Ruteplanlægning i hjemmeplejen

Danmarks Tekniske Universitet, DTU
Informatik og Matematisk Modellering, IMM

Af Clement Ryberg Lessel, s001633

Vejleder:
Jesper Larsen

IMM-M.Sc-2007-32

2. april 2007

Abstract

The aim of this project is to explore a possible two-phase approach in solving routing problems in the homecare sector. The idea behind the two-phase approach is first to divide all visits in a number of clusters based on geographical location, preferences, home care worker skills, joint visits etc. Every cluster is associated with one home care worker.

In the second phase every cluster is handled as a separate TSPTW connected through their mutual joint visits. A joint visit is treated as two normal visits placed in two different clusters which makes the connection between the clusters.

This project deals with the second phase because it is possible to describe this phase only using hard constrains.

A route generator is developed which quickly generates a number of routes for each home care worker. Based on these routes a weighted clustered clique problem is created. This problem describes the connection between each TSPTW. The solution to the weighted clustered clique problem is a set of routes which makes it possible for two home care workers to be at their mutual joint visit at the same time. To solve the problem a greedy restart heuristic is developed which finds good solutions in a short amount of time.

The results show that it is possible to find solutions which are 0-10 % from the optimal solution, for problems containing up to approximately 455 visits, 55 joint visits and 50 home care workers. The test problems used in this project is generated based on inspiration from Søllerød municipality.

Based on the results in this project a two-phase approach seems promising and it could be interesting to continue the work taking both phases into account.

Keywords: Homecare, joint visits, heuristics, routing, weighted clustered clique problem and two-phase approach

Resumé

Formålet med dette projekt er at undersøge muligheden for en to-fasedeling af ruteplanlægningsproblemer i hjemmeplejen. Ideen bag to-fasedelingen er, at besøgene i første fase inddeles i en række grupper ud fra geografisk placering, præferencer, plejerens kompetencer, fællesbesøg, osv. Til hver gruppe er der tilknyttet én plejer.

I anden fase betragtes hver enkelt gruppe som et separat TSPTW. De seperate problemer er koblede igennem indbyrdes fællesbesøg. Koblingen består i, at et fællesbesøg betragtes som to enkelt besøg og placeres i to forskellige grupperinger.

Dette projekt tager udgangspunkt i anden fase, da denne fase kan beskrives udelukkende ved hårde begrænsninger.

Der er udviklet en rutegenerator som vha. af en indsættelsesheuristik hurtigt genererer en større mængde ruter for hver plejer. Udfra disse ruter opstilles et såkaldt vægtet grupperet klike problem (VGKP). Dette problem beskriver koblingen mellem de enkelte TSPTW. Løsningen af problemet resulterer i, at der opnås et sæt ruter, der opfylder, at to plejere kan være hos deres indbyrdes fællesbesøg samtidig. Der udviklet en grådig genstartsheuristik, som på kort kan finde gode løsninger til VGKP.

Resultaterne viser, at der findes løsninger, der ligger maksimalt 0-10 % fra den optimale løsning, for problemer med op til ca. 455 besøg, ca. 55 fællesbesøg og 50 plejere. Instanserne brugt i dette projekt er konstruerede instanser med inspiration fra hjemmeplejen i Søllerød kommune.

Ud fra resultaterne i dette projekt virker en to-fasedeling af ruteplanlægningsproblemer i hjemmeplejen lovende og det kan være interessant at arbejde videre med samspillet mellem de to faser.

Nøgleord: Hjemmeplejen, fællesbesøg, heuristikker, ruteplanlægning, vægtet grupperet klike problem og to-fase problemindeling.

Forord

Dette projekt afslutter min uddannelse på DTU som civilingeniør. Projektet er på 35 ETCS point og er udarbejdet fra den 4 september 2006 til den 2. april 2007 ved institut for Informatik og Matematisk Modellering (IMM). Projektet er udarbejdet under vejledning af Jesper Larsen, lektor på IMM.

Jeg vil gerne takke Jesper for al den tid, han har brugt på at vejlede mig under projektet. Gennem hele projektet har jeg sat meget stor pris på vejledningen, som har gjort projektet spændende og udfordrende fra start til slut.

Jeg vil gerne takke min kæreste, Cecilie Terese Holst, for hendes støtte gennem hele projektforløbet. Hun har været en stor hjælp både som en generel støtte i hverdagen og i forbindelse med korrekturlæsning på projektet. Jeg er meget taknemmelig.

Jeg vil gerne takke Olav Holst for korrekturlæsning på projektet, Professor Jens Clausen for hans støtte i opstartsfasen, og lektor Thomas Stidsen for hjælp til tekniske detajler.

Jeg vil til sidst gerne takke min familie for støtte under projektet og forståelse for manglende overskud i projektets afsluttende fase.

Symbolliste

Symbolerne brugt i rapporten er angivet nedenfor og er præsenteret i alfabetisk rækkefølge.

- $[a_v^{u_l}, b_v^{u_l}]$, besøgstidsvinduet for et besøg v. a_v er det tidligste starttidspunkt og b_v det seneste starttidspunkt på en rute u_l, således at resten af besøgene på ruten kan foretages indenfor deres tidsvinduer.
- $[A_v, B_v]$, tidsvinduet for et besøg v, hvor en given service s_v skal starte indenfor.
- d_{ij}, den euklidiske afstand mellem to besøg i, j.
- h, hastigheden, plejerne i problemet antages at rejse med.
- k_S, antallet af manglende kanter i løsningen S.
- l, er et indeks, der angiver en specifik gruppe af besøg. Da der er tilknyttet præcis én plejer til hver gruppe, skelnes der ikke mellem gruppe og plejer.
- L, antallet af grupper/plejere.
- m_l, en gruppering af genererede ruter m_l for en given plejer l.
- M, mængden af grupperinger m_l. Er i en begræsning desuden brugt som store M notation brugt i heltalsprogrammering.
- $\mathrm{N}(u_l)$, mængden af borgere i en gruppering l, der endnu ikke er tilføjet ruten u_l.
- O arealet, som alle besøg v befinder sig indenfor.
- $Q(S)$, nabolaget til en løsning S.
- q, antallet af genstarter i genstartsheuristikken.
- s_v, servicetiden for besøget v.
- S, en løsning på VGKP.
- t_{lj}, angiver starttidspunktet for en servicering af besøg j på rute u_l for plejeren l. Ikke at forveksle med ankomsttidspunktet T_{lj}.
- T_{lj}, ankomsttidspunktet til besøget j på rute u_l for plejeren l.
- u_l, en rute for plejer l. Ruten bestemmer rækkefølgen, besøgene skal foretages. i.

- U, samlet mængde af genererede ruter u.
- $|u_f|$, antallet af knuder/ruter i kandidatsættet $C(m, |u_f|)$ for grupperingen m i konstruktionsheuristikken KH2.
- v, et besøg.
- V mængden af besøg v.
- v_{fb}, betegner et besøg som fællesbesøg.
- V_l, aen gruppering af besøg tildelt en specifik plejer l. Denne gruppering skal ikke forveksles med en gruppering af genererede ruter m_l for en given plejer l.
- V_{fb}, mængden af fællesbesøg i problemet.
- x_u, er en binær variabel der en 1 hvis ruten u er med en VGKP-løsningen.
- y_{ij}, en binær variabel som er lig 1 hvis plejeren rejser fra besøg i til besøg j.
- β, sandsynlighed for at der bliver tilføjet et ekstra fællesbesøg mellem to grupperinger.
- ΔAB, størrelsen på tidsvinduerne angivet i timer.
- γ, størrelsen på grupperingerne af borgerne inden tildeling af fællesbesøg.
- ω_{u_l}, den euklidiske længde på ruten u_l.
- ρ, vægten, som en manglende kant i en løsning S tildeles i omkostningsfunktionen.
- Υ, antallet af borgere i problemet, hvilket adskiller sig fra antallet af besøg V, *efter* der indkluderes fællesbesøg.

Indhold

1 Introduktion

I den danske hjemmeplejesektor er der et ønske om, at ældre borgere skal have mulighed for at blive boende i deres eget hjem så længe som muligt, hvis de ønsker det.

For at imødekomme dette ønske stiller kommunen en service i form af hjemmepleje til rådighed. Hjemmepleje består af, at der kommer en plejer hjem til den ældre et antal gange om ugen efter behov. Under disse besøg hjælpes de ældre borgere med gøremål, som de ikke længere selv er i stand til at ordne. Mange ældre ser også disse besøg som et kærkomment socialt visit, da de kan være meget ensomme. Derfor er der som regel et ønske om, at det er den/de samme plejer, der besøger dem fra gang til gang.

I de kommende år vil den ældre generation blive større og større, da gennemsnits levealderen er steget. Denne stigning vil ydermere blive forstærket, efterhånden som den store 50'er generation går på pension. Der foregår også i disse år en større sammenlægning af kommuner til såkaldte storkommuner. Kompleksiteten af planlægningen indenfor hjemmeplejen vil derfor også stige markant.

Udover den enorme planlægningsopgave som hjemmeplejen står overfor, er der i øjeblikket også problemer med at finde folk til at besætte stillingerne bl.a. pga. hårde arbejdsforhold, lav prestige og lave løninger. Dette er en slags ond spiral, da en underbemandet hjemmeplejesektor slider ekstremt hårdt på de plejere, der allerede er ansat, og øger risikoen for diverse arbejdsskader med resulterende førtidspension. Samtidig lider de ældre under manglen på plejere, og dette er igen med til at sænke moralen hos plejerne.

Med en effektivisering af ruteplanlægningen i hjemmeplejen er håbet at lave en bedre planlægning af besøg, således at den enkelte plejer bliver mindre stresset og har mere tid og overskud til de enkelte besøg. Problemet er, at effektiviseringer som regel betyder, at i stedet for at anvende effektivitetsgevinsten til mere tid til det enkelte besøg, skal en plejer nå flere besøg på en dag, og servicen for de ældre forbedres derved ikke.

En effektivisering af ruteplanlægningen kan evt. også frigøre økonomiske ressourcer, der er bundet i administrationen i forbindelse med planlægning af besøgene for plejerne. Disse ressourcer kan istedet bruges til ansættelse af flere plejere, hvis nødvendigt.

2 Præsentation af problemet

Traditionelt er planlægning af ruter i hjemmeplejen blevet gjort manuelt af en såkaldt planner. Plannerens job er at tildele hvert besøg til en plejer og planlægge en fornuftigt rækkefølge af besøg, samtidig med besøgene bliver fortaget indenfor deres tidsvinduer. Dette er ret tidskrævende, da det er en kompleks problemstilling. Der opstår derfor hurtigt problemer, hvis en eller flere plejere ligger sig syg og planlægningen derved skal revideres. Hjemmeplejen er derfor efterhånden ved at gå til automatisk planlægning.

Ved behandling af ruteplanlægningsproblemer i hjemmeplejen er der ofte en række såkaldte bløde begrænsninger, der skal tages højde for. Disse begræsninger kan f.eks. være, at en borger får tildelt den samme plejer hver dag eller, at den tildelte plejer har været på diverse kurser for at kunne dække behovet for den pågældende borger. Dette kunne være et demenskursus, et kursus i hørehæmmede ældre eller lignende. Disse begrænsninger skal forsøges overholdt, men det er ikke et krav. Denne type begrænsninger kan være svære at tage højde for i en matematisk model.

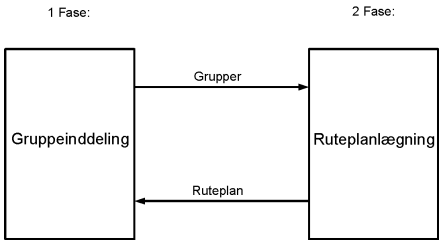
Problemet indeholder derudover behov, der skal opfyldes, givet ved en række hårde begrænsninger. Det kan f.eks være hjælp til at komme op om morgenen, rengøring, madlavning osv på bestemte tidspunkter. Til bestemte besøg kan der være behov for mere end en plejers tilstedeværelse. Dette kan f.eks. være, hvis patienter skal løftes ud af sengen eller lignede. Denne type begrænsninger er lettere at inkludere i en matematisk model, da de altid skal være opfyldt.

Tidligere har det været forsøgt at behandle problemet som et Vehicle Routing Problem with Time Windows (VRPTW), som kombinerede både gruppeinddeling og ruteplanlægning, se afsnit 2.2. Problemet med denne fremgangsmåde er, at det er utroligt svært at tage højde for de bløde begrænsninger. Et alternativ er at dele problemet op i to faser: En gruppeinddelingsfase og en rutegeneringsfase og derefter behandle hver del seperat.

I den første fase skal besøgene inddeles i grupper, således at hver besøg er tilknyttet en specifik plejer. Denne inddeling skal så vidt muligt laves med mht. de bløde begrænsninger og derudover tage højde for tidsvinduer og geografisk placering på besøg.

I anden fase af problemet skal der udelukkende arbejdes med ruteplanlægning indenfor de enkelte grupper. Fordelen er, at ude i den virkelige verden indeholder denne del kun hårde begrænsninger og er derfor lettere at arbejde med udfra et operationsanalytisk synspunkt.

To-fasedelingen af problemet er illustreret på figur 1.



Figur 1: *Figuren illustrerer en to deling af ruteplanlægningsproblemet i hjemmeplejen.*

Figur 1 viser de to faser. I første fase inddeles besøgene i grupper. Grupperne overføres til anden fase, hvor der laves en samlet ruteplan. Ruteplanen kan derefter sendes tilbage som en form for feedback, og en ny gruppeinddeling kan foretages hvis nødvendigt. En ny gruppeinddeling kunne være relevant, hvis der ikke kunne findes gyldige ruteplaner i fase 2, eller hvis forskellige gruppeinddelingsstragier ønskes afprøvet.

I dette projekt fokuseres der udelukkende på, fase 2 af problemstillingen; planlægningen af ruter, der kan imødekomme behovet for fællesbesøg. Det antages derfor, at inddelingen af grupper allerede er foretaget.

I det følgende afsnit defineres den præcise problemstilling, der arbejdes med i dette projekt.

2.1 Problemdefinition

Problemstillingen tager udgangspunkt i, at der eksisterer en mængde ældre borgere, der har behov for en service fra hjemmeplejen. En plejer, der yder en service hos en borger, betegnes et besøg v. Besøgene antages allerede at være inddelt i L grupper, betegnet l, efter forskellige præferencer. Disse præferencer kan f.eks være geografisk placering eller forskellige behov. Til hver gruppe l er der tilknyttet en plejer, der skal servicere alle besøg i grupperingen i løbet af en arbejdsdag. Der skelnes derfor ikke mellem grupper og plejer og begge dele betegnes derfor l. Til hvert besøg v hos en given borger er der tilknyttet en service tid s_v og et tidsvindue $[A_v, B_v]$, hvor den pågældende service skal være påbegyndt. Servicetiden kan variere efter, hvilken service der skal ydes; bad, rengøring, madlaving, etc. Tidsvinduet bestemmer kun, hvornår en service skal påbegyndes, og det er derfor tilladt at serviceringen

færdiggøres undenfor tidsvinduet, dvs. efter B_v.

Et besøg kan enten kræve en eller to plejeres tilstedeværelse. Et besøg der kræver to plejeres tilstedeværelse, kaldes et fællesbesøg og betegnes v_{fb}. Hvis der påkræves to plejere til et besøg, kan serviceringen ikke startes, før de begge to er til stede. Borgere med behov for et fællesbesøg placeres i to grupperinger og bliver derved knyttet til to plejere. Denne borger kommer derved til at være tilknyttet to besøg i problemet.

For hver gruppering l betegnes en rute u_l. Ruten u_l er en ordnet mængde af besøg v, ordnet efter den rækkefølge borgerne bliver besøgt i. En løsning på ruteplanlægningsproblemet er givet ved en rute for hver gruppering. Denne mængde af ruter betegnes et rutesæt. For en given rute kan et nyt tidsvindue beregnes for hver besøg. Dette betegnes *besøgstidsvindue* og er givet $[a_v^{u_l}, b_v^{u_l}]$. I besøgstidsvinduet beskriver $a_v^{u_l}$ det tidligste ankomsttidspunkt for plejeren til besøg v, og $b_v^{u_l}$ det seneste ankomsttidspunkt for plejeren til besøg v på ruten u_m der gør at ruten stadig er gyldig. Der tages i disse ankomsttidspunkter højde for rejse og serviceringstid for kunder der er placeret før besøg v på ruten. Besøgstidsvinduerne $[a_v^{u_l}, b_v^{u_l}]$ er for gyldige ruter indkluderet i tidsvinduet $[A_v, B_v]$ for et givent besøg. Modsat $[A_v, B_v]$ er besøgstidsvinduerne direkte tilknyttet til ruten, hvilket betyder, at en ændring i ruten vil betyde en ændring i besøgstidsvinduerne. Til hver rute er tilknyttet en længde ω_{u_l} givet ved den samlede euklidiske længde på ruten. Længden bruges til at vurdere kvaliteten af en given rute i forhold til andre ruter indenfor samme gruppering. Plejeren antages at rejse med en konstant hastighed h. Denne hastighed giver, sammen med den euklidiske afstand d mellem to besøg, tiden, det tager at komme fra besøg til besøg. Tiden for en plejers ankomst til et besøg hos en borger betegnes T_v. Alle plejere antages at starte deres ruter udfra det samme centralt beliggende udgangspunkt.

Problemstillingen kan nu defineres på følgende måde:

Givet en mængde borgere inddelt i grupperinger, søges der et rutesæt, således at alle tidsvinduer bliver overholdt, og alle behov for fællesbesøg bliver mødt. Antaget at disse behov er opfyldt, søges det rutesæt, der minimerer den samlede rutelængde.

Dette er den konkrete problemstilling, der behandles i dette projekt.

2.2 Litteratur om ruteplanlægning i hjemmeplejen

Ruteplanlægning i hjemmeplejen er et problem, der ikke er behandlet særlig meget i litteraturen. Det har derfor ikke været muligt at finde noget litteratur, der beskæftiger sig med 2 fase delingen af problemet, hvor der samtidig tages hensyn til fællesbesøg, hvilket behandles i dette projekt. Et udvalg af litteratur omhandlende ruteplanlægning i hjemmeplejen gennemgås i dette afsnit.

En del af inspirationen til dette projekt er kommet gennem [10], som er et andet eksamenprojekt skrevet på DTU i 2006 omhandlende ruteplanlægning i hjemmepejen. Projektet blev lavet i samarbejde med Zealand Care der laver ruteplanlægningsløsninger indenfor hjemmeplejen. Problemet betragtes som et Vehicle Routing Problem with Time Windows and Shared Visits (VRPTWSV)

Til løsning af problemet bruges tabusøgning, og løsningerne sammenlignes med løsninger, der er fundet med Zealand Care's eksisterende ruteplanlægningsværktøj ABP programme. I løbet af dette projekt er det blevet klart at de såkaldte fællesbesøg er problematiske. Fællesbesøgene er her behandlet som to enkelt besøg med ekstra betingelsen at to plejer skal være tilstede samtidig for at besøget kan serviceres. Problemet opstår, hvis et fællesbesøg i en given løsning fjernes fra en rute under lokalsøgningen. Det er nu svært at finde en ny rute at indsætte besøget på pga. fællesbesøgene ikke kun påvirker en rute, men to ruter, det kan derfor være svært at finde passende ruter at indsætte besøget på. Problematikken forbundet med fællesbesøg i[10] var en af inspirationskilderne til dette projekt.

Problematikken forbundet med ruteplanlægning i hjemmeplejen er behandlet i [4]. I problemet skelnes der mellem fuldtid og deltidsansatte. Forskellen mellem fuldtid og deltid er, at deltid er timeansat og fuldtid får penge for en hel dag, uanset om de arbejder hele dagen eller ej. Fuldtidsansatte skal også betales for evt. overarbejde, hvor deltidsansatte får deres normale timeløn. Frokostpauserne bliver i artiklen behandlet som et yderligere besøg i opgaven, der er tilknyttet til en bestemt plejer.

Artiklen behandler problemet som et VRPTW, hvor der er indført frokostpausebegrænsninger, og der laves et skel mellem fuldtid og deltid plejere. Til løsning af problemet bruges en to-fase heuristik. I første fase dannes der en initiel løsning vha. af en grådig algoritme. Denne løsning forbedres derefter med en lokalsøgning. Anden fase af løsningmetoden søger, at reducere ventetiden ved besøgene inden en service kan begynde for deltidsansatte plejere, og reducere overtiden for fuldtidsansatte. På en given rute findes det besøg der giver plejeren den største ventetid. Besøget forsøges indsat på en anden

rute således den samlede omkostning reduceres. Dette gentages, så længe der findes bedre løsninger.

Metoden er testet på en række små problemer med op til 4 plejere og 10 besøg, der er løst med Cplex. Derudover er der lavet test på på tre store problemer op til 900 besøg og 294 plejere genereret af et "fortune 500 company"som der skrives i artiklen. Løsningerne bliver ikke sammenlignet med eksisterende løsninger, og der er kun angivet om der blev fundet en gyldig løsning eller ej.

I [3] er problematikken forbundet med fællesbesøg medtaget. Artiklen omhandler et interaktivt software kaldet LAPS CARE udviklet til automatisk ruteplanlægning i den svenske hjemmepleje. Problemet bliver i denne artikel formuleret som et modificeret VRPTW. Modellen er modificeret så den indkluderer fællesbesøg, præferencer om en bestemt plejer, behov for bestemte kompetencer hos plejeren og en tilknytning mellem plejer og et bestemt område.

Der bruges i artiklen repeated matching som løsningmetode. Ideen er i hovedtræk at opfatte plejere og besøg som elementer i en matching. Hvert besøg og hver plejer har initielt tilknyttet en rute. Matching bruges til at kombinere besøg med plejer. Fællesbesøg bliver i modellen behandlet som to besøg og får tildelt et fast tidspunkt besøget skal finde sted. Præferencer om en bestemt plejer og behov for bestemte kompetencer hos plejeren bliver behandlet ved at tildele disse besøg til en bestemt plejer der opfylder kravene. Besøgene kan godt foretages af en anden plejer mod en straf i samlet rutelængde. Plejeren vil derfor, hvis muligt, blive fortrukket under matchingen. Besøg uden plejer tildeles en meget høj straf således at ruter besøg og plejer bliver kombineret uden matchingen. Matchingen gentages indtil løsningen ikke længere forbedres.

Metoden er testet på virkelige problemer fra den svenske hjemmepleje. Problemstørrelserne er 14-20 plejere og 86-123 besøg. For disse problemstørrelser sparer de omkring 7% arbejdstid og 20% arbejdstid.

To-deling af ruteplanlægning i hjemmeplejen er behandlet i [5]. Delingen består i, at besøgene første bliver tilknyttet en plejer og der findes herefter gode rutesekvenser og starttider. I artiklen bliver der taget højde for specielle behov hos de enkelte besøg, som f.eks. tid til snak, køn præferencer, samme tidspunkt hver gang osv. Bløde begrænsninger er medtaget, men kan ignoreres mod en straf. I modellen tages der ikke højde for fællesbesøg.

Tildelingen af besøg til plejerne bliver gjort udfra tidsvinduerne, således at ingen besøg overlapper og der er tid til at rejse fra besøg til besøg. Denne

tildeling gøres ved brug af heuristikker. Herefter findes gode starttider og rutesekvenser ved brug af en kombineret constraint programming og linear programming fremgangsmåde

Problemet er løst for op til 600 besøg og 50 plejere. Her sammelignes hvordan deres forskellige søgnings strategier klares sig. Der sammenlignes ikke med eksisterende løsninger eller en nedre grænse.

I [11] er der udviklet et beslutnings-støttesystem til ruteplanlægning i hjemmeplejen. Beslutnings-støttesystemet har en grafisk brugerflade baseret på Geographic Information-Systems (GIS) teknologi. Problemet behandlet i artiklen indkluderer ikke tidsvinduer og der er ikke behandlet evt. fællesbesøg.

Problemet er formuleret som et VRP, som skal løses fra dag til dag. Problemet løses med indsættelsesheuristik, hvor borgerne bliver indsat baseret på besparelsesliste mellem kundepar. De resulterende ruter bliver opdateret med en TSP nearest neighbour heuristik. Efter ruterne er konstrueret er det muligt for planneren at ændre i ruterne gennem en grafisk brugerflade.

I [12] er der behandlet et problem der minder om ruteplanlægning der indkluderer fællesbesøg. Artiklen omhandler manpower allocation with time windows and job-teaming constrains (MAPTWTC). Problemet består i at der skal udføres en række opgaver på forskellige geografiske lokationer i en havn i Singapore. Til disse opgaver er der brug for forskellige job teams der sammensat af arbejdere med forskellige kvalifikationer. Et job kan først starte når alle arbejdere er til stede og alle skal blive indtil jobbet er udført. Målet er at minimere den samlede rejselængde for alle arbejdere og minimere antallet af arbejdere, der skal bruges til at udføre jobbet.

Problemet løses ved brug heuristikker. Der præsenteres i artiklen to konstruktions heuristikker. Begge heuristikker tager en tilfældig gyldig rækkefølge af jobs og danner udfra denne rækkefølge en gyldig løsning. Ved at ændre en smule på rækkefølgen af jobs dannes der nye løsninger med konstruktionsheuristikken. Disse nye løsninger betegnes som naboløsninger til den forgående løsning. Der bruges derefter simuleret udglødning til at bevæge sig rundt mellem naboløsningerne.

Denne problemstilling minder en del om problemstillingen i dette projekt, men adskiller sig på en række væsentlige punkter. I dette projekt er det antaget at besøgene i problemet er inddelt i grupperinger til at starte med. Antallet af plejere er derfor defineret på forhånd. I MAPTWTC er arebjdsstyrkens størrelse ikke fastlagt på forhånd. I MAPTWTC er der udelukkende jobs der kræver to eller arbejdere i dette projekt er det overvejende besøg der kræver en enkelt plejer med et par fællesbesøg på hver rute. Artiklen er

dog interessant da den behandler problemstillingen forbundet med at finde naboløsninger, når der kræves mere end en arbejder/plejer til et job/besøg.

