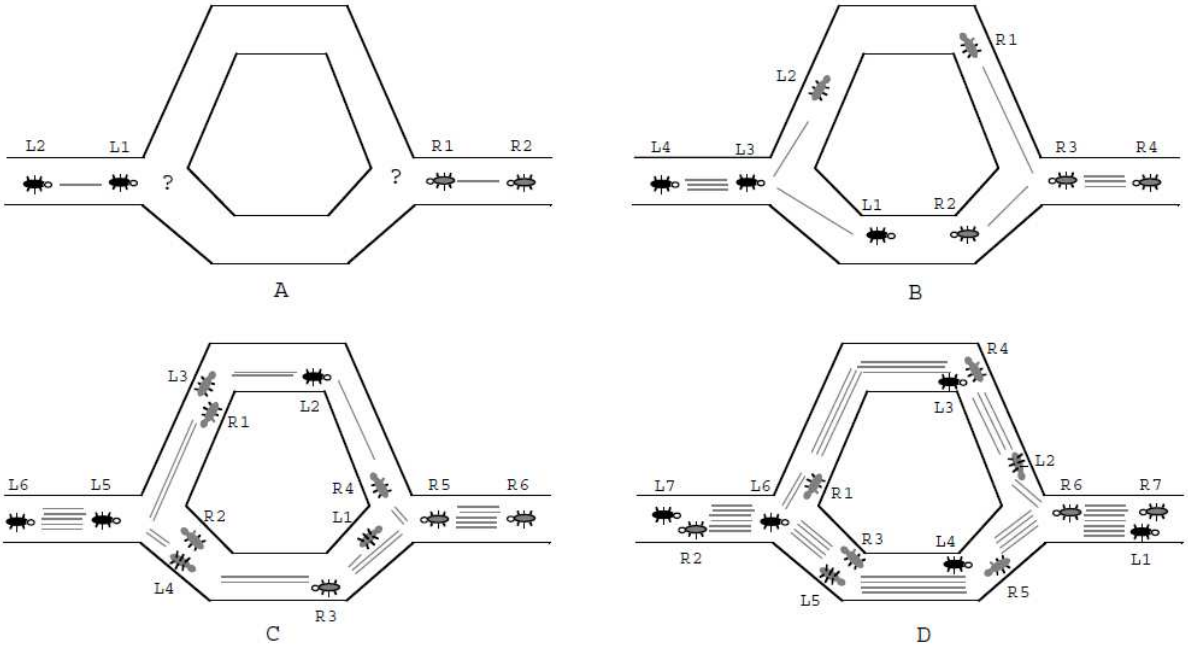
Herunder har vi opsummeret GA i et antal trin:

1. Generér en initialpopulation
2. Bestem og evaluér fitness-værdien for hver løsning i populationen.
3. Dan en ny population ved at gentage følgende trin.
   - 3.1. Vælg to forældreløsninger fra populationen på baggrund af et udvælgelseskriterium.
   - 3.2. Producér to afkom fra de to forældreløsninger ved brug af et crossover.
   - 3.3. Foretag en mutation på de to afkom.
   - 3.4. Evaluér afkommenes fitness-værdi.
   - 3.5. Vælg det bedste afkom.
   - 3.6. Hvis adgangskriteriet er tilfredsstillet for det valgte afkom:
     - 3.6.1. Vælg et populationsmedlem der skal erstattes.
     - 3.6.2. Afkommet indgår i populationen og det valgte medlem kasseres.

4. Gentag trin 2-3 indtil et stopkriterium er mødt.

6.5 Ant Colony Optimization

Ant colony optimization (ACO) er en metaheuristik, der bygger på hvordan myrer formår at finde de korteste ruter fra fødekilden til deres tue, ikke ved visuelle midler, men ved at lægge et spor af feromon, et stof der udskilles langs en myres rute og som andre myrer udnytter til at finde vej. Nedenstående figur illustrerer hvorledes sådan en korteste bliver dannet. I starten har myrerne lige stor sandsynlighed for at tage den venstre eller højre bane, men pga. den venstre banes kortere længde, vil myrerne hurtigere komme frem til fødekilden og/eller hurtigere tilbage til tuen, såfremt vi antager en gennemsnitlig konstant hastighed. Så mens de andre myrer fedter rundt i den højre bane, er de første allerede kommet igennem den venstre (måske adskillige gange), og dermed lagt et kraftigere spor af feromon end det tilsvarende antal myrer i den højre bane. På dette tidspunkt begynder det kraftigere spor af feromon at tiltrække andre myrer til også at vælge den venstre bane, som til sidst vil være den eneste benyttede bane.



Figur 24: Illustration af hvorledes styrken af feromon-sporet (repræsenteret af antal grå linjer) udvikler sig over tid fra A til D. Dorigo og Gambardella (1997).

Denne adfærd har ligget til inspiration for adskillige ACO algoritmer, bl.a. "ant system", en algoritme først designet af Dorigo (1992) til at løse TSP. Denne er siden hen blevet udbygget af Dorigo og Gambardella (1997). Da de to algoritmer oftest ligger til grund for ACO metaheuristik for VRP, vil vi i det følgende kort redegøre for disse.

6.5.1 "Ant system" for TSP

I "ant system" (AS) er hver kant associeret med en nærhedsfaktor η_{ij} og et feromonspor τ_{ij}. Nærhedsfaktoren definers som den inverse længde af kanten mellem i og j, og er dermed en konstant gennem hele problemet, mens feromonsporet ændres dynamisk. Ideen her er, at m myrer placeres tilfældigt rundt om på ruten og konstruerer nu parallelt hver deres rute ved en sandsynlighedsbaseret nærmeste-nabo heuristik. Myre k går således videre til den node j blandt mængden af mulige noder fra i, N_i^k, hvis kant scorer mest på hhv. afstand og mængden af feromon. Når alle myrer har dannet en komplet rute ψ_k der inkluderer samtlige noder, opdateres feromonsporet globalt: Alle kanter mister en andel af deres feromon og hver myre lægger en mængde feromon på de kanter der indgår i dens rute. På denne vis fremhæves de kanter der indgår i de fleste ruter, mens de kanter, der ikke bruges nok, langsomt udgår fra enhver løsning. Processen gentages indtil et stopkriterium er opfyldt.

Heuristikken der benyttes til at konstruere ruterne er bygget op om (3.2.1), der angiver sandsynligheden for at en myre k bevæger sig fra node i til j:

(6.2)
$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij} \cdot [\eta_{ij}]^\beta}{\sum_{u \in N_i^k} \tau_{iu} \cdot [\eta_{iu}]^\beta} & ,hvis\ j \in N_i^k \\ 0 & ,ellers \end{cases}$$

hvor $\beta > 0$ er en brugervalgt parameter der bestemmer den relative betydning af afstand i forhold til feromonmængden. Ved at bestemme sandsynligheden som et produkt af feromonmængden og den inverse længde af den pågældende kant, foretrækkes dermed kanterne med den største mængde af feromon og den mindste længde. Når m ruter er konstrueret, lad følgende betegne den globale opdateringsregel:

(6.3)
$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

hvor

(6.4)
$$\Delta\tau_{ij}^k = \begin{cases} [L_k]^{-1} & ,hvis (i,j) \in \psi_k \\ 0 & ,ellers \end{cases}$$

Her betegner ρ andelen af feromon der bortfalder ved opdateringen og L_k er længden af myre k's rute. Såfremt en bestemt kant indgår i adskillige myrers ruter, får denne således en andel feromon fra hver af myrerne. De kanter der ikke indgår i en løsning vil langsomt miste feromon i forhold til ρ.

6.5.2 "Ant colony system" for TSP

"Ant colony system" (ACS) er en udvidelse til det oprindelige AS, men adskiller sig på 3 områder: 1) Feromonspor bruges nu både til udforskning af nye kanter (3.2.1) med sandsynlighed $1-q_0$ og til udnyttelse af de hidtil bedste kanter til løsningen med sandsynlighed q_0. 2) Kun kanterne i den korteste rute tilføjes feromon i den globale opdateringsregel. 3) Modsat den globale opdatering der sker efter konstruktionen af m ruter, tilføjes nu en lokal opdateringsregel i selve konstruktionsfasen, hvor feromonsporet langsomt forsvinder når en kant benyttes. Dermed vil de første myrers ruter ligge forholdsvis tæt på foregående korteste rute, mens senere myrer vil udforske nye kanter, idet feromonsporet på de bedste kanter på dette tidspunkt vil være reduceret. Dermed får vi flere forskelligartede løsninger i stedet for kun at søge i et snævert nabo-område omkring den foregående korteste rute.

Opsummerende ser algoritmen således ud:

1) Fordel myrerne tilfældigt ud på ruten.
2) Lad en myre konstruere en fuldstændig rute på baggrund af en sandsynlighedsbaseret nærmeste-nabo heuristik, og mellem hver tilføjet node opdateres feromonsporet lokalt.
3) Vælg en myre der endnu ikke har dannet en rute og fortsæt fra trin 2 indtil samtlige myrer har konstrueret en rute.
4) Feromonsporet opdateres globalt ved at benytte den bedste løsning og algoritmen itereres fra trin 1 indtil et stopkriterium er mødt.

Lad os se på reglerne der definerer konstruktionsfasen, den globale opdatering og den lokale opdatering. Den nærmeste nabo j til en node i defineres som:

(6.5)
$$j = \begin{cases} \arg\max_{u \in N_i^k} \left\{ \tau_{iu} \cdot \left[\eta_{iu} \right]^{\beta} \right\} & ,hvis\, q < q_0 \\ J & ,ellers \end{cases}$$

hvor q er et tilfældigt tal genereret fra en standard ligefordeling i intervallet $]0,1[$, q_0 er sandsynligheden for at bruge den hidtil bedste kant til at vælge et j, og J er en stokastisk variabel der returnerer en node fra sandsynlighedsfordelingen i (6.2). Den øverste del af (6.5) svarer til udnyttelsen af *a priori* viden om den bedste kant fra i, mens den nederste til udforskningen af nye kanter baseret på en sandsynlighed, hvor den største sandsynlighed tillægges kanter med den bedste vægtning af hhv. afstand og feromon.

Efter m ruter er konstrueret, påbegyndes en global opdatering som følger:

(6.6)
$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho \Delta \tau_{ij}$$

hvor

(6.7) $$\Delta\tau_{ij} = \begin{cases} \left[L_{gb}\right]^{-1}, hvis(i,j) \in \psi_{gb} \\ 0, ellers \end{cases}$$

Her betegner ψ_{gb} og L_{gb} hhv. den globalt bedste rute og længden af denne. Dermed får kun de kanter der er medtaget i den bedste løsning et ekstra tilskud af feromon, mens de øvrige mister en andel af deres akkumulerede mængde.

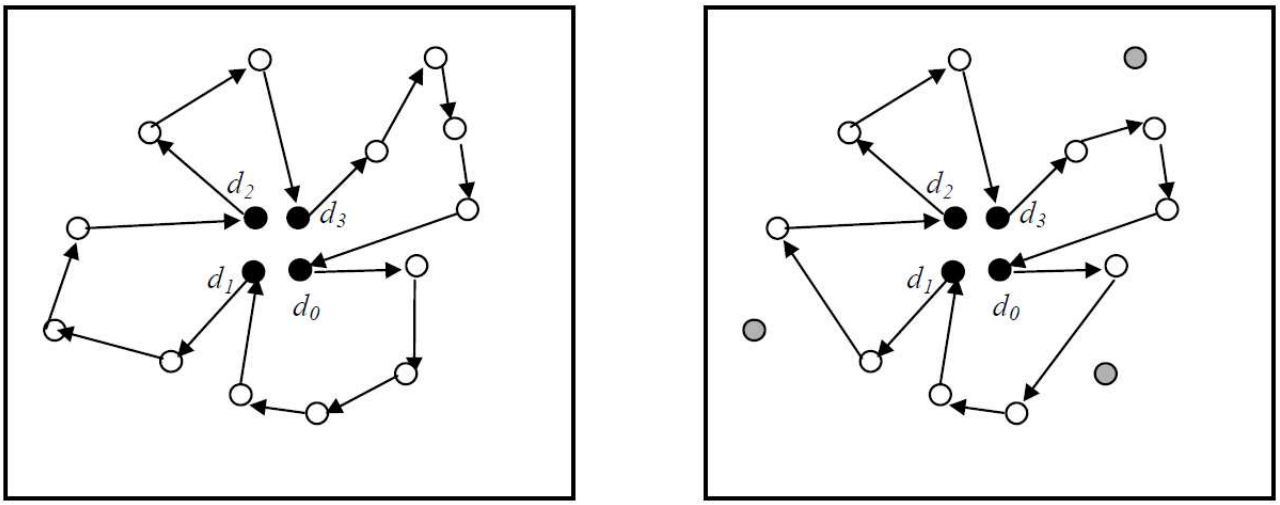
Hver gang en myre tilbagelægger en afstand fra i til j, opdateres feromonsporet for denne kant på følgende vis:

(6.8) $$\tau_{ij} = (1-\sigma)\tau_{ij} + \sigma\tau_0$$

hvor τ_0 er en parameter der repræsenterer initialværdien af feromonsporet og σ betegner andelen af feromon der bortfalder ved den lokale opdatering.

6.5.3 ACO-algoritmer for VRP

Når det kommer til implementeringen af ACO for VRP, kan det være en fordel at huske på at forskellen fra TSP ikke er særlig stor. Vi har tidligere været inde på, at et VRP i grunden er ækvivalent med et *m*-TSP med kapacitetsbegrænsning, hvor *m* sælgere besøger en bestemt by (eksempelvis et depot) *m* gange og de resterende én gang. En transformation fra et sådant problem til et TSP blev foreslået af bl.a. Lenstra og Rinnooy Kan (1975), hvilket i grunden går ud på at duplikere depotet et antal gange (se figur 25), svarende til antal køretøjer, hvorefter problemet kan løses ved almindelige TSP løsningsmetoder. Det er denne logik der gør, at de her gennemgående ACO-algoritmer for TSP oftest relativt simpelt kan transformeres til at løse VRP og dets udvidelser.



Figur 25: Illustration af duplikering af et depot. Bemærk, at de duplikerede depoter rent faktisk er placeret på præcist samme sted. Gambardella et al. (1999).

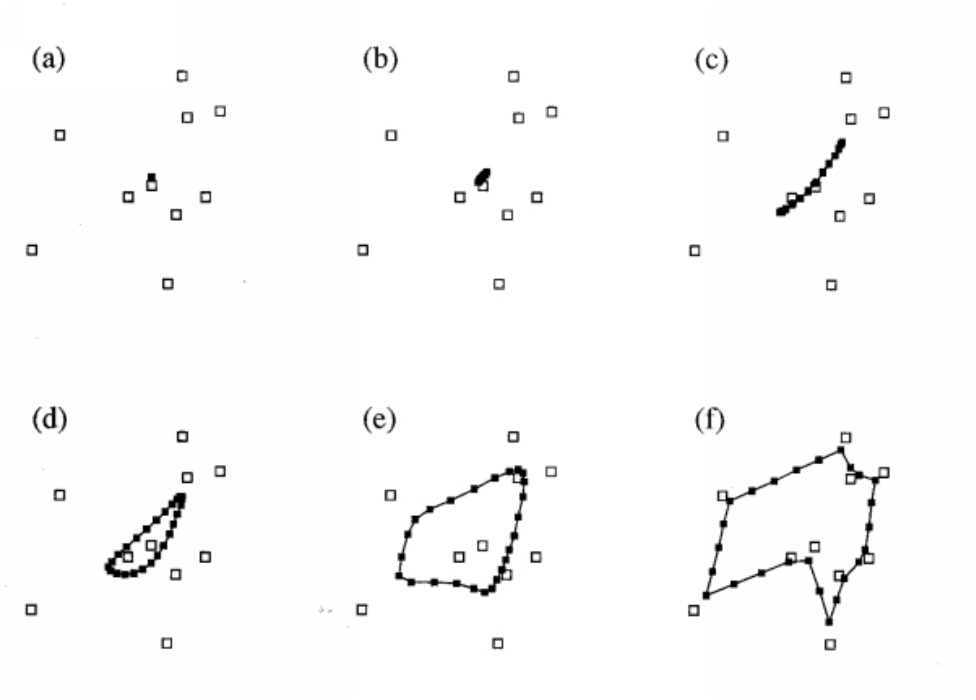
6.6 Neurale Netværk

Neurale netværk er vores måde at beskrive og simulere, hvorledes den menneskelige hjerne fungerer. Vores hjerne består af op til 100 milliarder nerveceller, også kaldes neuroner, og hver neuron er forbundet til tusinde andre neuroner og kommunikerer med dem via elektrokemiske signaler. En neuron modtager konstant signaler fra disse inputs. Et input er karakteriseret ved dets vægt for den pågældende neuron, hvor en vægt kan beskrives som et sæt koordinater. Det er disse vægte der kort fortalt bestemmer, hvorvidt en neuron udsender et signal til netværket.

Et kunstigt neuralt netværk består således af kunstige neuroner. Det der specielt gør netværket interessant er den lærende proces på baggrund af skrevne regler, hvor et netværk bruger en trinvis justering af vægtene til at ændre parametre i modellen. Jo længere vi kommer i algoritmen, des langsommere er læringsprocessen. Eksempler på sådanne netværk er det elastiske net (Durbin, Willshaw (1987)) og et "self-organizing map" (Kohonen (1988)), to tidlige modeller der er brugt til at løse TSP.

6.6.1 Et tidligt neuralt netværk for TSP

Nedenfor er givet et grafisk eksempel på det elastiske net af Durbin og Willshaw (1987), hvor de hvide kvadrater er noder og de sorte er enheder i modellen. Enhederne starter som en lukket rute i midten af kundemængden og enhedernes placering justeres nu trinvist, således at ruten strækkes ud for til sidst at tilpasse kunderne og derigennem definere en rute. Denne geometriske tilpasning er baggrunden for algoritmens navn, det elastiske net.



Figur 26: Det elastiske net. Durbin et al. (1989).

Hver enhed bevæger sig simultant ved at være påvirket af to kræfter; én der trækker den mod de nærmeste kunder og én der trækker den mod sine naboenheder for at danne så jævne mellemrum som muligt. Initialt har mere eller mindre alle kunder samme indflydelse på enhederne. Senere får kunderne længere væk mindre prioritet og hver enhed vil være tiltrukket mest af de nærmeste kunder. Det svarer meget til måden hvorpå vi sænker temperaturen gradvist i simuleret udglødning. Denne indflydelse (eller temperatur) justeres med en skalaparameter K, som vi lader være høj i starten men reduceres trinvist gennem algoritmen.

Lad $X_i, i = 1,\ldots,n$ være en vektor der repræsenterer placeringen af de n kunder gennem et sæt koordinater i planen. Ligeledes betegner $Y_j, j = 1,\ldots,m$ placeringen af de m enheder i modellen, hvor $m \geq n$ idet hver kunde skal kunne associeres med hver sin enhed. I hvert trin i algoritmen opdateres enhedernes placering ved følgende ændringsvektor:

(6.9) $$\Delta Y_j = \alpha \sum_i \omega_{ij} \left(X_i - Y_j \right) + \beta K \left(Y_{j+1} + Y_{j-1} - 2Y_j \right),$$

hvor

(6.10) $$\omega_{ij} = \frac{e^{-\left|X_i - Y_j\right|^2 / 2K^2}}{\sum_k e^{-\left|X_i - Y_k\right|^2 / 2K^2}}.$$

Her angiver første led i (6.9) den kraft, der trækker enhed j mod den nærmeste kunde i, og det andet led den kraft, der forsøger at holde enhederne mere eller mindre jævnt fordelt ud over ruten. α og β er konstanter, der vægter styrken af de to kræfter mod hinanden, og K er en skalaparameter, der, ved at blive justeret i hvert trin af algoritmen, ændrer prioriteringen af de to kræfter. ω_{ij} repræsenterer styrken af forbindelsen mellem kunde i og enhed j, normaliseret så den totale indflydelse af hver kunde på j er den samme. Denne vægt er altså en funktion af afstanden mellem i og j, hvilket betyder at en lavere afstand medfører en større vægt i (6.9) og dermed foretager enhed j et større spring i retning mod i i forhold til de andre kunder. Bemærk at samtlige kunder trækker i enhed j, men de kunder længst væk trækker kun marginalt, afhængig af deres relative afstand til j.