Artiklen [3] medtager fællesbesøg i deres model og artiklen [5] bruger en 2-fasedeling til løsning af ruteplanlægning i hjemmeplejen. Det har ikke været muligt at finde artikler der kombinerer 2-fasedelingen af problemet samtidig med at der tages højde for fællesbesøg, som der arbejdes med i dette projekt.

Problemet forbundet med at planlægge ruter i hjemmeplejen der indkluderer muligheden for fællesbesøg kan i princippet opstilles som et VRPTW. Ved at plejerne opfattes som køretøjer og besøgene som kunder kan problemet opstilles som VRPTW. Da artikler omhandlende ruteplanlægning i hjemmeplejen er begrænset, kunne det være interessant, hvis det var muligt at finde artikler om VRPTW, der medtog en form for fællesbesøg. Fællesbesøg kunne i denne henseende være to lastbiler der skulle mødes i løbet af en rute for at udveksle last.

Det har desværre ikke været muligt at finde artikler, der behandlede denne problematik, hvilket sammen med [10] indikerer at fællesbesøg kan være utrolig svære at tage højde for i VRPTW.

2.3 Hjemmeplejen i Søllerød kommune

I dette projekt tages der udgandspunkt i information omkring hjemmeplejen i Søllerød. Antallet af besøg i hjemmeplejen i Søllerød kommune er i omegnen af 200. Til at betjene disse besøg er der tilknyttet ca. 20 plejere. I forbindelse med disse besøg opereres der kun med tre store tidsvinduer: et morgentidsvindue fra kl. 08.00-11.00, et middagstidsvindue fra kl. 11.00-14.00 og eftermiddagstidsvindue fra kl. 14.00-17.00. Servicetiden for disse besøg er bestemt helt ned til minuts nøjagtighed og er meget varierende alt efter typen af besøg. I dette projekt vil der blive anvendt konstant servicetid for alle besøg.

Dette er det grundlæggende problem der tages udgangspunkt i, mht. tidsvinduer, antal besøg og antallet af plejere. Informationen er opnået gennem Jesper Larsens arbejde med optimering af hjemmeplejen i Søllerød kommune.

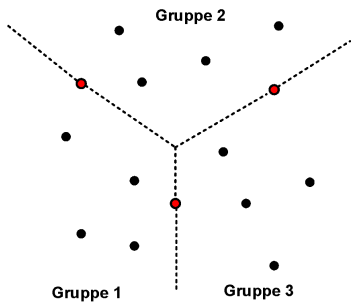
2.4 Løsningsmetode

Løsningsmetoden, der bruges til at behandle problemstillingen opstillet i afsnit 2.1, vil blive skitseret i dette afsnit. Metoden kan deles i følgende tre dele som vil blive behandlet separat i de kommende

1. Planlægning af ruter.
2. Vægtet Grupperet Klike Problem
3. Gyldighedsproblemet

afsnit. Den følgende skitsering skal danne et overblik over delproblemerne, således at det er lettere for læseren at placere de enkelte dele i den overordnede løsningsmetode.

Udgangspunktet er, som beskrevet tidligere, at der eksisterer en mængde borgere med behov for besøg fra hjemmeplejen. Disse besøg er inddelt i grupper efter præferencer, geografi osv. Planen med dette projekt er ikke at behandle denne inddeling og det antages derfor at inddelingen er foretaget. I figur 2 er præsenteret et lille eksempel.



Figur 2: *Figuren viser et eksempel på gruppering af en mængde besøg. Der er tre grupperinger med hver to fællesbesøg. Til hver gruppe er der tilknyttet en plejer. Fællesbesøgene for plejerne er lokaliseret på overgangen mellem grupperingerne markeret med den stiplede linie*

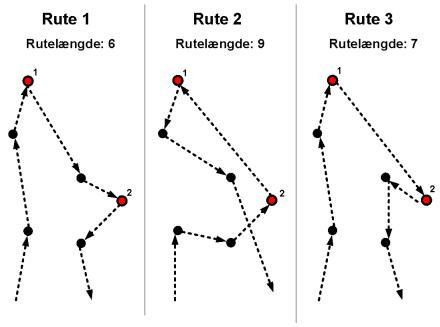
Figur 2 illustrerer, hvordan en mængde besøg kan være grupperet, og hvordan et fællesbesøg er tilføjet til to forskellige grupperinger. Fællesbesøgene er markeret med rødt og er placeret på grænsen mellem grupperingerne markeret med stiplede linier.

2.4.1 Ruteplanlægning

Det første delproblem i løsningsmetoden er bestemme en række ruter for hver gruppering.

Grundideen er, at der skal genereres tilpas mange ruter for hver gruppering, så det er muligt at finde et rutesæt, der opfylder alle behov for fællesbesøg. Det betyder, at der i første omgang skal genereres en større mængde ruter til hver enkelt gruppering. Til dette formål foretrækkes en heuristik fremfor en optimal metode. Dette er der to grunde til: For det første er, ruteplanlægningen kun et delproblem af det overordende problem, så ruterne skal findes hurtigt, hvilket er oplagt at gøre med en heuristik. For det andet skal der dannes mere end en rute for hver gruppering. En optimal metode vil kun finde én løsning.

På figur 3 er et eksempel på nogle mulige ruter for gruppe 1. Gruppen er taget fra figur 2.



Figur 3: *Figuren viser tre forskellige ruter indenfor gruppering 1. De røde knuder angiver fællesbesøg og er betegnet 1 og 2.*

Betragtes figur 3 ses det, at rute 1 og 3 er meget ens, og plejeren på ruten vil i begge tilfælde have næsten identiske besøgstidsvinduer. Dette kan f.eks. ses ved at rækkefølgen af besøg for rute 1 og 3, inden fællesbesøg 1 er helt identiske, hvilket betyder, at a_v vil være den samme for begge ruter.

For rute 2 vil besøgstidsvinduerne derimod være markant anderledes da rækkefølgen og mængden af besøg inden og efter fællesbesøg 1 er anderledes. Det betyder, at fællesbesøg 1 vil have et anderledes besøgtidsvindue end de to andre ruter. Rute 2 giver derfor mulighed for, at plejeren kan være ved fællesbesøgene på helt andre tidspunkter end rute 1 og 3. En sådan forskel på besøgstidsvinduerne er en fordel, når der på et tidspunkt skal findes et

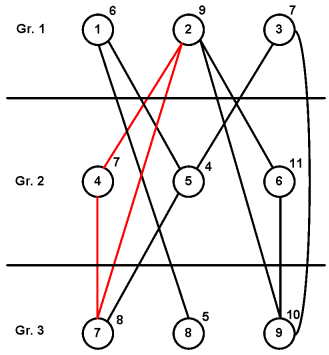
gyldigt rutesæt. Eksemplet er dog ikke så realistisk, da tidsvinduerne ikke gør det muligt at ændre så markant på rækkefølgen af besøg, men det skal illustererer betydningen af forskellige ruter.

Målet er at lave tilpas mange ruter for en given gruppering, således at (fælles)besøgstidsvinduerne $[a_{v_{fb}}, b_{v_{fb}}]$ for alle ruterne dækker så meget som muligt af det oprindelige tidsvindue $[A_{v_{fb}}, B_{v_{fb}}]$. Hvis der er stor diversitet i (fælles)besøgstidsvinduerne for alle grupperingerne, bliver det nemmere at finde et rutesæt, der kan opfylde behovet for fællesbesøg.

Målet med ruteplanlægningen er derfor ikke udelukkende at finde de bedste ruter, men lige så meget at finde forskellige type af ruter. Da det kan være svært at lave en heuristik, der finder forskellige typer af løsninger kan det evt. blive nødvendigt at lave tilpas mange ruter for derved at opnå den ønskede diversitet. Ruteplanlægningen er behandlet i afsnit 3.

2.4.2 Vægtet Grupperet Klike Problem

Efter ruterne er genereret, skal der vælges et rutesæt, således at behovet for fællesbesøg tilfredstilles og den samlede rutelængde minimeres. Problemet kan optilles som en graf ved, at hver rute angives som en knude, og kompatibilitet mellem ruter angives med kanter. Kompatiblitet skal forstås således, at to ruter tilhørende to forskellige grupper med et fællesbesøg, har overlappende besøgstidsvinduer for fællesbesøget. På figur 4 er opstillet et eksempel, hvordan en graf kan se ud for besøgene fra figur 2.



Figur 4: *Figuren viser en graf med tre forskellige ruter for hver af de tre grupperinger. Ruterne i gruppe 1 skal illustrere ruterne fra figur 3. Tallet inden i knuden angiver rutenummeret, og tallet udenfor ruten angiver længden af ruten. De røde kanter angiver det bedste gyldige rutesæt.*

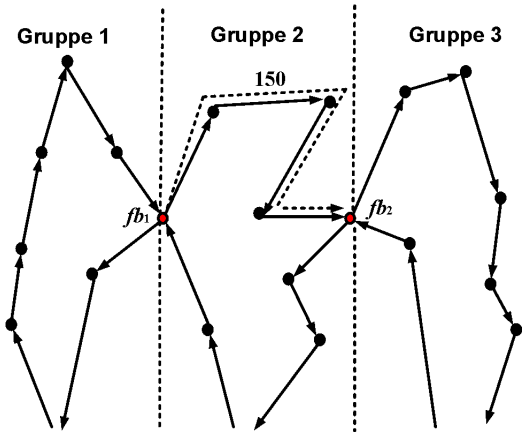
Figur 4 viser en graf med tre ruter for hver gruppering/plejer betegnet Gr.1-3. Knuderne for gruppe 1 skal repræsentere ruterne vist i figur 3. Resten af knuderne skal tilsvarende opfattes som ruter. Hvis der ikke er kant mellem to knuder/ruter, betyder det en af to ting: Enten er det ruter, der er tilknyttet samme plejer, eller også er det to ruter med fællesbesøg, hvor besøgstidsvinduerne $[a_{v_{fb}}, b_{v_{fb}}]$, for fællesbesøget ikke overlapper.

Et gyldigt rutesæt findes nu ved at finde en komplet delgraf med præcis en knude tilhørende hver plejer i problemet. Problemet med at finde en sådan delgraf vil i dette projekt blive betegnet vægtet grupperet klike problem.

I eksemplet er der to gyldige løsninger, nemlig knuderne nr. 2, 4, 7 og nr. 2, 6, 9. Den bedste løsning er angivet med røde kanter og har værdien 24. Metoder til at finde løsninger til dette delproblem er behandlet i afsnit 4.

2.4.3 Gyldighedsproblemet

Et gyldigt rutesæt, som f.eks. rutesættet præsenteret i figur 4, er desværre ikke nok til at sikre en gyldig løsning på det overordnede problem. Problemet er, at grafen kun angiver parvis kompatibilitet. I det efterfølgende er der angivet et eksempel, der illustrere hvorfor parvis kompatilitet mellem 3 grupper ikke nødvendigvis betyder en overordnet kompatibilitet.



Figur 5: *Figuren viser tre ruter for tre forskellige grupper. Fællesbesøg er markeret med rødt og betegnet hhv. fb_1 og fb_2. Tiden, det tager at komme fra fb_1 til fb_2 er sat til 150 minutter.*

Figur 5 viser tre grupper med to fællesbesøg betegnet fb_1 og fb_2. Tiden,

det tager for plejeren i gruppe 2 at komme fra fb_1 til fb_2, er 150 minutter.

Eksemplet på figuren betragtes nu for følgende værdier angivet i minutter:

- Tidsvinduerne på fællesbesøgene er givet ved $[A_{fb_1}, B_{fb_1}] = [0; 180]$ og $[A_{fb_2}, B_{fb_2}] = [180; 360]$.
- Besøgstidsvinduerne for fb_1 er givet ved $[105; 150]$ og $[45; 125]$ for hhv. ruten i gruppe 1 og ruten i gruppe 2.
- Besøgstidsvinduerne for fb_2 er givet ved $[195; 270]$ og $[180; 220]$ for hhv. ruten i gruppe 2 og ruten i gruppe 3.

Da fællesbesøg kun kan påbegyndes, når begge plejere er tilstede, betyder det, at fb_1 og fb_2 får samme besøgstidsvindue for begge ruter givet ved fællesmængden af besøgstidsvinduerne for de to ruter. Fællesmængden mellem besøgstidsvinduerne for fb_1 er $[105; 125]$ og $[195; 220]$ for fb_2. Da det tager plejeren i gruppe 2 150 minutter at komme fra fb_1 til fb_2, betyder det, at plejeren tidligst kan ankomme hos fb_2 til tiden 255 hvilket gør at det forenede besøgstidsvindue for fb_2 givet ved $[195; 220]$, ikke kan overholdes.

Dette er et eksempel på, hvordan ruterne angivet i figur 5 kan være parvis kompatible, men ikke er kompatible, når der tages højde for evt. ventetid ved fællesbesøg. For at tjekke den gyldighed opstilles et såkaldt gyldighedsproblem. Denne problemstilling er behandlet i afsnit 5.

Hvis det viser sig at det fundne rutesæt ikke er gyldigt, findes der et nyt rutesæt og der opstilles et nyt gyldighedsproblem. Dette gentages, indtil der findes en gyldig løsning på gyldighedsproblemet. Det kan være, at det igennem denne metode ikke er muligt at finde løsninger på problemet gennem denne metode. En undersøgelse af mulige tiltag hvis dette er tilfældet, ligger udenfor omfanget af dette projekt.

2.5 Rapportens opbygning

I dette afsnit gennemgås rapportens opbygning. Rapporten er opbygget som følger:

- I afsnit 3 bliver planlægningen af ruter i grupperingerne gennemgået.
- Afsnit 4 omhandler problemet forbundet med at finde et gyldigt rutesæt. Dette problem opstilles som et vægtet grupperet klike problem og løses vha. en genstartsheuristik.
- Gyldighedsproblemet forbundet med rutesættet behandles i afsnit 5. Problemet opstilles som et lineært programmeringsproblem og løses ved at kalde cplex fra hovedprogrammet.
- Det samlede hovedprogram præsenteres i afsnit 6.
- I afsnit 7 beskrives testinstanserne genereret i rapporten, og der laves parametertuning på programmet.
- Afsnit 8 indeholder resultaterne fundet med hovedprogrammet ved at løse forskellige testinstanser.
- I afsnit 9 diskuteres problemstillinger forbundet med projektet.
- Afsnit 10 foreslås områder der kan arbejdes videre med i forbindelse med dette projekt.
- Projektet afrundes i afsnit 11 med en konklusion.

3 Ruteplanlægning

Som beskrevet i afsnit 2 antages det, at alle besøg er inddelt i grupper til at starte med. Da alle fællesbesøg er delt op i enkeltbesøg og placeret i to forskellige grupperinger, betyder det, at ruteplanlægningen for den enkelte plejer kan behandles separat. Ruteplanlægningen for hver plejer kan opstilles som et Traveling Salesman Problem with Time Windows (TSPTW). I dette projekt antages problemet at være symmetrisk, altså at afstanden fra a til b er den samme som fra b til a. I virkeligheden er dette dog ikke altid tilfældet, f.eks. ved ensretning af enkelte veje.

I de følgende afsnit opstilles en matematisk model for TSPTW. Der præsenteres en konstruktionsheuristik for TSPTW, og der introduceres en metode til at generere flere ruter for hver plejer. En metode til generering af fiktive testinstanser bliver gennemgået. Heuristikkens løsninger af disse testinstanser sammenlignes herefter med en optimal løsning for at give et billede af kvalititeten af de heuristiske metoder.

3.1 Litteratur om Traveling Salesman Problem with Time Windows

Traveling Salesman Problem (TSP) er et de mest kendte og bedst gennemanalyserede optimeringsproblemer. Det meget beslægtede problem - Traveling Salesman Problem with Time windows - er derfor også behandlet en del i litteraturen. Dette er, både gennem tilknytningen til TSP, men også pga. af det optræder som et delproblem i andre optimeringsproblemer. En TSPTW heuristik kan f.eks bruges til reoptimere de enkelte ruter i VRPTW. Litteraturen på området er omfattende og der vil i dette afsnit kun blive præsenteret et udsnit af artikler om området.

I [9] præsenteres en guided lokalsøgings heuristik til løsning af traveling salesman problem. Der tages i artiklen ikke højde for tidsvinduer. Artiklen giver et godt indblik i forskellige heuristiske teknikker, tabusøgning, simuleret udglødning, etc., men pga. at der ikke taget højde for tidsvinduer kan metoderne præsenteret i artiklen ikke benyttes direkte til ruteplanlægningen.

Heuristikker til løsning af TSPTW er behandlet i [13], [14] og [15]. Artiklerne forsøger at løse problemet med hhv. en indsættelsesheuristik med efterfølgende lokalsøgning, simuleret udglødning og genetiske algoritmer.

Opbygning af gyldige ruter til Vehicle Routing Problem med Time Windows (VRPTW) er behandlet i [8]. Ruterne genereres med en indsættelsesheuristik. Heuristikken tilføjer et besøg af gangen til en initielt tom rute

et besøg af gangen. Det sikres samtidig, at indsætttelsen er gyldig og den bedst mulige. Heuristikken beregner ydermere besøgstidsvinduerne for hver rute. Metoden er hurtig og effektiv. TSPTW er et specialtilfælde af VRPTW hvor der kun er en lastbil med ubegrænset kapacitet. Indsættelsesheuristikken præsenteret i [8] kan derfor osse bruges til at lave TSPTW løsninger.

3.2 Matematisk Model

Lad $G(V, E)$ være en ikke-orienteret graf, hvor V er mængden af knuder v givet ved besøgene tilhørende en bestemt plejer. $E = V \times V$ er mængden af kanter e, der angiver en forbindelse mellem besøgene. Hver kant e har en vægt betegnet d_e svarende til afstanden mellem de to besøg, som kanten e forbinder. Hastigheden, som plejeren bevæger sig med, er betegnet h. Tidsvinduet, som det enkelte besøg skal serviceres indenfor, er betegnet $[A_v, B_v]$. Til hvert besøg er knyttet en serviceringstid betegnet s_v, der betegner tiden, plejeren opholder sig hos besøg v. Ankomsttiden hos besøg v er betegnet T_v. Beslutningsvariablen y_{ij} er givet ved følgende:

$$y_{ij} = \begin{cases} 1, & \text{hvis plejeren tager direkte fra besøg } i \text{ til besøg } j. \\ 0, & \text{ellers.} \end{cases}$$

Den matematiske model for TSPTW kan nu skrives som følger:

$$\begin{aligned}
\min \quad & \sum_i \sum_j d_{ij} y_{ij} && (1)\\
& \sum_i y_{ij} = 1 \qquad j \in V && (2)\\
& \sum_j y_{ij} = 1 \qquad i \in V && (3)\\
& \sum_{i \in R} \sum_{j \notin R} y_{ij} \geq 1 \quad \text{for} \quad R \subset V, R \neq \emptyset && (4)\\
& T_i \leq B_i \qquad i \in V && (5)\\
& T_i \geq A_i \qquad i \in V && (6)\\
& y_{ij}(T_i + s_i + h \cdot d_{ij}) \leq y_{ij} T_j \qquad i, j \in V, \quad i \neq j && (7)\\
& y_{ij} \in \{0, 1\} \qquad i, j \in V
\end{aligned}$$

Målfunktionen (1) minimere den samlede rutelængde der rejses. Begrænsningerne (2) og (3) skal sikre, at plejeren både ankommer til og forlader alle besøg på ruten. Begrænsning (4) sørger for, at ruten ikke deles op i flere delruter. Med delruter menes der, at ruten bliver delt op i to eller flere ruter, der tilsammen inkluderer alle besøg, men ikke er forbundet. Dette sikres ved, at der for en given delmængde $R \subset V$ altid skal være en kant, der forbinder denne delmængde med resten af V. Tidsvinduerne sikres overholdt

vha. begrænsningerne (5) og (6). Begrænsning (7) sikrer kobling mellem efterfølgende besøg på ruten.

Bemærk, at antallet af begrænsninger, der skal eliminere delruter givet i (4), stiger eksponentielt med antallet af besøg i problemet.

Begrænsningerne givet i (7) er for overskuelighedens skyld opstillet på ikke lineær form. Begrænsningen kan lineariseres ved brug af store M-metoden hvilket er vist nedefor:

$$T_i + s_i + h \cdot d_{ij} - (1 - y_{ij})M \leq T_j \qquad i, j \in V, \quad i \neq j \tag{8}$$

Hvis problemet skal løses med søjle generering, lagrange relaxation eller lignende, vil det være nødvendigt at foretage en linearisering, således at løsningen kan baseres på LP-løser. Den matematiske model skal dog ikke bruges direkte i løsningsmetoden. Modellen er primært opstillet til at præsentere problemet og til definering af notationen, og begræsningen kan derfor sagtens opskrives på ulineær form.

Pga. den begrænsede mængde af besøg en given plejer kan nå i løbet af en vagt, vil gruppestørrelserne næsten aldrig overstige 15 besøg. For et TSPTW med 15 besøg vil det ikke være noget problem i sig selv at generere optimale løsninger. Optimale løsninger indenfor de enkelte grupperinger er isoleret set ikke det vigtigste i dette projekt. De kan bruges som sammenligningsgrundlag og må meget gerne indgå i den mængden af ruter der genereres, men målet er primært, at der skal genereres mange forskellige ruter for hver TSPTW.

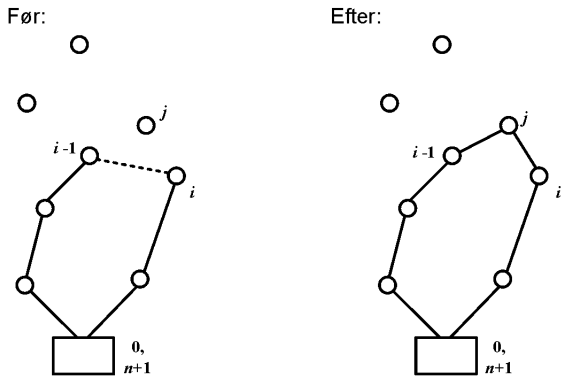
Ruteplanlægningen skal gå forholdsvis hurtigt da det kun optræder som et delproblem. Da der ydermere skal genereres mange ruter, er det oplagt at bruge en heuristik til dette formål. Heuristikken til rutegenerering er præsenteret i de følgende afsnit.

3.3 Heuristik til generering af ruter

Til generering af ruter tages der udgangspunkt i en konstruktionsheuristik lavet til VRPTW i [8] beskrevet i afsnit 3.1.

Heuristikken er baseret på en grundlæggende indsættelsesheuristik. Ideen bag en indsættelseheuristik, er at der startes med en tom rute. Ruten opbygges derefter ved at tilføje et besøg ad gangen. Et besøg tilføjes kun, hvis det ikke gør ruten ugyldig, og indsættes det sted på ruten, der forlænger ruten mindst.

Til beskrivelse af indsættelsesheuristikken indføres følgende notation. Besøget, der ønskes indsat på den pågældende rute u, betegnes j. Rutesekvensen for plejeren betegnes som $(0, 1, 2, \ldots, i-1, i, \ldots, n+1)$, hvor 0 og $n+1$ betegner det sted, der er udgangspunkt for alle plejere. Alle plejere skal både starte og slutte her. Indekset $i-1$ og i indekset betegner de to besøg, hvor besøg j forsøges indsat imellem. Indsættelsen forsøges mellem alle besøg (inkl. udgangspunktet), der allerede er tilføjet på ruten. På figur 6 er det illustreret et eksempel på brug af disse indeks under en indsættelse af et besøg på en rute.



Figur 6: *Tv. er ruten illustreret inden indsættelse af besøg j. Th. er ruten illustreret efter indsættelse af besøg j. Plejerens udgangspunkt er firkanten betegnet $0, n+1$. Den stiplede linie indikerer, hvor der overvejes at indsætte besøg j*

Figur 6 viser et billede af ruten før og efter indsættelse af et besøg på ruten. Det ses, at hvis besøget indsættes lige netop som på figuren, er det den indsættelse der forlænger ruten mindst.

Indsættelsen af besøg j forlænger ruten med afstanden Δd, givet ved følgende:

$$\Delta d = d_{i-1,j} + d_{j,i} - d_{i-1,i} \tag{9}$$

I hver iteration bestemmes Δd for alle besøg endnu ikke tilføjet ruten og for alle placeringer på ruten. Heuristikken benytter derfor en grådig fremgangsmåde.

Under indsættelsesproceduren evalueres hver enkelt besøg endnu ikke tilføjet til en rute. Hvert enkelt af disse besøg forsøges indsat på alle pladser på ruten. Det besøg med gyldig indsættelse og med mindst Δd bliver tilføjet på den fundne plads. Dette gøres i hver iteration og indsættelsesproceduren

fortsætter, indtil alle besøg er inkluderet i ruten.

Indsættelsesproceduren anvendes for alle grupper i problemet, og derved opnås en gyldig rute for hver plejer. På denne måde bliver alle besøg i problemet udført.

Heuristikken kan for en given rute u tilknyttet en given plejer beskrives ved følgende pseudokode, der gælder for hver plejer l:

Indsættelsesheuristik(u_l):

N(u_l): Mængden af besøg v i V_l der endnu ikke er tilføjet ruten.
V_l: Mængden af besøg v i gruppen l.
u_l: Ruten, som besøgene bliver tilføjet.

```
1:    N(u_l) = V_l
2:
3:    while ( N(u_l) ≠ ∅ )
4:        p* = - ∞
5:        for ( j ∈ N(u_l) )
6:            for ( (i − 1, i) ∈ u_l )
7:                if ( Feasible(i, j) and Profit(i, j)> p* )
8:                    i* = i
9:                    j* = j
10:                   p* = Profit(i, j)
11:               end if
12:           end for
13:       end for
14:       Insert(i*, j*)
15:       N = N \ j*
16:   end while
```

Pseudokoden beskriver, hvordan en enkelt rute genereres. Ruterne for de enkelte plejere kan genereres uafhængigt af hinanden, da der i første omgang ikke tages højde for fællesbesøg. Ovenstående procedure kan derfor køres for hver plejer, og der opnås derved et sæt ruter, der servicerer alle besøg.

Pseudokoden benytter tre funktioner: Profit, Feasible og Insert. Profit funktionen udregner $-\Delta d$. Feasible funktion vurderer om en indsættelse er gyldig eller ej, og Insert funktionen sørger for besøgstidvinduerne bliver opdateret efter en indsætteklse. I de næste to afsnit beskrives, hvordan hhv. Feasible og Insert funktionerne fungerer.