Vi har her beskrevet algoritmen som et TSP i planen, men den fungerer ydermere som en generel metode til at matche en mængde arbitrært forbundne punkter til en anden mængde punkter, placeret i et geometrisk rum af enhver dimension.

6.6.2 Neurale netværk for VRP

Vi kan generalisere denne type algoritmer for TSP til også at omfatte VRP ved at benytte os af adskillige lukkede ruter, der hver repræsenterer sin rute. Da vi nu har flere ruter, vil køretøjerne være nødt til at konkurrere om kunderne gennem en stokastisk mekanisme, hvor sandsynligheden for at tildele en

kunde et bestemt køretøj forandres gennem en lærende proces. For eksempel opstiller Ghaziri (1993) følgende algoritme for CVRP:

1) Betragt næste kunde (start forfra hvis sidste kunde er nået) og gem denne som den nuværende kunde.
2) Lad hver rute have en sandsynlighed for at blive valgt.
3) Vælg en rute på baggrund af den i trin 2 definerede sandsynlighedsfordeling.
4) Forsøgsvis tildel den nuværende kunde den nærmeste enhed på den valgte rute og flyt enheden samt nogle naboenheder på denne rute mod kunden.
5) Gentag fra trin 1 indtil hver kunde er tilpas tæt på en enhed.
6) Tildel hver kunde den nærmeste enhed og lad rækkefølgen af enhederne i hver sin rute definere løsningen.

Såfremt ingen mulig løsning eksisterer, kan yderligere ruter tilføjes og problemet atter forsøges løst.

Sandsynligheden for at en kunde tildeles en bestemt rute ændres dynamisk gennem algoritmen. Initialt er afstanden mellem kunden og den nærmeste enhed på en rute den mest dominerende faktor, hvor den nærmeste rute således har den største sandsynlighed for at blive valgt. Senere, som flere og flere kunder tildeles en rute, bliver kapacitetsbegrænsningerne af højere prioritet, således at de ruter der ikke kan servicere kunden uden at overskride køretøjets kapacitet får en lavere sandsynlighed. Til sidst vil kun mulige ruter have en betydelig sandsynlighed for at blive valgt.

6.7 Metaheuristik for VRPTW

Udviklingen indenfor heuristikker og metaheuristikker er gået utrolig hurtigt, og inden man for alvor fik taget fat på mere specifikke udvidelser til det oprindelige CVRP, var man gået over til metaheuristik. Derfor ser vi ikke mange heuristikker til de mere specielle udvidelser, netop fordi metaheuristik straks blev den nye dille indenfor VRP-området.

Tabu search blev som sagt først introduceret af Glover (1986) og i 1994 blev metoden allerede introduceret til at løse VRPTW (Garcia (1994)). Hvor genetiske algoritmer allerede i Thangiah (1991) blev tilpasset VRPTW, så blev denne type løsningsmetoder først konkurrencedygtige hen imod slutningen af 90'erne gennem hybrider med andre kendte heuristikker og metaheuristikker, som fx konstruktive heuristikker (Blanton and Wainwright 1993, Berger et al. 1998), lokal søgning (Thangiah 1995a, b; Thangiah et al. 1995; Potvin and Bengio 1996; Jung and Moon 2002), tabu search (Wee Kit et al. 2001) og ant colony systems (Berger et al. 2003).

Det er utrolig svært at vælge et lille antal metaheuristikker ud for VRPTW, idet antallet af foreslåede heuristikker for denne type problemer er ekstremt omfattende, og desuden er ingen løsningsmetode strengt bedre på alle områder. Vi har valgt først at se på en udvidelse af den tidligere omtalte ACS-algoritme for TSP, nu tilpasset VRPTW, kaldet "multiple ant colony system for vehicle routing problems with time windows" eller blot ved dens forkortelse, MACS-VRPTW. Dernæst vil vi tage fat på en nyere metaheuristik kaldet "active guided evolution strategies" (AGES), der har vist sig at være særdeles

effektiv for specielt store problemer og som stadig den dag i dag er en af de bedste løsningsmetoder for VRPTW.

6.7.1 "Multiple ant colony system"

Denne ACO-algoritme af Gambardella et al. (1999) for VRPTW bygger på den tidligere beskrevne ACS-algoritme for TSP. Vores VRP transformeres til et TSP på den måde beskrevet i afsnittet "ACO-algoritmer for VRP", hvorefter problemet nemt bør kunne løses. Men der er et twist, idet det rent faktisk foreslås at benytte 2 ACS-algoritmer, én til at minimere antal køretøjer (ACS-VEI) og én til at minimere den totale rejsetid (ACS-TIME). Begge mål søges nået samtidig ved simultant at benytte den ene og den anden myrekoloni, således at ACS-VEI finder en mulig løsning med et mindre antal køretøjer og ACS-TIME optimerer rejsetiden med det nuværende antal køretøjer.

De to kolonier har hver deres feromon-spor, der opdateres uafhængig af hinanden. Den eneste kommunikation mellem de to kolonier er gennem variablen ψ_{gb}, der hele tiden holder den globalt bedste løsning opdateret, dvs. den bedst mulige løsning i form af både antal køretøjer og rejsetid, hvor antal køretøjer altid har højeste prioritet.

Initialt er ψ_{gb} bestemt gennem en nærmeste-nabo heuristik med et uendeligt antal køretøjer. De to processer ACS-VEI og ACS-TIME kører nu simultant, men dog fuldkommen uafhængigt. ACS-VEI forsøger at finde en mulig løsning ved at benytte sig af ét køretøj mindre end antal køretøjer i ψ_{gb}. ACS-TIME forsøger at optimere den totale rejsetid ved at bruge præcist det samme antal køretøjer som i ψ_{gb}. ψ_{gb} opdateres herfra på følgende vis: Hver gang ACS-TIME finder en ny forbedret løsning, opdateres ψ_{gb} og der forsøges at finde en løsning med endnu lavere rejsetid. Hver gang ACS-VEI finder en mulig løsning, standses både ACS-VEI og ACS-TIME og to nye kolonier konstrueres, der nu begge arbejder med et køretøj mindre. På denne måde itereres denne overordnede algoritme indtil et stopkriterium er nået.

Opsummeret kan algoritmen trinvist opstilles på følgende vis:

1) Bestem en initial bedste løsning ψ_{gb} med en nærmeste-nabo heuristik med et uendeligt antal køretøjer.
2) Gem antal køretøjer i ψ_{gb} i variablen v.
3) Aktivér ACS-VEI $(v-1)$ og ACS-TIME (v).
4) Finder ACS-VEI eller ACS-TIME en ny bedste løsning, opdateres ψ_{gb}.
    a. Såfremt antal køretøjer i ψ_{gb} er mindre end v, deaktiveres ACS-VEI og ACS-TIME. Fortsæt fra trin 2, medmindre et stopkriterium er opfyldt, i hvilket tilfælde algoritmen standses og ψ_{gb} returneres.
5) Gentag trin 4 så længe både ACS-VEI og ACS-TIME er aktive.

Lad os nu dykke ned i hvorledes de to myrekolonier, ACS-VEI og ACS-TIME, fungerer.

ACS-TIME er en traditionel ACS-algoritme (se afsnit 6.5), hvis mål er at finde en så kort rute som overhovedet muligt. m myrer aktiveres og konstruerer hver sin løsning. Der tjekkes hvorvidt den totale længde af myre k's løsning $L_k < L_{gb}$, i hvilket tilfælde løsningen returneres til den overordnede algoritme som en løsning fra ACS-TIME. Der foretages herefter en global opdatering som i (3.2.5), m nye myrer aktiveres og processen itereres indtil et stopkriterium er opfyldt.

ACS-VEI's mål er imidlertid blot at finde en mulig løsning med ét køretøj mindre end den nuværende bedste løsning. Dette gøres ved at maksimere antallet af besøgte kunder ved hver gennemkørsel og såfremt antal besøgte kunder er større end alle hidtidige løsninger, gemmes løsningen i variablen $\psi_{ACS-VEI}$. Der vil imidlertid for det meste være tale om løsninger for hvilke nogle kunder ikke er tilknyttet en rute, men så snart en mulig løsning er fundet, returneres denne til den overordnede algoritme.

Maksimeringen af antal kunder fungerer på den måde, at hver gang en myre har produceret en løsning, opdateres en variabel IN, der gemmer antal gange en kunde ikke indgår i en løsning, således at disse kunder får større prioritet for de efterfølgende myrer.

Bemærk, vi har ikke specificeret et passende stopkriterium for hverken den overordnede algoritme eller ACS-TIME og ACS-VEI. Sådan et kan vælges på baggrund af den pågældende situation, f.eks. kan vi vælge at acceptere den nuværende bedste løsning, hvis algoritmen har kørt en tid uden at finde en forbedring.

ACS-TIME og ACS-VEI benytter samme konstruktive algoritme, *new_active_ant*, der opbygger en myres rute på meget lig den måde vi gjorde for TSP. Hver myre starter ved et tilfældigt valgt depot og går ved hvert trin videre ad kanter, der følger de tidligere beskrevne sandsynlighedsbaserede mekanismer for udforskning og udnyttelse i et ACS. Hvor nærhedsfaktoren η_{ij} før udelukkende afhang reciprokt af afstanden mellem to noder, lad os imidlertid omdefinere nærhedsfaktoren η_{ij} til at inkludere rejsetid t_{ij}, tidsvinduer $[a_i, b_i]$ samt prioritetsfaktoren IN:

(6.11) $$\eta_{ij} = \frac{1}{d_{ij}},$$

hvor

(6.12) $$d_{ij} = \max\left(1, \left(\left(w_j - w_i\right) \cdot \left(b_j - w_i\right) - IN_j\right)\right),$$

(6.13) $$w_j = \max\left(w_i + s_i + t_{ij}, a_j\right).$$

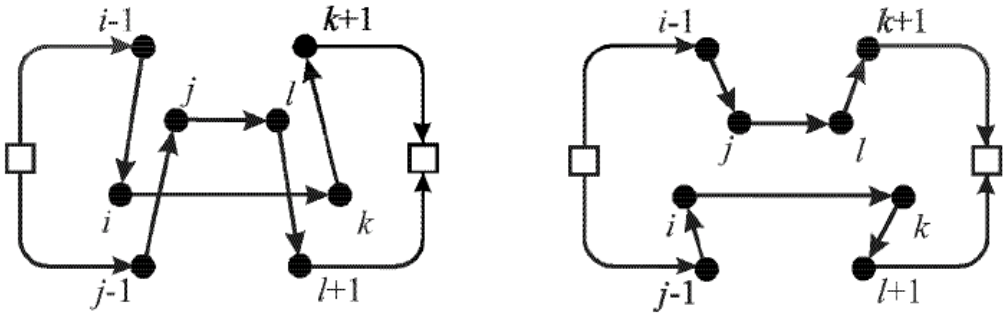
Her er afstanden d_{ij} mellem node i og j modificeret. Lad w_i være en kontinuert variabel der angiver start af service for kunde i, da betegner w_j start af service af kunde j, som afhænger dels af starten

af tidsvinduet og hvornår køretøjet tidligst kan være fremme[23]. Lad nu følgende indgå i afstandsvariablen d_{ij}: Tidsforskellen $w_j - w_i$ mellem start af service for de to kunder, den resterende tid indtil tidsvinduet slutter samt prioriteten af at kunde j indgår i ruten. Den oprindelige afstand mellem kunde i og j indgår her implicit i rejsetiden der antages at være proportionel med afstanden.

En myre vælger på baggrund af ovenstående mekanisme en kunde j fra mængden af mulige kunder, dvs. kunder der ikke overskrider tidsvinduerne ($a_j \le w_j \le b_j$) eller kapacitetsbegrænsningerne ($B_j = B_i + q_j \le Q$), hvor B_i og B_j betegner et køretøjs totale, leverede mængde efter hhv. kunde i og j, q_j den leverede mængde til kunde j og Q køretøjets kapacitet. Her initialiserer vi B_i med $B_i = 0$. Kun såfremt en myre ikke står i et depot, indgår de duplikerede depoter også blandt de mulige overgange. Så snart en kant (i, j) er valgt, foretages en opdatering af w_i og B_i ($w_i \leftarrow w_j$, $B_i \leftarrow B_j$) samt en lokal opdatering af feromonsporet, hvorefter en ny kunde j vælges. Bemærk at kun ACS-VEI benytter sig af variablen IN, som derfor i ACS-TIME sættes til nul.

Algoritmen standser når der ikke findes flere mulige overgange. Såfremt der findes kunder ikke tilknyttet en rute, forsøges disse herefter indsat med en indsættelsesheuristik sorteret i faldende orden efter efterspurgt mængde. Indsættelsesproceduren standses når der ikke findes flere mulige indsættelser.

For ACS-TIME iværksættes ydermere en forbedringsalgoritme kaldet en "CROSS-exchange", oprindeligt foreslået af Taillard et al. (1997). Nedenstående figur illustrerer idéen bag forbedringen. To mængder af kanter fra hver sin rute udvælges og bytter plads. Én mængde kan potentielt være tom, hvor karakteren af forbedringen skifter til en regulær indsættelse.



Figur 27: Kundesegmenterne (i,k) og (j,l) bytter simultant til plads i de to ruter. Dette udføres ved at erstatte kanterne $(i-1,i)$, $(k,k+1)$, $(j-1,j)$ og $(l,l+1)$ med kanterne $(i-1,j)$, $(l,k+1)$, $(j-1,i)$ og $(k,l+1)$. Bräysy, Gendreau (2002).

Herefter returneres løsningen til de respektive algoritmer for ACS-TIME og ACS-VEI. Trinvise opstillinger for hhv. ACS-TIME, ACS-VEI og new_active_ant kan findes i Gambardella et al. (1999).

[23] Bemærk, den oprindelige algoritme af Gambardella et al. (1999) tager ikke højde for servicetiden s_{ij}.

I dag er denne metaheuristik mere eller mindre forældet og findes kun som bedste løsningsmetode til en enkelt af de 56 af Solomon præsenterede problemer. Men da den først blev præsenteret i 1999, fandt den en ny bedste løsning til 13 af problemerne og var stærkt konkurrencedygtig med de øvrige. I artiklen præsenteres den bedste løsning efter flere længder af tidsforløb, og det viser sig at algoritmen meget hurtigt konvergerer mod en god løsning, noget som bestemt er én af de vigtigste egenskaber ved denne metaheuristik.

6.7.2 Active guided evolution strategies

AGES-algoritmen af Mester et al. (2005) er specielt designet til at løse problemer med et større antal kunder og leverer da også stærkt konkurrencedygtige resultater for problemer med 200 og op til 1000 kunder.

Evolutionsstrategier

Ligesom genetiske algoritmer er en evolutionsalgoritme, er evolutionsstrategier (ES) det også. Dog er der væsentlige forskelle. For det første bruges reelle beslutningsvektorer til at repræsentere en løsning, modsat genetiske algoritmer som vi har set enten bruger binære strenge eller (for VRP) brugerdefinerede diskrete repræsentationer. For det andet foregår udvælgelsen af forældreløsninger deterministisk, hvilket øger risikoen for at ende i lokale minima. For at undgå dette lægges større vægt på mutationer.

Hver person i ES er knyttet et sæt vektorer, $v = (x, \sigma)$, hvor x angiver et punkt i løsningsrummet og dermed en specifik løsning, mens σ er et sæt såkaldte strategiparametre, der angiver hvorledes mutationen foregår. For reelle beslutningsvariable x, er σ standardafvigelser i en normalfordeling og mutation udføres ved at erstatte x med $x_{t+1} = x_t + N(0, \sigma)$.

VRPTW hører dog til en klasse af diskrete optimeringsproblemer, hvorfor ovenstående nødvendigvis må justeres en smule. For det første består v nu af diskrete variable x og σ. For det andet indeholder σ oplysninger om hvorledes vi kommer fra en oprindelig løsning S til en anden løsning S' i et nabo-område $N(S)$ omkring S. Og for det tredje består området omkring S der gennemsøges ved mutationer, $N(S)$, af mulige løsninger genereret ved at ændre nogle egenskaber for S ved én enkelt iteration.

I AGES-algoritmen benyttes en såkaldt (1+1)-evolutionsstrategi, som består af en populationsstørrelse på 1 der genererer ét nyt afkom gennem mutation. Der forekommer derfor ingen udvælgelse, udelukkende mutationer af den oprindelige population. Mutationen foregår gennem en tilfældigt valgt mutationsstrategi, der udføres et tilfældigt antal gange. Ændringerne foretages én ad gangen og hver gang evalueres afkommet på baggrund af dets fitness-værdi, hvorefter det enten kasseres eller erstatter forældreløsningen.

Guided local search

"Guided local search" (GLS) evaluerer en kandidatløsning S på baggrund af en modificeret objektfunktion, der består af den oprindelige objektfunktion $g(S)$ samt et straf-led der straffer

bestemte karakteristika ved den fundne løsning som ikke bør findes i en tæt på optimal løsning, f.eks. ekstremt lange kanter mellem noder. Jo flere gange en bestemt karakteristik er blevet straffet, des mindre er sandsynligheden for at den bliver straffet yderligere, idet egenskaben tydeligvis er fremtrædende blandt mange mulige løsninger. Hver karakteristik i har en omkostning c_i og en straf p_i tilknyttet. c_i kan oftest bestemmes fra den oprindelige objektfunktion, f.eks. hvis en karakteristik er "hvorvidt kanten mellem node j og k findes i løsningen", vil en fornuftig omkostning være afstanden mellem noderne. p_i sættes initialt til 0 og opdateres for alle i hver gang et lokalt minimum nås.

Lad $g(S)$ være den oprindelige objektfunktion der knytter en numerisk værdi til en kandidatløsning S. Definér nu en ny objektfunktion $h(S)$ på følgende vis:

(6.14) $$h(S) = g(S) + \lambda \sum_i (p_i I_i(S)),$$

hvor λ er en parameter der regulerer styrken af straf-leddet, p_i er antal gange egenskaben i er blevet straffet og $I_i(S)$ er en indikatorfunktion defineret som:

(6.15) $$I_i(S) = \begin{cases} 1, \text{hvis } S \text{ udviser egenskab } i \\ 0, \text{ellers} \end{cases}.$$

Når søgningen er fanget i et lokalt minimum S^*, undslippes ved at forsøge at udfase de værste egenskaber ved den pågældende løsning ved at straffe disse. På den måde forsøges at gøre det lokale minimum mindre attraktivt i forhold til de dele af det omkringliggende løsningsrum, der ikke udviser disse egenskaber. For ikke at komme til at straffe de vigtigste egenskaber for en mulig løsning, tages der højde for hvor mange gange egenskaberne er blevet straffet tidligere, og tillægger dem, der er straffet flest gange, mindre vægt. Den egenskab i der straffes kan bestemmes fra følgende nyttefunktion:

(6.16) $$U_i(S^*) = \frac{I_i(S^*) c_i}{1 + p_i},$$

Blandt de egenskaber i som det lokale minimum S^* udviser, udvælges den egenskab k, vi har den største nytte af at straffe, hvor nytten afhænger af omkostningen for i samt det antal gange vi tidligere har straffet i. Nu opdateres straffen:

(6.17) $$p_k = p_k + 1.$$

GLS benyttes sammen med en lokal søgningsmetode, f.eks. en forbedringsheuristik, der med udgangspunkt i S returnerer en ny løsning. (6.14) er udelukkende en metode til evaluering af kandidatløsninger.