3.4 Feasible funktionen

Denne funktion skal vurdere, om indsættelse af besøg j imellem to på hinanden følgende besøg $i-1$ og i er gyldig eller ej. Til denne vurdering beregnes to tidspunkter: Første tidspunkt angiver, hvornår plejeren tidligst kan servicere besøg j, samtidig med at tidsvinduerne for besøg $0, 1, 2, \ldots, i-1$ overholdes. Grunden til, at der tages højde for besøg, der ligger før besøg j på ruten, er, at udelukkende de bestemmer det tidligste tidspunkt plejeren, kan ankomme til besøg j på. Det andet tidspunkt angiver hvornår plejeren senest kan servicere besøg j, samtidig med at tidsvinduerne for besøg $i, \ldots, n+1$ overholdes. Tilsvarende tages der kun højde for besøg, der ligger efter besøg j på ruten.

Det tidligste tidspunkt a_j plejeren kan servicere besøg j, er givet ved:

$$a_j = \max(A_j, a_{i-1} + h \cdot d_{i-1,j} + s_{i-1}) \tag{10}$$

Tidsvinduets start for besøg j er givet ved A_j . Det tidligste tidspunkt plejeren kan ankomme til besøg j er givet ved $a_{i-1} + h \cdot d_{i-1,j} + s_{i-1}$. Servicetiden er s_{i-1} for det forrige besøg, $h \cdot d_{i-1,j}$ er tiden, det tager komme fra det forrige besøg på ruten, og a_{i-1} er det tidligste starttidspunkt fra det forrige besøg på ruten. Der vurderes, om dette tidspunkt er senere end starten af tidsvinduet A_j. Hvis dette er tilfældet, angiver $a_{i-1} + h \cdot d_{i-1,j} + s_{i-1}$ det nye tidligste starttidspunkt ellers beholdes tidsvinduets start A_j.

Det seneste tidspunkt b_j plejeren kan servicere besøg j, er givet ved:

$$b_j = \min(B_j, b_i - h \cdot d_{j,i} - s_j) \tag{11}$$

Tilsvarende overvejelser gør sig her gældende. Tidsvinduets afslutning for besøg j er angivet ved B_j. Det seneste tidspunkt, plejeren kan ankomme til besøg j, er givet ved $b_i - h \cdot d_{j,i} - s_j$. Her er s_j servicetiden, $h \cdot d_{j,i}$ er tiden, det tager komme til det næste besøg på ruten, og b_i er det seneste tidspunkt, plejeren kan ankomme hos det næste besøg på ruten. Det vurderes, om dette tidspunkt er senere end afslutningen af tidsvinduet B_j. Hvis dette er tilfældet, er det seneste starttidspunkt givet ved B_j, ellers er det givet ved $b_i - h \cdot d_{j,i} - s_j$.

Til sidst kan gyldigheden af en indsættelse bestemmes ved, om det tidligst mulige ankomst tidspunkt a_j er mindre end det seneste mulige afgangstidspunkt b_j. Hvis uligheden $a_j < b_j$ ikke er opfyldt, kan plejeren simpelthen ikke overholde tidsvinduerne ved indsættelse af besøg j mellem besøg $i-1$ og besøg i, og indsættelsen er ikke gyldig.

Det er værd at bemærke, at tiden, det tager at tjekke gyldigheden af en mulig indsættelse, er konstant.

3.5 Insert funktionen

Når evalueringen af gyldigheden er overstået, indsættes den valgte besøg j på den fundne plads. Denne indsættelse forårsager et skift i a og b for de andre besøg på ruten og de skal derfor opdateres.

Opdateringen for a behøves kun foretaget for alle besøg efter det tilføjede besøg j, startende med besøg i på den pågældende rute. Grunden til, at de påvirkes, er, at der er indsat et besøg før dem, og det tidligste tidspunkt, plejeren kan være hos dem, kan nu have ændret sig. Opdateringen kan foretages på følgende måde:

$$a_k = \max(a_k, a_{k-1} + h \cdot d_{k-1,k} + s_{k-1}) \quad \text{for} \quad k = i, \ldots, n+1 \tag{12}$$

Opdateringen af tidspunkterne givet ved a skal foretages rekursivt. For $k = i$ bestemmes a_i ved brug af a_j givet i (10). På samme måde bruges a_i til at bestemme a_{i+1}. Indeks i, j er stadig defineret som i afsnit 3.4. Opdateringen forsættes indtil $k = n + 1$.

Opdateringen for b behøves derimod kun foretaget for alle besøg før besøg j på den pågældende rute. Besøgstidsvinduet b er et udtryk for den senest mulige start på servicering, og derfor må alle b tilknyttet til besøg efter besøg j på ruten være upåvirket. Denne opdatering kan gøres på følgende måde:

$$b_k = \min(b_k, b_{k+1} - h \cdot d_{k,k+1} - s_{k+1}) \quad \text{for} \quad k = i-1, \ldots, 0 \tag{13}$$

Den første opdatering for $k = i - 1$ er baseret på b_j givet i (11), hvor $i - 1$ er det forrige besøg til j. Herefter foretages opdateringen rekursivt fra b_{i-1} til b_0.

Ved brug af ligning (12) og (13) opnås en fuld opdatering af besøgstidsvinduerne efter en indsættelse af besøg j.

De besøg, der endnu ikke er tilknyttet ruten, evalueres igen. Dette fortsættes, indtil alle besøg tilhørende den pågældende plejer er tilknyttet ruten.

Når alle besøg v i gruppering l er tilknyttet ruten u, er det disse a,b værdier, der angiver besøgsvinduet $[a_v^{ul}, b_v^{ul}]$.

3.6 Rutegenerering

Som beskrevet i afsnit 2.4, er planen at generere en række forskellige ruter for den enkelte plejer. Med forskellige ruter menes der ruter, der har forskellige besøgstidsvinduer for de enkelte besøg. Grunden til at der ønskes

forskellige ruter er, at muligheden for at finde en samlet lovlig løsning, bliver større med mange forskellige ruter for hver plejer. Disse ruter skal udover at være forskellige, stadig være korte, eftersom målet er at lave en samlet lovlig løsning med så lille samlet rutelængde som muligt.

Der tages udgangspunkt i konstruktionsheuristikken beskrevet i afsnit 3.3, en plejer betragtes af gangen. Ideen er, at efter den første rute er genereret findes den billigste kant på ruten. Vægten på denne kant sættes til et stort tal, $\min_{e \in u_l}\{d_e\} = K$. K skal opfylde følgende:

$$K > d_e \quad \forall\ e$$

Konstruktionsheuristikken køres derefter igen. Da den billigste kant nu er blevet til den dyreste kant, vil denne kant ikke blive valgt til den nye rute, og der dannes derved en ny, anderledes rute. Samme betragtning kan laves for den nye rute, hvor en ny kant bliver sat til K. Konstruktionsheuristikken køres derefter igen, og en ny rute forskellig fra de to andre er genereret. Denne procedure kan gentages, til det ønskede antal ruter for en given plejer er opnået. Nedenfor er dette beskrevet i pseudokode:

Rutegenerator(K, RN):

```
1:   RN: Antallet af ruter der skal genereres.
2:   U: Mængden af alle ruter der genereres.
3:   L: Mængden af alle grupperinger l.
4:   u_l: Ruten, for grupperingen l som besøg bliver tilføjet.
5:
6:     for (l ∈ L)
7:         count = 0
8:         while (count ≤ RN )
9:             v = Indsættelsesheuristik(u_l)
10:              Sæt Værdi På Billigste Kant I ForrigeLøsning til K
11:              U = U ∪ u_l
12:              count++
13:        end while
14:    end for
```

Den ovenstående metode bruges til at generere ruter i hovedprogrammet. Det gode ved rutegeneratoren er, at der er god chance for, at de fundne løsninger er meget forskellige, da der ved hver ny løsning bliver fjernet minimum en kant i forhold til løsningen fundet med konstruktionsheuristikken. Dette skaber diversitet mellem ruterne, hvilket er vigtigt, når der skal findes kompatible rutesæt senere hen.

Tildelingen af vægten K til den billigste kant kunne alternativt sættes til

kun at vare et par iterationer, hvor efter kanten genvandt sin oprindelige værdi. Dette ville evt. finde bedre løsninger, da det igen bliver attraktivt at inkludere den bedste kant i løsningen. Til gengæld er der en stor risiko for, at heuristikken kører i ring og producerer de samme løsninger. Dette kan i givet fald omgås med en tabuliste eller lignende.

Et andet alternativ kunne være at fastsætte kanten med den største vægt i den foregående løsning til K, $\max_{e \in ul}\{d_e\} = K$. Her skulle den dyreste kant være en kant, der ikke allerede var sat til værdien K.

Problemet med dette alternativ er, at løsningerne kan risikere ikke at ændre sig særligt meget fra iteration til iteration. Det kan tænkes, at kun et par kanter vil blive skiftet ud, mens resten af løsningen bliver bibeholdt. På den måde vil alle løsninger ligne hinanden meget, og den ønskede diversitet vil ikke blive opnået.

Heuristikker til TSPTW er et område, der ofte er analyseret, både isoleret set og i forbindelse med VRPTW eller lignende problemer. Der findes en del forskellige måder at generere ruter på, men problemet er, at det ikke er til at vide, hvilken der er den bedste, før den bliver testet i den overordnede problemstilling. Det er selvfølgelig muligt at teste på værdien af ruterne i forhold til den optimale rute, men gode ruter er kun halvdelen af problemet, da gode ruter, der ikke kan kombineres til et gyldigt rutesæt, er ubrugelige.

Dette er grundlaget for at vælge den ovenstående rutegenereringsheuristik. I det efterfølgende afsnit vil heuristikken blive testet mht. diversiteten af ruterne og mht. ruternes længde.

Alt kode der er produceret i løbet af dette projekt er brændt ned på en CD-rom og vedhæftet bagerst i rapporten.

3.7 Test

Det har desværre ikke været muligt at skaffe data fra virkelige problemstillinger til dette projekt. For at kunne lave tests er det derfor nødvendigt selv at generere testinstanser. Målet er at generere instanser, der afspejler den virkelige verden så realistisk som muligt, men samtidigt er helt tilfældige indenfor en given ramme af parametre.

Som udgangspunkt placeres et antal borgere tilfældigt over et areal på 10 km^2. Arealet betragtes som et kartesisk koordinatsystem, og borgenes placering er givet ved et sæt x og y koordinater. Alle plejere i problemet starter og slutter deres ruter i origo.

3.7.1 Gruppeinddeling

Det er som udgangspunkt antaget, at alle besøg er inddelt i grupperinger. Dette betyder, at gruppeinddelingen af besøg i testinstanserne kan foretages frit. så længe de afspejler et virkeligt problem. Besøgene vil derfor blive grupperet på baggrund af deres geografiske placering.

Grupperingen af besøg foretages med en såkaldt "sweep"algoritme. Antag at den ønskede gruppestørrelse er γ. Sweep algoritmen finder grupperinger ved at udvælge γ besøg, der danner den mindste vinkel til den positive x-akse. Disse γ besøg tilføjes den første gruppe. Derefter findes de næste γ besøg med de γ mindste vinkler, der ikke allerede er tilknyttet en gruppering. Disse γ besøg tilføjes den anden gruppering. Algoritmen fortsætter, til alle besøg er tilknyttet en gruppering. Hvis den sidste gruppering ikke har den ønskede gruppestørrelse, fjernes denne fra problemet. Nedenfor er en pseudokode af algoritmen præsenteret:

Sweep Algoritme:

N: Mængden af besøg n der endnu ikke er tilføjet en gruppering.
V_l: Grupperingen, som besøgene bliver tilføjet.
L: Mængden af grupperinger, V_l.
$\alpha(n)$: Vinklen, som besøg n danner med den positive x-akse.
γ: Ønsket gruppestørrelse.

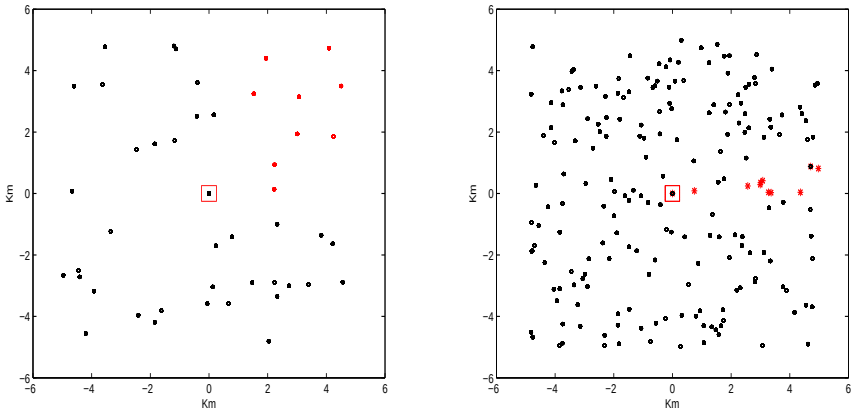
```
1:    while ( N ≠ ∅ )
2:        if ( |V_l| < γ )
3:            α* = ∞
4:            for ( n ∈ N)
5:                if (α(n) < α* )
6:                    α* = α(n)
7:                    n* = n
8:                end if
9:            end for
10:           V_l = V_l ∪ n*
11:           N = N \ n*
12:       end if
13:       else
14:           L = L ∪ V_l
15:           V_l = ∅
16:       end else
17:   end while
```

Med sweep alogritmen kan besøgene grupperes efter geografisk placering

og det er disse grupperinger, der skal repræsentere tilknytningen mellem besøg og plejer. Dvs. at antallet af plejere i problemet er direkte bestemt af antallet af grupperinger.

3.7.2 Spredning af besøg

Under testen af ruteplanlægningsheuristikken er det værd at overveje, hvor mange besøg der skal være i det givne testinstans. Problemet er, at spredningen af besøg indenfor den enkelte gruppering ændrer sig markant med problemstørrelsen. I figur 7 er to forskellige instansstørrelser illustreret.



Figur 7: *Tv. 50 besøg tilfældigt fordelt over 10 km^2.Th. 200 besøg tilfældigt fordelt over 10 km^2. De røde punkter er de ti besøg, der tilhører den første gruppering efter brug af sweep algoritmen. Plejerens startpunkt er markeret med en rød firkant.*

Figuren viser den første gruppering efter brug af sweep algoritmen for to forskellige problemstørrelser.

Til venstre på figur 7 er der illustreret en testinstants bestående af fem grupperinger. Den første gruppering markeret med røde punkter. I dette tilfælde er besøgene spredt udover hele første kvadrant.

Til højre er et testinstants med 200 besøg 20 grupperinger præsenteret. De 200 besøg er spredt over samme areal som testinstanset med 50 besøg.
Det ses, at området, som grupperingen er spredt over, er reduceret til tynd tange. Denne tendens vil blive tydeligere og tydeligere, jo større problem der arbejdes med. For overskuelighedens skyld er det kun illustreret for den første gruppering, men denne tendens gælder for alle grupperinger.

Det er tydeligt på baggrund af figur 7, at spredningen indenfor den enkelte gruppering er afhængig af antallet af besøg i det overordnede problem. Det kan tænkes, at kvaliteten af heuristikkens løsning afhænger af spredningen af besøg indenfor grupperingen. Pga. dette fænomen vil der blive genereret flere instanser med samme gruppestørrelse, men med forskelligt antal besøg i problemet til at teste rutegenereringsheuristikken.

Da alle grupper har stort set samme spredning i samme testinstans, er det ikke nødvendigt at teste alle grupperinger. Der vil derfor blive generet testinstanser som vist i figur 7, men kun lavet tests for den første gruppering i hver testinstans. Dette skal sikre, at besøgene har samme spredning i de tests lavet for ruteplanlægningsheuristikken, som de ville have i de endelige tests.

3.7.3 Tidsvinduer

Til at repræsentere tidsvinduerne anvendes vinduerne fra hjemmeplejen i Søllerød kommune beskrevet i afsnit 2.3. Tidsvinduerne er, givet ved et morgentidsvindue fra kl. 08.00-11.00, et middagstidsvindue fra kl. 11.00-14.00 og eftermiddagstidsvindue fra kl. 14.00-17.00. I testinstanserne tages der udgangspunkt i disse tidsvinduer, men det er selvfølgelig interessant at undersøge, om det er muligt at gøre vinduerne mindre, og derfor opfattes tidsvinduerne som en parameter til testinstanserne. Mindre tidsvinduer vil i praksis betyde en bedre service for borgere, da de ikke risikerer at vente i op til tre timer på en servicering.

Tidsvinduerne bruges direkte til at beskrive længden på en arbejdsdag for en given plejer. Plejerens vagtlængde dækker alle tre tidsvinduer og kan i princippet gå fra 8.00 til 17.00 plus serviceringstiden af det sidste besøg på ruten. Dette betyder, at plejerens vagtlængde ikke ligger nogen restriktioner på løsningen af problemet. Dette betyder også, at hvis tidsvinduerne gøres mindre, forkortes vagtlængden også.

Tidsvinduerne bliver tildelt, efter grupperingen er fortaget, og bliver fordelt ligeligt over antallet af besøg i gruppen. Dvs. hvis der er syv besøg i en gruppe, bliver to tildelt et morgentidsvindue, to bliver tildelt et middagstidsvindue, og to bliver tildelt et eftermiddagstidsvindue. Det sidste besøg bliver tildelt et tilfældigt tidsvindue.

På denne måde er det ikke nødvendigt, at tage højde for, at der evt. kan opstå grupper indeholdende besøg med udelukkende de samme tidsvinduer.

3.7.4 Parametre

Testinstanser, der bruges til at teste ruteplanlægningsheuristikken, er defineret ud fra følgende parametre:

- Størrelsen på tidsvinduerne
- Størrelsen på grupperne
- Servicetiden for et besøg
- Antal besøg i problemet
- Rejsehastigheden h
- Areal for besøgenes placering, O.

De tre første parametre påvirker hinanden i forhold til, om det er muligt at finde gyldige ruter. Hvis f.eks gruppestørrelsen bliver øget, risikeres det, at antallet besøg indenfor samme tidsvindue bliver så stort, at rejsetiden kombineret med servicetiden gør det umuligt at finde gyldige ruter. Det er derfor vigtigt fastholde et passende forhold mellem de tre parametre, således at der ikke bliver genereret instanser, der ikke kan løses. Da målet er at teste ruteplanlægningsheuristikken for stigende gruppestørrelser, er det nødvendigt at justere servicetiden eller tidsvinduerne, således at testinstanserne forbliver mulige at løse. Servicetiden sættes derfor udfra forholdet:

$$s_v = \frac{240}{|V_l|} \text{ min} \tag{14}$$

Hvor servicetiden er betegnet s_v, og gruppestørrelsen er betegnet $|V_l|$. Servicetiden kunne i virkeligheden betegnes s, da s_v er konstant for alle besøg. Dette er nødvendigt, idet en variabel servicetid ville resultere i en stor risiko for at generere uløselige testinstanser.

Når servicetiden er defineret til at afhænge af gruppestørrelsen, kan tidsvinduerne tilpasses, så der ikke genereres uløselige testinstanser. Til testene i dette afsnit sættes de tre typer tidsvinduer, til hhv. $[0; 180]$, $[180; 360]$ og $[360; 540]$ svarende til tidsvinduer der er typsike for hjemmeplejen. Vinduerne er angivet i minutter, og plejerens starttidspunkt sættes til tidspunktet 0.

Arealet, som besøgene er spredt over, fastsættes for alle testinstanser til 10 km^2. Dette areal skal repræsentere lokalområdet, som besøgene er tilknyttet. Hastigheden er ikke en direkte parameter til testinstanserne, men da størrelsen på arealet, besøgene er spredt over, bestemmer transportmiddel

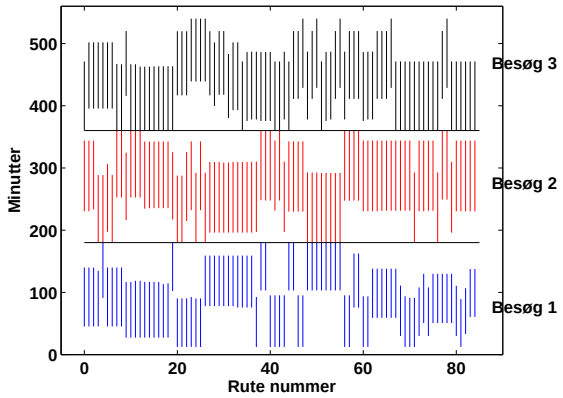
og derved hastighed, tages den med alligevel. For alle testinstanser fastsættes hastigheden til 15 km/t. Denne hastighed skal svare til, at plejerne cykler rundt til deres besøg.

For større lokalområder ville plejerne formentlig benytte en bil og rejsehastigheden ville blive forøget. Rejsetiden mellem besøgene ville derfor være nogenlunde den samme som for mindre lokalområder, hvor plejerne benytter cykler. Der er selvfølgelig tilfælde, hvor det ikke passer, men for overskuelighedens skyld laves der kun testinstanser, hvor borgerne er spredt over 10 km^2, og hastigheden fastsættes til 15 km/t.

3.7.5 Testresultater

I forbindelse med ruteplanlægningen er der to mål: Det første er at skabe diversitet i ruterne dvs. besøgstidsvinduerne for en enkelte besøg er så forskellige som muligt. Det andet mål er, at ruterne skal være så gode som muligt målt på rutelængden.

Det kan være interessant at undersøge, om rutegenerationsheuristikken rent faktisk producerer ruter, der har forskellige besøgstidsvinduer for hvert besøg. På figur 8 er der illustreret besøgstidsvinduerne for tre besøg indenfor samme gruppe og for 85 forskellige ruter.



Figur 8: *Figuren viser besøgstidsvinduerne for tre forskellige besøg indenfor samme gruppering. De tre besøg er placeret i hvert deres tidsvindue afgrænset af de vandrette linier ved 180 og 360 minutter. Besøgstidsvinduerne for de tre besøg er angivet for 85 forskellige ruter.*

På figur 8 er der angivet besøgstidsvinder for tre besøg på 85 forskellige

ruter. Gruppestørrelsen er i eksemplet sat til ti besøg. I dette eksempel er der valgt tre besøg 1, 2 og 3 der ligger hhv. i de tre forskellige tidsvinduer $[0; 180]$, $[180; 360]$ og $[360; 540]$ minutter. Figuren skal forstås sådan, at på f.eks. rute nummer 0, har besøg 1 (markret med blå) et besøgstidsvindue på $[45; 140]$. Besøg 2 (markeret med rød) har et besøgstidsvindue på $[230, 344]$ og besøg 2 (markeret med sort) har et besøgstidsvindue på $[360; 471]$. Tilsvarende er angivet for 84 andre ruter.

På figur 8 ses hvordan besøgstidsvinduerne varierer fra rute til rute. F.eks. ses det at selvom besøgstidsvinduerene for besøg 1 og 2 er stort set konstante for rute nummmer ca. 25-35, varierer de for besøg 3. Hvis der tages højde for variation i besøgstidsvinduerne for de resterende syv besøg i gruppen, kan det fastslås, at rutegeneratoren finder ruter med en vis diversitet. Figuren giver i det hele taget et meget godt billede af variationen i besøgstidsvinduerne for de forskellige ruter.

Ovenstående resultater viser, at rutegeneratoren opfylder det første mål da ruterne har en variation i besøgstidsvinduerne. Det skal derfor testes, om ruterne opfylder det andet mål nemlig at ruterne skal være så gode som muligt målt på rutelængden.

I tabel 1 er forskellen mellem den bedste løsning fundet med rutegeneratoren og den optimale løsning angivet. Testen er lavet for problemstørrelserne 400, 100 og til sidst instanser med én enkelt gruppering.

Afvigelse					
	besøg per gruppe				
borgere ialt	7	8	9	10	11
gr. størrelsen	0.50	0.25	3.73	2.78	3.47
100	0.60	1.11	2.52	2.37	5.48
400	0.00	0.81	0.26	1.01	2.73

Tabel 1: *Tabellen angiver den procentuelle forskel mellem den bedste løsning fundet med rutegeneratoren og den optimale løsning. Resultaterne er angivet som et gennemsnit over ti kørsler på samme problem.*

I tabel 1 er der angivet den procentuelle forskel mellem den bedste rute ud af 30 ruter genereret med rutegeneratoren og den optimale løsning. Resultaterne er angivet som et gennemsnit over ti kørsler på samme problem.

Tabellen viser en lille tendens, til at ruterne bliver en anelse dårligere, når spredningen af besøg vokser. Denne tendens er dog ikke helt klar, da der tilsyneladende findes bedre ruter for problemer med 100 borgere i forhold til problemer med kun den ene gruppe af besøg.

Det ses, at de bedste ruter fundet med rutegeneratoren ligger forholdsvis tæt på den optimale løsning. Problemet er, at det kun har været muligt at generere optimale løsninger for op til 11 besøg på rute. Grunden til dette er, at de genereres udfra en såkaldt brute force metoden. Metoden genererer alle permutationer af besøg rækkefølger og tjekker derefter, om de er gyldige. Den bedste af gyldige løsninger angiver den optimale løsning, da alle permutationer er afprøvet. Køretiden for metoden stiger eksponentielt med antallet af besøg, og gruppestørrelsen er derfor begræsnet til maksimalt 11.

Problemet er at den anvendte type grådig algoritme, som regel laver dårligere og dårligere løsninger, når antallet af besøgene øges. Med lidt god vilje kan denne tendens godt anes i tabel 1, men er på ingen måde klar.

I tabel 2 er kørselstiden brugt på generere de 30 ruter brugt til resultaterne i tabel 1.

Kørselstid					
	besøg per gruppe				
borgere ialt	7	8	9	10	11
gr. størrelsen	0.011	0.013	0.017	0.019	0.026
100	0.010	0.013	0.016	0.022	0.024
400	0.012	0.012	0.014	0.023	0.026

Tabel 2: *Tabellen angiver den samlede kørselstid angivet i sekunder for at generere de 30 ruter brugt til resultaterne i tabel 1.*

Tabel 2 viser, at kørselstiden for at generere 30 ruter for en enkelt gruppering, ligger mellem 1-3 hundrededele sekund for disse gruppestørrelser. Kørselstiden må betegnes som acceptabel til rutegenerering for hver gruppering.