Active guided evolution strategies

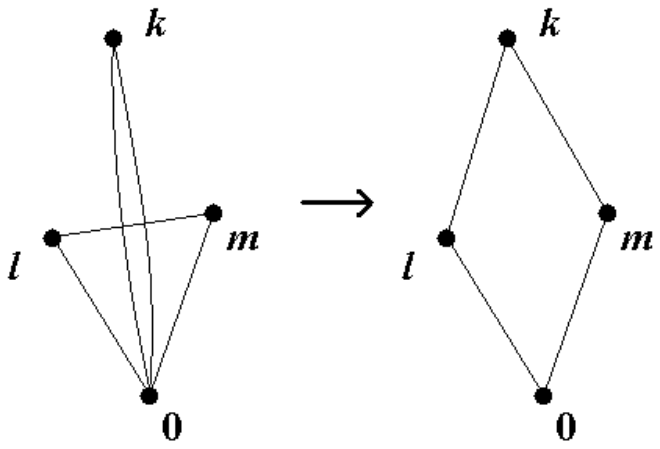
Mester og Bräysy (2005) foreslår en to-fase metaheuristik, som i første fase bestemmer en initial løsning ved en hybrid indsættelsesheuristik efterfulgt af en reduktion af antal ruter og i anden fase forbedrer løsningen med en såkaldt "active guided evolution strategies (AGES)" metaheuristik, der skal ses som en kombination af ovenstående metaheuristikker, GLS og ES.

Konstruktion af en initial løsning

Her benyttes en hybrid indsættelsesheuristik meget lig Clarke og Wrights "savings"-heuristik. Der lægges ud med en løsning, hvor hver kunde er tildelt sin egen rute, hvorefter en modificeret besparelse udregnes for hver kunde k, givet ved:

(6.18) $$\left(d_{ik} + d_{kj} - d_{lm}\right) - \alpha\left(d_{lk} + d_{km} - d_{ij}\right),$$

hvor d_{ij} er afstanden mellem kunde i og j, og hvor kunde k, der lige i øjeblikket serviceres mellem i og j, vurderes at kunne indsættes mellem de to tilstødende kunder l og m. Til forskel fra den oprindelige savings-heuristik, som kun sætter ruter sammen i forlængelse af hinanden, har vi altså her en heuristik der indsætter én kunde ad gangen, til stedet med den største besparelse. Tag følgende som eksempel:



Figur 28: Eksempel

Her er besparelsen således givet ved $\left(d_{0k} + d_{k0} - d_{lm}\right) - \alpha\left(d_{lk} + d_{km}\right)$, hvor første led repræsenterer de kanter der bliver erstattet med kanterne i det andet led. Er denne besparelse positiv, som i dette tilfælde, accepteres indsættelsen af kunde k.

α er en parameter hvis værdi afhænger af problemstørrelsen. I artiklen betragtes og præsenteres resultater for problemer med 200, 400, 600, 800 samt 1000-kunder. For problemer af størrelsen 200 kunder forsøges alle værdier mellem 0,6 og 1,4 med intervaller af 0,2. For problemer med 400 eller 600 kunder er det værdier mellem 0,8 og 1,2, mens der for 800 og 1000 kunder kun undersøges for $\alpha = 1$. På denne måde forsøges at begrænse beregningstiden for de største problemer.

For hver gang 20 % af kunderne er blevet indsat, benyttes to forbedringsheuristikker skiftevist indtil det ikke længere er muligt at finde en forbedring. Den første kaldes en "relocate" operator, og er i grunden blot en flytning af en kunde fra sin nuværende plads til enhver anden placering, enten på sin nuværende rute eller en anden. Den anden kaldes en "1-interchange" og bytter placeringen af to kunder, én i hver rute.

Af løsningerne opnået med forskellige værdier for α vælges nu den bedste løsning som en initial løsning. Inden vi fortsætter med næste fase forsøges at reducere antallet af ruter med en påfyldningsstrategi, der går ud på at maksimere summen af ruternes kvadrerede antal kunder. Proceduren indsætter kunder fra ruter med et lavt antal kunder til ruter med et højt antal kunder, for på den måde at forsøge at eliminere nogle af de mindste ruter. For yderligere oplysninger om denne procedure henvises til Bent og Van Hentenryck (2004).

Forbedringsfase

Forbedringsfasen består af to stadier der anvendes iterativt. Første stadie er en GLS i samarbejde med de ovennævnte "relocate" og "1-interchange" operatorer. Det andet stadie er en modificeret ES. For de præcise parameterindstillinger henviser vi til metodens forfattere, D. Mester og O. Bräysy (2005). Den generelle metodik i de to stadier gennemgås i det følgende.

GLS med lokale søgningsheuristikker

Denne søgningsheuristik er baseret på de to forbedringsheuristikker "relocate" og "1-interchange". Disse to heuristikker benyttes skiftevist til at bevæge os fra én løsning til en anden, mens vi hele tiden holder styr på, hvorvidt vi har fundet en ny bedste løsning, vurderet i forhold til den oprindelige objektfunktion.

De egenskaber vi vælger at straffe i et lokalt minimum er de længste kanter, bestemt ud fra nyttefunktionen i (6.16), hvor omkostningen c_i er bestemt som længden af kant i og p_i er antal gange i er blevet straffet tidligere. Det nabo-område, der betragtes af søgningen fra et lokalt minimum, består udelukkende af kunderne på ruter der ligger tættest på den kant der er blevet straffet. Intuitionen er den, at formentlig vil én af kunderne for enden af denne kant skifte til en nærliggende rute, mens resten vil forblive som hidtil. Dette nabo-område kaldes her for et "penalty variable neighborhood (PVN)", og størrelsen af dette område reguleres dynamisk gennem søgningen. Ved hver iteration er nabo-områdets størrelse N^k givet ved:

(6.19) $$N^k = N_{\min} + \left(N_{\max} - N_{\min}\right)a^2,$$

hvor $N_{\min}$ er et antal kunder således at mindst 3 ruter er inkluderet i nabo-området, $N_{\max}$ er det mindste af følgende: Antal kunder blandt de udvalgte naboruter og 60 % af problemets størrelse (målt i antal kunder), og a er en stokastisk variabel ligefordelt på intervallet 0 til 1. Denne definition betyder for det meste relativt små PVN. For at reducere antal ruter, lades der være en vis sandsynlighed i hver iteration for, at PVN består af samtlige kunder.

Hvis der ikke er registreret en forbedret løsning gennem et brugervalgt antal iterationer, og PVN for denne iteration er større end 25 kunder, standses første stadie og det andet startes (nuværende PVN bibeholdes). Dermed undgås at vi blot har været uheldige med nogle meget små nabo-områder som har begrænset søgningen. Desuden er andet stadie mere grundig i sin søgning og dermed beregningstidsmæssigt mere krævende, hvorfor vi kun bevæger os ind i dette stadie, hvis en større del af nabo-området (PVN) i forvejen er blevet gennemsøgt, dvs. hvis det kan undgås, foretrækkes første stadie frem for andet.

Modificeret ES

Denne ES ligner LNS på den måde, at kunder fra det nuværende PVN (fra første stadie) fjernes fra løsningen og genindsættes til optimale omkostninger. Der benyttes her 3 strategier til at vælge præcis hvilke kunder der skal reorganiseres; disse vil blive gennemgået herunder.

I de første 2 strategier begrænses det antal kunder der kan fjernes med parameteren β:

(6.20) $$\beta = (0.2 + 0.5a)n,$$

hvor a er en ligefordelt stokastisk variabel på intervallet 0 til 1 og n er antal kunder i det nuværende PVN.

Den første strategi foretager en tilfældig udvælgelse fra PVN. Den anden danner to cirkler fra depotet med tilfældige radier, og fjerner nu de kunder fra PVN som ligger mellem de to cirkler. Den tredje og sidste strategi fjerner alle kunder fra det nuværende PVN. Hvor den første og anden strategi bygger på næsten fuldkommen tilfældighed, og desuden kun udvælger en begrænset mængde af kunder, så er den tredje strategi mere desperat og fjerner alle kunder med 100 % sikkerhed.

Når de kunder der skal fjernes er valgt, benyttes den hybride indsættelsesheuristik fra konstruktionsfasen til at genindsætte dem blandt ruterne, for forskellige værdier af α. Når alle kunder er indsat, evalueres løsningerne med den modificerede objektfunktion (6.14), og den bedste løsning forbedres med GLS-metaheuristikken i første stadie.

Proceduren gentages i 14 iterationer ad gangen, hvor de første 7 iterationer udelukkende benytter de første to strategier til at fjerne kunder (hver med 50 % sandsynlighed), og de sidste 7 udelukkende benytter den tredje strategi. I hver iteration sammenholdes afkommet med forældreløsningen ved at sammenligne værdierne fra den modificerede objektfunktion, og såfremt afkommet er bedst, træder denne løsning i stedet og danner udgangspunkt for næste iteration, hvor et nyt antal kunder udvælges fra et nyt PVN omkring den nye løsning til at blive fjernet på baggrund af en ny strategi. Herefter gentages processen og 14 nye iterationer gennemføres, og dette fortsætter indtil vi gennem et antal iterationer ikke har fundet en bedre løsning, hvorefter vi fortsætter fra første stadie.

Såfremt andet stadie ikke kommer frem til en forbedring, kan vi risikere at den samme kant bliver straffet når første stadie starter forfra og vi dermed bliver ved med at cirkulere i samme løsningsområde. I dette tilfælde straffes en kant tilfældigt.

Den overordnede algoritme

Den overordnede algoritme kan beskrives som følger:

Fase 1:

1) Konstruér en initial løsning S^0 med en hybrid indsættelsesheuristik.
2) Initialisér strafparametrene og sæt $c^1 = 0, c^2 = 0, i = 0$.
3) Reducér antal ruter hos S^0 ved påfyldningsproceduren indtil det ikke er muligt at eliminere flere.
4) Gem den globalt bedste løsning $S^{gb} = S^0$.

Fase 2:

5) Nulstil strafparametrene, sæt $i = 0, c^2 = 0, S^i = S^{gb}$ og definér $c^2_{\max}$ som et tilfældigt tal mellem 5.000 og 50.000.
6) Lokalisér en kant i S^i til at blive straffet og opdater strafparametrene.
7) Definér et PVN til S^i.
8) Konstruér en ny løsning S^{ny} fra et PVN omkring S^i vha. GLS med lokale søgningsheuristikker.
9) Hvis S^{ny} er bedre end S^{gb} i forhold til den oprindelige objektfunktion, sæt $S^{gb} = S^{ny}$.
10) Hvis S^{ny} er bedre end S^i i forhold til den modificerede objektfunktion, sæt $i = i + 1, c^1 = 0, c^2 = 0, S^i = S^{ny}$ og fortsæt fra trin 6.
11) Hvis $c^1 < c^1_{\max}$ eller størrelsen af PVN < 25, sæt $c^1 = c^1 + 1, c^2 = c^2 + 1$ og fortsæt fra trin 6.
12) Konstruér en ny løsning S^{ny} fra PVN omkring S^i vha. en modificeret ES.
13) Hvis S^{ny} er bedre end S^{gb} i forhold til den oprindelige objektfunktion, sæt $S^{gb} = S^{ny}$.
14) Hvis S^{ny} er bedre end S^i i forhold til den modificerede objektfunktion, sæt $i = i + 1, c^2 = 0, S^i = S^{ny}$ og fortsæt fra trin 12.
15) Hvis stopkriteriet er opfyldt, standses algoritmen og den bedste løsning S^{gb} returneres.
16) Sæt $c^1 = 0, c^2 = c^2 + 1$.
17) Hvis $c^2 < c^2_{\max}$, fortsæt fra trin 6, ellers fra trin 5.

Her er trin 6-11 første stadie af forbedringsalgoritmen, mens trin 12-14 angiver andet stadie. I første omgang får første stadie muligheden for at forbedre den bedste løsning og dette stadie får lov til at køre indtil vi igennem et antal iterationer ikke har kunnet registrere nogen forbedringer. På dette tidspunkt tager andet stadie over med en mere grundig eftersøgning af løsningsrummet, og når også dette stadie har opbrugt sine muligheder for at forbedre løsningen, starter første stadie forfra. Dette fortsætter iterativt indtil vi gennem adskillige iterationer ikke har kunnet registrere nogen forbedringer, hvorefter alle parametre nulstilles og hele fase 2 startes forfra med udgangspunkt i den nuværende bedste løsning S^{gb}. Overgangen mellem de to stadier styres således af de brugervalgte parametre $c^1_{\max}$ og $c^2_{\max}$, hvor

$c^1_{\max}$ styrer det tidspunkt hvor første stadie opgives og andet stadie igangsættes, mens $c^2_{\max}$ styrer det tidspunkt hvor strafparametrene nulstilles og anden fase igangsættes på ny.

Afsluttende kommentar

Mester og Bräysy har testet denne metaheuristik på 300 teoretiske VRPTW problemer og 2 praktiske problemer. I 2005 da denne metaheuristik blev præsenteret, returnerede algoritmen den bedst-kendte løsning i 259 ud af 300 (86 %) af de teoretiske problemer og i begge de praktiske.

Metaheuristikken står stadig i dag som én af de mest succesfulde metoder til at løse VRPTW. Herunder ses på tværs af problemstørrelsen, andelen af problemer som Mester og Bräysy's metaheuristik stadig den dag i dag har leveret den bedste løsning til.

200 kunder	400 kunder	600 kunder	800 kunder	1000 kunder	I alt
9 ud af 60	12 ud af 60	23 ud af 60	17 ud af 60	21 ud af 60	82 ud af 300
15 %	20 %	38 %	28 %	35 %	27 %

Tabel 5: Kilde: http://www.sintef.no/projectweb/top/

Vi kan tydeligt se, at der hvor metoden er blevet slået er på de mindste problemer, mens den stadig står stærkt på de store problemer. Dette må siges at være rimelig godt gået, når man tænker på det enorme antal metaheuristikker der er blevet præsenteret for VRPTW over de seneste to årtier.

6.8 Metaheuristik for VRPPDTW

Metaheuristikken i næste afsnit er baseret på en udvidelse af LNS. Den består af en række konkurrerende under-heuristikker, der anvendes med en hyppighed svarende til deres historiske performance. Som algoritmen skrider frem, vælges de heuristikker, der har vist sig særligt gode frem for andre, hvilket er årsagen til at denne algoritme kaldes "Adaptive Large Neighborhood Search" (ALNS). Hyppigheden, hvormed en bestemt heuristik benyttes, justeres dynamisk som søgningen skrider frem, således at den valgte heuristik tilpasses det tilfælde man har på daværende tidspunkt. Ved at bruge terminologien fra LNS, vil det sige, at ALNS udvider LNS ved at tillade flere nabo-områder inden for den samme søgning.

6.8.1 En "adaptive large neighborhood search" for VRPPDTW

Herunder opstilles ALNS for problemet VRPPDTW, tilpasset af Røpke og Pisinger (2006).

1. En mulig initialløsning x fundet på baggrund af en simpel konstruktionsheuristik.
2. Sæt $x^b = x$, $\rho^- = (1,...,1)$ og $\rho^+ = (1,...,1)$.
3. Gentag følgende indtil et stopkriterium er mødt.
   3.1. Vælg en ødelæggelses- og en reparationsmetode, $d \in \Omega^-$ og $r \in \Omega^+$, ved brug af ρ^- og ρ^+.
   3.2. Fjern q ordrer fra løsningen x ved ødelæggelsesmetoden.
   3.3. Genindsæt de fjernede ordrer ved brug af den valgte reparationsmetoden.
   3.4. Gem den nye løsning i x^m, hvor $x^m = r(d(x))$.
   3.5. Hvis x^m accepteres, sæt $x = x^m$.
   3.6. Hvis $c(x^m) < c(x^b)$, sæt $x^b = x^m$.
   3.7. Opdatér ρ^- og ρ^+.
4. Returnér den bedste løsning.

Algoritmen initialiseres med en løsning, konstrueret på baggrund af en konstruktionsheuristik. I andet trin introduceres to nye variable ρ^- og ρ^+, som gemmer vægten af hver enkelt ødelæggelses- og reparationsmetode. I trin 3.1 anvender man vægtene ρ^- og ρ^+ til at udvælge en ødelæggelsesmetode og en reparationsmetode. Første gang algoritmen kører har alle metoder dog samme vægt. Trin 3.2 og 3.3 hhv. fjerner og genindsætter nogle ordrer fra løsningen. Parameteren $q \in \{0,...,n\}$ bestemmer størrelsen af nabo-området. Hvis q er nul, udføres ingen søgning eftersom der ikke fjernes nogen ordrer. På den anden side, hvis q er lig n løses problemet fra bunden af i hver iteration. Generelt kan det siges, at jo større q, des lettere er det at bevæge sig i løsningsrummet, men medfører samtidig en længere beregningstid for indsættelsen. I trin 3.5-3.6 besluttes ved brug af acceptkriteriet for SA om den nye løsning skal accepteres eller ej, og den hidtil bedste løsning opdateres. Dernæst opdateres vægtene til ødelæggelses- og reparationsmetoderne. Algoritmen afsluttes når et bestemt stopkriterium er opfyldt, hvorefter den bedste løsning returneres.

Vi vil i de efterfølgende afsnit kort beskrive nogle ødelæggelses- og reparationsmetoder samt angive en metode til udvælgelse af en under-heuristik til den aktuelle iteration. Udvælgelsesmekanismen styres af indsamlet statistik under søgningen, som er hvad det sidste afsnit omhandler.

6.8.2 Fjernelse af ordrer

Røpke og Pisinger foreslår tre heuristikker til fjernelse af ordrer fra en løsning. De kræver alle en initialløsning og et heltal q som input. Metoderne "Shaw removal" og "worst removal" kræver yderligere en parameter p, som er et mål for tilfældigheden i heuristikken.

"Shaw's removal"-heuristik

Shaw's metode er blevet modificeret for at tilpasse VRPPDTW. Den generelle idé går ud på at fjerne ordrer, som er nogenlunde ens, da det forventes at være rimelig nemt at blande sådanne ordrer og dermed danne nye (og bedre) løsninger. Fjernes ordrer, der er meget forskellige fra hinanden, kan det resultere i en dårligere løsning eller den samme som tidligere. Ligheden mellem to ordrer i og j kan defineres ved brug af en Lighedsfaktor $L(i,j)$. Jo lavere $L(i,j)$, des mere ens er de to ordrer. Faktoren ser således ud:

(6.21)
$$L(i,j) = \varphi\left(d_{A(i),A(j)} + d_{B(i),B(j)}\right) + \chi\left(\left|T_{A(i)} - T_{A(j)}\right| + \left|T_{B(i)} - T_{B(j)}\right|\right) + \psi\left|l_i - l_j\right| + \omega\left(1 - \frac{\left|K_i \cap K_j\right|}{\min\left\{\left|K_i\right|, \left|K_j\right|\right\}}\right)$$

Lighedsfaktoren består af fire led, hver vægtet med hhv. φ, χ, ψ og ω; et afstandsled, et tidsled, et kapacitetsled og et led der angiver de køretøjer, som kan benyttes til servicering af de to ordrer. $A(i)$ og $B(i)$ betegner afhentnings- og leveringsstedet for ordre i, T_i angiver det tidspunkt, hvor stedet i besøges, og beregnes som følgende: $T_i = \sum_{k \in K} \sum_{j \in V_k} S_{ik} \cdot x_{ij}^k$. S_{ik} er et ikke-negativt heltal, som angiver det tidspunkt, hvor køretøj k starter service af i, og x_{ij}^k er en binær variabel som er 1, hvis kanten mellem node i og j tilbagelægges af køretøj k og 0 ellers. K_i er mængden af køretøjer, der er i stand til at servicere ordre i. d_{ij} betegner afstanden mellem i og j, og l_i er den mængde af varer, der skal læsses på køretøjet hos den pågældende node.

Leddet, der vægtes med φ, måler den samlede afstand mellem to ordrers afhentningssteder samt mellem to ordrers leveringssteder. Leddet vægtet med χ måler den tidsmæssige forskel mellem de to ordrer, bestemt som tidsforskellen mellem afhentningsstedernes besøg og tidsforskellen mellem leveringsstedernes besøg. Leddet vægtet med ψ måler forskellen i den efterspurgte mængde og leddet vægtet med ω belønner et sæt af to ordrer, hvis mange køretøjer har mulighed for at servicere begge ordrer samtidig.

I første iteration vælges en tilfældig ordre til at blive fjernet, hvorefter man i de efterfølgende iterationer forsøger at fjerne de ordrer, der ligner de allerede fjernede ordrer. Man kan anvende parameteren $p \geq 1$, som angiver graden af tilfældighed ved udvælgelse af første ordre.

Random removal

Denne algoritme udvælger tilfældigt q ordrer og fjerner dem fra løsningen. Metoden kan ses som et specialtilfælde af Shaw's removal heursitik hvor $p = 1$, idet et lavt p svarer til en høj grad af tilfældighed.

Worst removal

Til en given ordre i, der serviceres af et eller andet køretøj i løsningen s, defineres en marginalomkostning $\text{cost}(i,s) = f(s) - f_{-i}(s)$, hvor $f_{-i}(s)$ svarer til omkostningen uden ordre i. Herefter fjernes ordrer med høje marginalomkostninger.

6.8.3 Genindsættelse af ordrer

Herunder præsenteres indsættelsesheursitikker, der alle tilhører klassen af parallelle heuristikker, dvs. bygger flere ruter op samtidig.

"Basic greedy"-heuristik

Grådighedsalgoritmen er en simpel konstruktionsheuristik, hvor der kun indsættes én ordre ad gangen i hver iteration, hvorfor der maksimalt udføres n iterationer. Ved at indsætte ordre i i rute k sker en ændring i objektfunktionsværdien, betegnet ved $\Delta f_{i,k}$. Hvis det ikke er muligt at indsætte i i rute k, sættes $\Delta f_{i,k} = \infty$. Indsættelsen sker nu i den position, som øger objektfunktionsværdien mindst, dvs.

(6.22) $$\min_{i \in U}(c_i),$$

hvor U er mængden af ordrer, der ikke er tildelt en rute og $c_i = \min_{k \in K}\{\Delta f_{i,k}\}$ er omkostningen ved at indsætte ordre i i sin bedste position. Man vælger således ordren med den mindste stigning i objektfunktionsværdien, og indsætter denne i den bedste position. Processen gentages indtil alle ordrer er genindsat, eller det ikke længere er muligt at indsætte flere ordrer. En ulempe med heuristikken er, at den ofte udskyder indsættelse af dyre ordrer (dvs. høj c_i) til de sidste iterationer, hvor antallet er muligheder er begrænset. Den næste heuristik prøver at undgå dette problem.

Fortrydelsesheuristik

Fortrydelsesheuristikken (eng. "regret heuristic") forsøger at forbedre "basic greedy"-heuristikken ved at inkorporere en form for information, der ser "fremad" inden valget af indsættelsesordren. Lad $x_{ik} \in \{1,...,m\}$ være ruten for hvilken ordre i har den k'te laveste indsættelsesomkostning, dvs. $\Delta f_{i,x_{ik}} \leq \Delta f_{i,x_{ik'}}$ for $k \leq k'$. Heraf kan vi udtrykke $c_i = \Delta f_{i,x_{i1}}$. I fortrydelsesheuristikken defineres en fortrydelsesværdi $c_i^* = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}$, som angiver forskellen i omkostningerne ved at indsætte ordren i dennes bedste rute og den næstbedste rute. I hver iteration udvælger fortrydelsesheuristikken en ordre i der maksimerer

(6.23) $$\max_{i \in U} c_i^*.$$

Med andre ord vælges den ordre, hvis indsættelse vi ville fortryde mest, hvis ikke det skete omgående. Denne ordre indsættes da i den position, der minimerer omkostningen.

Det er også muligt at udvide fortrydelsesheuristikken til en såkaldt "regret-k"-heuristik, hvor vi i hvert konstruktionstrin tjekker følgende:

1. Hvis der er ordrer som ikke kan indsættes i mindst $m-k+1$ ruter, indsættes dén ordre der kan indsættes i færrest antal ruter.
2. Ellers indsættes den ordre i, der maksimerer udtrykket:

(6.24) $$\max_{i\in U}\left\{\sum_{j=2}^{k}\left(\Delta f_{i,x_{ij}}-\Delta f_{i,x_{i1}}\right)\right\}$$

3. Den valgte ordre (fra trin 1 eller 2) indsættes i positionen med den mindste omkostning.

Heuristikken i starten af dette afsnit svarer til en "regret-2"-heuristik. For $k>2$ undersøges de k bedste ruter for omkostningen ved indsættelse af en ordre, hvorefter man indsætter den ordre med størst forskel mellem omkostningen ved indsættelse i den bedste rute og indsættelse i den $k-1$'te rute er størst. Fordelen ved denne heuristik er, at de ordrer, som kun har få gode indsættelsesmuligheder, får højere prioritet, og dermed undgår vi at sidde tilbage med nogle ordrer, der ikke kan indsættes i de sidste iterationer.

6.8.4 Valg af fjernelses- og indsættelsesheuristik

Det er en mulighed at benytte én fjernelses- og én genindsættelsesheuristik gennem hele søgningen, men det anbefales at veksle mellem forskellige heuristikker, idet hver heuristik virker bedre end andre på forskellige typer af situationer. Måden man udvælger en heuristik på, er ved at tildele hver metode en vægt, hvorefter "Roulette Wheel"-metoden bliver brugt som baggrund for udvælgelsen. Hvis man har med k heuristikker med vægte $w_i, i\in\{1,2,...,k\}$ at gøre, udvælges heuristik j med sandsynlighed

(6.25) $$\frac{w_j}{\sum_{i=1}^{k}w_i}.$$

Det skal bemærkes at indsættelsesheuristikken og fjernelsesheuristikken vælges uafhængigt af hinanden.

6.8.5 "Adaptive weight adjustment"

Hvis der benyttes indtil flere indsættelses- og/eller fjernelsesheuristikker, kan det være en fordel at justere vægtene dynamisk, således at de heuristikker, der tilsyneladende har den bedste performance, vælges med større sandsynlighed. Idéen er blot, at der holdes styr på en heuristiks seneste performance ved en score, hvor en høj score svarer til en succesfuld heuristik.

Hele søgningen inddeles først i et antal segmenter, hvor et segment består af et antal iterationer. Initialscoren sættes til 0 for alle heuristikker i starten af hvert segment. Scoren øges nu med én af følgende parametre: σ_1, σ_2 og σ_3, som hver svarer til én af følgende situationer:

Parameter	Beskrivelse
σ_1	Når den sidste fjern-indsæt operation har resulteret i en ny global løsning.
σ_2	Når den sidste fjern-indsæt operation har resulteret i en løsning, der ikke er blevet accepteret før. Omkostningen for den nye løsning er bedre end omkostningen for den aktuelle løsning.
σ_3	Når den sidste fjern-indsæt operation har resulteret i en løsning, der ikke er blevet accepteret før. Omkostningen for den nye løsning er værre end for den aktuelle løsning, men accepteres alligevel.

Tabel 6: Parameterbeskrivelser

Situationen for σ_1 er enkel, idet heuristikken har været i stand til at finde en ny overordnet løsning. Hvis en heuristik har fundet en løsning der endnu ikke er blevet besøgt og bliver accepteret af ALNS acceptkriterium, betragtes heuristikken som succesfuld, da denne har ført søgningen fremad, og den belønnes med σ_2. Man foretrækker naturligvis heuristikker der forbedrer løsningen, men belønner også heuristikker i stand til at diversificere søgningen, her med σ_3.

Scoren for både fjernelses- og indsættelsesheuristikken opdateres med samme parameter, da det ikke er muligt at vide, hvorvidt det har været fjernelses- eller indsættelsesmetoden, som har været årsagen til en god løsning. I slutningen af hvert segment omdannes scorerne til nye vægte.

Vi lader nu w_{ij} være vægten for heuristik i brugt i segment j, hvor w_{ij} er den samme vægt som i formel (6.25). I første segment vægtes alle heuristikker ens, men efter det j'te segment er overstået, opdateres vægten for alle heuristikker i. De nye vægte anvendes derefter i segment $j+1$. Opdateringen sker som følgende:

(6.26) $$w_{i,j+1} = w_{ij} \cdot (1-r) + r \cdot \frac{\pi_i}{\theta_i}.$$

π_i betegner den score der er opnået i løbet af det sidste segment for heuristik i og θ_i angiver antal gange man har forsøgt at anvende heuristik i i løbet af det sidste segment. Formlen har en reaktionsfaktor r, som bestemmer hvor hurtigt vægtjusteringen reagerer på ændringer i effektiviteten af heuristikkerne. Hvis $r = 0$, ser man fuldstændig bort fra scorerne og holder sig til de initiale vægte. For $r = 1$ bestemmes vægtene udelukkende på baggrund af scoren, hvorfor man kan risikere, at en heuristik, der blot var "heldig" første gang, vil blive brugt i de resterende iterationer også. Fastsættelsen af r er derfor en balancegang.

6.8.6 Afsluttende kommentar

Vi kan sammenligne ALNS med andre heuristikker ved at bruge Li og Lims (2001) problemsæt af 354 problemer som benchmark. Metoden står den dag i dag som værende den bedste løsningsmetode for VRPPDTW i mere end 50 % af problemerne. Herunder ses en tabel over andelen af problemer, hvor denne metaheuristik har leveret den bedste løsning, afhængig af problemstørrelsen.

100 kunder	200 kunder	400 kunder	600 kunder	800 kunder	1000 kunder	I alt
0 ud af 56	25 ud af 60	41 ud af 60	51 ud af 60	47 ud af 60	51 ud af 58	215 ud af 354
0 %	41,7 %	68,3 %	85 %	78,3 %	87,9 %	60,7 %

Tabel 7: Kilde: http://www.sintef.no/projectweb/top/

Af tallene ses det, at metoden er særlig attraktiv for store problemstørrelser, men for mindre problemer overgås den af andre heuristikker.

6.9 Metaheuristik for VRPBTW

Denne udvidelse har modtaget meget begrænset opmærksomhed. Gelinas et al. (1995) har udvidet en branch-and-bound–algoritme, oprindeligt udviklet til VRPTW, til at klare backhauling. Simple konstruktions- og forbedringsalgoritmer er foreslået af Thangiah et al. (1996), mens Duhamel et al. (1997) har forslået en TS-algoritme til at takle problemet. Vi har dog valgt at beskrive en "Ant System"-algoritme til løsning af VRPBTW, baseret på Reimann et al. (2002).

6.9.1 AS-algoritme til VRPBTW

Denne algoritme består hovedsageligt af følgende tre trin:

1. Myrernes generering af løsninger på baggrund af et feromonspor.
2. Anvendelse af lokal søgning på myrernes løsninger.
3. Opdatering af feromonsporet.

Herunder gives en mere detaljeret beskrivelse af de tre trin.

Generering af løsninger

Reimann et al. (2002) indarbejder en indsættelsesalgoritme i AS som en løsningsgenererende teknik, i stedet for en nærmeste-nabo heuristik. Den heuristik som anvendes er udledt af Solomon's indsættelsesheuristik til VRPTW[24].

Denne heuristik fungerer således: I starten udvælges en "seed"-kunde til dannelsen af den første rute, og er fra dette tidspunkt eneste kunde i ruten. De resterende kunder indsættes én efter én i ruten indtil der ikke er flere mulige indsættelser mht. tidsvinduer, kapacitet eller rutelængde. På dette tidspunkt

[24] Blandt de undersøgte konstruktionsheuristikker, førte indsættelsesheuristikken til de bedste resultater.

initialiseres en ny rutemed en ny "seed"-kunde, hvorefter indsættelsesproceduren gentages for de resterende kunder, der endnu ikke tilhører en rute. Algoritmen stopper når alle kunder er tildelt en rute.

I denne indledende procedure skal der foretages to beslutninger. Den første er valget af en "seed"-kunde for hver rute, det andet beregningen af attraktiviteten for indsættelsen af en kunde i en rute. Beslutningerne er baseret på følgende kriterier, hvor η_i svarer til kunde i's attraktivitet:

Ruteinitialisering

(6.27) $$\eta_i = d_{0i} \qquad \forall i \in N_u,$$

hvor d_{0i} betegner afstanden mellem depotet og kunde i og N_u betegner mængden af kunder, som endnu ikke tilhører en rute. Denne ruteinitialisering foretrækker "seed"–kunder, som er placeret langt fra depotet.

Kundeindsættelse

(6.28) $$\eta_i = \max\left\{0, \max_{j \in R_{l_i}}\left[\alpha \cdot d_{0i} - \beta \cdot \left(d_{ji} + d_{is_j} - d_{js_j}\right) - (1-\beta)\cdot\left(b^i_{s_j} - b_{s_j}\right)\right]\right\},\ \forall i \in N_u,$$

hvor s_j er den kunde der besøges umiddelbart efter kunde j i den aktuelle løsning. $b^i_{s_j}$ er den faktiske ankomsttid hos kunde s_j, hvis i indsættes mellem kunde j og s_j, mens b_{s_j} er ankomsttiden hos kunde s_j før indsættelsen af kunde i. R_{l_i} betegner mængden af de kunder i den aktuelle rute, hvor det er muligt at indsætte kunde i. α og β er brugervalgte parametre. Reglen angiver for hver kunde det bedst mulige indsættelsessted. Eneste forskel i reglen fra Solomons indsættelsesheuristik ligger i, at afstanden mellem depotet og kunden har en indflydelse, så kunder langt fra depotet får en højere prioritet og indsættes tidligere. Vi bestemmer herefter den bedste kunde $i*$ at indsætte, således at $\eta_{i*} \geq \eta_i \ \ \forall i \in N_u$.

Det skal bemærkes, at det for hver kunde er nødvendigt at tjekke, om det er muligt at indsætte denne i rute i forhold til tidsvinduer. Specielt når antal kunder i ruterne er stort, kan man med fordel benytte sig af Solomons tjek for ”time feasibility”.

Da vi også skal tage højde for backhaul, forøges attraktiviteten på følgende vis:

(6.29) $$\eta_i = \max\left\{0, \max_{j \in R_{l_i}}\left[\alpha \cdot d_{0i} - \beta \cdot \left(d_{ji} + d_{is_j} - d_{js_j}\right) - (1-\beta)\cdot\left(b^i_{s_j} - b_{s_j}\right) + \gamma \cdot type_i\right]\right\},$$
$$\forall i \in N_u,$$

hvor $type_i$ er en binær indikatorvariabel, som angiver om en kunde i er linehaul- ($type_i = 0$) eller backhaul-kunde ($type_i = 1$). Intuitionen er, at man gerne vil diskriminere mellem linehaul- og backhaul–kunder, idet linehaul skal ske før backhaul.

Vi kan nu tilpasse ovenstående algoritme inden for rammerne af AS. Tilpasningen tillader en mulighed for et sandsynlighedsbaseret valg i hvert beslutningstrin. Algoritmen omskrives til følgende:

Første trin:

For hver kunde, som endnu ikke tilhører en rute, udregnes η_i kundens bedste indsættelsessted i den aktuelle rute som følgende:

(6.30)

$$\eta_i = \max\left\{0, \max_{j \in R_{l_i}}\left[\left(\alpha \cdot d_{0i} - \beta \cdot \left(d_{ji} + d_{is_j} - d_{js_j}\right) - (1-\beta)\cdot\left(b^i_{s_j} - b_{s_j}\right) + \gamma \cdot type_i\right) \cdot \frac{\tau_{ij} + \tau_{is_j}}{2 \cdot \tau_{js_j}}\right]\right\}, \quad \forall i \in N_u,$$

hvor τ_{ij} betegner styrken af feromonsporet på kanten, der forbinder i og j. Feromonsporet indeholder information om, hvorvidt kanten mellem i og j tidligere har resulteret i gode løsninger Bemærk, at udtrykket $\frac{\tau_{ij} + \tau_{is_j}}{2 \cdot \tau_{js_j}}$ er større end 1, hvis den gennemsnitlige feromonværdi af de to tilføjede kanter er større end feromon-værdien af den slettede kant. På den måde medtages feromonsporet i bestemmelsen af næste kunde til indsættelse.

Andet trin:

Der anvendes en "Roulette Wheel" udvælgelse for alle kunder, der endnu ikke tilhører en rute og har en positiv η_i. Beslutningsreglen ser således ud:

(6.31)
$$P_i = \begin{cases} \dfrac{\eta_i}{\sum_{h|\eta_h>0} \eta_h} & , hvis \ \ \eta_i > 0 \\ 0 & , ellers \end{cases}$$

Den valgte kunde i indsættes derefter i den aktuelle rute, i dennes bedst mulige indsættelsesposition.

Lokal søgning

Efter at en myre har konstrueret sin løsning, anvendes en lokal søgningsalgoritme til det formål at forbedre løsningskvaliteten. De anvendte forbedringsoperatorer er "swap" og "move". Swap-operatoren forbedrer løsningen ved at udveksle en kunde i med en anden kunde j. Move-operatoren prøver at fjerne en kunde i fra dennes aktuelle position og indsætte den et andet sted.

Operatorerne anvendes i en specifik rækkefølge, hvor forfatterne først anvender move og derefter swap. Operatorerne anvendes indtil der ikke er flere forbedringer. En vigtig bemærkning er, at ikke-

mulige løsninger ikke accepteres. Den lokale søgning i denne algoritme er begrænset til $\lambda = 1$, idet de ekstra begrænsninger der hører til VRP-udvidelserne fører til ikke-mulige løsninger for $\lambda > 1$.

Opdatering af feromonsporet

Feromonsporet opdateres på basis af de løsninger myrerne har konstrueret ved følgende opdateringsregel:

(6.32) $$\tau_{ij} = \rho \cdot \tau_{ij} + \sum_{\mu=1}^{p} \Delta\tau_{ij}^{\mu} + \sigma\Delta\tau_{ij}^{*}$$

hvor $0 \le \rho \le 1$ er kantens vedholdenhed og $\sigma = p + 1$ er antallet af gange kanten i den hidtil bedste løsning skal medtages. Der lægges feromon ad to typer af løsninger. Den første bruger den hidtil bedste løsning til at opdatere sporet, som om σ myrer havde konstrueret denne løsning. Mængden af feromon som lægges er $\Delta\tau_{ij}^{*} = 1/L^{*}$, hvor L^{*} er objektfunktionsværdien for den hidtil bedste løsning. For det andet bruges iterationens p bedste myrer til at opdatere sporet. Mængden, som disse myrer lægger, afhænger af deres rang μ samt deres løsningskvalitet L^{μ}, således at den μ'te bedste myre lægger $\Delta\tau_{ij}^{\mu} = \frac{(p - \mu + 1)}{L^{\mu}}$. Kanter der ikke indgår i disse myrers løsninger, mister feromon med en rate på $(1 - \rho)$.