Det kan fastslås, at rutegeneratoren både genererer forskellige og gode løsninger op til 11 besøg per gruppering. Da et sted mellem 8 og 11 besøg per rute ikke er unormalt i virkeligheden og ruterne findes på hundrededele af et sekund, er rutegeneratorens mål opfyldt.

4 Vægtet Grupperet Klike Problem

I foregående afsnit blev der præsenteret en rutegenerator til generering af en mængde ruter for hver gruppering. Der blev ikke taget højde for fællesbesøg mellem grupperingerne (se evt. figur 2). Problematikken forbundet med at finde et passende rutesæt således, at det er muligt for plejerne med fællesbesøg at være der samtidig, vil blive behandlet i dette afsnit.

4.1 Vægtet Grupperet Klike Problem

I dette afsnit indføres begrebet kompatibilitet mellem to ruter, tilknyttet hver deres plejer med et indbyrdes fællesbesøg. Kompatibilitet skal forstås sådan, at hvis to ruter gør at plejerne kan være hos dette fællesbesøg samtidig, er ruterne kompatible. Hvis der er mere end et fællesbesøg mellem to ruter, er ruterne kun kompatible, hvis begge plejere kan være hos alle indbyrdes fællesbesøg samtidig.

Kompatibiliteten mellem to ruter kan bestemmes udfra de såkaldte besøgstidsvinduer $[a_v^{u_l}, b_v^{u_l}]$, der beregnes i ligning (10) og (11) i indsættelsesheuristikken. Hvis besøgstidsvinduerne overlappper, er det muligt for de to plejere at servicere fællesbesøget samtidigt, og ruterne er derved kompatible. Da en servicering ved et fællesbesøg først kan påbegyndes, når begge plejere er til stede, bliver det tidligste starttidspunkt $a_v^{u_l}$ for begge plejere givet ved det seneste for fællesbesøget. Dette kan skrives som:

$$a_{v_{fb}}^{u_1} = a_{v_{fb}}^{u_2} = \max\left(a_{v_{fb}}^{u_1}, a_{v_{fb}}^{u_2}\right)$$

Problematikken forbundet med denne ændring i besøgtidsvinduet er behandlet i afsnit 5.

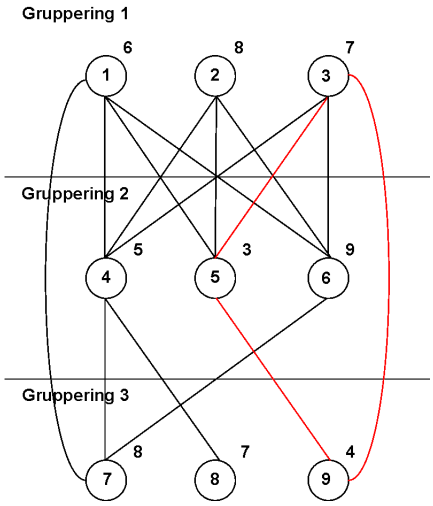
Til at præsentere kompatibilitet mellem ruterne opstilles en kompatibilitetsgraf. En knude i en kompatibilitetsgraf repræsentere en rute. Hvis der er kompatibilitet mellem to ruter, forbindes de med en kant. Ruter, der tilhører samme plejer, har derved ingen fælles kan. Ruter, der ikke har fællesbesøg, har altid fælles kant.

Kompatibilitetsgrafen danner grundlag for et optimeringsproblem, der i denne rapport vil blive betegnet som et Vægtet Grupperet Klike Problem (VGKP). Sammenhængen mellem VGKP og kompatibilitetsgrafen er beskrevet i det følgende.

VGKP er inddelt i en række grupperinger. Grupperingerne består af et antal knuder med en given vægt. I praksis svarer antallet af grupperinger til antallet af plejere i det givne problem. I en gruppering betegner hver knude

en bestemt rute, den enkelte plejer kan tildeles. En knude svarer derfor til en bestemt ordning af besøgene på ruten. Hver rute har tilknyttet en vægt svarende til længden (omkostningen) af ruten.

For en gyldig løsning skal der gælde, at alle ruter i løsningen skal være indbyrdes kompatible. Kompatibiliteten mellem de forskellige ruter findes i kompatibilitetsgrafen. Målet er at finde en gyldig løsning, der minimerer den samlede rejselængde, plejerne skal bruge og samtidig overholde, kompatibilitetsbegrænsningerne. På figur 9 er der givet et eksempel på VGKP opstillet som kompatibilitetsgraf. Det ses på figuren, at løsningen er en kom-



Figur 9: *Figuren viser et eksempel på VGKP for tre grupperinger/plejere og ni knuder/ruter. Kompatibiliteten mellem ruterne er angivet som kanter, og den optimale løsning er angivet med rødt. Tallet inde i knuderne er knudernes respektive numre, og tallet udenfor knuderne er længden på ruten, som knuden repræsenterer.*

plet delgraf af problemet, og at antallet af knuder i løsningen præcis svarer til antallet af grupperinger. For gyldige løsninger til VGKP skal dette altid være opfyldt.

Figur 9 viser, at gruppering 1 og gruppering 2 er uafhængige af hinanden, da alle knuder i de to grupperinger har en kant til fælles. Dette vil i praksis enten betyde, at plejeren, der servicerer borgere i gruppering 1, ikke har et fællesbesøg med plejeren i gruppering 2 eller, at alle deres respektive ruter er indbyrdes kompatible.

Den viste løsning svarer til, at plejer 1 vælger rute nr. 3, plejer 2 vælger

rute nr. 5, og plejer 3 vælger rute nr. 9. Det ses, at der er alternative rutevalg; plejer 1 kan f.eks. vælge rute nr. 1, plejer 2 kan vælge rute nr. 6, og plejer 3 kan vælge rute nr. 7. Dette rutevalg vil også være en gyldig løsning på problemet.

Det viste rutevalg gør, at tidsvinduerne for de enkelte besøg bliver overholdt. Plejere med fællesbesøg ankommer indefor et overlappende tidsrum hos det pågældende besøg, så de kan nå at servicere vedkommende og stadig nå resten af besøgene på deres respektive ruter.

For den type VGKP, der ønskes behandlet, kan problemstørrelserne nemt blive meget store. Dette sker pga. det store antal lovlige ruter, der kan genereres per plejer. Antallet af ruter, der genereres for hver plejer, skal være stort nok til at sikre, at det er muligt at finde et sæt af kompatible, gode ruter, der tilsammen danner en gyldig løsning.

Problemstørrelsen vil typisk være en form for kompromis, da antallet af mulige ruter for hver gruppering stiger eksponentielt med antallet af besøg i en given gruppering. Det kan derfor være nødvendigt kun at medtage en mindre mængde af det samlede antal ruter. Kompromiset består i kun at generere en brøkdel af det samlede antal ruter og derved reducere problemstørrelsen. Dette betyder at det risikeres at løsningen der opnås, er mindre god. Det er derfor vigtigt, at ruteplanlægningsheuristikken præsenteret i afsnit 3 finder gode og kompatible ruter, så de ruter, der inkluderes i VGKP, stadig kan lave gode og gyldige løsninger.

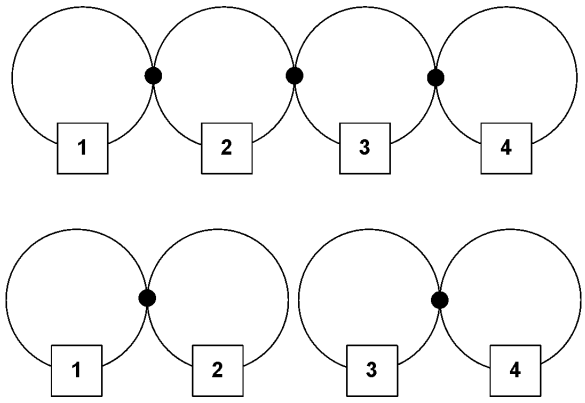
4.2 Sammenhængende problemer

Udover det ovenstående er det nødvendigt at betragte følgende eksempel:

- Der betragtes fire grupper tildelt plejer 1, 2, 3 og 4.
- Plejer 1 har fællesbesøg med plejer 2, og plejer 3 har fællesbesøg med plejer 4

Hvis ikke plejer 3 enten har fællesbesøg med 1 eller 2, eller plejer 4 har fællesbesøg 1 eller 2, er plejer 1 og 2 uafhængige af plejer 3 og 4. Grunden til dette er, at ligegyldig hvilken rute der vælges for plejer 1 og 2, påvirker det ikke hjemmehjælper 3 og 4. I figur 10 er eksemplet illustreret med og uden fællesbesøg mellem plejer 2 og plejer 3.

Udfra figur 10 ses det, at hvis plejer 2 og plejer 3 ikke har fællesbesøg, kan problemet deles op i to, og de to dele kan opstilles som uafhængige VGKP problemer og løses hver for sig. Der betragtes derfor kun problemer, hvor alle



Figur 10: *Figuren viser de fire grupper. Kun fællesbesøgene betragtes og er angivet som punkter. Øverst er tilfældet, hvor plejer 2 og 3 har fællesbesøg illustreret. Nederst er tilfældet, hvor plejer 2 og 3 ikke har fællesbesøg illustreret, men plejer 1 og 2 og plejer 3 og 4 har stadig fællesbesøg.*

plejere er afhængige af hinanden som angivet øverst i figur 10. Udfra dette følger, at hvis en gruppe ikke har nogen fællesbesøg, er den uafhængig af problemet og skal ikke betragtes mht. kompatibilitet. Det optimale rutevalg fremkommer ved at løse TSPTW for den enkelte gruppe.

Udfra eksemplet ses det, at alle grupperinger undtagen to skal minimum have to fællesbesøg, og alle grupperinger skal som minimum have et fællesbesøg. Problemet vil fremover angtages at være sammenhængdende.

For at ruterne i et rutesæt skal være indbyrdes kompatible, skal alle ruter i rutesættet have en kant tilfælles i kompatibilitetsgrafen. Målet er derfor at finde en komplet delgraf i kompatibilitetsgrafen med præcis én knude for hver plejer. Dette optimeringsproblem er behandlet i de efterfølgende afsnit.

4.3 Litteratur om Vægtet Grupperet Klike Problem

Det har ikke været muligt at finde litteratur omhandlende VGKP. Der findes derimod masser af artikler om det relaterede problem, Maximal Clique Problem (MCP).

I MCP forsøges det at finde den største klike i en graf. Problemet er bl.a. behandlet [6] og [7]. Artiklerne behandler problemet med hhv. tabusøgning og en grådig algoritme.

Problemet er, at MCP adskiller sig fra VGKP på et vitalt område. I MCP

søges der en maximal klike, hvor der i VGKP søges en klike med fast størrelse, nemlig antallet af grupperinger. I VGKP er løsningen ydermere betinget af at løsningen skal indeholde præcis en knude i hver gruppering.

Disse forskelle gør at det ikke er muligt bruge MCP løsningsmetoder til løsning af VGKP.

4.4 Matematisk model

Lad $G(U, E)$ være en ikke-orienteret graf, hvor U er mængden af knuder u og E er mængden af kanter e. Hver knude u har en vægt betegnet ω_u. En gruppering af ruter tilknyttet plejer l er givet ved en delmængde betegnet m_l, og mængden af alle grupperinger er givet ved M. Antallet af grupperinger er betegnet L.
For grupperinger gælder der, at de skal være en ægte delmængde af mængden af knuder, dvs. $m_l \subset U$. De enkelte grupperinger må ikke have fælles knuder $m_i \cap m_j = \emptyset, \quad \forall\, (m_i, m_j) \in M$, og fællesmængden af alle grupperinger skal være lig med mængden af kanter $m_1 \cup m_2 \cup \ldots \cup m_L = U$. Variablen x_u er givet ved følgende:

$$x_u = \begin{cases} 1, & \text{hvis de ruten } u \text{ vælges til rutesættet.} \\ 0, & \text{ellers.} \end{cases}$$

Den matematiske model for VGKP kan skrives som følger:

$$\min \quad \sum_u \ \omega_u x_u \tag{15}$$

$$x_{u_i} + x_{u_j} \ \leq \ 1, \quad \text{for} \quad e_{u_i,u_j} \notin E, \ \ u_i \in m_i, \ u_j \in m_j \tag{16}$$

$$\sum_u x_u \ = \ 1, \qquad u \in m_l, \qquad l = 1, \ldots, L \tag{17}$$

$$x_u \ \in \ \{0, 1\} \quad \forall\, u \in U \tag{18}$$

Målfunktionen (15) minimerer den samlede vægt på knuder der indgår i løsningen. Begrænsningen givet i (16) sikrer, at der er en kant mellem alle knuder, der indgår i løsningen. For at sikre, at der bliver valgt præcis en knude i hver gruppering, indføres begrænsningen (17).

4.5 Heuristikker

VGKP er i dette projekt tiltænkt som et delproblem, der skal løses som del af et større problem. Det er derfor vigtigt, at der kan dannes gode løsninger på kort tid. Til dette formål bliver der i dette afsnit præsenteret en heuristisk metode. En heuristisk metode er en metode til at finde løsninger på

et problem udfra en viden omkring problemets struktur. I forbindelse med VGKP kan denne viden f.eks. være, at problemet er inddelt i grupperinger, hvor der skal vælges præcis én knude fra hver gruppering. Ved at anvende en heuristik til at finde løsninger til VGKP er håbet, at der kan findes gode løsninger på kort tid.

Som nævnt tidligere har det ikke været muligt at finde litteratur omkring VGKP, og det har derfor været nødvendigt at eksperimentere med, hvilken type heuristik der kan være god til denne type problem. Der vil i dette afsnit blive præsenteret overvejelser omkring konstruktions- og forbedringsheuristikker.

4.6 Konstruktionsheuristikker

En konstruktionsheuristik bruges til at generere løsninger til et givent problem helt fra bunden. Dette gøres udfra en viden om problemstrukturen. Den kan have forskellige mål alt efter, hvad den skal konstruere løsninger til. Hvis den f.eks. skal generere en løsning til en forbedringsheuristik, behøver det ikke altid være en god løsning eller måske ikke engang en gyldig løsning. I dette afsnit præsenteres to konstruktionsheuristikker til VGKP.

4.6.1 Første konstruktionsheuristik KH1

I første omgang er ideen at lave en simpel konstruktionsheuristik inspireret af en grådig algoritme. Denne heuristik behøver ikke nødvendigvis at generere gyldige løsninger, da det er meningen, at en forbedringsheuristik skal kunne arbejde med ugyldige løsninger. På den måde skal det være forbedringsheuristikkens opgave at gøre løsningerne gyldige. Ideen bag denne konstruktionsheuristik er, at knuderne til løsningen udvælges grådigt en gangen. Initielt vælges knuden med laves vægt i en tilfældig gruppering. Udvælgelsen består derefter i, at der i hver iteration udvælges knuden lavest vægt i en tilfældig gruppering. Udvælgelsen gøres under forudsætning af, at grupperingen ikke har været valgt i de foregående iterationer, og knuden skal være kompatibel med knuden valgt i forrige iteration. Den første konstruktionsheuristik fungerer på følgende måde:

S: Vektor indeholdende de knuder, der udgør løsningen.
F: Matrix, der indeholder information om hvilke knuder der er indbyrdes kompatible.
U: Mængden af knuder u.
Ω: Vektor med vægten ω_u af den pågældende knude u.
M: Mængden af grupperinger m.

1. $u = \arg\min_{u \in U,\ m \in M}\{\omega_u\},\ S \leftarrow S \cup \{u\}$.

2. $u = \arg\min_{u \in U,\ m \in M}\{\omega_u\},\ S \leftarrow S \cup \{u\}$ for en gruppering m der ikke har været valgt i en foregående iteration og for en knude u der ikke har været prøvet indsat i S.

3. Hvis $|S| + 1 = L$, gå til trin 4. Ellers tjek i F, om den valgte knude u i trin 2 er kompatibel med knuden sidst tilføjet i S. Hvis den valgte knude er kompatibel, tilføj den i S, ellers gå tilbage til trin 2.

4. Hvis den valgte knude er kompatibel med knuden valgt i trin 1, tilføj den i S og stop. Ellers tjek, om alle knuder i m har været prøvet. Hvis alle knuder har været prøvet, gå til trin 5. Ellers gå tilbage til trin 2.

5. Returner en fejlmelding og en tilfældig startløsning. Stop.

Denne heuristik er simpel og meget hurtig. Heuristikken sikrer ikke gyldige løsninger. Dette kan ske, fordi der kun tjekkes, om den valgte knude i trin 2 er kompatibel med den forrige. Der tjekkes ikke for kompatibilitet med de resterende knuder tilføjet i S.
For bestemte problemtyper kan det være umuligt at finde en kompatibel knude i en given iteration. Dette håndteres ved at returnere en fejlmelding og en tilfældig startløsning. Denne startløsning genereres ved at tage en tilfældig knude i hver gruppering og tilføje den til løsningen.

4.6.2 Anden konstruktionsheuristik KH2

Det næste naturlige skridt er at finde en konstruktionsheuristik, der kan finde gode, gyldige startløsninger. Problemet med den forrige konstruktionsheuristik er, at den producerer startløsninger, der mangler mange kanter i forhold til at danne en komplet delgraf i kompatibilitetsgrafen og stiller derved store krav til forbedringsheuristikken. Der konstrueres derfor en ny konstruktionsheuristik, der er en udbyggelse af den forrige. Ideen er, at i hver iteration skal en knude u kun tilføjes til løsningen S, hvis den er kompatibel med alle knuderne, der allerede er valgt til løsningen. Der indføres til dette formål et kandidatsæt $C(m, |u_f|)$, der betegner de $|u_f|$ billigste kompatible knuder i en gruppering m. Grupperingerne m vælges tilfældigt en af gangen i hver iteration, og der vælges en tilfældig knude fra kandidatsættet $C(m, |u_f|)$, der tilføjes til løsningen S. Konstruktionsheuristikken er opstillet nedenfor:

S: Vektor indeholdende de knuder, der udgør løsningen.
F: Kompatibilitetsmatrix.
U: Mængden af knuder u.
M: Mængden af grupperinger m.

$|u_f|$: Antallet af knuder i kandidatsættet $C(m, |u_f|)$.
$C(m, |u_f|)$: Vektor med et kandidatsæt indeholdende de $|u_f|$ billigste kompatible knuder til S for den givne gruppering m.

1. Find en tilfældig gruppering m. Vælg en tilfældig knude i $C(m, |u_f|)$ og tilføj den i S.
2. Find en tilfældig gruppering m, som ikke har været valgt. Vælg en tilfældig knude i $C(m, |u_f|)$ og tilføj den i S.
3. Trin 2 gentages indtil $|S| = L$. Hvis det resulterer i en gyldig løsning, så stop, ellers fjern knuderne i S og gå tilbage til trin 1.

Hvis ikke kandidatsættet er for restriktivt, eller problemet har få gyldige løsninger, vil denne konstruktionsheuristik finde gyldige startløsninger til det givne problem. Hvis nødvendigt kan den køres flere gange for at opnå bedre og/eller gyldige startløsninger. Tilfældigheden i valg af gruppering og valg af knude indenfor kandidatsættet vil gøre, at der højst sandsynligt vil genereres en ny løsning, hvis heuristikken køres igen.

4.7 Forbedringsheuristikker

I dette afsnit præsenteres overvejelser omkring forskellige forbedringsheuristikker til løsning af VGKP. Nogle af heuristikkerne er blevet kasseret indledningsvis, da de viste for dårlige resultater, men bliver alligevel præsenteret, da de giver en god illustration af, hvor problemerne i at løse VGKP ligger.

Der indføres lokalsøgning til at forbedre den eksisterende løsning fundet med konstruktionsheuristikken. Elementerne der indgår i forbedringsheuristikken gennemgås i de følgende afsnit.

4.7.1 Omkostningsfunktion

For at kunne evaluere kvaliteten af de genererede løsninger, skal der defineres en omkostningsfunktion. Denne omkostningsfunktion skal kunne straffe ugyldige løsninger. Dette håndteres ved at undersøge, om knuderne i en given løsning danner en komplet subgraf i kompatibilitetsgrafen. Hvis dette ikke er tilfældet, bliver hver manglende kant straffet med en konstant ρ. Grunden til, at manglende kanter straffes, er, at en gyldig løsning er givet netop ved en komplet graf. Hvis k_S betegner antallet af manglende kanter i en løsning S, kan man ved brug af notationen i (15) skrive omkostningsfunktionen således:

$$\sum_u \omega_u x_u + \rho k_S \tag{19}$$

Ideen med denne omkostningsfunktion er, at heuristikken skal vægte ugyldige knuder medtaget i løsningen forskelligt. Heuristikken skal kunne favorisere knuder, der er kompatible med mange andre knuder i løsningen frem for knuder, der er kompatible med få. Med denne omkostningsfunktion skulle heuristikken gerne nærme sig gyldige løsninger. Det er værd at bemærke, at for gyldige løsninger bliver omkostningsfunktionen den samme som målfunktionen i den oprindelige matematiske model.

4.7.2 Nabolag

Der skal nu laves en lokalsøgning til forbedring af løsningen genereret med konstruktionsheuristikken. For at kunne lave en lokalsøgning skal der defineres et nabolag, hvor søgningen skal foregå. Et nabolag er en mængde af løsninger, der kan opnås vha. en veldefineret ændring udfra en given løsning. På grund af problemets grupperingsstruktur er et oplagt nabolag til en bestemt løsning, at en knude erstattes med en knude indenfor samme gruppering. Alle knuder i denne gruppering prøves i løsningen i stedet for den knude, der oprindeligt var valgt til løsningen. Nedenfor er dette nabolag opstillet med matematisk notation:

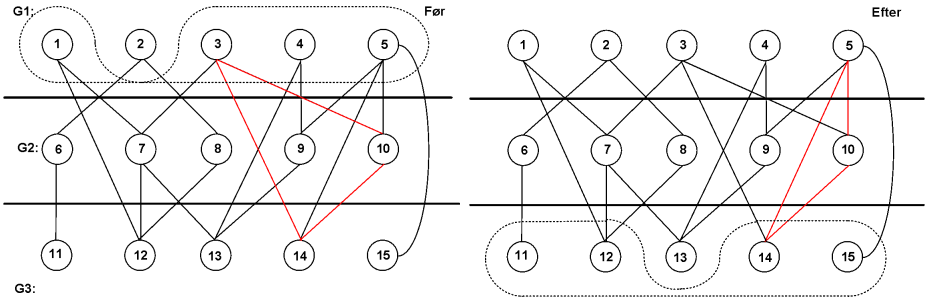
$$Q(S) = \{\overline{S} \subset U : \overline{S} = S \cup \{u_{m_i}\} \text{ for } u_{m_i} \notin S \text{ og } \overline{S} = S \backslash \{u_{m_j}\} \text{ for } u_{m_j} \in U_m\}$$

Ovenfor er nabolaget Q for en løsning S angivet. Naboløsningen $\overline{S}$ opnås ved at fjerne en knude u_{m_j} fra løsningen og derefter erstatte knuden med en knude u_{m_i} indenfor samme gruppering m. U_m angiver en delmængde af U indeholdende alle knuder i grupperingen m.

Et problem forbundet med dette nabolag er, at det vokser lineært med størrelsen af grupperinger. Dette er i sig selv ikke noget problem, men der skal ved hver knudeombytning beregnes en kompatibilitetsgrad k_S, som skal indgå i omkostningsfunktionen (19). Kompleksiteten på denne beregning er givet ved $O(2US^2)$. Grunden til at kompleksiteten, er, så stor er at for hver knude u i løsningen S skal der findes kompatibilietetsgraden i kompatibilitetsmatricen med størrelsen $U \times U$. Da beregningen skal laves ved hver eneste ombytning, kan evalueringen af nabolaget hurtigt blive meget tungt. VGKP optræder som et delproblem, og derved kan store nabolag og derved lange kørselstider blive problematiske.

I stedet for, at alle knuder i hver gruppering betragtes, betragtes et konstant antal knuder, som vælges tilfældigt. På denne måde er det muligt at justere nabolaget, så det får en passende størrelse i forhold til en ønsket kørselstid.

Der er tre karakteristika for en given knude. Det første er, hvilken gruppering den tilhører, det andet er knudens vægt, og det sidste er knudens kompatiblitet. Knudernes tilhørsforhold til en bestemt gruppering ligger fast. Hvis der fjernes en knude i en given gruppering, skal den altid erstattes med en knude fra samme gruppering. De to andre karakteristika tages der højde for i omkostningsfunktionen og kan derfor ikke bruges som vurderingsværktøj i denne sammenhæng. Det er derfor svært at finde en fornuftig delmængde indenfor en bestemt gruppering, og derfor udvælges knuderne tilfældigt.



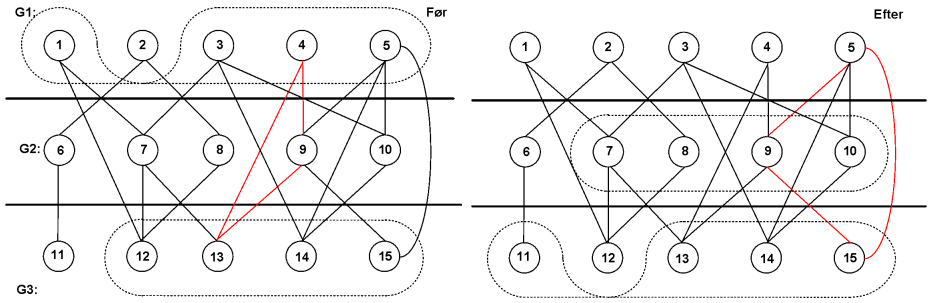
Figur 11: *Figuren viser et 1-knude nabolag. Nabolaget er givet ved knuderne indrammet af den stiplede linie. De røde kanter repræsenterer den aktuelle løsning. Tv. løsningen før evaluering og th. er løsningen efter evaluering. Grupperingerne er nummeret G1-G3.*

På figur 11 er nabolaget illusteret. Den aktuelle løsning er til venstre angivet ved knude nr. 3, 10, 14 forbundet med de røde kanter. Nabolaget til løsningen er indrammet med den stiplede linie. Den nye løsning findes ved at erstatte knude nr. 3 med en af de andre knuder i nabolaget. Den nye løsning betegnes som en naboløsning til den forrige.