6.9.2 Afsluttende kommentar

Da denne udvidelse kun i begrænset omfang er blevet behandlet i litteraturen, kan det umiddelbart være svært at vurdere, hvorvidt den valgte metode er konkurrencedygtig. Reimann et al. (2002) sammenligner deres metaheuristik med den klassiske heuristik af Thangiah et al. (1996). Resultatet viser, at Reimanns heuristik overperformer i 10 ud af 15 tilfælde, noget der igen indikerer at metaheuristikker er at foretrække frem for den klassiske heuristik.

6.10 Metaheuristik for OVRPTW

Alle forrige udvidelser vi indtil nu har behandlet har været med hårde tidsvinduer. I forbindelse med OVRPTW er der i litteraturen ikke tilstrækkeligt tilgængeligt materiale, der behandler problemet med hårdre tidsvinduer, hvorfor vi vil tage udgangspunkt i en metaheuristik for OVRPSTW (eng. "soft time windows").

6.10.1 Initialløsning

Til løsning af OVRPSTW benyttes en TS-algoritme af Fenghua og Xiaonian (2008), hvor den lokale søgning starter fra en initialløsning, baseret på en "Farthest First"-heuristik (FFH). Da ruterne i OVRPSTW er åbne, ser man ofte i de bedste løsninger, at kunden længst fra depotet er den sidste kunde i ruten. Baseret på denne observation, er det i litteraturen foreslået, at nye ruter altid dannes fra kunden længst fra depotet.

Den grundlæggende idé med FFH er følgende: En ny rute starter fra kunde i, som er den kunde længst fra depotet som endnu ikke tilhører en rute. Derefter konstrueres den korteste rute tilbage til depotet fra kunde i ved at tilføje kunder til den aktuelle rute, indtil køretøjet er fyldt nok op. Hvis køretøjet ikke fyldes tilstrækkeligt op, afvises ruten, hvorefter den næst-korteste rute overvejes. Hvis denne heller ikke accepteres, undersøges den tredje-korteste rute, osv. Til at bestemme en sådan k'te-korteste rute kan f.eks. benyttes Lawler (1972). Processen gentages så længe en rute accepteres. Proceduren er beskrevet mere detaljeret herunder.

1. Bestem den korteste afstand mellem depotet og hver kunde.
2. Start en ny rute fra den kunde i, der er længst fra depotet og endnu ikke tilhører en rute. Sæt $k = 0$.
3. $k = k + 1$.
4. Kunder j langs den k'te-korteste rute tilbage fra i til depotet tilføjes nu én for én den aktuelle rute, så længe både $\left(d_i + \sum_j d_j \le C\right)$ og den totalt tilbagelagte afstand for ruten $\le L$.
5. Hvis køretøjets restkapacitet er mindre end $\min\left(\text{ethvert ikke tildelt } d_l, (C - C_{avg})\right)$ eller alle kunder er tilføjet, accepteres ruten, hvor l er den kunde endnu ikke tilføjet ruten med den laveste efterspørgsel.
6. Gentag trin 3-5 indtil en rute accepteres.
7. Hvis alle kunder tilhører en rute afsluttes FFH, ellers forsæt fra trin 2.

Målet er at holde antal køretøjer mindst mulig for en løsning, hvorfor det ikke er økonomisk fordelagtigt at starte en ny rute, indtil det aktuelle køretøj er fyldt nok op. Lad $\sum_{i \in V \setminus \{0\}} d_i$ være den samlede efterspørgsel af alle kunder og $K_{\min}$ det mindste antal køretøjer nødvendigt for at servicere alle kunder. Ofte bestemmes $K_{\min}$ som $K_{\min} = \sum_{i \in V \setminus \{0\}} d_i \big/ C$, hvorfor vi kan angive den gennemsnitlige kapacitet nødvendig for at servicere alle kunder med det mindst mulige antal køretøjer som

(6.33) $$C_{avg} = \frac{\sum_{i \in V \setminus \{0\}} d_i}{K_{\min}}$$

I et tjek om køretøjet er fyldt "nok" op, sammenholdes restkapaciteten med forskellen mellem køretøjets kapacitet og den gennemsnitlige kapacitet nødvendig for at servicere alle kunder med det mindste antal køretøjer.

6.10.2 Nabo-områdets struktur

Der kan anvendes forskellige træk på den aktuelle løsning, som svarer til at bevæge sig til en naboløsning. Disse træk er (a) node-omplacering, (b) node-swap, (c) 2-opt for TSP og (d) en "hale"-swap. Et eksempel på hver af disse træk ses herunder, hvor de valgte noder er markeret med en understreg.

Udvælg to tilfældige noder (kunde eller depot) og udfør tilfældigt ét af de fire følgende træk:

a) *Nodeomplacering:* Fjern den først valgte node fra dennes aktuelle position i ruten, og indsæt den et andet sted før (eller efter for pickup-problemer) den anden valgte node. Eksempel: $X_1 = (01\underline{3}560479028\underline{)} \rightarrow X_2 = (015604790\underline{38})$.

b) *Node-swap:* Positionen af de to valgte noder udveksles. Dvs. $X_1 = (0135\underline{0}6\underline{4}79028) \rightarrow X_2 = (0135\underline{4}6\underline{0}79028)$.

c) *2-opt:* Byt om på rækkefølgen af alle elementer mellem de to valgte noder. Eksempel: $X_1 = (01\underline{3}56\underline{4}079028) \rightarrow X_2 = (01\underline{4}65\underline{3}079028)$.

d) *"Hale"-swap:* Her udveksles "halen" efter to valgte noder (inkl. de valgte noder), begge skal være kunder fra hver sin rute. Eksempel: $X_1 = (013\underline{5}60\underline{4}79028) \rightarrow X_2 = (013\underline{9}90\underline{4}56028)$.

Bemærk, at (a)-(c) er træk, der muligvis fører til en reduktion af antal ruter.

6.10.3 Løsningsevaluering

Uanset total rutelængde, dominerer en mulig løsning med et mindre antal køretøjer altid en løsning med et større. Mellem løsninger med samme antal køretøjer vælges den mindste totale omkostning. For at lette undersøgelsen af søgningsrummet, tillades træk, der resulterer i ikke-mulige løsninger i forhold til kapacitet eller rutelængde. Graden af den ikke-mulige løsning kan måles ved at inkorporere kapacitetsbegrænsningerne og den maksimale rutelængde i objektfunktionen ved siden af tidsvinduerne.

6.10.4 Tabuliste

En tabuliste indeholder trækkets attribut for løsninger inden for de sidste 5-10 iterationer. En mængde af $(n+1)\times(n+1)$ matricer kan evt. konstrueres til det formål at gemme de tabuaktive elementer. F.eks. hvis node i vælges til trækket (a), gemmes tabu–statussen, repræsenteret som et antal iterationer, i matricens element (i,i). Hvis man derimod vælger node i og j til trækkene (b)-(d), gemmes tabu-statussen i matricens element (i,j). Efter hver iteration tilføjes tabu-statussen for det sidste træk til listen, mens statussen for de andre aftager med én iteration, indtil de er lig nul og dermed ikke er tabu længere.

6.10.5 Stopkriterier

Algoritmen stoppes, hvis der enten er forløbet et på forhånd specificeret antal iterationer eller et vist antal iterationer uden en forbedring. Følgende variable anvendes til at angive stopkriterierne i de enkelte trin af TS-heuristikken:

- $Iter$ *:* Det nuværende antal af iterationer.
- Max_iter *:* Det maksimale antal iterationer.
- $Cons\_iter$ *:* Det nuværende antal af iterationer i træk uden forbedring af den hidtil bedste løsning.
- Max_cons_iter *:* Det maksimale antal af iterationer i træk uden forbedring af den hidtil bedste løsning.
- $Cand\_list$ *:* Det aktuelle antal naboløsninger der indgår i en kandidatliste.
- Max_cand_list *:* Det maksimale antal kandidater i kandidatlisten.

6.10.6 TS-algoritme

1. Generér en initialløsning enten tilfældigt eller på baggrund af FFH algoritmen. Sæt denne løsning som den aktuelle og den hidtil bedste løsning.
2. Sæt $Iter = 0$ og $Cons\_iter = 0$.
3. Så længe $\left(Cand\_list \le Max\_cand\_list\right)$, udfør følgende:
    a. Vælg to tilfældige noder.
    b. Vælg tilfældigt ét af de fire nabotræk.
    c. Udfør trækket og tilføj naboløsningen til kandidat listen.
4. Vælg den bedste ikke-tabu løsning fra kandidatlisten. Tabulisten tilsidesættes, hvis et tabutræk har ført til en ny hidtil bedste løsning.
5. Den nye løsning sættes som den aktuelle. Opdatér tabulisten og sæt $Iter = Iter + 1$.
6. Hvis den nye løsning er bedre end den hidtil bedste løsning, opdateres sidstnævnte og sæt $Cons\_iter = 0$, ellers sæt $Cons\_iter = Cons\_iter + 1$.
7. Hvis $\left(Iter \le Max\_iter\right)$ og $\left(Cons\_iter \le Max\_cons\_iter\right)$, fortsæt fra trin 3, ellers standses algoritmen.

Denne TS-algoritme benytter sig af en simpel, men stærk, nabostruktur. I løbet af søgningen vælges nabo-området tilfældigt blandt de fire typer træk, mens tabulængden varieres tilfældigt mellem 5 og 10 iterationer. Denne mekanisme introducerer en stokastisk type diversificering. Endvidere, når der veksles mellem trækkene i hver iteration, udforskes et større nabo-område i løbet af få iterationer. Anvendelsen af en straf-funktion tillader undersøgelse af ikke-mulige løsninger.

6.10.7 Hårde tidsvinduer

Ovenstående algoritme kan formentlig levere en løsning for hårde tidsvinduer, men hvis man vil være sikker på en tidsmæssig mulig løsning, vil vi være nødt til at tjekke for ”time feasibility” ved indsættelse af hver kunde i FFH-algoritmen (vi antager, at det er muligt at servicere enhver kunde indenfor depotets tidsvindue). Indsættelsen sker herefter én ad gangen præcis som metoden foreskriver. Enhver indsat kunde (inkl. den første kunde) antager sit senest mulige serviceringstidspunkt, således at indsættelse af nye kunder kan ske tidsmæssigt før. Når en kunde vurderes til indsættelse, ved vi at den kan nås fra depotet, men tjekker at de efterfølgende kunder tidsmæssigt ikke skubbes ud over deres tidsvinduer.

Efter dannelsen af den initiale rute, kunne man afslutningsvis inkludere en forbedringsalgoritme, som har til formål at effektivisere placeringen af et udvalg af kunder, ikke mindst i forhold til deres tidsvinduer. Som eksempel kan bruges en λ-opt, der udelukkende præsenterer mulige forbedringer. Heuristikken gentages, indtil der ikke er flere forbedringer.

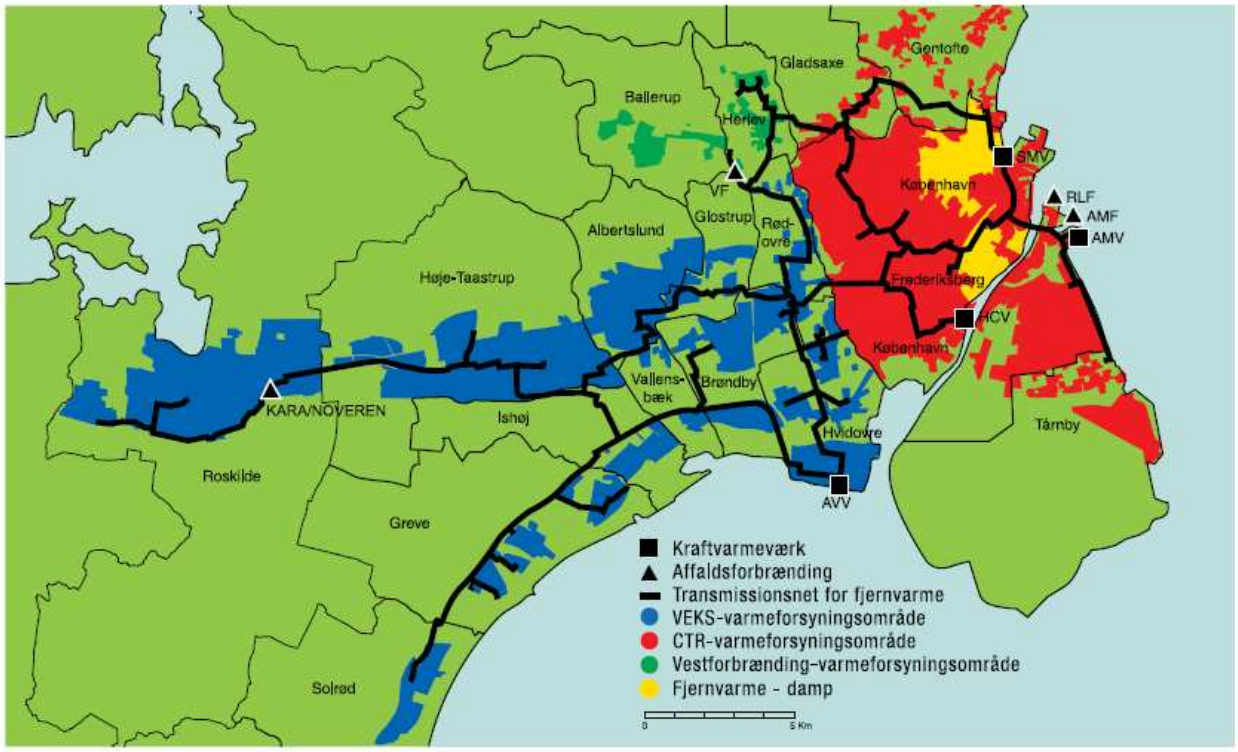
I TS-algoritmen indsættes et trin mellem 3b og 3c som tjekker for ”time feasibility”, og således om tidsvinduet er overholdt efter hvert af de træk der kan foretages. Dermed vil hver eneste kandidatløsning være ”time feasible”, men ikke nødvendigvis mulig mht. kapacitetsbegrænsninger eller en evt. maksimal rutelængde.

6.10.8 Afsluttende kommentar

Ligesom for metaheuristikken for VRPBTW findes få eller ingen muligheder for at sammenligne med andre algoritmer til denne type udvidelser. I praktiske situationer er der naturligvis mange typer begrænsninger, som man vil være nødt til at tage højde for i en løsningsmetode. Den metaheuristik vi her har præsenteret, er ikke udarbejdet til at fungere for hvert enkelt problemtilfælde man kan støde på. Derfor vil det i praksis være et spørgsmål om at udvælge en metode, der overordnet har en høj performance, men også let kan tilpasses sådanne ekstra begrænsninger. Vores mål med forslagene er blot et forsøg på at tilpasse metoden til en simpel faktor som tidsvinduer.

Kapitel 7 – Et Fjernvarmenetværk

Som afslutning på denne afhandling har vi valgt at beskæftige os med et VRP fra den virkelige verden, om end et noget alternativt problem. På nedenstående figur ses et transmissionsnetværk til at forsyne København og omegn med fjernvarme. Netværket er opdelt i tre forskellige ansvarsområder, hvert tilhørende de tre transmissionsselskaber, CTR, VEKS og Vestforbrænding. Det er et kompliceret netværk, dannet på baggrund af mange forskellige hensyn og forbehold, hvilket gør det vanskeligt, men – som vi vil vise på de følgende sider – ikke umuligt at betragte problemet som et VRP.



Figur 29: Transmissionsnetværk til transport af fjernvarme rundt til kommunerne i København og omegn.

Selve netværket er naturligvis lagt fast; rørene er gravet ned og omkostningerne ved at omarrangere er enorme. Vi skal derfor forestille os, at vi sidder med i beslutningsfasen tilbage i begyndelsen af 80'erne, hvor planlægningen til et nyt transmissionsnetværk stod på. Hvordan kunne vi forestille os dette netværk skulle se ud, hvis vi bedst muligt skulle transportere det varme vand fra kraftvarmeværkerne og ud til forbrugerne samt tilbage igen? Det er dette de følgende afsnit handler om.

7.1 Københavns fjernvarmenetværk i store træk

Vi kan kort beskrive fjernvarme som leveringen af varme og varmt vand til et antal bygninger fra et centralt varmeværk. Varmen fra værket transporteres og distribueres som varmt vand der cirkulerer i et netværk, bestående af et dobbelt rørsystem. Gennem det ene rør sendes varmt vand til forbrugerne og når vandet har afgivet sin energi i form af varme, returneres det kolde vand via det andet rør til genopvarmning i varmeværket.

Disse varmeværker findes typisk som kraftvarmeværker, dvs. kraftværker, der udnytter den overskudsvarme som opstår i forbindelse med produktion af el til at producere varme. Overskudsvarmen skulle ellers have været kølet med havvand eller i køletårne, med andre ord var den gået til spilde, men ved at udnytte overskudsvarmen udnytter man altså brændslet langt bedre. Hvor et typisk kraftværk udnytter ca. 40 % af energien til produktion af el, så kan 55 % udnyttes til fjernvarme, hvilket bringer den samlede udnyttelse op i nærheden af 95 %.

Der findes andre typer varmeproducenter, som forbrændingsanlæg, geotermianlæg, spids- og reservelastanlæg og nogle industrielle virksomheder kan ligeledes udnytte overskudsvarmen fra produktionen og levere til fjernvarmenettet. I et forbrændingsanlæg udnyttes overskudsvarmen som dannes ved afbrændingen af store mængder affald, og i et geotermianlæg udnyttes varmen i undergrunden til at hente varmt vand op og distribuere det som fjernvarme, hvorefter det afkølede vand pumpes ned i undergrunden igen. Et spids- og reservelastanlæg sættes kun i gang, når det er særlig koldt, og behovet for varme er større end leverancerne fra grundlastanlæggene; kraftvarmeværker, geotermianlæg og forbrændingsanlæg.

Varmen sendes fra varmeværkerne ud til et omfattende rørnet, et såkaldt transmissionsnet, som forbinder varmeproducenterne med boligområderne gennem et antal centrale underjordiske vekslerstationer. Vekslerstationernes formål er at sikre, at varmen overføres fra transmissionsnettet til distributionsnettet og videre til forbrugerne.

Nogle vekslerstationer inkluderer såkaldte boosterpumper til at presse vandet igennem rørene, og som er nødvendige for en kontinuert cirkulation af vandet i transmissionsnettet. Det er typisk brugt til transport af vand fra lavere til højere beliggende områder med lavere tryk eller, og nok mere relevant for Københavnsområdet, på særligt lange strækninger i netværket.

Nogle områder af København modtager fjernvarme i form af damp i stedet for vand. Men dette er et helt separat netværk, hvorfor vi ikke yderligere vil beskæftige os med dette.

7.2 Fokusområde for casestudie

Vi har valgt specifikt at beskæftige os med det overordnede transmissionsnetværk, som forbinder varmeproducerende værker med vekslerstationerne, hvilket har én primær årsag: Vi kan af praktiske årsager ikke beskæftige os med hver enkelt distributionsledning, da dette kan opgøres i mange tusinder ledningssektioner med en længde på i alt 1.142 km[25].

VRP består derudover udelukkende af tre typer aktører; depot, kunder og køretøjer. Da strømmen af vand er køretøjerne der kører langs ruter af transmissionsledninger, skal vi reelt fordele anlæggene på de sidste to aktører, en kraftig simplificering af problemet. Vi har valgt at lade kraftvarmeværk, forbrændingsanlæg og geotermianlæg indgå som depoter pga. deres vedvarende daglige leveringer af varme til transmissionsnettet, mens spidslastanlæg, vekslerstationer og boosterpumpestationer indgår som kunder i problemet.

Forkortelse	Navn	Adresse
AMV	Amagerværket	Kraftværksvej 50, 2300 København S
AVV	Avedøreværket	Hammerholmen 50, 2650 Hvidovre
HCV	H.C. Ørstedværket	Tømmergravsgade 4, 2450 København SV
SMV	Svanemølleværket	Lautrupsgade 1, 2100 København Ø
VEV	Vestforbrænding	Ejby Mosevej 219, 2600 Glostrup
KNV	KARA/NOVEREN	Håndværkervej 70, 4000 Roskilde

Tabel 8: placeringen af de 6 centrale varmeproducerende værker i København[26].

For at passe problemet ind i en VRP-sammenhæng, har vi ladet samtlige spidslastanlæg indgå som vekslerstationer, da disse kun tages i brug i særligt kolde perioder, hvor efterspørgslen er langt højere end normalt, og dermed ikke kan regnes blandt de daglige faste varmeproducenter. Spidslast udgjorde i 2008 og 2009 kun mellem 1-2 % af varmen i CTR's transmissionsnet i forhold til grundlast, og i tidligere år endnu mindre. Fordi der er så stor usikkerhed med hensyn til, hvornår spidslast er nødvendig, har vi derfor valgt denne løsning som alternativ til de to ekstremer, helt at se bort fra spidslastanlæg og at lade dem indgå som et fast pålideligt varmeværk.

Samme lokalitet har ofte flere anlæg[27], f.eks. både forbrændingsanlæg og vekslerstation, eller veksler- og boosterpumpestation. Da det ikke er relevant at lade flere af sådanne anlæg med samme placering indgå – de ændrer ikke løsningen i nævneværdig grad – vil vi kun lade den vigtigste station indgå i løsningen, baseret på følgende rækkefølge i faldende orden efter relevans: Kraftvarmeværk, forbrændingsanlæg, geotermianlæg, spidslastanlæg, vekslerstation, boosterpumpestation.

Boosterpumper er her ofte placeret i forbindelse med en vekslerstation, men i nogle tilfælde også som en separat station i netværket. I forbindelse med en anden station gælder relevanskriteriet ovenfor og

[25] http://www.ke.dk
[26] Se bilag 1 for udvidet tabel med samtlige anlæg.
[27] Vi definerer to anlæg til at have samme lokalitet, såfremt de ligger indenfor 300 m fra hinanden.

kun den vigtigste tages med. Som separat station lader vi den indgå som en separat kunde i problemet, på højde med både spidslastanlæg og vekslerstationer.

7.3 Transmissionsnetværkets struktur

Det nuværende transmissionsnetværk er langt fra optimalt placeret i forhold dets anlæg, hvilket kommer sig af, at dets struktur er stærkt påvirket af de mange forskellige forhold der har spillet ind og fortsat spiller ind. Mange af disse forhold er ikke praktisk mulige at tage højde for, andre vil kræve oplysninger som ikke umiddelbart er til at få fat på. Vores forslag til et mere optimalt transmissionsnetværk vil derfor formentlig ikke være praktisk anvendeligt af mange forskellige årsager.

Fjernvarmenetværket er naturligvis blevet bygget lang tid efter København og dens omfattende infrastruktur. Bygningerne står, parker og søer er lagt i overensstemmelse med en overordnet byplan og der er i forvejen et kompliceret net af forsyningsledninger til blandt andet kloak, vand, el og telefon, specielt i de tættest beboede områder, hvorfor rørene sådanne steder er nødt til ofte at skifte retning. Det er derfor ikke muligt rent praktisk at lægge rørsystemet efter forgodtbefindende og optimalt. Derfor ses oftest at rørene er placeret i forhold til vejnettet, men vi finder også rør under parker, søer og på tværs af vejnettet nogle af de steder hvor det er muligt – der er med andre ord improviseret i visse områder af nettet. De muligheder, vi har rent beregningsmæssigt, kan ikke tage højde for disse forhold, og vi er nødt til at vælge, hvorvidt vores rute skal bestemmes som fugleflugtslinjer mellem stationerne eller ved brug af det overordnede vejnet (som et køretøj normalt ville følge). Da rørene som sagt overvejende er lagt under gader og veje, har vi valgt sidstnævnte løsning.

Øvrige forhold som f.eks. eksisterende netværk har vi valgt ikke at beskæftige os med af hensyn til problemets i forvejen tilstrækkeligt komplicerede natur. Som eksempel kan nævnes den nye Metro Cityring, hvor det har vist sig nødvendigt at ombygge dele af transmissionsledningerne ved Vasbygade og Østerport Station for at gøre plads. Yderligere detaljer vedrørende omlægningerne kendes ikke.

Ydermere er transmissionssystemet bygget af en sammenslutning af interessentkommuner, kaldet hhv. CTR og VEKS (deres ansvarsområder ses af figur 1). Politisk skal byggerierne godkendes og tilladelser gives, et område med potentielt væsentlige forskelle mellem kommunerne. I beslutningsfasen tog det således også væsentlig tid at blive enige om placeringen af kraftvarmeværkerne. Dernæst viste det sig umuligt for kommunerne at blive enige om etableringen af ét fælles transmissionsselskab, hvilket i sidste ende ledte til dannelsen af CTR og VEKS. Således spiller politiske forhold også ind og har påvirket netværket til hvad det er i dag.

Det skal bemærkes, at en del økonomiske overvejelser ligeledes har været inde og påvirke strukturen. Transmission af fjernvarme drejer sig i bund og grund om køb og salg af varme, hvor det enkelte transmissionsselskab køber varmen hvor det er billigst; her er affaldsvarme billigst og spidslast forholdsvist dyrere at anvende. Det er også muligt for selskaberne at købe og sælge varme af hinanden, alt efter behovet og hvorvidt der er ledig kapacitet at bruge af. For at dette fortsat skal være muligt, er vi nødt til at sørge for at hele netværket er forbundet, dels på tværs af de varmeproducerede værker og

dels på tværs af transmissionsselskabernes ansvarsområder. Hvordan vi vil sikre at netværket er således forbundet, skal vi komme ind på senere.

Disse forbindelser er sådan set ikke nødvendige i den forstand, at såfremt varmeværkerne har tilstrækkelig høj kapacitet, kan de levere varme til hvert deres område. Men der er visse økonomiske fordele ved at lade alle disse anlæg være forbundet, blandt andet en fleksibilitet der gør, at hvis efterspørgslen er større end produktionsanlægget kan levere, kan der trækkes på andre værker for at supplere den nuværende produktion.

Én af disse overvejelser er mundet ud i, at en større andel af transmissionssystemet er lagt som et 54 km langt ringsystem. Beslutningen er baseret på en række overvejelser af dels økonomisk karakter, men også af hensyn til et pålideligt udbud, der gør, at der for det første til ethvert tidspunkt kan trækkes varme fra det mest økonomisk fordelagtige produktionsanlæg og at der for det andet findes alternativt udbud, hvis der skulle ske et brud på en rørledning eller et varmeværk skulle sætte ud. I sådanne tilfælde kan strømmen i rørsystemet vendes, så varmen har mulighed for at sendes begge veje rundt i ringen. Logisk vil dette også betyde, at der vil være et sted i denne ring, hvor strømmen står stille. Igen er dette svært at implementere i en løsningsmetode, dels at strømme kan gå begge veje og vendes efter behag, dels at nogle dele af netværket er lagt som ringsystemer og andre som strækninger. Vi har valgt at abstrahere fra strømmens retning i netværket samt at se bort fra muligheden for et sådant ringsystem, som principielt ikke er nødvendigt – blot en sikkerhedsforanstaltning.

7.4 Efterspørgslens betydning for problemet

Her er der tale om en efterspørgsel efter en mængde varmt vand, hvor mængdebegrebet oftest er MJ/s, som er en energienhed eller varmeeffekt. Kommunernes fjernvarmeefterspørgsel er forskellig og det er forskelligt hvor meget der skal leveres til hver enkelt vekslerstation. Så vidt vi er bekendt med, er der ingen truende begrænsninger på de enkelte vekslerstationer ang. hvor meget der kan behandles. Der er imidlertid en kapacitet på den samlede varmemængde, der kan leveres i sekundet fra alle varmeværkerne tilsammen. Hos CTR er kapaciteten af de tilsluttede produktionsanlæg 1.931 MJ/s og i den time i 2009, hvor CTR havde den maksimale leverance til interessentkommunerne, var den samlede efterspørgsel 1.368 MJ/s – dvs. en reserve på 563 MJ/s. Dermed kommer vi end ikke i nærheden af kapaciteten.

Forbrændingsanlæggenes varmeproduktion er en væsentlig usikkerhedsfaktor for netværkets samlede kapacitet og leveret varme, fordi ”varmekvaliteten” varierer meget grundet enorme svingninger i brændselsværdien af affaldet. Da affaldsvarmen i gennemsnit udgør ca. 30 % af den samlede varmeproduktion i CTR-området, er dette bestemt en væsentlig ubekendt.

Efterspørgslen afhænger til gengæld af en meget længere række forhold, hvor vi først og fremmest finder vejret, der let kan skifte fra det ene øjeblik til det næste. Temperaturen har en del at sige, og alt andet lige vil periode der er gennemsnitligt varmere end en anden have en lavere varmeefterspørgsel. Temperaturens påvirkning måles i form af graddage, der er et generelt mål for en bygnings varmebehov.

Det bestemmes som en forskel mellem en indendørstemperatur på 17 grader og den gennemsnitlige udendørstemperatur.

Vejret betyder at efterspørgslen varierer fra time til time, vinter til sommer, år til år. Derudover er der en række systematiske svingninger i efterspørgslen på baggrund af døgnets rytme. Om morgenen bruges en del varme til bad samt te/kaffe og anden madlavning, og det samme sker for de fleste hen på eftermiddagen efter arbejdstid. I weekenderne er almindelige mennesker hjemme og der forbruges tilsvarende mere. Ydermere findes en række forskelle på tværs af geografisk beliggenhed.

Det meste af Københavnsområdet er allerede tilknyttet fjernvarmenetværket, men tilslutningen af nye kunder kan også have en permanent effekt på efterspørgslen. Effekten sker dog forholdsvis langsomt, i takt med at nettet udvides. Specielt i yderområderne findes stadig mange muligheder for ekspansion af netværket.

Således er der utrolig mange faktorer der påvirker efterspørgslen, som vi ikke kan tage med i vores formulering af problemet. Da efterspørgslen på trods af alle disse påvirkninger stadig er væsentlig under kapaciteten, har vi vurderet at vi med fordel kan se bort fra sådan en begrænsning.

Det står nu klart, at vi kan formulere problemet uden kapacitetsbegrænsninger, og dermed slipper vi også for at holde styr på den leverede mængde, da denne altid vil kunne lade sig gøre i sådanne omgivelser. Desuden varierer den konstant og besværliggør problemet væsentligt. I praksis reguleres og overvåges den leverede mængde da også manuelt af en døgnbemandet driftscentral hos både CTR og VEKS.

7.5 Klassificering af problemtype

Af ovenstående beskrivelse af netværket, vil de stærkeste karakteristika være dem for et OVRP, men altså uden hverken kapacitetsbegrænsninger eller tidsvinduer, hvilket reducerer problemet til et spørgsmål om at konstruere den korteste vej gennem netværket. Baggrunden for denne betragtning ligger i transmissionsnetværkets dobbelte rørsystem, der transporterer varmt og koldt vand til hhv. vekslerstationerne og tilbage ad den selvsamme rute. Som vi husker, er det grundlæggende tema i det åbne VRP, at køretøjerne enten ikke vender tilbage til depotet efter at have besøgt kunderne i ruten eller at de gør det ved at besøge samtlige kunder i omvendt rækkefølge. Denne sidste beskrivelse passer godt til et fjernvarmerørsystem.

Som det ses af figur 1, ligner problemet grafisk ikke er OVRP, netop pga. de strukturmæssige overvejelser man har gjort sig mht. dets udseende, som beskrevet tidligere. Den indre del af København minder således mere som et klassisk VRP i den forstand, at køretøjerne vender tilbage til depotet og dermed danner ringe. Men der er også her strukturmæssige ”afstikkere” fra denne formulering. Vi har herfra valgt at gå videre med OVRP-betragtningen, som klart er den mest teoretiske tiltalende af de to.

Ydermere har vi adskillige varmeværker i netværket, som hver udgør et depot i problemet. Reelt har vi derfor en kombination af et OVRP og et såkaldt ”multi-depot VRP” eller MDVRP, som vi kalder et OMDVRP. De egenskaber de to standard problemer har tilfælles kan stilles op som følgende:

1) Homogene køretøjer med kendt kapacitet.
2) Et ubegrænset antal køretøjer.
3) Hver kunde besøges én gang af ét og kun ét køretøj.
4) Kunderne i en rute har en samlet efterspørgsel der er mindre end køretøjets kapacitet.
5) Hvert køretøj har kun én rute (ingen dobbeltafgange).
6) Bestem mængden af ruter og minimér den totale afstand.

I dette tilfælde har vi ingen kapacitetsbegrænsninger, hvorfor punkt 4) er overflødig. Bemærk, løsningen til et OMDVRP vil ikke være et forbundet netværk, men en række separate OVRP-løsninger for hvert depot og eneste sammenhæng er, at hver kunde kun må tildeles ét køretøj og besøges én gang. Dette står i skarp kontrast til vores problem, som indikerer et tæt forbundet netværk. I næste afsnit vil vi præsentere en løsningsmetode for sådan en variant af et OMDVRP.

7.6 Præsentation af løsningsmetode

Én af de nyere software-løsninger der eksisterer for VRP er et internetbaseret ruteplanlægningsprogram kaldet "logvrp"[28]. Men fordi det er et forholdsvist nyt produkt, har det også en del mangler og begrænsninger, heriblandt muligheden for kun at afsætte ét depot og et meget begrænset sæt af algoritmer til løsning af problemet. Til gengæld kan sidstnævnte bestemme løsninger til en bred vifte af problemer, blandt andet OVRP, hvilket gør at vi på trods af disse væsentlige mangler alligevel vælger at benytte os af det.

Den største udfordring med at tilpasse problemet til programmet ligger i, at programmet ikke automatisk kan behandle flere depoter. Det betyder at vi selv er nødt til at indlede løsningen med at inddele kunderne i clusters, som hver serviceres af ét depot, en inddeling som er fast gennem løsningen af problemet.

Den anden udfordring er, at netværket i vores situation skal være forbundet og ikke et antal separate OVRP-netværk. Disse forbindelser vil vi være nødt til at fastlægge efter ruterne er dannet.

Vores løsningsmetode består således af 3 faser, bestående af:

1) Tildel alle kunder et depot,
2) konstruér et antal OVRP-løsninger og
3) forbind løsningerne til ét stort netværk.

[28] http://logvrp.com

7.6.1 Tildel alle kunder et depot

Vi vil præsentere to muligheder for dannelsen af initiale clusters: (1) Første metode er blot en simpel tildeling af kunderne efter nærmeste afstand til depoterne. (2) Den anden sker i 2 faser; først tildeles en mængde kunder sit nærmeste depot, dernæst tildeles resten iterativt i forhold til deres afstand til nærmeste cluster.

Første fase: For hvert depot $d_i \in D$, dan en cirkel med radius r:

(7.1) $$r = \eta \cdot \min_{j \in D \setminus \{i\}} \left(\left| d_i, d_j \right| \right),$$

hvor $\eta > 0$ betegner andelen af afstanden til nærmeste andet depot, der som radius i en cirkel om depotet specificerer mængden af kunder der initialt tildeles nærmeste depot.

Anden fase: Bestem for alle kunder, som allerede er tildelt et depot, afstanden til hver kunde, som endnu ikke er tildelt et depot. Vælg den mindste afstand og tildel den nye kunde det pågældende depot. Gentag processen iterativt indtil alle kunder er tildelt et depot.

Bemærk, at for $\eta > 0{,}5$ vil nogle cirkler overlappe hinanden og noget af styrken i algoritmen vil gå tabt, idet det mere og mere vil ligne den første clustermetode. Målet med første fase er, at forholdsvis isolerede depoter bør trække flere kunder med sig, end de depoter der ligger op ad hinanden. Det medfører, at såfremt kunderne er mere eller mindre ligeligt fordelt i problemet, vil sådanne depoter, der har andre depoter tæt på sig, i højere grad vil få tildelt deres kunder gennem anden fase.

Fordelen ved sidstnævnte clustermetode er, at vi i første fase beholder en geografisk sammenhæng blandt depoterne og de nærmeste kunder, mens vi i anden fase tager højde for at kunderne helst skal knyttes til sine nærmeste naboer. Dette giver en mere effektiv tildeling af kunderne.

7.6.2 Konstruér et antal OVRP-løsninger

Efter den indledende gruppering af kunderne i clusters, tildeles de hver en fiktiv efterspørgsel på 1 enhed. Hvert depot har nu et ubegrænset antal køretøjer til rådighed, som hver har en uendelig kapacitet, således at den fiktive efterspørgsel ikke kommer i spil[29].

Programmet leverer nu 2 løsninger, én baseret på metaheuristikkerne JDAM og ALNS, hvoraf sidstnævnte er præsenteret tidligere. Hvorledes den førstnævnte fungerer, vil vi ikke komme nærmere ind på her, men henviser til logvrp's hjemmeside[30]. Vi lader programmet køre 10 gange for hver af de to løsningsmetoder.

[29] Antal kunder tilknyttet et depot fungerer således både som en øvre grænse for antal køretøjer og kapaciteten, og er derfor reelt ækvivalent med en "ubegrænset" værdi.
[30] http://logvrp.com.

7.6.3 Forbind løsningerne til ét stort netværk

Vi er naturligvis interesseret i at forbinde OVRP-løsningerne via så få antal overgange som muligt, for at minimere den totale længde af rørsystemet, dvs. i alt $|D|-1$ forbindelser, hvor D er mængden af depoter. På denne måde har vandet stadig mulighed for at strømme rundt i hele netværket. Algoritmen kører som følger:

1) For hver OVRP-løsning, bestem afstanden mellem hver af noderne og den nærmeste node tilhørende en anden OVRP-løsning.
2) Vælg overgangen med den mindste afstand, der repræsenterer en forbindelse mellem to løsninger, endnu ikke forbundet, og forbind disse.
3) Gentag trin 1-2 indtil alle OVRP-løsninger er forbundet til en anden.