I dette eksempel er der kun en gyldig løsning indefor nabolaget nemlig løsningen angivet ved knude nr. 5, 10, 14. Til højre ses den nye løsning efter evaluering af nabolag.

Det kan tænkes, at nabolaget har svært ved at undslippe strukturen givet i den initielle løsning, da der kun ændres på en knude ad gangen i løsningen.

Et alternativt nabolag kan derfor være at vælge to tilfældige knuder og fjerne dem fra løsningen. Knuderne erstattes derefter med to andre knuder valgt i en tilfældig delmængde indenfor samme gruppering. Dette er illustreret i figur 12. Figuren viser ændringen før og efter en evaluering.



Figur 12: *Figuren viser et 2-knude nabolag. Nabolaget er givet ved knuderne indrammet af den stiplede linie. De røde kanter repræsenterer den aktuelle løsning. Tv. løsningen før evaluering og Th. er løsningen efter evaluering. Grupperingerne er nummeret G1-G3.*

Hvis figur 11 og 12 sammenlignes, ses det, at dette nabolag har større mulighed for at ændre på en given løsning i hver evaluering. For nogen problemtyper kan dette være en fordel, og for andre problemtyper er det en ulempe. Hvis problemet har lav kompatibilitet, kan heuristikken meget nemt blive tvunget til at vælge løsninger, der ikke er gyldige. Hvis der på figur 12 f.eks. ikke var nogen kant mellem knude nr. 9 og 15, ville det ikke være muligt at danne en gyldig løsning indefor nabolaget. Hvis kompatibiliteten er tilstrækkelig lav, kan heuristikken risikere at bevæge sig længere og længere væk fra gyldige løsninger i hver iteration.

Dette nabolag bliver desværre hurtig beregningstungt. Hvis 2-knude nabolaget sammenlignes med 1-knude nabolaget, skal kørselstiden kvadreres i hver evaluering af nabolagene. Dvs. antallet af knuder, der betragtes i hver gruppering, skal være væsentlig mindre for 2-knude nabolag for, at heuristikken kan beholde samme hastighed. Dette er et problem, fordi løsningen kan forværres betydeligt, hvis heuristikken er tvunget til at vælge to dårlige knuder i en iteration.

4.7.3 Lokalsøgning

En lokalsøgning er en søgning, der koncentrerer sin søgning om løsninger i nabolaget af en løsning S. I det foregående afsnit er der angivet to mulige måder at definere forskellige naboløsninger til en løsning S. Løsningen S erstattes efter lokalsøgningen med den bedste naboløsning fundet i nabolaget. I dette projekt er den bedste naboløsning den løsning med lavest omkostning givet ved omkostningsfunktionen i (19).

For at teste nabolagene og få et indblik i, hvordan de opfører sig, genereres

derfor en lokalsøgningsheuristik. Den fungerer meget simpelt: Først vælges en tilfældig knude i løsningen. Den valgte knude erstattes med en knude ad gangen fra nabolaget. Dette fortsættes, indtil alle knuder har været prøvet. Den knude i nabolaget, der resulterer i den mindste omkostningsfunktionsværdi, erstatter den valgte knude i løsningen. Herefter vælges en ny tilfældig knude i løsningen, som ikke har været valgt i forrige iteration, og det hele gentages. Dette gentages i et fast antal iterationer.

Lokalsøgning med to-knude nabolag undersøger alle kombinationerne af ændringer indenfor de to grupperinger. Det knudepar i nabolaget, der resulterer i den mindste omkostningsfunktionsværdi, erstatter knuderne fra løsningen. Dette gentages i et fast antal iterationer. Nedenfor er der præsenteret en pseudokode af heuristikken.

LokalSøgning(S):

S: Midlertidig løsning.
S*: Hidtil bedste løsning.
S_{nabo}: Løsning med nabolag i en bestemt gruppering, *nabo*.
S_{nabo*}: Hidtil bedste naboløsning.
MaxNabo: Antallet af nabolag der skal undersøges.

```
1:      while (AntalNabo < MaxNabo)
2:          nabo = TilfældigGruppering();
3:          |S_nabo*| = EtMegetStortTal;
4:          S = Q(S_nabo);
5:          while (iterationer < MaxIterationer)
6:                    if (|S| < |S*|)
7:                    {S* = S;}
8:                    else if (|S| < |S_nabo*|)
9:                    {S = S_nabo*;}
10:                   else
11:                   {S = Q(S_nabo);}
12:                   iterationer++;
13:           S = S_nabo*;
14:           AntalNabo++;
15:     return S*;
```

Lokalsøgningen har som input en løsning S, f.eks. fundet med KH2. Denne løsning erstattes med en naboløsning til S fra en tilfældig gruppering. Hvis denne løsning er bedre end den hidtil bedste løsning S*, sættes S* = S. Ellers tjekkes der, om den midlertidige løsning S er bedre end den hidtil bedste naboløsning S_{nabo*}. Hvis dette er tilfældet, erstatter den midlertidige løsning den hidtil bedste naboløsning, $S = S_{nabo*}$. Hvis de to førnævnte be-

tingelser ikke er opfyldt, findes en ny, midlertidig løsning i nabolaget $S = Q(S_{nabo})$. Dette fortsættes et fast antal iterationer givet ved MaxIterationer. Når MaxIterationer er nået, sættes den midlertidige løsning lig med den bedste naboløsning $S = S_{nabo*}$, og hele metoden gentages for en ny tilfældig gruppering.

Med omkostningsfunktionen givet i (19) kan den ovenstående lokalsøgningsheuristik arbejde med ugyldige løsninger og nabolaget kan være begge nabolag beskrevet tidligere i afsnittet.

4.7.4 Simuleret udglødning

Med lokalsøgningen implementeret er det forholdvis simpelt at lave simuleret udglødning. Simuleret udglødning er inspireret af kontrolleret nedkøling af et materiale, så der opnås en ændring i krystalstrukturen. Grundideen i simuleret udglødning er, at der startes med en høj temperatur. Ved denne temperatur accepteres stort set alle løsninger i nabolaget. Herefter falder temperaturen med en given faktor per iteration. Efterhånden som temperaturen falder, accepteres der med mindre og mindre sandsynlighed dårligere løsninger. Dvs. i starten er det som en tilfældig søgning, der gradvist går over i en lokalsøgning dog med en lille sandynlighed for undslippe lokalområdet.

For at fastslå, om simuleret udglødning er en brugbar heuristik til at løse VGKP, er en simpel version implementeret. Nabolagene er som givet i 4.7.2, og temperaturen er sat til at aftage med en fast faktor. Denne version af simuleret udglødning er beskrevet i [1].

En af styrkerne ved simuleret udglødning er, at der bliver taget valg ind imellem som resulterer i dårligere løsninger for at undslippe lokale optimum. Dette viste sig at være meget problematisk, da heuristikken havde svært ved at finde tilbage til gyldige løsninger. Selvom ugyldige løsninger blev straffet ekstremt hårdt, fandt heuristikken aldrig tilbage til gyldige løsninger, og heuristikken befandt sig stort set hele tiden i det ugyldige område af løsningsrummet. Dette gælder for brug af begge konstruktionsheuristikker, begge nabolag og med forskellige versioner af straf i omkostningsfunktion. Derudover blev en version, der slet ikke accepterede ugyldige løsninger, afprøvet. I denne version blev de bedste løsninger altid fundet med konstruktionsheuristikken.

Der var ingen resultater, der indikerede, at simuleret udglødning var værd at arbejde videre med, og derfor blev brugen af denne meta-heuristik kasseret.

4.7.5 Generelle betragtninger

I forsøget på at lave en heuristik til VGKP er en række fremgangsmåder blevet prøvet. Mange af dem har ikke virket, men de har alle bidraget til at få et bedre indblik i problemet. I løbet af denne fase er der lavet følgende betragtninger.

Da der arbejdes med at finde komplette delgrafer med en fastlagt størrelse og med grupperingsforudsætningerne, er antallet af ugyldige løsninger i forhold til antallet af gyldige løsninger meget stort. Hvis der arbejdes med ugyldige løsninger, gør det, at heuristikken skal være meget effektiv til at søge tilbage til det gyldige løsningsrum i sin søgning. Nogen gange kan heuristikken tvinges til at søge tilbage til gyldige løsninger ved at tildele en straf efter, hvor mange iterationer der har returneret ugyldige løsninger. Altså jo længere tid, der returneres ugyldige løsninger, desto dyrere bliver det. Der blev også testet strafmetoder med en konstant ρ af forskellig størrelse. Ingen af disse metoder har vist sig at være synderligt effektive.

Forklaringen skal formentlig findes i nabolaget. Hvis alle muligheder i nabolaget er så dårlige, at løsningen bliver langt fra gyldig, kan det være svært at finde tilbage til gyldige løsninger. Dette kan meget vel være tilfældet her. I problemet findes der kun få gyldige løsninger i forhold til problemstørrelsen, og der kan derved godt opstå situationer, hvor heuristikken er tvunget til at vælge knuder med meget lav kompatibilitet. Hvis heuristikken tilstrækkelig tit løber ind i nabolag, hvor alle muligheder forringer løsningen meget, søges der aldrig tilbage til gyldige løsninger uafhængigt af strafmetode. Derfor virker det mest fornuftigt at lave en heuristik, der primært arbejder med gyldige løsninger.

Under de initielle test af nabolag og de forskellige forbedringsheuristikker er det blevet klart, at der er et fælles problem. Problemet består i, at de gode løsninger altid forekommer i de første iterationer, og heuristikkerne bevæger sig væk fra gyldige løsninger og kommer sjældent tilbage til nye gyldige løsninger. Dette understøtter, at problemet nok ligger i nabolagene kombineret med, at der accepteres gyldige løsninger.

I afsnittet om nabolag blev der præsenteret to forskellige nabolag. Et nabolag, hvor der bliver fjernet en knude af gangen fra løsningen, og et, hvor der bliver fjernet to. Nabolaget, hvor der bliver fjernet to knuder af gangen, klarede sig markant dårligere i alle sammenhænge. Grunden til dette er, at strukturen på den konstruerede løsning bliver ændret totalt i hver iteration. Det er meget svært for heuristikken at finde tilbage til en gyldig komplet delgraf, når der bliver påtvunget at fjerne to knuder fra løsningen i hver iteration.

Generelt virker det som om, at nabolagene er meget svage mht. at til forbedre eksisterende løsninger. Dette gør, at alle typer forbedringsheuristikker der bruger nabolagene, vil klare sig dårligt. Problemet er, at det er svært at finde en anden type nabolag, da problemet er opdelt i grupperinger, hvilket virker som en slags naturligt nabolag. Hvis der startes med gode løsninger, fundet med KH2, finder forbedringsheuristikkerne sjældent en bedre løsning. Hvis der startes med dårligere løsninger, kan en lokalsøgning med et stort nabolag godt forbedre løsningen, men generelt bliver de gode løsninger fundet af konstruktionsheuristikken KH2.

4.7.6 Genstartsheuristik

På baggrund af de generelle betragtninger virker det hensigtsmæssigt at lave en form for genstartsheuristik. Denne genstartsheuristik tager udgangspunkt i KH2, da målet er at arbejde med gyldige løsninger. Efter KH2 har genereret en løsning, køres der en lokalsøgning som angivet i afsnit 4.7.3, der kun arbejder med gyldige løsninger. Lokalsøgningen stopper, når løsningen ikke har forbedret sig i et fast antal iterationer. Herefter genstartes heuristikken med KH2, som genererer en ny løsning. Da KH2 vælger tilfældige knuder fra kandidatsættet i hvert skridt, opnås der forkellige gyldige løsninger hver gang, der genstartes. Lokalsøgningen bliver derefter genoptaget, og sådan gentager heuristikken sig selv indenfor den ønskede tidsramme. Nedenfor er der præsenteret en pseudokode af heuristikken.

Genstartsheuristik(q):

S*: Hidtil bedste løsning
S: Midlertidig løsning

```
1:    MaxGenstart = q;
2:    GenstartNo = 0;
3:    while (GenstartNo ≤ MaxGenstart)
4:            S* = KH2();
5:            S = S*;
6:            S = lokalsøgning(S);
7:            if (|S| < |S*|)
8:                {S*=S};
9:            GenstartNo ++;
10:   return S*;
```

Heuristikken er meget simpel, men har i de indledende test vist sig at være klart overlegen til de andre heuristikker, der har været forsøgt. De indledende

forsøg viste også, at det var yderst sjældent, at lokalsøgningen havde nogen indflydelse på resultatetet. De gode løsninger bliver næsten hver gang fundet direkte af KH2. En meget vigtig detalje er, at løsningerne bliver fundet på meget kort tid selv for store problemer. Typisk tager det nogle sekunder alt efter indstillinger. Derfor vil dette blive den heuristik, der arbejdes med fremover til løsning af VGKP.

I det efterfølgende afsnit vil genstartsheuristikken blive testet.

4.8 Test

I dette afsnit laves forskellige tests forbundet med løsningen af VGKP. Først undersøges forskellen mellem optimale IP-løsninger i forhold til LP-relakseringen fundet vha. GAMS. Denne undersøgelse bruges til at lave en kvalitetsvurdering af løsningerne genereret med genstarts heuristikken.

4.8.1 Parametrene

Genstartsheuristikken beskrevet i forrige afsnit er defineret udfra følgende tre parametre:

- Størrelsen på kandidatsættet $|u_f|$ i konstruktionsheuristikken KH2
- Antallet af genstarter i genstartsheuristikken q
- Mængden af knuder betragtet i nabolaget for lokalsøgningen

I første omgang sættes parametrene til en fast værdi. En mere grundig undersøgelse af parametrernes indflydelse på den endelige program er foretaget i 7.2. Parametrene fastsættes til $|u_f| = 10$, $q = 20$ og mængden af knuder betragtet i nabolaget fastsættes til en tredjedel. Parametrerne er fastsat udfra initielle test, som indikerer, at valget af parametre er fornuftigt.

4.8.2 Test problemer

Til udførelsen af de ønskede tests skal der opstilles en række test problemer. Problemerne er en række tilfældige VGKP opstillet som kompatibilitetsgrafer. Til dette formål er der lavet et program, der opstiller tilfældige problemer på en sådan form, at de kan løses direkte i GAMS. Filindlæseren til heuristikken er konstrueret således, at den kan læse problemet direkte ud fra GAMS notation. Programmet har følgende parametre:

- Antallet af knuder i problemet U.
- Antallet af grupperinger i problemet L.
- Procentdel af kompatibilitet mellem to grupperinger med fællesbesøg.

Størrelsen på grupperingerne er konstant og bestemt udfra $\frac{U}{L}$. Procentdel af kompatibilitet mellem to grupperinger med fællesbesøg angiver sandsynligheden for at to ruter u_i, u_j med indbyrdes fællesbesøg, er kompatible.

Det skulle ud fra dette program være muligt at generere tilfældige problemer, der har den samme struktur, som problemerne der ønskes løst. Det kan dog være svært at vurdere, hvor stor procentdel kompatibilitet der skal være imellem to grupperinger med fællesbesøg. I det efterfølgende afsnit er der opstillet tests med forskellige værdier for disse parametre.

4.8.3 LP-relaksering

Det har ikke været muligt at finde litteratur omhandlende VGKP, og det har derfor heller ikke været muligt, at finde datasæt indeholdende problemer og deres hidtil bedste / optimale løsninger.

For at kunne lave kvalitetsvurdering af heuristikker til VGKP er der brug for et sammenligningsgrundlag. Dette kan gøres vha. en nedre grænse for problemet. Denne grænse kan f.eks fastsættes ved at tage de L billigste knuder i problemet eller den billigste knude i hver gruppering. Problemet med denne type nedre grænse er, at der er en risiko for at grænsen befinder sig langt fra den optimale løsning og er derfor ikke altid velegnet til kvalitetsvurdering. Et alternativ kan være at generere optimale løsninger ved brug af GAMS (General Algebraic Modeling System). Disse løsninger kan danne grundlag for en kvalitetsvurdering af de heuristikker, der senere bliver implementeret. Dog er problemet med at finde optimale løsninger til binære problemer (IP-problemer) med GAMS, at det kun kan lade sig gøre for begrænsede problemstørrelser, $U = 600$ og $L = 6$ (se evt. tabel 3).

For denne størrelse problemer kan der findes optimale løsninger ved brug af GAMS. Disse løsninger kan sammenlignes direkte med heuristikkens løsninger. Hvis en heuristik klarer sig dårligt på små problemer, kan det med en vis rimelighed antages, at det ikke bliver bedre for større problemer. Dette gør, at der kan laves en vurdering af, om en ide er værd at arbejde videre med.

For at kunne lave kvalitetsvurdering på større problemer er en anden fremgangsmåde nødvendig. En fremgangsmåde kan være at lave en LP-relaksering. Relakseringen opnås ved at erstatte (18) i den matematiske model med:

$$0 \leq x_u \leq 1$$

Da det er muligt at løse mindre problemer til optimalitet, kan der laves en vurdering af forskellen mellem IP-løsninger og løsningen på det relakserede problem. Hvis forskellen mellem den optimale værdi for IP-løsningen og den optimale værdi for LP-relakseringen er tilstrækkelig lille, kan den optimale værdi for LP-relakseringen bruges som sammenligningsgrundlag til heuristikken i stedet for løsningen til IP problemet.

4.8.4 Test af optimale IP-løsninger vs. LP-relaksering

Som beskrevet i foregående afsnit er det nødvendigt at kunne lave en form for kvalitetsvurdering af heuristikker. I dette afsnit undersøges forholdet mellem de optimale værdier for IP problemet og LP-relaksering, når parametrene for testproblemerne ændres.

For at kunne lave test, som er brugbare i en større sammenhæng, er det nødvendigt at lave en række testsæt, der er tilfældige, men stadig har de samme karakteristika, som en kompatibilitetsgraf for et virkeligt problem besidder. Som beskrevet i afsnit 4.2 er det nødvendigt, at problemet er sammenhængende. Fællesbesøg er derfor sat til to per gruppering og er ordnet på følgende måde: Hvis der er tre grupperinger, gruppering 1, 2 og 3, har gruppering 1 fællesbesøg med gruppering 2, gruppering 2 har fællesbesøg med gruppering 3 og gruppering 3 har fællesbesøg med gruppering 1. Dvs. at grupperingerne kan ordnes således, at en given gruppering altid har fællesbesøg med grupperingen før og efter, og den første gruppering altid har fællesbesøg med den sidste gruppering.

Det er ikke til at vide på forhånd, hvor stor kompatibilitet, der forekommer mellem ruterne for to plejere med fælles patient. Det skal derfor overvejes, hvor stor del af ruterne der har fællesbesøg, som er indbyrdes kompatible Dette vides først, når der er dannet ruter for alle plejere, og kompatibilitetsgrafen er opstillet. Derfor bliver der testet i et stort kompatibilitetsinterval.

Knuder	Kompatibilitet				Grupperinger
U	0.1	0.2	0.3	0.4	L
300	20.40	13.06	11.25	8.58	6
450	29.02	16.03	6.70	6.74	6
600	22.73	10.60	7.32	6.68	6
250	46.36	21.77	11.08	5.58	5
375	22.78	16.54	10.35	6.62	5
500	20.83	8.33	7.44	3.02	5

Tabel 3: *Tabellen viser forskellen i procent mellem IP- og LP-løsninger. For hver problemstørrelse er dette angivet for 10, 20, 30 og 40% kompatibilitet mellem grupperingerne. F.eks. betyder det for en problem størrelse på 600 knuder og 6 grupperinger med 0.3 kompatibilitet er forskellen mellem den optimale IP- og LP-løsning i gennemsnit 7.32 %.*

I tabel 3 er test for forskellige problemstørrelser, antal grupperinger og karakteristika angivet. Tallene er et gennemsnit baseret på tre forskellige problemer af samme type. Dvs. at der f.eks er genereret tre problemer med 600 knuder og seks grupperinger med 30 % kompatibilitet. Tallet givet i tabellen er derfor den gennemsnitlige forskel mellem IP- og LP-løsninger for de tre problemer. Dette er dog med undtagelse af problemer med kompatibilitet 0.1. Her er der ikke tale om et gennemsnit, men forskellen er blot baseret på et enkelt problem. Dette er gjort, fordi kørselstiderne er for lange, og resultaterne for dårlige. F.eks kan det ses i tabel 4, at kørselstiden kan være 136817 sekunder $\approx$ 1.5 døgn for at løse et problem med 600 knuder, seks grupperinger og 0.1 kompatibilitet. Det ses også i tabel 4, at generelt bliver kørselstiden længere, jo mindre kompatibilitet der er i problemet. Der kan dog være undtagelser, hvilket ses i tabel 4 for hhv. 350 og 500 knuder ved 0.2 kompatibilitet (angivet med **fed**). Variationen kan skyldes at GAMS bruger en form for branch and bound algoritme. Kørselstiden er derfor meget afhængig af hvor gode bounds der bliver lavet i starten dette kan variere meget fra probleminstans til probleminstans.

	Knuder U					
Kompatibilitet	300	450	600	250	375	500
0.1	12060	63104	136817	8774	12054	57096
0.2	1310	4069	74242	553	**5501**	**2740**
0.3	492	2190	75305	353	1240	1823
0.4	162	926	13446	91	414	1196
Grupperinger L	6			5		

Tabel 4: *Tabellen angiver gennemsnitskørselstiden for at løse problemerne fra tabel 3 til optimalitet i GAMS.*

Udfra tabel 3 ses det, at jo større kompatiblitet mellem knuderne og jo flere ruter, der er genereret per gruppering, jo mindre forskel bliver der mellem værdien på den optimale IP-løsning i forhold til den optimale af LP-relakseringen. Dette var også som forventet, da der ved større kompatibilitet er der større chance for, at de gode knuder kan inkluderes i løsningen på IP-problemet. Chancen for at LP-relakseringen bryder binaritetsbegrænsningen er derfor knap så stor da flere af de gode knuder kan indkluderes i løsningen.

Der er selvfølgelig nogle små afvigelser, men de vil formentlig forsvinde hvis resultaterne bliver baseret på flere end tre problemer. Dette er dog alt for tidskrævende, og tendensen synes at være rimelig klar allerede ved en gen-

nemsnitlig afvigelse baseret på tre problemer.

Det er tydeligt, at for en kompatibilitet på 0.1 og 0.2 er forskellen meget stor mellem værdien på den optimale IP-løsning og LP-relakseringen. Dette betyder, at for problemer med en kompatibilitet på 0.2 eller mindre kan denne metode ikke bruges som kvalitetsvurdering. Dette behøver ikke nødvendigvis være noget problem, da det ikke er til at vide med sikkerhed, hvor stor kompatibilitet der forekommer mellem ruterne, før det samlede problem er opstillet.

4.8.5 Test af genstartsheuristik mod optimale IP-løsninger

Der er i forrige afsnit fundet optimale løsninger til en række mindre problemer. Problemerne og deres optimale løsninger er blevet gemt. Der er dog kun gemt en af hver problemtype, da problemerne er meget pladskrævende.

I første omgang er det oplagt at teste, hvordan heuristikken klarer sig på disse problemer i forhold til deres optimale løsninger. I tabel 5 er forskellen mellem heuristikkens løsning og den optimale løsning angivet i procent.

knuder	Kompatibilitet				Grupperinger
U	0.1	0.2	0.3	0.4	L
300	2.88	0.00	0.00	0.00	6
450	0.00	4.76	0.00	0.00	6
600	5.26	0.00	0.00	0.00	6
250	0.00	7.78	0.00	0.00	5
375	0.00	5.19	0.00	0.00	5
500	9.10	0.00	0.00	0.00	5

Tabel 5: *Tabellen angiver forskellen mellem heuristikkens løsning og den optimale IP-løsning i procent. Forskellen mellem heuristikkens løsning og den optimale LP-løsning for et problem med 300 knuder, fem grupperinger og 0.1 kompatibilitet, er f.eks 2.88%. Heuristikkens værdier er baseret på et gennemsnit mellem tre kørsler på samme instans.*

Tabellen viser, at heuristikken udelukkende finder optimale løsninger for en kompatibilitet på 0.3 og 0.4. For mindre kompatibilitet begynder det at blive sværere for heuristikken at finde optimale løsninger. Alle løsninger ligger under 10% fra de optimale løsninger. Heuristikken skal gerne kunne bruges for

problemer der er min. 3-4 gange så store, som de anvendte eksempler. Selvom der findes optimale løsninger i mange tilfælde er det ikke overbevisende. Der skal dog hele tiden tages højde for, at kompatibiliteten kan vise sagtens sig at være større end 0.2 når det samlede problem opstilles, og resultaterne for lav kompatibilitet vil derfor ikke have betydning.

Det ser dog ud som, at heuristikkens løsning er meget afhængig af det givne problem. Der er f.eks forskel mellem heuristikkens løsning og den optimale løsning på 4.76 % for 450 knuder, seks grupperinger og en kompatibilitet på 0.2. Hvis man kigger på det tilsvarende problem med en kompatibilitet på 0.1 er forskellen 0 %, hvilket er lidt overraskende. Dette må betyde, at kvaliteten af heuristikkens løsninger også afhænger af probleminstansen, da det formodes, at heuristikken ville klare sig mindst ligeså dårligt på et tilsvarende problem med en kompatibilitet på 0.1.

Forklaringen kan evt. findes i KH2's metode til at danne løsninger: Det antages nu, at der er en række mindre gode knuder, der skal indgå i den optimale løsning. I første iteration er KH2's kandidatsæt for hver gruppering givet ved de $|u_f|$ bedste knuder i grupperingen. Det kan tænkes, at mange af de knuder i den optimale løsning ikke indgår i kandidatsættet for deres gruppering i første iteration, og hvis deres gruppering bliver valgt i første iteration, kan den optimale løsning ikke opnås. Hvis der er tilstrækkelig mange af sådanne knuder, kan det blive svært at finde optimale løsninger.

Sandsynligheden for, at disse knuder indgår i den optimale løsning, bliver større ved lav kompatibilitet. Dette er dog ikke kun afhængigt af kompatibilitet, men også afhængigt af probleminstansen og kan derfor godt være en forklaring på at der findes bedre løsninger for instanser med 0.1 kompatibilitet i forhold tilinstanser med 0.2 kompatibilitet.