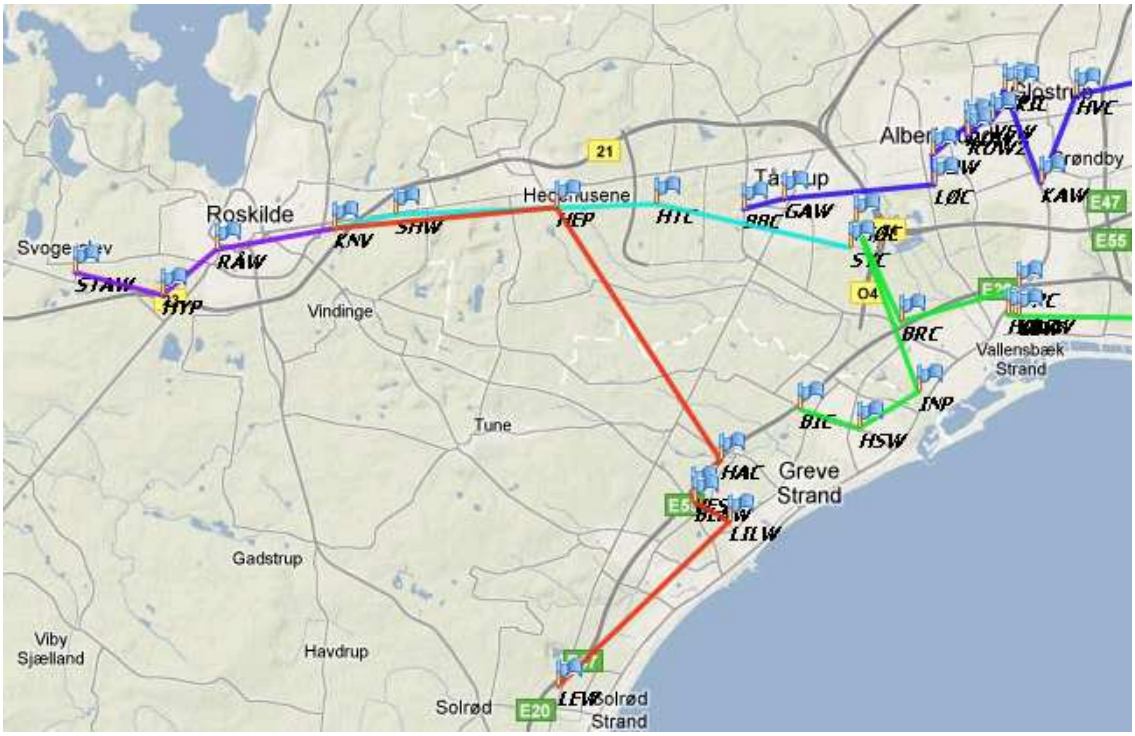
7.7 Resultater

Vi har kørt ovenstående 3-fase-model for 3 forskellige sæt af initiale clusters: Det første sæt antager strategien (1), det andet og tredje sæt strategien (2) for hhv. $\eta = 0{,}5$ og $\eta = 0{,}25$. I nedenstående tabel præsenteres den totale afstand for forskellige sæt af initiale clusters og løsningsmetoder i logvrp.

Total afstand (km)	Cluster 1	Cluster 2	Cluster 3
JDAM	230,7	205,2	205,9
ALNS	228,2	205,8	204,3

Tabel 9: Resultater

Vi kan se af ovenstående tabel, at den første clustermetode ikke er særlig hensigtsmæssig, idet der udelukkende tages højde for afstanden til et depot og ikke afstanden til de nærmeste kunder, kunder som man ville forvente burde ligge i forlængelse af hinanden. Som eksempel ses herunder et udsnit af løsningen, dannet på baggrund af cluster 1, JDAM:

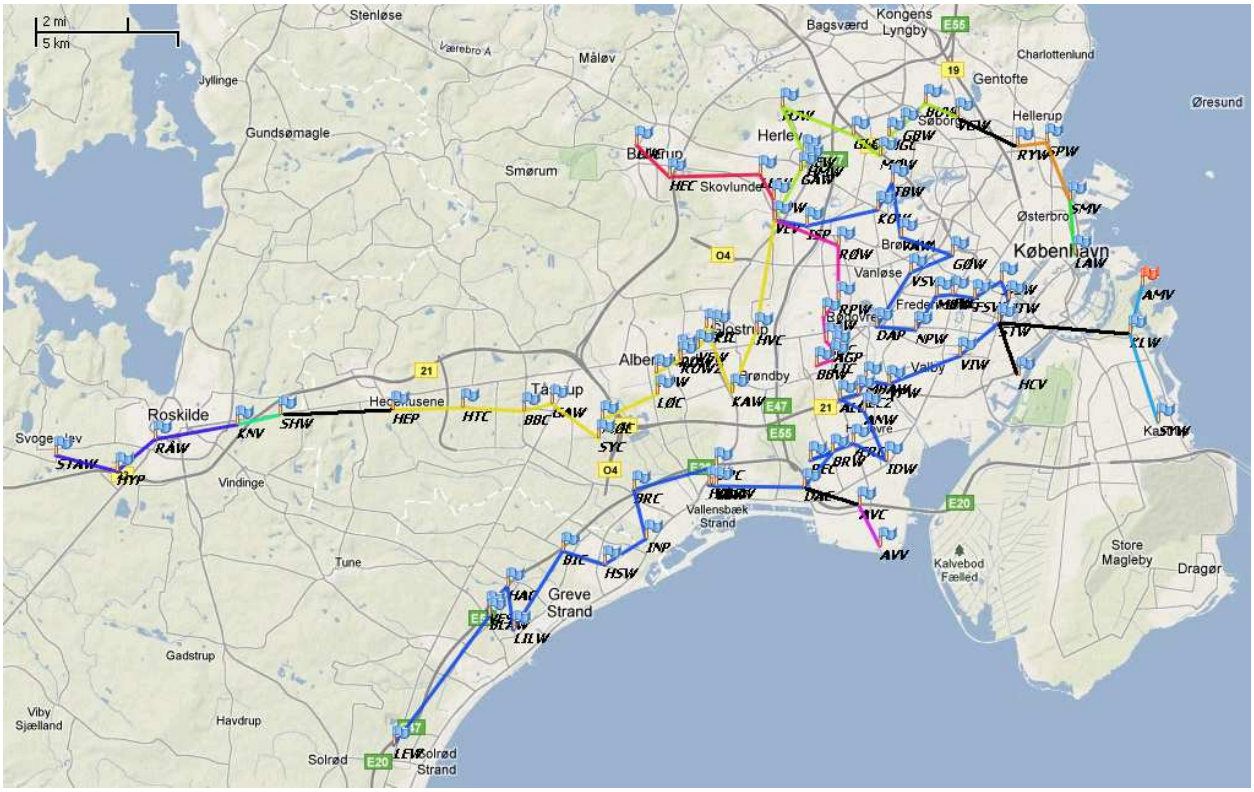


Figur 30: Udsnit af løsningen dannet på baggrund af cluster 1, JDAM.

Her ses et dårligt træk ved løsningen forårsaget af clustermetoden. Afstanden HEP → HAC er nødvendig for at kæde noderne nederst i figuren til forbrændingsanlægget i Roskilde, KNV, idet noderne til højre er tildelt andre varmeværk. Det er også tydeligt at denne lange kant er unødvendig i forhold til den kortere BIC → HAC. Det er disse kortere kanter der tilskyndes i clustermetode 2 og 3.

Så viser den anden clustermetode sig en del mere effektiv, både for $\eta = 0{,}5$ og $\eta = 0{,}25$. Her leder $\eta = 0{,}5$ til, at 42 ud af 84 kunder er tildelt et depot inden fase to sættes i gang, mens dette forhold kun er 10 ud af 84 for $\eta = 0{,}25$. I sidstnævnte tilfælde har det lave antal initialt fordelte kunder medført, at kunder, der ligger forholdsvist tæt på hinanden, er blevet tildelt samme depot. Det har til gengæld også medført, at nogle kunder er at finde ret langt fra sit depot.

Lad os betragte den bedste løsning, nemlig den der findes for clustermetode 3, ALNS. Af hensyn til overskueligheden har vi valgt at illustrere løsningen herunder med kanterne værende fugleflugtslinjer mellem noderne.

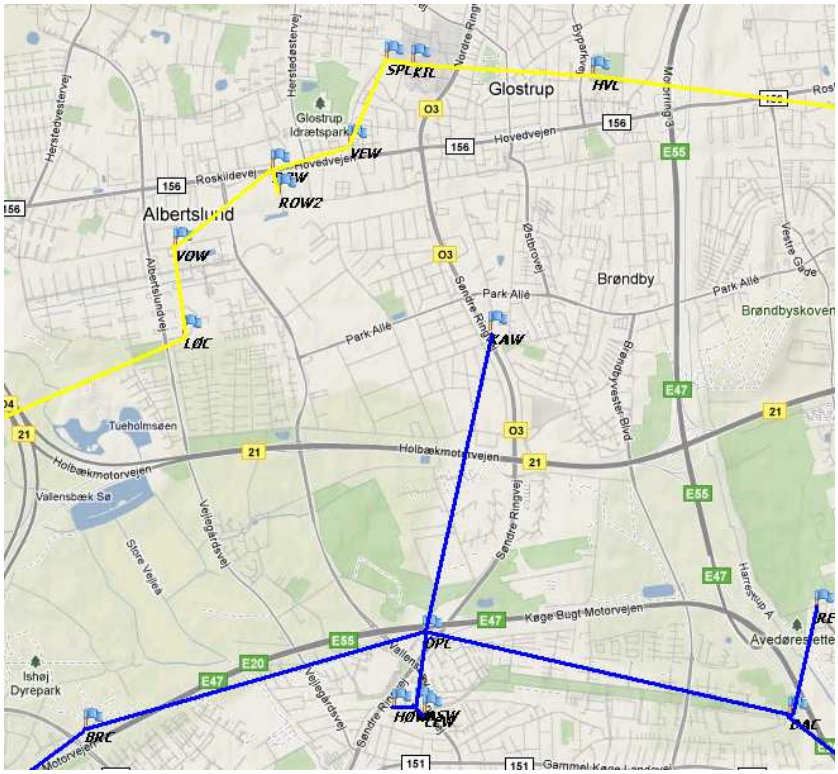


Figur 31: Den bedste løsning præsenteret ved hjælp af fugleflugtslinjer mellem noderne i netværket.

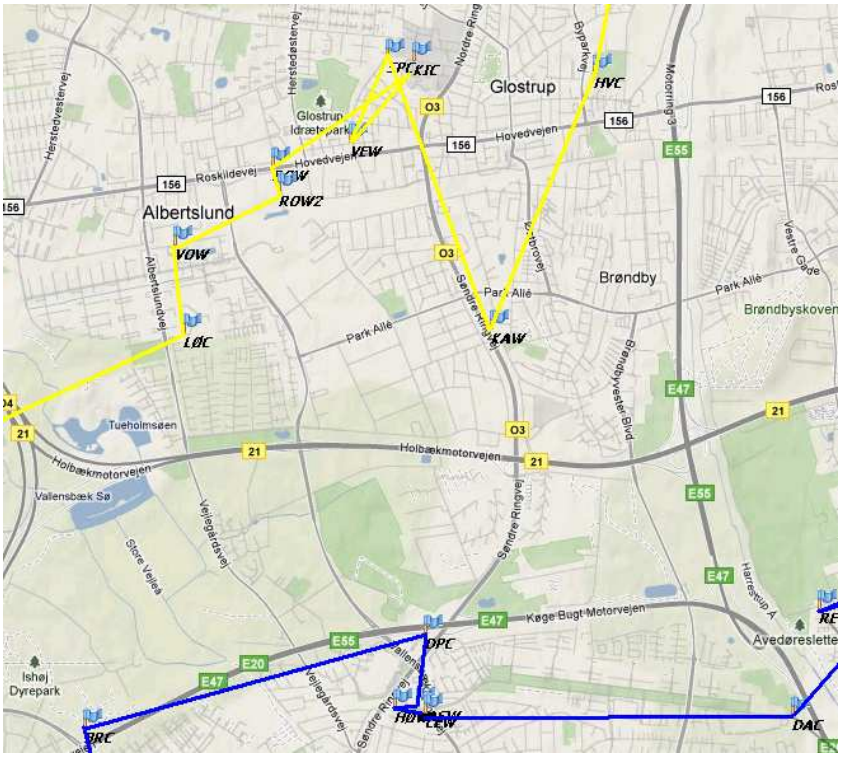
Løsningen ligner på mange måder det eksisterende netværk (se figur 29). Andelen af kanter i løsningen, som samtidig findes i det eksisterende transmissionsnetværk, udgør 66 %. Foruden de mindre detaljer, der udgør langt størstedelen af forskellene, er der dog nogle væsentlige, som er værd at bemærke.

En sådan forskel er, at kanten VEV → HVC i den nordvestlige del af løsningen ikke er en del af det eksisterende netværk. Kanten er da også væsentlig længere end alternativet, BBW → HVC, som udgør en forbedring til løsningen.

I det eksisterende netværk udgår en transmissionsledning fra DPC, der udelukkende servicerer KAW (figur 32). I løsningen vurderes det bedst at KAW indsættes mellem HVC og SPC (figur 33). Ifølge afstandsmatricen er dette dog ikke tilfældet, idet kanterne DPC → KAW (3,7) og HVC → SPC (2,5) tilsammen er kortere end HVC → KAW (3,3) og SPC → KAW (3,4). Grunden til dette ikke er præsenteret i løsningen er, at en sådan "afstikker" ikke er muligt i et OMDVRP, idet det medfører at en node (her: DPC) således har 3 kanter tilknyttet.



Figur 32: Illustration af det eksisterende netværk omkring vekslerstationen KAW.



Figur 33: Illustration af den bedste løsning omkring vekslerstationen KAW.

Af ovenstående illustrationer ses også, hvordan VEV, KIC og SPC er et slags knudepunkt i løsningen, idet kanterne imellem dem krydses flere steder. Men det hænger sammen med, at afstanden mellem punkterne er udregnet på baggrund af det overordnede vejnet. I ovenstående figur er kanterne repræsenteret som fugleflugtslinjer, og giver således ikke helt det korrekte billede af rørlægningen.

Andre forskelle skyldes, at vi har brugt vejnettet til at bestemme afstande mellem anlæggene, mens man i praksis har improviseret og fortaget en lang række valg af overgange, der ikke opfylder denne strenge regel. Et eksempel på dette er kanten KLW → STW ved østkysten, som i løsningen udgør forbindelsesleddet mellem Amagerværket (AMV) og resten af netværket. I det eksisterende netværk er rørene dog lagt under kanalen som en rørledning langs kanten AMV → LAW. På kortet i figur 31 virker denne kant da også kortere, men skal rørene følge vejnettet er dette dog ikke tilfældet, idet programmet automatisk vil vælge at krydse den nærmeste bro. Ved at basere afstandene på vejnettet, bliver vores løsning således automatisk en del længere.

Vi henviser til bilag 2 for specifik angivelse af kanterne i OVRP-løsningerne samt de valgte forbindelsesled i fase 3 af algoritmen.

7.8 Diskussion

Vi valgte fra starten at reducere problemet så vidt muligt ved at fjerne kunder, der ligger indenfor 300 m af et anlæg med højere prioritet (se afsnittet ”Fokusområde for casestudie”). Dermed har vi reduceret problemet fra at være af størrelsesordenen 124 noder til at omfatte 90 noder, hvoraf 6 er depoter og 84 er kunder, uden at vi har mistet særlig høj grad af nøjagtighed.

Blandt de udfordringer vi er stødt på i forbindelse med løsningen af problemet, er en af de væsentligste relateret til datamaterialet fra logvrp, hvis indbyggede GIS (”Geographic Information System”) ikke leverer afstande fra vejnettet i alle 8010 beregnede afstande. Således i 661 ud af 8010 tilfælde er der brugt en statistisk tilnærmelse der hedder 1,4 gange afstanden i fugleflugt. Det er klart, at denne tilnærmelse ikke er lige god for alle kanter, afhængig af om en forholdsvis lige strækning er tilgængelig, eller adskillige omveje er nødvendige. Dette knytter en væsentlig usikkerhed til evalueringen af den totale afstand, når vi på denne måde ikke kan regne med, at afstanden for alle noder er lige nøjagtig.

Et problem relateret til logvrp er, at nogle løsninger kan indeholde genopfyldning af køretøjer hos et andet depot, end køretøjerne tilhører. I værste fald sker det ved, at efter endt rute køres ad en forholdsvis lang vej til det nærmeste depot til genopfyldning, hvorefter køretøjet starter en ny rute fra dette depot. Det er umiddelbart ikke værre i vores tilfælde, end at vi kan se bort fra en sådan ”genopfyldningskant”, for når ruten fortsættes fra det nye depot, kan denne rute ses som en separat rute.

Det er svært at vurdere, hvorvidt resultatet er tilfredsstillende. Vi ved at VEKS’ og CTR’s rørsystemer strækker sig over i alt 158 km (104 km hhv. 54 km). Vestforbrændingen’s rørnet kendes ikke længden af, men den bedste løsning præsenterer den samlede afstand af denne til 18,7 km. Det giver i alt en længde på 176,7 km. Vores bedste løsning er 15,6 % større end denne værdi og er dermed ikke optimal.

7.9 Konklusion

Vi har i dette kapitel præsenteret et transmissionsnetværk for transport af fjernvarme og formået at formulere dette som en VRP-variant, der indeholder egenskaber fra både OVRP og MDVRP. Problemet er yderst kompliceret og en række abstraheringer har været nødvendige, ikke mindst for at passe problemet ind i en VRP-sammenhæng, men også i forbindelse med implementeringen af de løsningsmetoder der har været til rådighed. Ydermere har vi præsenteret en 3-fase algoritme til at løse dette såkaldte OMDVRP og konstrueret en løsning, der på mange måder ligner det eksisterende netværk, dog med en 16 % længere afstand. Set i lyset af de fejlkilder, der er både i datamaterialet og ikke mindst de abstraheringer, vi har været nødt til at foretage undervejs, har løsningen været tilfredsstillende.

Kapitel 8 – Konklusion

Denne afhandling fokuserer på løsningen af VRP og nogle af dets mest almindelige udvidelser vi støder på i praksis. I den forbindelser har vi præsenteret en række heuristiskker til løsning af CVRP og VRPTW problemerne. Men når vi bevæger os over i større problemer samt mere komplicerede udvidelser, er metaheuristiske løsningsmetoder, om end ligeledes mere komplicerede, også langt mere effektive, hvorfor vi har beskrevet metaheuristikker for hver af de valgte udvidelser; VRPTW, VRPPDTW, VRPBTW og OVRPTW. For VRPTW hhv. VRPPDTW findes et antal benchmarks, som viser at de præsenterede løsningsmetoder stadig den dag i dag er blandt de bedste på området, på trods af hhv. 6 og 7 år på bagen.

Herefter har vi præsenteret Københavns fjernvarmenetværk, som værende en variant af VRP med egenskaber fra både OVRP og MDVRP. Problemet er yderst kompliceret og en række abstraheringer har været nødvendige, ikke mindst for at passe problemet ind i en VRP-sammenhæng, men også i forbindelse med implementeringen af løsningsmetoderne til rådighed. Herefter har vi præsenteret en 3-fase algoritme og på den måde konstrueret en løsning, der på mange måder ligner det eksisterende netværk, dog med en 16 % længere afstand. Det skal bemærkes, at fordi primært vejnettet er brugt til at bestemme afstande mellem anlæggene, mens rørnettet mellem to anlæg i virkeligheden og i høj grad er præget af improviserede forbindelser, som ikke følger det overordnede vejnet. Vores metode har været rent mekanisk og derfor er nogle strækninger betydeligt længere i den præsenterede løsning. Set i dette lys er løsningen tilfredsstillende.

Referencer

Andersen (2003): "Vognrumsproblemet", kandidatafhandling, Datalogisk Institut, Københavns Universitet (DIKU).

Backchi et al. (2001): "Real life routing – en strategi for et vrp", Opgave, Roskilde universitetscenter (RUC), Institut for studiet af matematik og fysik samt deres funktioner i undervisning, forskning og anvendelse.

Baker, Ayechew (2003): "A genetic algorithm for the vehicle routing problem", Computers & Operations Research, Vol. 30, pp. 787–800.

Bent, Hentenryck (2004): "A two-stage hybrid local search for the vehicle routing problem with timewindows", Transportation Science, Vol. 38, No. 4, pp. 515–530.

Bertsimas (1991): "A Vehicle Routing Problem with Stochastic Demand", Massachusetts Institute of Technology, Cambridge, Massachusetts. Operations Research, Vol. 40, No. 3, pp. 574-585.

Bjarnadóttir (2004): "Solving the Vehicle Routing Problem with Genetic Algorithms", Kandidatafhandling, Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Bräysy (2001): "Genetic Algorithms for the Vehicle Routing Problem with Time Windows", Arpakannus, special issue on Bioinformatics and Genetic Algorithms.

Bräysy (2002): "Fast Local Searches for the Vehicle Routing Problem with Time Windows", INFORMS Vol. 40, No. 4, pp. 319-330.

Bräysy (2003): "A Reactive Variable Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows", INFORMS Journal on Computing, Vol. 15, No. 4, pp. 347-368.

Bräysy, Gendreau (2002): "Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows", Sociedad de Estadística e Investigación Operativa, Top, Vol. 10, No. 2, pp. 211-237.

Bräysy, Gendreau (2005): "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms", Transportation Science, Vol. 39, No. 1, pp. 104–118.

Bräysy, Gendreau (2005): "Vehicle Routing Problem with Time Windows, Part II: Metaheuristics", Transportation Science, Vol. 39, No. 1, pp. 119-139.

Breedam (1994): "An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints", Ph.d., University of Antwerp.

Christofides et al. (1979): "The vehicle routing problem", Combinatorial Optimization, Wiley, Chichester, pp. 315-338.

Coene et al. (2008): "The Periodic Vehicle Routing Problem: A Case Study", FETEW Research Report KBI_0828, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven.

Cordeau (2002): "A guide to vehicle routing heuristics", The Journal of the Operational Research Society, Vol. 53, No. 5, pp. 512-522.

Cordeau et al. (2004): "Transportation on Demand", CRT.

CTR (2004): "The Main District Heating Network in Copenhagen".

CTR (2009): "Klimavenlig varme i verdensklasse".

CTR: Årsrapport 2009.

Dantzig, Ramser (1959): "Truck dispatching problem", Management Science, Vol. 6, No. 1, pp. 80-91.

De Boer (2008): "Approximate Models and Solution Approaches for the Vehicle Routing Problem with Multiple Use of Vehicles and Time Windows", the graduate school of natural and applied science of middle east technical university.

De Jong (1975): "An analysis of the behavior of a class of genetic adaptive systems", Ph.d., University of Michigan.

Dorigo (1992): "Optimization, learning and natural algorithms", Ph.d., DEI, Politecnico di Milano.

Dorigo, Gambardella (1997): "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", IEEE Transactions on Evolutionary Computation, Vol.1, No.1.

Duhamel et al. (1997): "A Tabu Search Heuristic for the Vehicle Routing Problem with Backhauls and Time Windows", Transportation Science, Vol. 31, pp. 49–59.

Durbin et al. (1989): "An Analysis of the Elastic Net Approach to the Traveling Salesman Problem", Neural Computation, Vol. 1, pp. 348-358.

***Durbin*, *Willshaw* (*1987*):** "An analogue approach to the travelling salesman problem using an elastic net method", Nature, Vol. 326, pp. 689-691.

Eglese, Li (2005): "A New Tabu Search Heuristic for the Open Vehicle Routing Problem", The Journal of the Operational Research Society, Vol. 56, No. 3, pp. 267-274.

Fenghua, Xiaonian (2008): "On Open Vehicle Routing Problem with Soft Time Windows and Tabu Search", Logistics Research and Practice in China- Proceedings of 2008 International Conference on Logistics Engineering and Supply Chain.

Gambardella et al. (1999): "MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows", In New ideas in optimization, D. Corne, M. Dorigo and F. Glover, eds., McGraw-Hill, London, pp. 63–76.

Gelinas et al. (1995): "A new branching strategy for time constrained routing problems with application to backhauling", Annals of Operations Research, Vol. 61, pp. 91–109.

Gendreau et al. (2008): "Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography, The Vehicle Routing Problem: Latest Advances and New Challenges", Operations Research/Computer Science Interfaces Series, Vol. 43, s. 143-169.

Gendreau, Potvin (2005): "Chapter 6: Tabu search", Introductory Tutorials in Optimization and Decision Support Techniques, 1st ed., 2nd printing.

Ghaziri (1993): "Algorithmes connexionnistes pour l'optimisation combinatoire", Phd. École Polytechnique Fédérale de Lausanne.

Gillett, Miller (1974): "A heuristic algorithm for the vehicle-dispatching problem", Operations Research, Vol. 22, No. 2, pp. 340-349.

Glover (1986): "Future Paths for Integer Programming and Links to Artificial Intelligence", Computers and Operations Research, Vol. 13, issue 5, pp. 533–549.

Glover, Laguna (1997): "Tabu Search".

Glover, Marti (2006): "Chapter 3: Tabu search", Metaheuristic Procedures for Training Neural Networks, pp. 53-70.

Goldberg (1989): "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley Publishing Company Inc., New York.

Hasle, Kloster (2007): "Industrial Vehicle Routing", In Geometrical Modeling, Numerical Simulation, and Optimization: Industrial Mathematics at SINTEF", pp. 397-436.

Helsgaun (1998): "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic", datalogiske skrifter nr. 81.

Holland (1975): "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor, MI.

Keiding (2002): "Lærebog i operationsanalyse", Operationsanalyse, DJØFs forlag. Kan findes via link: http://www.econ.ku.dk/keiding/.

Kirkpatrick et al. (1983): "Optimization by Simulated Annealing", *Science,* Vol. 220, no. 4598, pp. 671-680.

Knudsen, Jørgensen (2004): "Tabusøgning til effektivisering af eksakt VRP algoritme baseret på søjlegenerering", Kandidatafhandling, Institut for Regnskab, Finansiering og Logistik, Handelshøjskolen i Århus.

Kohonen (1988): "Self-Organizing and Associative Memory", 3rd ed.

Laporte (1992): "The Vehicle Routing Problem: An overview of exact and approximate algorithms", European Journal of Operational Research, Vol. 59, pp. 345-358.

Larsen (2000): "The Dynamic Vehicle Routing Problem", Ph.d., Department of Mathematical Modeling, Technical University of Denmark (DTU).

Lawler (1972): "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem", Management Science, Vol. 18, pp. 401-405.

Lenstra, Rinnooy Kan (1975): "Some simple applications of the travelling salesman problem", Operational Research Quarterly, Vol. 26, pp. 717-734.

Lin, Kernighan (1973): "An effective heuristic algorithm for traveling-salesman problem", Operations Research, Vol. 21, No. 2, pp. 498-516.

Li, Lim (2001): "A Metaheuristic for the Pickup and Delivery Problem with Time Windows", The 13th IEEE Conference on Tools with Artificial Intelligence, ICTAI, Dallas, USA, pp. 160-170.

Miller, Goldberg (1995): "Genetic Algorithms, Tournament Selection, and the Effects of Noise", Complex Systems.

Min Wen (2010): "Rich Vehicle Routing Problems and Applications", Ph.d., DTU Management Engineering.

Mester, Bräysy (2005): "Active guided evolution strategies for large-scale vehicle routing problems with time windows", Computers & Operations Research Vol. 32, pp. 1593–1614.

Metropolis et al. (1953): "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, Vol. 21, No. 6, pp. 1087–1092.

Osman (1993): "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", Annals of Operations Research, Vol. 41, pp. 421-451.

Pisinger, Røpke (2009): "Large neighborhood search", In Jean-Yves Potvin and Michel Gendreau, editors, Handbook of Metaheuristics.

Prins (2004): "A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem", Computers & Operations Research, Vol. 31, Issue 12, pp. 1985-2002.

Reimann et al. (2002): "Insertion Based Ants for Vehicle Routing Problems with Backhauls and Time Windows", ANTS, LNCS 2463, pp. 135–148.

Repoussise et al. (2007): "The open vehicle routing problem with time windows", Journal of the Operational Research Society, Vol. 58, pp. 355–367.

Rich (1999): "A computational study of vehicle routing applications". Ph.d., Rice University.

Russell (1995): "Hybrid heuristics for the vehicle routing problem with time windows", Transportation Science, Vol. 29, pp. 156–166.

Røpke (2005): "Heuristic and Exact algorithms for vehicle routing problems", Ph.d., Datalogisk Institut, Københavns Universitet (DIKU).

Røpke, Pisinger (2006): "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows", Transportation Science, Vol. 40, pp. 455-472.

SINTEF: http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/

Shaw (1998): "Using constraint programming and local search methods to solve vehicle routing problems", In CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), Lecture Notes in Computer Science, Vol. 1520, pp. 417–431.

Solomon (1987): "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints", Operations Research, Vol. 35, No. 2, pp. 254-265.

Thangiah et al. (1996): "Heuristic approaches to Vehicle Routing with Backhauls and Time Windows", Computers and Operations Research, Vol. 23, pp. 1043–1057.

Thompson, Psaraftis (1993): "Cyclic transfer algorithms for multi-vehicle routing and scheduling problems", Operations Reasearch, Vol. 41, pp. 935-946.

Toth, Vigo (2002): "The Vehicle Routing Problem".

VEKS: Årsrapport 2009.

Vigo (1994): "A heuristic algorithm for the asymmetric capacitated vehicle routing problem", European Journal of Operational Research, Vol. 89, pp. 108-126.

Yuh-Jen Cho, Sheng-Der Wang (2005): "A Threshold accepting meta-heuristic for the vehicle routing problem with backhauls and time windows, Journal of the Eastern Asia Society for Transportation Studies, Vol. 6, pp. 3022 - 3037.

Bilag 1

Forkortelse	Navn	Adresse	Postnr.
AMV	Amagerværket	Kraftværksvej 50	2300 København S
AVV	Avedøreværket	Hammerholmen 50	2650 Hvidovre
HCV	H.C. Ørstedværket	Tømmergravsgade 4	2450 København SV
SMV	Svanemølleværket	Lautrupsgade 1	2100 København Ø
VEV	Vestforbrænding	Ejby Mosevej 219	2600 Glostrup
KNV	KARA/NOVEREN	Håndværkervej 70	4000 Roskilde
BUW		Hagavej 1	2860 Søborg
DAP		Roskildevej 211	2500 Valby
FSW		Falkoner allé 28	2000 Frederiksberg
FVC		Stæhr Johansens Vej 38	2000 Frederiksberg
GBW		Vandtårnsvej 59	2860 Søborg
GLC		Transformervej 9	2730 Herlev
GØW		Grøndalsvænge allé 13	2400 København NV
HGC		Gyngemosevej 3	2860 Søborg
JTW		Julius Thomsens plads	1925 Frederiksberg C
JÆW		Skyttegade 24	2200 København N
KLW		Raffinaderrivej 2	2300 København S
KOW		Korsager allé	2700 Brønshøj
LAW		Østbanegade 19	2100 København Ø
MBW		Malte Bruuns vej	2000 Frederiksberg
MØW		Gyngemose Parkvej 28	2860 Søborg
NPW		Nordens Plads 10, 2000 Frederiksberg, Danmark	
RYW		Niels Andersens vej 6	2900 Hellerup
SPW		Sankt Peders vej 7	2900 Hellerup
STW		Frederiksberg Alle 24, 1820 Frederiksberg, Danmark	
SYW		Vægtergangen 22	2770 Kastrup
TBW		Ved Bygården 5	2700 Brønshøj
VAW		Valløvej 18	2700 Brønshøj
VGW		Sønderengen 3	2860 Søborg
VIW		Vigerslev allé 10A	2500 Valby
VPW		Vigerslev allé 297	2500 Valby
VSW		Vanløse allé 51	2720 Vanløse
ISP		Islevbrovej 45	2610 Rødovre
RØW		Knudsbølvej 1	2610 Rødovre
RPW		Rødovre Parkvej 130	2610 Rødovre
BAW		Brandholms Alle 34	2610 Rødovre
AGC		Ruskær 57	2610 Rødovre
AGP		Agerkær 2	2610 Rødovre

LIC		Lindeager 10	2605 Brøndbyvester
BBW		Brøndbyøster Boulevard 29	2605 Brøndbyvester
ALC		Allingvej 100	2650 Hvidovre
ALC2		Allingvej 42	2650 Hvidovre
MBAW		M. Bechs Alle 6	2650 Hvidovre
ANW		Arn Nielsens Boulev 24A	2650 Hvidovre
ÆRC		Ærtebjergvej 107	2650 Hvidovre
IDW		Idrætsvej 84	2650 Hvidovre
AVC		Avedøreholmen 49	2650 Hvidovre
DAC		Daruplund 60	2660 Brøndby Strand
REC		Avedøre Tværvej 68-131	2605 Hvidovre
BRW		Brostykkevej 212	2650 Hvidovre
DPC		Delta Park 32	2665 Vallensbæk Strand
VASW		Vallensbæk Stationstorv 86	2665 Vallensbæk Strand
CEW		Centervej	2665 Vallensbæk Strand
HØW		Højstrupparken 59	2665 Vallensbæk Strand
KAW		Kirkebjerg Allé 92A	2605 Brøndbyvester
HVC		Hvissingevej 55	2600 Glostrup
KIC		Sportsvej 64B	2600 Glostrup
SPC		Sportsvej 62	2600 Glostrup
VEW		Vestergårdsvej 14	2600 Glostrup
ROW		Roskildevej 6	2620 Albertslund
ROW2		Roskildevej 25	2620 Albertslund
VOW		Vognporten 9	2620 Albertslund
LØC		Løkkekrogen 3	2625 Vallensbæk
MØC		Mølleholmen 5	2630 Taastrup
SYC		Sydskellet	2630 Taastrup
BRC		Brentevej 25	2635 Ishøj
BIC		Birkelyparken 150	2670 Greve
HSW		Hundige StorCenter 4	2670 Greve
INP		Industrivangen 34	2635 Ishøj
HAC		Hasselhegnet 100	2670 Greve
VESW		Vesthegnet 100	2670 Greve
BLÅW		Blåhegnet 46	2670 Greve
LILW		Lilleholm 2	2670 Greve
LEW		Lerbækvej 17	2680 Solrød Strand
BBC		Blekinge Boulevard 4	2630 Taastrup
GAW		Gasværksvej 5	2630 Taastrup
HTC		Potentilvej 11	2630 Taastrup
HEP		Charlottegårdsvej 4	2640 Hedehusene
SHW		Store Hedevej 1	4000 Roskilde
HYP		Hyrdehøj 100	4000 Roskilde

RÅW		Rådmandshaven 4	4000 Roskilde
STAW		Stamvejen 11	4000 Roskilde
LUW		Lundebjerg 40	2740 Skovlunde
TOW		Tonsbakken 10	2740 Skovlunde
GÅW		Gåseholmvej 69	2730 Herlev
HMW		Herlev Bygade 9	2730 Herlev
HEW		Herlevgårdsvej 13A	2730 Herlev
HJW		Dildhaven 40	2730 Herlev
HEC		Magleparken 3	2750 Ballerup
LUC		Telegrafvej 5A	2750 Ballerup

Bilag 2

OVRP-delløsninger for bedste løsning:

Ruter	Afstand (Km)	Ruter	Afstand (Km)	Ruter	Afstand (Km)
Rute 1		**Rute 7**		**Rute 8**	
1. AMV > KLW	2,2 Km	1. VEV > ISP	1,6 Km	1. VEV > HVC	5,6 Km
2. KLW > SYW	4,6 Km	2. ISP > KOW	3,5 Km	2. HVC > KAW	3,3 Km
Total	6,7 Km	3. KOW > TBW	1,6 Km	3. KAW > SPC	3,4 Km
		4. TBW > VAW	2,6 Km	4. SPC > VEW	0,8 Km
Rute 2		5. VAW > GØW	2,6 Km	5. VEW > KIC	1,8 Km
1. AVV > AVC	2,2 Km	6. GØW > VSW	2,2 Km	6. KIC > ROW	2,1 Km
Total	2,2 Km	7. VSW > DAP	3,1 Km	7. ROW > ROW2	0,3 Km
		8. DAP > NPW	1,5 Km	8. ROW2 > VOW	2,0 Km
Rute 3		9. NPW > MBW	2,8 Km	9. VOW > LØC	1,2 Km
1. SMV > SPW	2,6 Km	10. MBW > FVC	1,0 Km	10. LØC > MØC	3,0 Km
2. SPW > RYW	1,6 Km	11. FVC > FSW	1,1 Km	11. MØC > SYC	2,5 Km
Total	4,2 Km	12. FSW > JÆW	1,6 Km	12. SYC > GAW	2,8 Km
		13. JÆW > JTW	0,9 Km	13. GAW > BBC	1,4 Km
Rute 4		14. JTW > STW	1,5 Km	14. BBC > HTC	4,2 Km
1. SMV > LAW	2,7 Km	15. STW > VIW	2,7 Km	15. HTC > HEP	4,1 Km
Total	2,7 Km	16. VIW > VPW	2,9 Km	Total	38,5 Km
		17. VPW > MBAW	1,1 Km		
Rute 5		18. MBAW > ALC2	0,8 Km	**Rute 9**	
1. VEV > RØW	2,7 Km	19. ALC2 > ALC	0,9 Km	1. KNV > SHW	2,1 Km
2. RØW > RPW	2,5 Km	20. ALC > ANW	1,4 Km	Total	2,1 Km
3. RPW > BAW	1,3 Km	21. ANW > IDW	2,6 Km		
4. BAW > AGC	1,2 Km	22. IDW > ÆRC	1,8 Km	**Rute 10**	
5. AGC > AGP	0,5 Km	23. ÆRC > REC	1,7 Km	1. KNV > RÅW	3,5 Km
6. AGP > LIC	0,8 Km	24. REC > BRW	1,2 Km	2. RÅW > HYP	2,5 Km
7. LIC > BBW	0,9 Km	25. BRW > DAC	2,2 Km	3. HYP > STAW	3,1 Km
Total	9,9 Km	26. DAC > CEW	3,8 Km	Total	9,1 Km
		27. CEW > HØW	0,3 Km		
Rute 6		28. HØW > VASW	0,3 Km	**Rute 11**	
1. VEV > GÅW	2,7 Km	29. VASW > DPC	0,9 Km	1. VEV > TOW	1,0 Km
2. GÅW > HMW	0,5 Km	30. DPC > BRC	3,3 Km	2. TOW > LUW	1,4 Km
3. HMW > HEW	0,5 Km	31. BRC > INP	2,5 Km	3. LUW > HEC	4,5 Km
4. HEW > HJW	2,1 Km	32. INP > HSW	3,7 Km	4. HEC > LUC	2,2 Km
5. HJW > GLC	3,8 Km	33. HSW > BIC	2,3 Km	Total	9,1 Km
6. GLC > MØW	3,1 Km	34. BIC > LILW	4,1 Km		
7. MØW > HGC	2,9 Km	35. LILW > HAC	2,1 Km	**I alt**	184,4 km
8. HGC > GBW	0,9 Km	36. HAC > VESW	1,5 Km		
9. GBW > BUW	2,0 Km	37. VESW > BLÅW	0,3 Km		
10. BUW > VGW	1,3 Km	38. BLÅW > LEW	7,9 Km		
Total	19,8 Km	Total	80,1 Km		

Forbindelser fra fase 3 i algoritmen:

Kanter	Afstand (km)
AVC > DAC	2,9 Km
RYW > VGW	2,9 Km
KLW > STW	5,4 Km
SHW > HEP	4,8 Km
HCV > STW	3,9 Km
Total	19,9 Km