Resultaterne angivet i tabel 5 antyder, at heuristikken kunne være god. Specielt hvis problemtypen har kompatibilitet på mere 0.2. En endelig bekræftelse heraf kræver dog test på større problemer.

4.8.6 Test af genstartsheuristik mod optimale LP-løsninger

Ønsket er, at genstartheuristikken skal være i stand til at løse væsentlig større problemer, end der er præsenteret i forrige afsnit. Genstartheuristikken bliver derfor i dette afsnit testet på større problemer og sammenlignet med den optimale løsning på problemets LP-relaksering.

Forskellen mellem den optimale IP-løsning og den optimale LP-løsning er

forholdsvis stor for en kompatibilitet på 0.1 og 0.2 som præsenteret i tabel 3. Det er derfor ikke muligt at sige noget definitivt om kvaliteten af løsninger, og det giver derfor ikke meget mening at sammenligne heuristikken med optimale LP-løsninger for problemer med denne kompatibilitet.

Kompatibilitet	Knuder pr. gruppe			Grupperinger
	50	75	100	L
0.5	13.21	7.77	6.52	15
0.4	12.82	7.37	6.28	15
0.3	13.11	10.29	6.42	15
0.5	9.82	9.13	5.60	20
0.4	16.51	12.37	8.30	20
0.3	12.46	11.90	8.59	20
0.5	11.98	7.79	NA	25
0.4	18.80	13.35	NA	25
0.3	21.02	12.20	NA	25

Tabel 6: *Tabellen angiver forskellen mellem heuristikkens løsning og den optimale LP-løsning angivet i procent. Forskellen mellem heuristikkens løsning og den optimale LP-løsning for et problem med 75 knuder per gruppering, 20 grupperinger og 0.3 kompatibilitet, er f.eks 11.90%. Heuristikkens værdier er baseret på et gennemsnit af tre kørsler. NA i tabellen betyder, at problemet var for stort til at løse i GAMS på trods af relakseringen.*

I tabel 6 er der angivet forskellen mellem heuristikkens løsninger og optimale LP-løsninger. Antallet af grupperinger i problemet er nu forøget betragteligt, og der ses bort fra kompatibilitet mindre end 0.3.

Tabel 6 viser en klar tendens til, at jo flere knuder der er pr. gruppering, jo mindre bliver forskellen mellem LP-løsningen og heuristikkens løsning. Det er på trods af, at det samlede antal knuder i problemet stiger. Denne tendens er dog ikke helt uventet, da dette også var tilfældet for forskellen mellem LP- og IP-løsningerne præsenteret i tabel 3. Tabellen viser ydermere, at forskellen stiger med antallet af grupperinger ved fastholdte gruppestørrelser. Dette er, som forventet, da løsningen bliver mere kompleks, men antallet af knuder per gruppering forbliver den samme. Dvs. at løsnigen forringes med antallet af grupper i problemet, men ikke med gruppestørrelsen.

Det ses også udfra tabel 6, at en forøgelse af gruppestørrelse på 25 knuder tilsynelandende har større indflydelse på løsningen end en forøgelse fra

0.3 til 0.4 kompatibilitet. Dette er en smule overraskende og kan indikere, at heuristikken er mere følsom overfor en ændring i gruppestørrelse end en ændring i kompatibilitet.

I tabel 7 ses gennemsnitskørselstiden for løsningerne fundet med genstartsheuristikken præsenteret i tabel 6 .

	Knuder pr. gruppe		
Grupperinger L	50	75	100
15	2	3	7
20	3	7	13
25	5	12	NA

Tabel 7: *Tabellen angiver tiden i sekunder det tager for heuristikken at opnå løsningerne fra tabel 6. F.eks tager det 5 sekunder at løse et problem med 50 knuder per gruppering og 25 grupperinger. Kørselstiden er uafhængig af kompatibiliteten.*

Det ses, at heuristikken som ønsket finder løsningerne indenfor sekunder, hvilket giver håb om, at det endelige programs kørselstid også kan holdes nede.

Testene præsenteret i dette afsnit kan ikke garantere, at genstartsheuristikken laver gode løsninger. Testene giver dog en god indikation om, at løsningerne kan være gode. Kvaliteten af løsningerne afhænger i høj grad af, hvor langt LP relakseringen er fra den optimale løsning. Hvis resultaterne fra tabel 3 med kompatibilitet på 0.3 og 0.4 tages i betragtning, ses det, at forskellen mellem IP-løsningen og LP-løsningen er ca. 5-10 %. Hvis det antages, at denne forskel ikke bliver mindre for større problemer som præsenteret i tabel 6, kan løsningerne måske være ret tæt den optimale løsning.

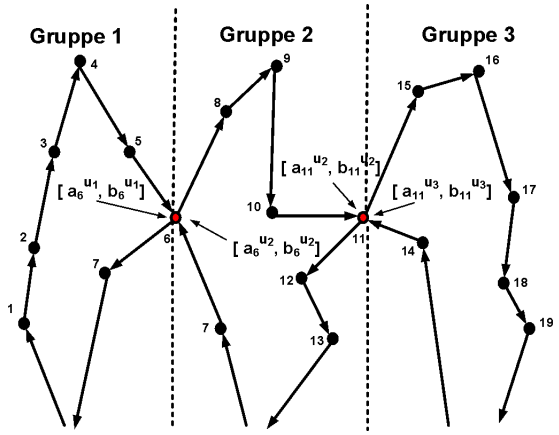
Baseret på testene præsenteret i dette afsnit virker det fornuftigt at arbejde videre med genstartsheuristikken som løsningsværktøj til VGKP i den overordnede løsning.

5 Gyldighedsproblemet

I de forrige afsnit er præsenteret heuristikker til generering af ruter og til løsning af det resulterende VGKP. I dette afsnit præsenteres et samlet program til løsning af problemstillingen beskrevet i afsnit 2. For at finde en samlet løsning for det samlede ruteplanlægningsproblem skal rutesættet fundet ved at løse VGKP undersøges nærmere. Desværre er en gyldig løsning til VGKP ikke nødvendigvis en gyldig løsning på det underlæggende ruteplanlægningsproblem. I det følgende afsnit behandles denne problematik.

5.1 Gyldige Løsninger

En gyldig løsning til VGKP betyder, at alle hjemmeplejere med fællesbesøg har mulighed for at være hos deres respektive fællespatient samtidig. Dette bliver afgjort vha. besøgstidsvinduerne $[a_v^{u_l}, b_v^{u_l}]$, for alle fællesbesøg, beskrevet i afsnit 3.4. Problemet med besøgstidsvinduerne, er at de kan ændre sig, hvis der er mere end ét fællesbesøg på en given rute. Dette er illustreres ved et eksempel med tre grupper angivet i figur 13.



Figur 13: *Figuren viser tre grupper med en tilhørende rute. Der er to fællesbesøg angivet med rødt. Besøgstidsvinduerne er betegnet f.eks.* $[a_6^{u_2}, b_6^{u_2}]$ *hvilket betyder besøgstidsvinduet for fællesbesøget hos borger nummer 6 på ruten i gruppe to.*

En gyldig VGKP-løsning på det ovenstående problem opfylder, at $a_6^{u_1} < b_6^{u_2}$, $a_6^{u_2} < b_6^{u_1}$ og at $a_{11}^{u_3} < b_{11}^{u_2}$, $a_{11}^{u_2} < b_{11}^{u_3}$. Denne løsning tager derimod ikke højde for, at hvis f.eks hjemmeplejer 1 ankommer senere end hjemmeplejer 2 til deres fællesbesøg, skal hjemmeplejer 2 vente på hjemmeplejer 1, før en servicering kan begynde. Fællesbesøgets tidligste serviceringsstart bliver

derved $a_6^{u_2}$ for begge hjemmeplejere. Denne forsinkelse kan forårsage en ændring i alle de tidligste starttidspunkter på resten af ruten og derved også $a_6^{u_2}$. Det kan derfor være, at kravet $a_{11}^{u_3} < b_{11}^{u_2}$, $a_{11}^{u_2} < b_{11}^{u_3}$ ikke længere er opfyldt, og løsningen ikke længere er gyldig. Se evt. eksemplet angivet på figur 5 i afsnit 2

I det følgende afsnit er der opstillet en matematisk model, der kan fastslå, om løsningen til VGKP er en gyldig løsning til det underlæggende ruteplanlægningsproblem eller ej.

5.1.1 Matematisk Model af Gyldigshedsproblemet

Til opstilling af modellen betragtes to mængder; mængden af alle hjemmeplejere L og mængden af alle besøg V. Mængden V_{fb} indføres til at betegne mængden fællesbesøg. Den enkelte plejer l er ud fra løsningen af VGKP tildelt en rute u_l. Ruten består af en ordnet mængde af besøgene i V_l. Ruten er ordnet, så den borger som skal serviceres først, er det første besøg i mængden; borgeren, der skal serviceres nummer to, er det andet besøg i mængden, etc. Ligeledes er L en ordnet mængde, hvor den første plejer i mængden har fællesbesøg med den anden plejer i mængden, og den anden plejer i mængden har fælllesbesøg med den tredje, osv.

Modellen skal tage højde for, at alle besøgenes tidsvinduer $[A_j, B_j]$ bliver overholdt samtidig med, at begge plejere skal være tilstede, før en fællesservicering kan begynde. Tiden, hvor den enkelte plejer ankommer til et besøg, angives T_{lj}, og serviceringstiden plus transporttiden er angivet c_{lj}. Desuden indføres en variabel, der angiver starttidspunktet for en servicering, t_{lj}. Da både T_{lj} og t_{lj} er lineære variable er tilsvarende et lineær model. Modellen kan opstilles som følger:

$$\min \quad \sum_l \sum_j \quad T_{lj} \tag{20}$$

$$T_{lj} \leq B_j \qquad l \in L,\ j \in u_l \tag{21}$$

$$T_{lj} \geq A_j \qquad l \in L,\ j \in u_l \tag{22}$$

$$T_{lj} \geq t_{lj} + c_{lj} \qquad l \in L,\ j \in u_l \tag{23}$$

$$t_{l,j+1} \geq T_{lj} \qquad l \in L,\ j \in u_l \tag{24}$$

$$t_{l,j+1} \geq T_{l+1,j} \qquad l \in L,\ j \in V_{fb} \cap u_l \tag{25}$$

Målfunktionen (20) minimere den samlede ankomsttid for alle besøg i problemet. Begrænsningerne (21) og (22) sikrer, at den enkelte hjemmeplejer overholder alle tidsvinduer for besøgene på sin rute. Begrænsningerne (23) og (24) sikrer, at T bliver opdateret for den næste besøg på ruten. Den sid-

ste begrænsning (25) sikrer sammen med (24), at en servicering først kan starte, når begge hjemmeplejere er til stede og opdaterer tiden for begge plejere derefter.

Løsningen på det ovenstående optimeringsproblem danner et sæt af ankomsttider T for alle hjemmeplejere, der sammen med ruterne udgør en gyldig løsning på det overordnede ruteplanlægningsproblem. Dette sæt af ankomsttider behøver enddda ikke at være den optimale løsning, da ethvert sæt af T'er, der opfylder begrænsningerne (21)-(25), vil være en gyldig løsning til det overordnede ruteplanlægningsproblem. Derfor kan problemet ved at vægte besøgene forskelligt beskrive forskellige målsætninger.

Ankomsttiden hos ekelte borgere med tidlige tidsvinduer kunne f.eks vægtes 100 gange tungere end ankomsttiderne i alle andre tidsvinduer. En stor andel af de tidlige tidsvinduer vil typisk være borgere, der skal bruge hjælp til at komme ud af sengen, og det ville måske være at foretrække, at disse borgere skulle serviceres så tidligt i deres besøgstidsvindue som muligt.

5.1.2 Cplex Concert Technology

Problemet beskrevet i forrige afsnit er et lineært optimeringsproblem og kan derfor løses hurtigt med en LP-solver. Ved brug af Ilog Cplex Concert Technology er det muligt at opstille et optimeringsproblem i et Java program og løse det ved at kalde Cplex-solveren.

Efter problemet er løst, er det muligt at hente information omkring den fundne løsning ind i Java programmet. Denne information kunne f.eks. være, om der findes gyldige løsninger til problemet. På denne måde er det muligt at bestemme, om der findes gyldige løsninger til problemet beskrevet i forrige afsnit på meget kort tid.

Som nævnt tidligere er der ikke nogen garanti for, at den fundne løsning til VGKP er gyldig. Det vil derfor være hensigtsmæssigt at gemme alle gyldige løsninger fundet med genstartsheuristikken under dens søgning og sortere dem efter samlet rutelængde.

Disse løsninger kan derefter undersøges for gyldighed en af gangen, i den prioriterede rækkefølge indtil en gyldig løsning findes. Dette er muligt på grund af, at Cplex Concert Technology gør det muligt at opstille modellen og kalde Cplex-solveren direkte fra programmet. Hvis der ikke findes en gyldig løsning mellem disse VGKP-løsninger kan programmet ikke finde gyldige løsninger på problemet.

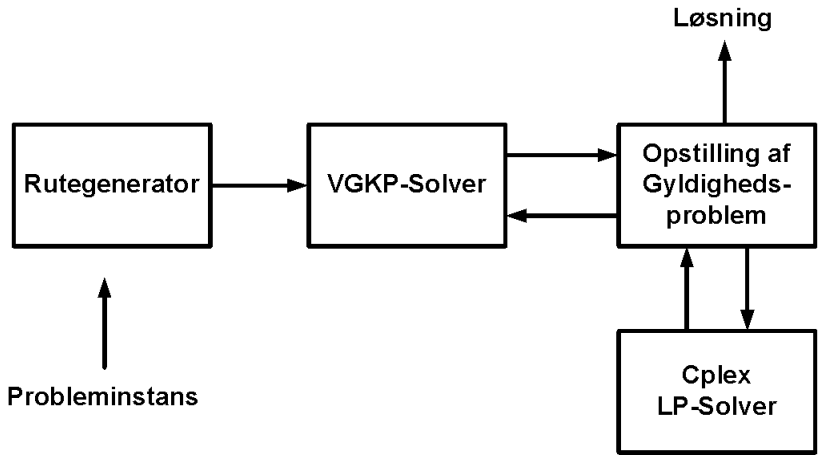
Undersøgelsen af VGKP løsningernes gyldighed foregår ekstremt hurtigt, da der er tale om relativt små lineære programmeringsproblemer. Problemet består af en begrænsning af typerne (21)-(24) for hver borger og to begrænsninger af typen (25) for hver borger med behov for fællesbesøg.

Antallet af variable er lig med antallet af borgere i problemet. Dvs. at størrelsen på problemet stiger lineært med antallet af borgere i problemet. Dette gør, at problemerne kan løses ekstremt hurtigt og derfor ikke påvirker den samlede kørselstid nævneværdigt.

6 Programmet

I de foregående afsnit er der præsenteret løsningsmetoder til forskellige delproblemer. I dette afsnit forklares det, hvordan løsningerne på enkelte delproblemer indgår i et samlet hovedprogram.

Hovedprogrammet er i stand til udfra en række givne grupperinger med fællesbesøg at lave en ruteplan for alle hjemmeplejere, således at alle tidsvinduer og behov for fællesbesøg bliver opfyldt. På figur 14 er det præsenteret, hvordan de forskellige delelementer i programmet hænger sammen.



Figur 14: *En illustration af, hvordan de forskellige delelementer indgår i hovedprogrammet.*

Figuren viser programmets hovedbestanddele. Problemet initieres ved, at data vedrørerende borgernes placering, gruppering, tidsvindue, serviceringstid og behov for fællesbesøg indlæses i rutegeneratoren. Rutegeneratoren finder udfra disse data en række gyldige ruter til hver gruppering. Udfra disse ruter opstilles den såkaldte kompatibilitetsgraf, som beskriver kompatibilitetsforholdet mellem alle ruter.

Kompatibilitetsgrafen overføres til VGKP-solveren, der bruger genstartsheuristikken beskrevet i afsnit 4.7.6 til at finde en løsning bestående af et rutesæt der opfylder VGKP-begrænsninger. Heuristikken støder under sin søgning på en række gyldige løsninger. Disse løsninger lagres til evt. senere brug.

Når genstartsheuristikken er færdig, opstilles optimeringsproblemet beskrevet i afsnit 5.1.1 udfra den fundne løsning. Der tjekkes, om der findes gyldige løsninger til problemet ved at kalde en Cplex-solver. Hvis ikke der findes gyldige løsninger, hentes den næstbedste løsning fundet, og der tjekkes igen for gyldighed. Denne procedure fortsættes, indtil der er fundet en gyldig løsning, eller alle løsninger fundet under genstartheuristikken har været prøvet. Programmet er nedenfor præsenteret i pseudokode:

Hovedprogrammet(PI):

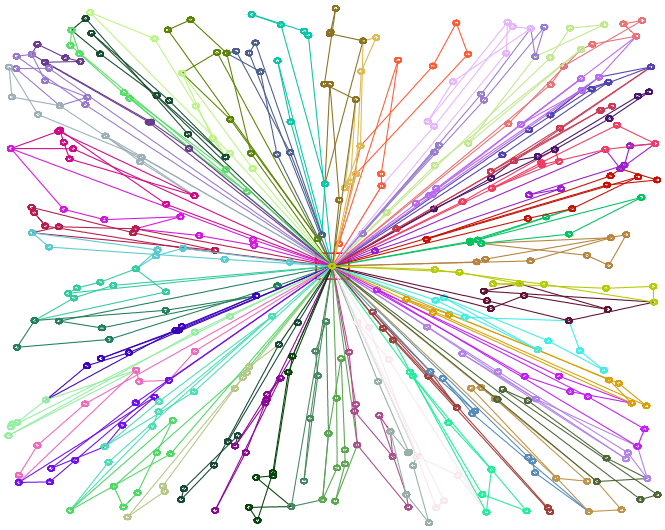
PI: Probleminstans indeholdende alle data for problemet.
LM_S: Løsningsmængde, der indeholder alle fundne løsninger til VGKP.
U: Mængden af ruter.
F: Kompatibilitetsmatrix.
GP: Gyldigshedsproblem.
S: Løsningen på VGKP.

```
1:    U = RuteGenerator(PI)
2:    F = Opstil Kompatibilitetsgrafen(U)
3:    LM_S = Genstartsheuristik(q)
4:
5:    while ( Gyldighed == false && LM_S ≠ ∅)
6:        S = hentLøsning(LM_S)
7:        GP = Opstil GyldigshedsModel(S)
8:        Gyldighed = CplexSolver(GP)
9:        LM_S = LM_S\S
10:   end while
```

Funktionen opstilGyldigshedsModel(s) opstiller modellen præsenteret i afsnit 5.1.1 i Cplex notation. CreateKompatibilitetsgraf(V) opstiller kompatibilitetsgrafen beskrevet i i afsnit 4.2. Pseudokoden til RuteGenerator og Genstartsheuristik er angivet i hhv. afsnit 3.6 og 4.7.6.

Når der er fundet en løsning til VGKP, der overholder gyldighedsbegrænsningerne (21)-(25), returneres de respektive ruter og en løsning til det overordnede ruteplanlægningsproblem er fundet.

Figur 15 illustrerer, et eksempel, på hvordan en sådan løsning kan se ud.



Figur 15: *Figuren viser et eksempel på en løsning for et ruteplanlægnings-problem med 450 besøg. Ruterne er angivet med hver sin farve.*

Figur 15 giver et godt billede af kompleksiteten af problemstillingen. Det ses, at ruterne ofte krydser sig selv. For normalt TSP kan det aldrig betale sig for en rute at krydse sig selv, men når der indkluderes tidsvinduer, kan det ofte være nødvendigt for at opnå gyldige ruter.

7 Tests

Før der kan laves tests af hovedprogrammet præsenteret i forrige afsnit, skal programmets parametre tunes. Når der er fastsat et passende sæt parametre for programmet, kan selve programmet testes på forskellige testinstanser. Til dette formål skal der genereres testinstanser, der skal afspejle et ruteplanlægningsproblem i hjemmeplejen, der er så tæt på virkeligheden som muligt. Der tages derfor udgangspunkt i instanser, der skal repræsentere hjemmeplejen i Søllerød beskrevet i afsnit 2.3. En beskrivelse af instanserne til tuning af parametre og kvalitetstest af hovedprogrammet er gennemgået i det efterfølgende afsnit.

7.1 Testinstanser

Testinstanserne behandlet i dette afsnit tager udgangspunkt i testproblemerne lavet til test af ruteplanlægnings heuristikken beskrevet i afsnit 3.7.

Testdata består af en mængde borgere med behov for besøg, tilfældigt fordelt ud over et givent areal. Dette areal betragtes som et koordinatsystem, og centrum betegner startpunktet for alle hjemmeplejere i problemet. I gennemgang af data er borgerne blevet inddelt i grupper med sweep algoritmen som beskrevet i afsnit 3.7.1, der inddeler borgere efter deres vinkel med den positive x-akse. Efter inddelingen i grupper tildeles hvert besøg et tidsvindue. Der antages at være tre store tidsvinduer ligesom i 3.7. Disse tidsvinduer bliver tildelt ligeligt til alle besøg i hver gruppering så der er stort set lige mange af hver type tidsvindue indenfor hver gruppering.

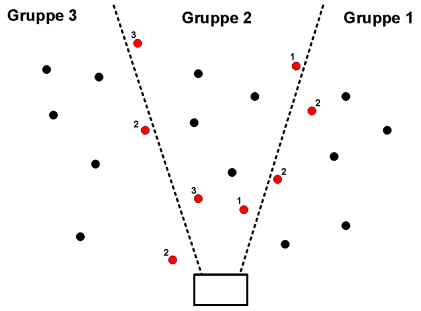
Til hver gruppe er der tilknyttet en hjemmeplejer, der skal besøge alle borgere i gruppen.

I afsnit 3 om ruteplanlægning er de enkelte grupper i testproblemerne uafhængige af hinanden. Grupperingerne kobles nu sammen gennem indbyrdes fællesbesøg. Disse fællesbesøg beskrives i det efterfølgende afsnit.

7.1.1 Fællesbesøg

Til at starte med tilføjes problemet et minimum af fællesbesøg. Alle grupper tildeles ét fællesbesøg med gruppen dannet umiddelbart før og umiddelbart efter med sweep algoritmen (se evt. figur 17). Dette skal sikre, at problemet er sammenhængende og ikke kan splittes op i mindre problemer som beskrevet i afsnit 4.2.

Under inddelingen af grupperingerne tilføjes de to borgere, der er tilføjet først, og de to borgere, der er tilføjet sidst i gruppen, til en mængde af borgere der er potentielle fællesbesøg. De første to tilføjet i mængden er potentielle fællesbesøg med gruppen før, og de sidste to tilføjet til gruppen er potentielle fællesbesøg med gruppen efter. På denne måde kommer fællesbesøg til at blive placeret geografisk tæt på ovengangene mellem grupperne. Det virker intuitivt rigtigt, at et fællesbesøg skal placeres mellem to grupper, og det kunne tænkes, at grupperne blev inddelt efter disse fællesbesøg i virkeligheden. De potentielle fællesbesøg er illustreret på figur 16.

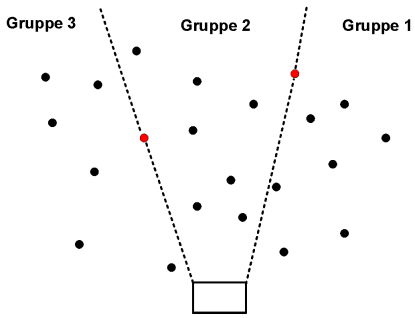


Figur 16: *Figuren viser tre grupper. Punkterne illustrerer borgere. De røde punkter viser potentielle fællesbesøg. Nummeret ved punkterne beskriver, hvilken gruppe de er potentielle fællesbesøg med.*

Figur 16 viser potentielle fællesbesøg. Fællesbesøgene er tilknyttet et nummer, der beskriver, hvilken gruppe de er potentielle fællesbesøg med. Fællesbesøg vælges tilfældigt ud fra disse sæt, således at gruppe 1 har fællesbesøg med gruppe 2, gruppe 2 har fællesbesøg med gruppe 3, osv.

Da fællesbesøgene kan ændre antallet af besøg i en given gruppe, tildeles fællesbesøg således, at hvis en borger i gruppe 2 bliver valgt til at være fællesbesøg med gruppe 1, så bliver en borger fra gruppe 3 valgt til at repræsentere fællesbesøg med gruppe 2. Således bliver hver gruppe forøget med lige mange borgere. Dette gøres for at styre antallet af besøg i de enkelte grupper, således at hjemmeplejeren kan nå alle borgere indenfor en vagt. På figur 17 er problemet fra figur 16 præsenteret efter udvælgelsen af fællesbesøg. Figuren viser, at der nu er tilføjet to fællesbesøg til problemet. En borger fra gruppe 2 er tilføjet til gruppe 1, og en borger fra gruppe 3 er tilføjet til gruppe 2. Tilsvarende tilføjelser laves for alle grupperinger i problemet, således at problemet bliver sammenhængende.

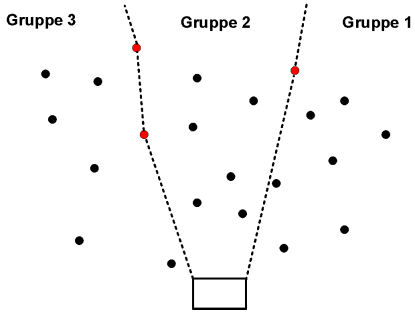
Da inddelingen af grupper sker vha. af en sweep algoritme, begrænses fællesbesøg til de to nabogrupper, der er geografisk beliggende på hver side



Figur 17: *Figuren viser de tre grupper fra figur 16 efter, der er valgt, hvilke borgere der skal have fællesbesøg. Det ses at gruppe 1 og 2 nu har fået et ekstra besøg efter tildeling af fællesbesøg.*

af en given gruppe. Dette betyder, at gruppe 1 fra figur 16 f.eks ikke kan have fællesbesøg med gruppe 3. Et sådan fællesbesøg ville betyde, at en hjemmeplejer fra gruppe 1 eller 3 skulle krydse gruppe 2 for at komme til pågældende fællesbesøg. Dette ville formentlig resultere i en anden gruppeinddeling fremfor, at hjemmeplejerne skulle krydse hinandens ruter for at komme til deres fællesbesøg. Der er derfor kun lavet testproblemer med fællesbesøg for nabogrupper. Alle ekstra fællesbesøg der tildeles problemet tages fra kandidatsættet af fællesbesøg illustreret på figur 16.

Tilføjelserne af fællesbesøg beskrevet hidtil er et minimum af fællesbesøg for et sammenhængende problem. Det kunne dog være interessant med problemtyper, der havde flere fællesbesøg. Det kunne f.eks være interessant, hvis to grupper kunne have mere end en borger til fælles, svarende til to hjemmeplejere, der havde mere end et fællesbesøg på deres ruter. På figur 18 er dette illustreret for det lille eksempel vist på figur 17.



Figur 18: *Figuren viser to fællesbesøg mellem de to samme ruter. Eksemplet er fortsat fra figur 17.*

Disse ekstra fællesbesøg tildeles med en given sandsynlighed β, og efter problemet er blevet tildelt initielle fællesbesøg som illustreret i figur 17. Dvs. sandsynligheden β er sandsynligheden for, at der er et ekstra fællesbesøg mellem to nabogrupper. Dette betyder, at der også kommer en smule variation i gruppestørrelser og antallet af fællesbesøg.

Indsættelse af fællesbesøg betyder at der nu skælnes mellem borgere og besøg, da antallet af besøg nu er forøget med antallet af fællesbesøg, men antallet af borgere i problemet er det samme. Antallet af borgere vil derfor fremover blive betegnet Υ.

7.1.2 Tidsvinduer

Testinstanserne antages af at have samme type tidsvinduer som benyttes i Søllerød Kommune beskrevet i afsnit 2.3. Her benyttes tre store tidsvinduer i umiddelbart forlængelse af hinanden.

Tidsvinduerne bliver tildelt besøgene på samme måde som i 3.7.3, altså inden tildeling af fællesbesøg til de enkelte grupper. Det betyder, at ved tildeling af fællesbesøg kan der opstå grupper med en stor del af besøg indenfor sammen tidsvindue, hvilket kan betyde, at der ikke findes nogen gyldig løsning til testinstanset. Dette kan være svært at styre, men ved at holde en lav hyppighed af ekstra fællesbesøg begrænses sandsynligheden for, at der genereres uløselige testinstanser.

I testinstanserne angiver tidsvinduernes størrelse direkte vagtlængden for plejerne. Dvs. hvis instanserne har tre tidsvinduer på tre timer er den tilsvarende vagtlængden på ni timer. En reduktion i tidsvinduerne til f.eks. 2.5 time vil derfor betyde at alle vagtlængder tilsvarende bliver reduceret til 7.5 time.

Ved at styre forholdet mellem tidsvinduer, servicetid, antallet af fællesbesøg og gruppestørrelser skulle det være muligt at undgå uløselige testinstanser.

7.1.3 Nedre grænse

Til at teste kvaliteten af det fundne rutesæt bruges en nedre grænse på den samlede rutelængde som sammenligningsgrundlag. Den nedre grænse bestemmes ved at finde den optimale rute for hver gruppe. Da det er umuligt at finde bedre ruter indenfor hver gruppering, må summen af de optimale rutelængder for hver gruppering fungere som nedre grænse til testinstanserne.

Til dette formål bruges TSPTW løseren anvendt i afsnit 3.7, der finder optimale TSPTW ved brug af brute force. Herved menes der, at alle permutationer af rutekombinationer genereres, og den korteste gyldige rute angiver den optimale løsning.

Dette sætter naturligvis en grænse på antal af besøg i en gruppering, da kørseltiden stiger eksponentielt med antallet af besøg i problemet. Det er dog muligt indenfor rimelig tid at finde optimale løsninger for gruppestørrelser op til 10 besøg og evt. op til 11. Hvis der er over 11 besøg, tager det over et døgn at finde den optimale løsning, men op til 11 besøg for en plejer kan godt bruges til at beskrive ruteplanlægningsproblemer i hjemmeplejen.

Det er ikke til at sige, hvor stor forskellen er mellem den optimale løsning på problemet og denne nedre grænse, men der er ikke andre oplagte muligheder for kvalitetsvurdering. Problemet med denne type nedre grænse er, at hvis der findes løsninger med værdier langt fra den nedre grænse, er det ikke til at sige, om det er fordi løsningen er dårlig, eller grænsen er dårlig. Hvis forskellen derimod er lille, kan det med sikkerhed siges, at det er gode løsninger.

Denne grænse bruges både til kvalitetsvurdering ved tuning af parametre og ved den endelige test af hovedprogrammet.

7.1.4 Parametre for konstruktion af testinstanser

Testproblemerne genereres udfra parametre, der er med til at bestemme typen af problemet. Parametrene er opstillet nedenfor:

- Antal af borgere i problemet, Υ.
- Størrelsen på grupperne inden tildeling af ekstra fællesbesøg, γ.
- Hyppighed af ekstra fællesbesøg, β.
- Størrelsen på tidsvinduerne, ΔAB.
- Servicetiden, s_v.
- Rejsehastigheden, h.
- Areal borgerne fordeles på, O.

Som det ses, er der mange parametre for testinstanser, hvilket betyder, at der kan genereres rigtig mange forskellige testinstanser. Testinstanser adskiller sig fra instanser præsenteret i afsnit 3.7.4 i to henseende: Den første forskel, er at gruppestørrelserne nu er antallet af borgere tildelt med sweep

algoritmen plus fællesbesøg, der gør problemet sammenhængende, beskrevet tidligere i afsnittet. Den anden forskel, er at der bliver tildelt ekstra fællesbesøg med en hvis sandsynlighed β, der yderligere forøger gruppestørrelsen. Ellers er parametrene de samme som i ruteplanlægningsafsnittet. Arealet og hastigheden er igen konstant sat til hhv. 10 km^2 og 15 km/t.

Tidsvinduernes størrelse er angivet ΔAB. Værdien $\Delta AB = 3$ betyder, at der eksisterer 3 tidsvinduer i intervallerne $[0; 180]$, $[180; 360]$ og $[360; 540]$. $\Delta AB = 2.5$ betyder at tidsvinduerne går fra $[0; 150]$, $[150; 300]$ og $[300; 450]$. Tidsvinduerne starter på denne måde altid på det samme tidspunkt, og tidsvinduerne ligger altid i forlængelse af hinanden. Som beskrevet tidligere angiver tidsvinduernes samlede længde også plejernes vagtlængde i problemet. F.eks. er vagtlængden for $\Delta AB = 3$ ni timer for alle plejere i problemet.

Antallet af plejere og antallet af besøg i problemet er direkte bestemt af parameterne givet ovenfor. Antallet af plejere er pga. sweep algoritmen givet ved

$$L = \lfloor \frac{\Upsilon}{\gamma} \rfloor$$

Dette betyder, at antallet af plejere L er givet udfra antallet af borgere, Υ, og basisstørrelsen på grupperne, γ, og skal derfor ikke opfattes som parameter til problemet.

Antallet af besøg i problemet bestemmes også udfra de givne parametrer ved følgende,

$$|V| \approx \Upsilon + L + \beta \cdot L \tag{26}$$

Antallet af besøg i problemet skal derfor heller ikke opfattes som en parameter til testinstanserne.

7.2 Parametertuning

Hovedprogrammet er afhængigt af en række parameter valg. For at programmet skal være alsidigt og kunne bruges på forskellige problemtyper, er disse parameter nødt til at blive tunet således, at der opnås de bedste gennemsnitsløsninger. I hovedprogrammet er følgende parametre anvendt:

- Størrelsen på kandidatsættet $|u_f|$ i konstruktionsheuristikken KH2.
- Mængden af knuder betragtet i nabolaget for lokalsøgningen.
- Antallet af genstarter i genstartsheuristikken q.
- Antallet af ruter genereret U_l for hver plejer l.

Lokalsøgningen har som nævnt tidligere ikke haft den store indflydelse på løsningen. Det er yderst sjældent at lokalsøgnigen resulterer i bedre løsninger end konstueret med konstruktionsheuristikken KH2. Det betyder ikke at lokalsøgningen er ligegyldig, det betyder bare at den ikke finder de bedste løsninger til VGKP. Lokalsøgningen bidrager stadig til mængden af mulige løsninger på gyldigshedsproblemet. På trods af dette betragtes mængden af knuder betragtet i nabolaget for lokalsøgningen ikke i parametertuningen. Dette gøres da det er utroligt svært at overskue data i fire dimensioner og pga. at de resterende parametre vurderes til at have større indflydelse på løsningen. Efter initielle eksperimenter bliver antallet af knuder betragtet i hver nabosøgning sat til en tredjedel af knuderne i grupperingerne.

Antallet af ruter, U_l, det er nødvendigt at generere med rutegenerationsheuristikken fastlægges ud fra en hårfin balance. Udfordringen er at generere en passende mængde ruter, så der ikke bliver genereret et stort antal ruter, hvoraf mange ruter er totalt ubrugelige. Det vil derfor blive undersøgt hvad et passende antal ruter kan være.

De indledende test har vist, at det er primært i de første par iterationer, at størrelsen på kandidatsættet $|u_f|$ har indflydelse. Efter de første iterationer er der som regel ikke meget mere end et par gyldige kandidater. Det betyder, at valget i de første iterationer er meget afgørende for, hvordan den endelige løsning kommer til at se ud. Kandidatsættet skal derfor være stort nok til, at der kan dannes løsninger til gyldighedsproblemet, men samtidigt være lille nok til, at løsningerne er gode løsninger. Der vil derfor blive testet for store og små kandidatsæt i parametertuningen.

Antallet af genstarter i genstartheuristikken er direkte knyttet til kandidatsættet u_f. Hvis kandidatsættet er meget stort kan kvaliteten af løsninger

forbedres med mængden af genstarter. Hvis kandidatsættet er meget restriktivt kan der evt. nøjes med få genstarter til af finde gode løsninger. En forøget mængde af genstarter kan aldrig forringe løsningen, så antallet af genstarter er kun besgrænset af kørselstiden. Det er derfor interessant at finde et passende antal genstarter der begrænser kørselstider og samtidigt resulterer i både gode og gyldige løsninger.

7.2.1 Instanserne

Til tuningen er der udvalgt seks forskellige problemtyper der skal repræsentere de problemtyper, som danner grundlag for de endelige tests. Valg af parameterværdier til konstruktion af testinstanserne er fastsat med inspiration fra hjemmeplejen i Søllerød kommune beskrevet i afsnit 2.3. Da parametrene i hovedprogrammet kan have forskellig indflydelse på løsningens kvalitet, alt efter problemtype, er det vigtig at udvælge et sæt af forskellige testinstanser. Nedenfor er parametrerne der beskriver de seks testinstanser givet:

Instansnavn	Parametre				
	Υ	β	ΔAB	s_v	γ
TI-50-0.1-2.5-30-7	50	0.1	2.5	30	7
TI-150-0-2.5-30-8	150	0	2.5	30	8
TI-200-0-3-40-7	200	0	3	40	7
TI-250-0-2.5-20-7	250	0	2.5	20	7
TI-300-0.1-3-30-8	300	0.1	3	30	8
TI-400-0.1-3-30-7	400	0.1	3	30	7

Tabel 8: *Testinstanser betegnes f.eks. TI-50-0.1-2.5-30-7, hvilket betegner det første instans givet ovenfor. Tallene i navnet betyder et instans genereret udfra parametrene* $\Upsilon = 50$, $\beta = 0.1$, $\Delta AB = 2.5$, $s_v = 30$ *og* $\gamma = 7$.

Testinstanserne angivet i tabel 11 er som sagt blevet konstrueret med inspiration fra den virkelige verden. Instanset TI-150-0-2.5-30-8 kan f.eks. repræsentere hjemmeplejen i Søllerød kommune. Antallet af besøg i problemet er udfra (26) givet ved:

$$|V| = 150 + \lfloor \frac{150}{8} \rfloor + 0 * (\lfloor \frac{150}{8} \rfloor) = 168$$

Antallet af fællesbesøg tildelt for at gøre problemet sammenhængende er samtidig lig med antallet af grupperinger/plejere. Dette kan beregnes til $\lfloor \frac{150}{8} \rfloor = 18$. Antallet af besøg tildelt hver plejer er i dette tilfælde ni. Dette er givet ved basisgruppestørrelsen $\gamma = 8$ plus et fællesbesøg, der gør problemet sammenhængende (se evt. figur 17). Tidsvinduerne i instanset er sat

til 2.5 time, hvilket skal repræsentere en indskrækelse af tidsvinduerne, på halv time i forhold til tidsvinduerne der tages udgangspunkt i beskrevet fra afsnit 2.3. Servicetiden er sat konstant til 30 minutter. Servicetiden antages konstant, så det er nemmere at styre, at der ikke bliver konstrueret grupper, der ikke kan nås i løbet af en vagt.

Instanset TI-300-0.1-3-30-8 kan repræsentere en evt. sammenlæggelse af hjemmeplejen i to mindre kommuner. I dette problem er antallet af besøg ca. givet ved $300 + \lfloor \frac{300}{8} \rfloor + 0.1 * \left(\lfloor \frac{300}{8} \rfloor \right) \approx 341$. Grunden til, at antallet af besøg ikke længere kan bestemmes eksakt, er, antallet af ekstra fællesbesøg bliver tildelt med vis sandsynlighed β. Dette gør også, at gruppestørrelserne nu kan variere fra ni til elleve besøg. Servicetiden er igen sat konstant til 30 minutter, og tidsvinduerne er sat til udgangspunktet tre timer. Dvs. instanset TI-300-0.1-3-30-8 repræsenterer i realiteten et problem med ca. 341 besøg i gennemsnit, med 37 plejere og en varierende gruppestørrelse mellem ni og elleve besøg.

Grunden til, at γ kun varierer fra syv-otte besøg i basisgruppestørrelse, er, at en basisgruppestørrelse på otte besøg kan for enkelte grupperinger betyde en endelig gruppestørrelse på op til elleve besøg efter tildeling af fællesbesøg. En gruppestørrelse på elleve besøg er på grænsen af, hvad der kan findes optimale TSPTW løsninger med brute force indenfor rimelig tid. For at kunne lave parametertuning er det nødvendigt, at der findes optimale TSPTW løsninger for alle grupperinger, således at der kan findes en nedre grænse, som resultaterne under tuningen kan sammenlignes med. Derfor kan basisgruppestørrelsen ikke sættes højere end $\gamma = 8$.

En basisgruppestørrelse på f.eks. $\gamma = 6$ er modsat ikke specielt interessant, da gruppestørrelserne typisk vil ende på omkring syv besøg, hvilket er lige i underkanten.

Testinstanserne er så vidt muligt valgt ud fra en betragtning af, hvordan virkelige problemer kunne se ud, men valget af gruppestørrelser er desværre begrænset af, at det skal være muligt at generere en nedre grænse til testinstanserne. Dette betyder ikke nødvendigvis, at programmet ikke kan løse testinstanser med større grupperinger, men kun at der er ikke nogen nedre grænse til at vurdere løsningskvaliteten.

7.2.2 Resultater fra tuningen

Under parametertuningen tunes der på tre parametre, u_f, q, og U_l beskrevet tidligere i afsnittet. Intervallet parametrene tunes i, er fundet efter en indledende grovtuning. For at begrænse kørselstiden er intervallet begrænset til

et mindre antal værdier. Værdierne er bestemt efter en indlende grovtuning.

Grovtuningen viste, at det ikke kunne betale sig at generere mere end 70 ruter per gruppering. Dette kan bl.a. skyldes, at for en gruppe med f.eks. otte besøg, hvor tre besøg ligger i første tidsvindue, tre besøg ligger i andet tidsvindue, og to besøg ligger i sidste tidsvindue, er antal permutationer af besøgene begrænset. For dette eksempel er det maksimale antal permutationer givet ved $3! \cdot 3! \cdot 2! = 72$, idet det ikke engang er sikkert at alle permutationer er en gyldig rækkefølge af besøg. Dette kan bl.a. forklare, at der ikke findes bedre løsninger ved at generere mere end 70 ruter per gruppering.

I VGKP afsnittet var kandidatsættet fastsat til $|u_f| = 5$. Det kan være interessant at undesøge om det evt. er for restriktivt og derfor medtages der større værdier i tuningen.

Antallet af genstarter tunes i håb om at finde et minimums antal genstarter, hvor der ikke opnås bedre løsninger ved øge antallet af genstarter. Værdierne er fundet ved den indlende grovtuning.

Parametrene og deres tuningsintervaller er angivet nedenfor.

- $U_l = \{10, 30, 50, 70\}$
- $|u_f| = \{5, 10, 15\}$
- $q = \{20, 25, 30, 35\}$

Der vil i dette afsnit kun blive prænseteret nogle nøgledata for tuningen. Alle data fra tuningen er vedlagt i appendix A.

Da det kan være svært at analysere data i tre dimensioner, er parametertuningen foretaget ved at fastlåse en parameter af gangen. Parametervalget kan derfor godt afhænge af rækkefølgen, hvori parametrenerne fastlåses. Rækkefølgen er i dette afsnit bestemt ved at vurdere hvilken parameter der har størst indfydelse på løsningerne og fastsætte denne parameter først. Dette gentages indtil alle parametre er fastsat.

Resultaterne fra parametertuningen viser tydeligt at antallet af ruter har den største effekt på kvaliteten af løsningen (Se evt. appendix A).

Betragtes antallet af ruter, viser det sig, at i det fleste tilfælde er det bedst at nøjes med at generere 50 ruter. I flere tilfælde forværres løsningen ved at generere flere ruter. I tabel 9 er præsenteret et udsnit af data fra parametertuningen der, illustrerer dette fænomen.

Ruter U_l	Antal genstarter q				$\lvert u_f \rvert$
	20	25	30	35	
10	11.0	10.76	9.45	9.77	5
30	7.26	7.67	8.45	6.12	
50	4.57	**4.57**	4.57	**4.57**	
70	4.57	5.97	4.57	5.34	
10	11.51	11.03	9.77	10.86	10
30	6.04	6.04	6.81	6.04	
50	4.57	**4.57**	*7.0*	4.57	
70	4.57	6.48	4.57	4.57	
10	9.45	11.51	11.51	9.77	15
30	6.04	6.04	6.04	6.32	
50	*4.57*	**4.57**	*6.34*	*4.57*	
70	4.23	6.48	4.57	4.47	

Tabel 9: *Tabellen viser forskellen mellem den nedre grænse beskrevet i 7.1.3 og programmets løsninger angivet i procent. Data er fra TI-150-0-2.5-30-7.*

Det ses, at løsningen bliver markant dårligere i alle tilfælde, hvis der genereres 30 i stedet for 50 ruter. Hvis der til gengæld genereres 70 ruter i stedet for 50, er løsningen stort den samme. Det ses i tabellen, at løsningen faktisk bliver bedre for 50 ruter i fire tilfælde (markeret med **fed**). Løsningen bliver tilsvarende bedre i fire tilfælde for 70 ruter (markeret med *kursiv*), hvor to af disse tilfælde endda kun er marginalt bedre. Dette betyder, at de sidste 20 ruter der genereres, ikke er af den store kvalitet i forhold til at opnå større kompatibilitet mellem grupperingerne og derved bedre VGKP løsninger. En grund kunne, være at de sidste ruter kan være meget lange. På lange ruter bliver besøgstidsvinduerne mindre, og det bliver sværere at opnå en overordnet gyldig løsning ved brug af disse ruter. Hvis ruterne ikke er brugbare, resulterer de i et større VGKP, og det bliver derved vanskeligere at finde de gode løsninger.

Der er en generel tendens for at løsningerne på alle testinstanserne i gennemsnit er ligeså gode for 50 ruter som for 70 ruter (se evt. appendix A). Mængden af ruter bliver derfor sat til 50.

Efter rutemængden for hver plejer U_l er fastsat, bliver det lidt lettere at overskue, hvordan de to sidste parametre påvirker kvaliteten af løsningen. I

tabel 10 er resultaterne for parametertuningen angivet ved 50 ruter for alle testinstanserne.

Antal ruter $U_l = 50$					
	Antal genstarter q				Instans
$\|u_f\|$	20	25	30	35	størrelse Υ
5	1.23	2.41	1.23	1.23	400
10	1.23	1.23	1.23	1.23	
15	1.23	1.23	1.23	1.23	
5	5.3	5.69	4.31	4.71	300
10	6.77	4.77	4.86	4.36	
15	5.87	5.5	4.73	5.4	
5	0.69	0.61	0.66	0.56	250
10	1.06	0.92	0.75	0.55	
15	1.28	2.02	0.94	0.72	
5	5.02	5.82	4.53	4.87	200
10	6.22	5.03	5.02	8.02	
15	6.15	4.92	4.64	4.51	
5	4.57	4.57	4.57	4.57	150
10	4.57	4.57	7.0	4.57	
15	4.57	4.57	6.34	4.57	
5	2.28	2.28	2.28	2.28	50
10	2.28	2.28	2.57	2.28	
15	2.28	2.28	2.28	3.31	
gns.	3.48	3.37	3.28	3.28	

Tabel 10: *Tabellen viser alle data genereret under parametertuningen for 50 ruter. Data er angivet som den procentuelle forskel mellem programmets løsning og den nedre grænse. Nederst er den gennemsnitlige afvigelse angivet for forskellige antal af genstarter. T.h. er størrelsen på de forskellige testinstanser angivet.*

Tabel 10 viser forskellen mellem løsningsværdierne, for forskellige parameterværdier, og den nedre grænse beskrevet i 7.1.3. Tallene er angivet i procentuel afvigelse. Nederst i tabellen er der angivet et gennemsnit over disse afvigelser for de forskellige antal genstarter.

Det ses, at det største gennemsnit findes ved 20 genstarter. Gennemsnittet falder derefter en smule for 25, 30 og 35 genstarter. Tallene indikerer,

at løsninger forbedres ved at forøge antallet af genstarter op til omkring 25 genstarter, hvorefter løsningskvaliteten stagnerer.

Tabellen viser derudover at dette primært gør sig gældende for de lidt strammere testinstanser, nemlig TI-200-0-3-40-6 og TI-300-0.1-3-7-30. I TI-200-0-3-40-6 er servicetiden s_v sat til 40 minutter, hvilket gør det væsentlig vanskeligere at finde kompatible ruter og at finde VGKP løsninger der samtidig er en løsning på gyldighedsproblemet fra afsnit 5.1.1.

TI-300-0.1-3-7-30 er et væsentlig større problem med lidt større gruppestørrelser og flere fællesbesøg. De mange fællesbesøg gør, at det igen er svært at finde VGKP løsninger, der samtidig er en løsning på gyldighedsproblemet. Dette betyder, at der skal lidt flere genstarter til at finde de bedste løsninger end for de andre problemtyper. Det ses også, at værdierne for disse to instanser er dem, der varierer mest for forskellige antal genstarter.

Antallet af genstarter fastsættes på baggrund af tabel 10 til $q = 30$. For overskuelighedens skyld opstilles en ny tabel over parametertuningsdata for $U = 50$ og $q = 30$. Denne tabel er betegnet tabel 11 og er angivet nedenfor.

$U_l = 50$, $q = 30$			
Borgere	$\lvert u_f \rvert$		
Υ	5	10	15
400	1.23	1.23	1.23
300	4.31	4.86	4.73
250	0.66	0.75	0.96
200	4.53	5.02	4.64
150	4.57	7.0	6.34
50	2.28	2.57	2.28
gns.	2.89	3.65	3.27

Tabel 11: *Tabellen viser data genereret under parametertuningen for 50 ruter og 30 genstarter. Data er angivet som den procentuelle forskel mellem programmets løsning og den nedre grænse. Nederst er der beregnet et gennemsnit for hver værdi af u_f.*

I tabel 11 ses det, at det ikke kan betale sig at gøre kandidatsættet større end 5 ruter. Det ses, at ikke nok med at gennemsnittet er lavest for 5 ruter. Der opnås også ligeså gode eller bedre løsninger for hver enkelt testinstans.

Parametersættet $U_l = 50$, $q = 30$ og $|u_f| = 5$ virker derfor som et fornuftigt valg og vil blive benyttet til test af hovedprogrammet.

8 Resultater

Til test af hovedprogrammet benyttes testinstanser som beskrevet i afsnit 7.1. Arealet som borgerne bliver placeret indenfor, O, er igen sat til 10 km^2, og hastigheden, h, fastsættes til 15 km/t. Parametrene i hovedprogrammet fastsættes til værdierne $U = 50$, $q = 30$ og $|u_f| = 5$ fundet i testafsnittet 7.2.

Alle resultater i dette afsnit er angivet som løsningernes procentuelle afvigelse fra den nedre grænse beskrevet i 7.1.3. Afvigelsen er et gennemsnit af løsningerne på tre forskellige testinstanser genereret udfra samme parameterindstilling. Hvis det ikke er muligt at finde en gyldig løsning for en eller flere af disse instanser, markeres det med det tilsvarende antal stjerner (*). Hvis det slet ikke har været muligt at finde en gyldig løsning for testinstanser med given parameterindstilling, markeres dette med en bindestreg (-).

8.1 Resultater for $\Delta AB = 3$ og $s_v = 30$

Som nævnt tidligere er der mange muligheder for at generere forskellige testinstanser. Der vil i dette afsnit blive præsenteret en række resultater for løsninger på et udsnit af alle disse testinstanser. I tabel 12 er angivet resultater for løsninger på instanser med $s_v = 30$ minutter og $\Delta AB = 3$ timer.

Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	2.43	0.64	2.53	1.84	1	3	6	23	0
7	2.14	1.55	3.94	5.31	0	2	9	34	0.1
8	3.81	3.63	3.56	4.15	1	2	10	45	0
8	2.97	3.98	4.50*	7.02	0	2	8*	49	0.1

Tabel 12: *Tabellen angiver t.v. den procentuelle forskel mellem programmets løsning og den nedre grænse beskrevet i afsnit 7.1.3. Forskellen er et gennemsnit over løsninger på tre forskellige instanser med samme parameterindstilling. Antallet af stjerner (*) indikerer hvor mange af de tre instanser, der ikke kunne løses. T.h. er kørselstiden som programmet bruger for at opnå løsningerne, angivet i sekunder .*

Tabel 12 angiver den procentuelle forskel mellem programmets løsning og

den nedre grænse beskrevet i afsnit 7.1.3. Forskellen er som nævnt et gennemsnit over løsninger på tre forskellige instanser med samme parameterindstilling. Dvs. at der bliver genereret tre forskellige instanser for f.eks. parameterindstillingen $s_v = 30$ minutter, $\Delta AB = 3$ timer, $\gamma = 7$, $\Upsilon = 200$ og $\beta = 0.1$. Hvis der bliver genereret et testinstans ved en bestemt parameterindstilling, som programmet ikke kan løse, er det lettere at se, om det evt. kan være et isoleret tilfælde, eller om parameterindstillingen generelt resulterer i testinstanser, som programmet ikke kan løse.

I tabel 12 ses det, at der er en tendens til, at løsningerne forværres, hvis der kommer flere fællesbesøg i problemet. Det ses også, at afstanden til den nedre grænse generelt stiger, jo flere borgere der er i testinstanserne. Løsningerne må betegnes som at være gode, da resultaterne er angivet i forhold til den nedre grænse og ikke i forhold til en optimal løsning. For et enkelt testinstans (markeret med stjernen *) kunne der ikke findes nogen løsninger med programmet. Dette står i kontrast til alle de øvrige løsninger og kan tyde på manglende programstabilitet.

Til højre i tabel 12 er kørselstiden angivet. Kørselstiden er gennemsnitskørselstiden angivet i sekunder for de tre testinstanser. Kørselstiden varierer mellem 0-1 sekunder for de små problemer til 23-49 sekunder for de store problemer. En kørselstid på 49 sekunder for et problem med 400 borgere, hvilket svarer til ca. 455 besøg, må betegnes som en acceptabel kørselstid.

Udfra tabel 12 ser det ud, til at løsningerne bliver dårligere, når antallet af fællesbesøg forøges. Det kunne derfor være interessant at øge sandsynligheden for ekstra fællesbesøg β. I tabel 13 er β forøget til 0.2 for de samme typer testinstanser præsenteret i tabel 12.

Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	0.70	3.48*	2.78*	1.73**	1	2*	6*	29**	0.2
8	3.16	4.38	3.84*	-	0	3	13*	-	0.2

Tabel 13: *Tabellen angiver resultater for instanser $\beta = 0.2$, men ellers samme parametre som i tabel 12. T.v. er den angivet den procentuelle afvigelse fra den nedre grænse, og t.h. er kørselstiden angivet i sekunder.*

Udfra tabel 13 ses det, at ved forøge mængden af ekstra fællesbesøg be-

gynder det at blive sværere for programmet at finde gyldige løsninger for testinstanserne. For testinstansen TI-400-0.2-3-30-8 bliver der f.eks slet ikke fundet nogen gyldig løsning. Ved at forøge sandsynligheden for ekstra fællesbesøg øges sandsynligheden for, at der er tre eller evt. fire fællesbesøg indenfor samme tidsvindue i samme gruppering, hvilket højst sandsynligt vil resultere i, at der ikke er nogen gyldig løsning på problemet.

Det ses til gengæld, at de løsninger, der findes med programmet, stadig er ganske udmærkede, liggende mellem ca. 1-5 % fra den nedre grænse, hvilket må være meget tæt på den optimale løsning. Det samlede problem må dog ikke overstige 200 borgere, før det bliver rigtig vanskeligt for programmet at finde gyldige løsninger. Kørselstiden er stadig acceptabel og under 30 sekunder.

I tabel 12 ses det, at programmet kan finde gode gyldige løsninger for stort set alle af de angivne testinstanser. Det kan være interessant at undersøge, om programmet kunne finde gode gyldige løsninger for større testinstanser. I tabel 14 er det testet for op til 800 borgere.

Afvigelse			Kørselstid		
	Borgere ialt Υ		Borgere ialt Υ		
γ	600	800	600	800	β
7	4.72*	5.31**	76*	135**	0
7	-	-	-	-	0.1
7	-	-	-	-	0.2
8	5.87*	-	76*	-	0
8	3.39**	-	85**	-	0.1
8	-	-	-	-	0.2

Tabel 14: *Tabellen angiver resultater for instanser med op til 800 borgere. T.v. er den angivet den procentuelle afvigelse fra den nedre grænse, og t.h. er kørselstiden angivet i sekunder.*

I tabel 14 ses det, at når mængden af borgere i problemet overstiger 400, begynder det at blive problematisk at finde gyldige løsninger. De gyldige løsninger bliver primært fundet for $\beta = 0$ og 600 borgere. Hvis enten β eller Υ gøres større, bliver det meget svært at finde en gyldig løsning, og for $\beta = 0.2$ bliver der slet ikke fundet nogen gyldig løsning ved disse problemstørrelser. Resultaterne ved disse problemstørrelser skal selvfølgelig ses i lyset, af at programmet ikke er tunet til at arbejde med så store problemer.

Tabel 14 viser, at de løsninger, der bliver fundet, tilsyneladende stadig er gode løsninger. Den højeste kørselstid er 135 sekunder for et problem med 800 borgere og 900 besøg, hvilket må betegnes som en acceptabel kørselstid.

Indtil nu er der kun præsenteret resultater for faste tidsvinduer ΔAB og en fast servicetid s_v. Det kan derfor være interessant at undersøge tidsvinduernes og servicetidens indflydelse på løsningerne.

I de efterfølgende afsnit er der præsenteret resultater for ændret servicetid.

8.2 Resultater for $\Delta AB = 3$ og $s_v = 40$

Hidtil har servicetiden været fastsat til 30 minutter. Det kan tænkes, at der er nogen typer af besøg, der kræver en længere servicetid. Det kan derfor være interessant at se, hvordan løsningen påvirkes, hvis servicetiden forøges. Servicetiden sættes op til 40 minutter i samme type testinstanser som præsenteret i tabel 12. Resultaterne er præsenteret i tabel 15.

Servicetid $s_v = 40$ min.									
Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	1.81	4.52	3.12	4.43	0	1	7	26	0
7	3.62	2.47	3.24	-	0	1	6	-	0.1
8	7.06	4.43	5.15	8.19	0	2	8	38	0
8	4.70	6.65	7.83*	7.28**	0	2	9*	38**	0.1

Tabel 15: *Tabellen viser resultater for de samme instanser som i tabel 12, men med ændret servicetid, $s_v = 40$ min. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Sammenlignes tabel 15 med tabel 12 ses det, at en forøgelse af servicetiden til 40 minutter resulterer i, at programmet får svært ved at finde gyldige løsninger for problemer på 400 borgere og $\beta = 0.1$. Dette er som forventet, da en forøgelse af servicetiden formindsker den overskydne tid indenfor den samlede vagttid. Dette gør, at det bliver mere problematisk at løse problemer med ekstra fællesbesøg tilføjet.

Hvis der ikke medtages ekstra fællesbesøg i problemet ($\beta = 0$), findes der stadig udelukkende gyldige løsninger. Forskellen mellem løsningerne og den nedre grænse er generererelt forøget, men det betyder ikke nødvendigvis, at løsningerne er blevet dårligere. Det kan ligeså godt være pga. at problemet er blevet strammere tidsmæssigt, og det derfor ikke længere er muligt at vælge helt så gode ruter som tidligere.

8.3 Resultater for $\Delta AB = 3$ og $s_v = 20$

Det ses, at ved at sætte servicetiden op til $s_v = 40$ bliver det sværere at finde gyldige løsninger, og løsningerne, der bliver fundet, er tilsynelandende blevet en smule dårligere. Det kan derfor være interessant at se, om det modsatte gør sig gældende ved nedsættelse af servicetiden. Servicetiden nedsættes nu til $s_v = 20$. I tabel 16 er resultaterner angivet for $s_v = 20$.

$s_v = 20$ minutter									
Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	2.56	2.08	2.00	0.93	0	1	6	22	0
7	1.51	4.17	1.78	2.53	0	1	6	28	0.1
8	3.58	3.06	2.94	3.11	0	2	8	33	0
8	3.52	2.36	3.24	4.12	0	2	10	41	0.1

Tabel 16: *Tabellen viser resultater for de samme instanser som i tabel 12, men med servicetiden nedsat til $s_v = 20$ minutter. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Tabel 16 viser, at der udelukkende findes gyldige løsninger for de valgte instanser. Med en maksimal afstand til den nedre grænse på 4.12 % er kvaliteten af løsningerne tilsyneladende ret god.

Da der udelukkende findes gode og gyldige løsninger for $s_v = 20$, kan det være interessant at undersøge, hvor store problemer der kan løses. I tabel 17 er der angivet resultater for instansstørrelser op til 800 borgere.

Tabel 17 viser, at der bliver fundet mindst én løsning i alle tilfælde. Hvis tabel 17 sammelignes med tabel 14, ses det, at der findes en del flere gyldige løsninger for problemer med op til 800 borgere, når servicetiden er reduceret til $s_v = 20$ minutter. Dette skulle også gerne være tilfældet, da der

Afvigelse			Kørselstid		
	Borgere ialt Υ		Borgere ialt Υ		
γ	600	800	600	800	β
7	1.20	1.95	50	88	0
7	3.85	2.22*	71	112*	0.1
8	3.44	2.73	99	161	0
8	3.45*	3.86**	94*	184**	0.1

Tabel 17: *Tabellen angiver resultater for instanser med op til 800 borgere. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

ved en reduktion i servicetid fra 30 min. til 20 min. er mere luft i problemet til evt. at vente ved fællesbesøg og vælge længere ruter hvis nødvendigt.

Det ses, at hvis der ikke tildeles ekstra fællesbesøg til problemet, dvs. $\beta = 0$, bliver der udelukkende fundet gyldige løsninger for problemer med helt op til 800 borgere. Det kan derfor være interessant at undersøge, hvor store problemer der kan løses ved denne lavere servicetid $s_v = 20$ minutter. I tabel 18 er der angivet resultater for helt op 2000 borgere uden ekstra fællesbesøg, $\beta = 0$.

Afvigelse				Kørselstid			
	Borgere ialt Υ			Borgere ialt Υ			
γ	1000	1200	2000	1000	1200	2000	β
7	2.63	1.86	4.65*	146	209	738*	0
8	2.41	2.67	4.90	247	341	1343	0

Tabel 18: *Tabellen angiver resultater for instanser uden ekstra fællesbesøg $\beta = 0$ med op til 2000 borgere. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Det ses udfra tabel 18, at hvis servicetiden er tilpas lav, og der ikke tildeles ekstra fællesbesøg, kan der også løses meget store problemer. Instanserne løst i tabel 18 skal dog tages med et gran salt, da den samlede servicetid for hver plejer er meget lav i forhold til, hvor meget tid den enkelte plejer har til rådighed. Hvis der f.eks. er ni besøg i en gruppering, bruges der ialt 180 minutter på service ud af de 540 minutter, plejeren har til rådighed. Der er

samtidig ikke tildelt nogen ekstra fællesbesøg, så det burde også være muligt at løse store instanser.

Resultaterne er medtaget for at illustrere, hvor meget sværere det bliver at løse instanserne, hvis der medtages ekstra fællesbesøg i probleminstanserne. Det ses i tabel 17, at hvis der medtages ekstra fællesbesøg for $\beta = 0.1$, begynder det at blive problematisk at finde gyldige løsninger ved 800 borgere. Hvis der ikke medtages ekstra fællesbesøg, begynder der først at være problemer med instanser omkring 2000 borgere.

8.4 Resultater for $\Delta AB = 3$, $s_{v_{fb}} = 20$ og $s_v = 40$

Forrige afsnit viste, at ved en nedsættelse af servicetiden til $s_v = 20$ minutter bliver det væsentlig nemmere at finde gyldige løsninger. En servicetid på 20 min. er ikke realistisk for alle typer besøg i hjemmeplejen. Nogle typer besøg kan dog godt ordnes på 20 minutter. Det kan tænkes, at fællesbesøg kan ordnes inden for 20 minutter, hvis besøget f.eks. udelukkende indebærer et løft af en ældre borger. En sådan borger vil formentlig få flere besøg af hjemmeplejen i løbet af dagen. Besøg der ikke krævede to plejere, kunne så gøres længere for at opnå en bedre service. Det kan derfor være interessant at se, hvad der sker med løsninger, hvis servicetiden for fællesbesøg sættes til $s_{v_{fb}} = 20$ samtidig med, at alle andre servicetider sættes til $s_v = 40$. Dette vil betyde lidt mindre tid til fællesbesøgene, men mere tid til enkeltbesøgene. I tabel 19 er resultaterne for disse servicetider angivet.

$s_{fb} = 20$ minutter, $s_v = 40$ minutter									
Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	3.55	2.98	1.48	2.72	0	1	5	23	0
7	2.43	1.39	4.30*	6.62**	0	1	10*	43**	0.1
8	2.80	3.25	4.38	5.88	0	2	10	37	0
8	2.80	3.12	1.59*	2.22**	0	2	8*	40**	0.1

Tabel 19: *Tabellen viser resultater for instanser med ændret servicetid,* $s_v = 40$ *min og* $s_{v_{fb}} = 20$ *min. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Hvis resultaterne i tabel 19 sammnelignes med resultaterne i tabel 15, ses det, at kvaliteten af løsningerne ved at nedsætte servicetiden for fællesbesøg

til $s_{fb} = 20$ min. tilsyneladende er blevet bedre, da afstanden til den nedre grænse er blevet mindre. Det ses til gengæld, at der er de samme problemer med at finde gyldige løsninger, specielt for parametrene $\gamma = 8$, $\Upsilon = 400$ og $\beta = 0.1$.

8.5 Resultater for $\Delta AB = 2.5$ og $s_v = 30$

Tidsvinduerne har hidtil været hold fast på tre timer. Tidsvinduer på tre timer betyder, at der er en stor sandsynlighed for, at en del borgere skal vente i op til tre timer, før de bliver serviceret. En reduktion af tidsvinduerne betyder i dette projekt også en reduktion af arbejdstiden for de enkelte plejere.

En reduktion i tidsvinduerne er derfor interessant både ud fra et service synspunkt og udfra et økonomisk synspunkt.

Det kan derfor være interessant at undersøge, hvordan løsningerne bliver påvirket ved en nedsættelse af størrelsen på tidsvinduet til $\Delta AB = 2.5$ timer. I tabel 20 er resultaterne for denne nedsættelse angivet.

$\Delta AB = 2.5$ timer									
Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	3.38	2.28	3.85	-	0	1	8	-	0
7	3.47	1.59	3.10	2.34**	0	1	5	27**	0.1
8	6.48	3.29	5.15	6.38*	0	2	10	40*	0
8	5.45	7.09*	6.36*	6.55*	0	2*	9*	41*	0.1

Tabel 20: *Tabellen viser resultater for de samme instanser som i tabel 12, men med ændrede tidsvinduer $\Delta AB = 2.5$ timer. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Tabel 20 viser, at ved indskrænkelse af tidsvinduerne bliver det problematisk at finde gyldige løsninger for problemer med 400 borgere. Dette gør sig også gældende for $\beta = 0$ og $\gamma = 7$, hvor der tidligere ikke har været problemer med at finde løsninger.

Det ses, at for $\beta = 0.1$, $\gamma = 8$ og ligeledes 400 borgere er der fundet to gyldige løsninger. Dette kunne godt tilskrives en tilfældighed, da der til gengæld er problemer med at finde gyldige løsninger for problemer helt ned 100

borgere. Ved denne parameterindstilling kan der være op til fire besøg i et tidsvindue, og hvis to af disse besøg er fællesbesøg, bliver det meget svært at finde gyldige løsninger på problemet. Med fire besøg inden for samme tidsvindue er den samlede servicetid på 120 min., og tidsvinduet er 150 min. Der er derfor kun 30 minutter til rejsetid og evt. ventetid hos de enkelt fællesbesøg. Dette kan nemt gøre problemet meget svært eller evt. umuligt at løse.

Kvaliteten af de løsninger, der bliver fundet, ser stadig fornuftig ud, og kørselstiden holder sig også på et pænt niveau.

8.6 Resultater for $\Delta AB = 2.5$, $s_{v_{fb}} = 20$ og $s_v = 30$

Resultaterne i tabel 20 viser at for $\beta = 0.1$ og $\gamma = 8$ er der problemer med at finde gyldige løsninger helt ned til problemer med 100 borgere. Det kunne derfor være interessant at undersøge om reduktion i servicetid til $s_{v_{fb}} = 20$ for fællesbesøgene ville betyde, at det var muligt at finde flere gyldige løsninger. I tabel 21 er resultaterne for $\Delta AB = 2.5$, $s_{v_{fb}} = 20$ og $s_v = 30$ angivet.

$\Delta AB = 2.5$, $s_{v_{fb}} = 20$ og $s_v = 30$									
Afvigelse fra nedre grænse					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
7	3.75	1.65	2.70	2.75*	0	1	5	28*	0
7	3.25	2.63	2.80	2.47**	0	1	8	25**	0.1
8	3.91	2.35	3.64	5.20*	0	2	8	47*	0
8	2.25	3.04	3.64*	5.30*	0	2	8*	38*	0.1

Tabel 21: *Tabellen viser resultater for de samme instanser som i tabel 12, men med ændrede tidsvinduer $\Delta AB = 2.5$ timer og ændret servicetid $s_{v_{fb}} = 20$ og $s_v = 30$. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Tabel 21 viser, at med en reduktion i servicetiden til $s_{v_{fb}} = 20$ bliver der fundet flere gyldige løsninger i forhold til resultaterne givet i tabel 20. Ved parametrene i tabel 21 er der for alle testinstanser op til 200 borgere kun et enkelt instans, der ikke kan findes en gyldig løsning til.

Disse resultater er interessante, da der findes gode gyldige løsninger for problemer med op til 200 borgere på trods af en reduktion i tidsvinduerne. En

problemstørrelse på 200 borgere kan sagtens være repræsentativ for antallet af borgere i en mindre kommune med behov for hjemmehjælp.

8.7 Større grupperinger

Selvom det ikke er muligt at generere en nedre grænse for problemer med større grupperinger, kan det stadigvæk være interessant at undersøge, om programmet kan finde gyldige løsninger, hvis gruppestørrelsen gøres markant større, og servicetiden nedsættes. I tabel 22 er antallet af gyldige løsninger ud af ti kørsler angivet for instanser med forskellige gruppestørrelser.

$\Delta AB = 3$, $s_v = 20$									
Antal gyldige løsninger					Kørselstid				
	Borgere ialt Υ				Borgere ialt Υ				
γ	50	100	200	400	50	100	200	400	β
12	10	10	10	9	1	3	13	61	0
12	10	10	10	8	0	3	13	72	0.1
13	10	10	9	9	1	4	17	67	0
13	10	10	10	9	2	4	18	69	0.1
14	10	10	10	8	1	4	16	65	0
14	10	10	10	6	1	5	15	68	0.1
15	9	10	10	8	0	3	15	61	0
15	10	10	10	6	0	3	15	67	0.1

Tabel 22: *Tabellen angiver antallet af gyldige løsninger fundet for ti forskellige testinstanser ved samme parameter indstilling. Tv. er den procentuelle afvigelse angivet. Th. er kørselstiden angivet i sekunder.*

Tabel 22 viser, at der for problemer med op til 200 borgere næsten altid findes gyldige løsninger. For instanserne på 400 borgere findes der gyldige løsninger for over halvdelen af alle problemerne.

Som beskrevet i 3.7.5 er der en god chance for, at ruterne fundet med rutegenereringsheuristikken bliver dårligere og dårligere ved forøgelse af gruppestørrelsen. På trods af, at alle gyldige løsninger fundet hidtil på instanser med mindre gruppestørrelser har været rigtig gode, kan der ikke siges noget om kvaliteten af løsningerne angivet i tabel 22. Tabellen er medtaget for at illustrere, at der trods alt kan findes løsninger for større gruppestørrelser.

9 Diskussion

I resultatafsnittet er der taget udgangspunkt i testinstanser med parametrerne $\Delta AB = 3$ timer, $s_v = 30$ minutter og resten af parametrerne variende. Resultaterne viste tydeligt, at programmets evne til at finde gyldige løsninger er meget afhængigt af de ekstra fællesbesøg, β, problemet tilføjes. Hvis bare 0.2 af grupperne får tildelt et ekstra fællesbesøg med en nabogruppe, er det svært at finde gyldige løsninger for problemstørrelser indeholdende helt ned til $\gamma = 100$ borgere.

Selv om $\beta = 0.2$ kun forøger mængden af ekstra fællesbesøg med ti fællesbesøg i gennemsnit for et problem med 50 grupper (ca. 400 borgere), bliver kompleksiteten af problemet meget større. Når der bliver tilføjet et ekstra fællesbesøg mellem to nabogrupper, bliver kompatibiliteten mellem de to grupper meget mindre. Der skal ikke længere kun være et overlap i besøgstidsvinduerne for et enkelt fællesbesøg, men for to fællesbesøg. Det kan tænkes, at størrelsen på disse overlap samtidig bliver mindre, hvilket hurtigt kan blive et problem, når gyldighedsproblemet betragtes. Evt. ventetid for plejeren hos et fællesbesøg vil meget hurtigt skabe problemer, hvis der kun er små overlap i besøgstidsvinduerne. Hvis der samtidig tages højde for, at der også er mindst et fællesbesøg med den anden nabogruppe, er det ikke så mærkeligt, at der opstår problemer med at løse problemer efter inkludering af relativt få ekstra fællesbesøg i testinstanserne.

Resultaterne viser, at når servicetiden i testinstanserne øges til $s_v = 40$ minutter, bliver det generelt svært at finde gyldige løsninger for instanser med over 200 borgere. Dette skyldes formentlig at mængden af overskydende tid i instanser bliver mindre når servicetiden sættes op. Hvis servicetiden sættes op for instanser med samme mængde besøg og samme tidsvinduer vil besøgstidsvinduerne blive markant mindre da de direkte er et billede af den mængde overskudstid der eksisterer indenfor det enkelte tidsvindue. Det er derfor ikke underligt at det bliver tilsvarende sværere at finde gyldige løsninger hvis servicetiden sættes op.

Hvis servicetiden nedsættes, bliver det derfor også tilsvarende lettere at finde gyldige løsninger til testinstanserne, da mængden af overskudstid i problemet bliver større. Dette er klart, når der kigges på resultaterne for $s_v = 20$. Det ses endda, at hvis der samtidig ikke tildeles ekstra fællesbesøg, er det muligt at finde gode og gyldige løsninger til instanser med op til omkring 2000 borgere, hvilket må betegnes som meget store problemer.

Problemet forbundet med for lidt overskudstid i instanserne opstår igen, når indskrænkede tidsvinduer, $\Delta AB = 2.5$, timer betragtes. Ved denne ind-

skrænkelse af tidsvinduerne er det kun muligt at løse instanser indeholdende op til ca. 200 borgere. Herefter begynder det at blive problematisk at finde gyldige løsninger. En indskrænkelse af tidsvinduerne har i princippet den samme effekt som en forøgelse servicetiden. Tiden, der skal bruges til at rejse fra besøg til besøg, og tiden plejerne har mulighed for at vente ved de enkelte fællesbesøg, bliver indskrænket. Dette kan også ses direkte i resultaterne. Hvis tabel 15 og 20 sammenlignes, ses det dog, at en forøgelse i servicetiden gør problemet mere følsomt overfor tildeling af ekstra fællesbesøg i forhold til en indskrænkelse af tidsvinduer. Dette gør sig primært gældende for instanser indeholdende 400 borgere.

I resultatafsnittet er der præsenteret resultater for indskrænkede tidsvinduer, $\Delta AB = 2.5$, timer med reduceret servicetid for fællesbesøg, $s_v = 20$ minutter. Der er to grunde til at reducere servicetiden på fællesbesøg. Den første grund er, at hvis disse besøg primært indebærer løft af tungere ældre, burde dette kunne gøres rimelig hurtigt, når først begge plejere er tilstede. Besøg af mere social karakter kan evt. begrænses til enkeltbesøg og kan tildeles en længere servicetid. Den anden grund til at nedsætte servicetiden er, at den samlede servicetid indenfor tidsvinduerne med fællesbesøg generelt bliver mindre. Dette burde give lidt mere tid til, at en plejer kan vente på den anden plejer ved fællesbesøgene. Hvis tabel 20 og tabel 21 sammenlignes, kan effekten af denne reduktion i servicetid ses. Forskellen er ikke overvældende, men det ser ud til, at der nemmere findes gyldige løsninger på større instanser ved en reduktion i servicetid. Derudover indikerer tabellerne, at kvaliteten af løsningerne også forøges ved denne nedsættelse.

Generelt set er alle gyldige løsninger, der bliver fundet med programmet, rigtig gode. De ligger for de mindre komplekse problemer mellem $0 - 5\%$ fra nedre grænse og for de mere komplekse problemer mellem $5 - 8\%$ fra den nedre grænse. Dette må betegnes som rigtig gode løsninger. Grunden til, at løsningerne er så gode, kan være, at de instanser, der findes gyldige løsninger til, ikke er komplekse nok. Det er på den anden side heller ikke til at vide, om de instanser, der ikke kan løses med programmet, er uløselige eller bare for komplekse til programmet. I det hele taget er det et problem, at programmet går fra at finde rigtig gode løsninger til slet ikke at kunne finde gyldige løsninger. Den bratte overgang skaber et stabilitetsproblem i tilfældet af, at programmet skal behandle virkelige problemstillinger der kan variere meget fra kommune til kommune.

Dette er i det hele taget problematikken under generering af fiktive testinstanser. Instanserne skal være så svære at løse, at de udgør en udfordring. Hvis de derimod bliver så komplekse, at der ikke findes gyldige løsninger med programmet, er det så pga. programmets utilstrækkelighed, eller er det pga. instanset slet ikke har gyldige løsninger. Problemet er, at der i dette

projekt er antaget, at gruppeinddelingen er foretaget, men en mere intelligent gruppeindeling ville måske sikre en større stabilitet i forbindelse med at finde gyldige løsninger. Generering af testinstanserne er en balancegang, og i dette projekt er der genereret instanser, der tester programmet til grænsen af dets formåen - og også et stykke over.

Problematikken bliver forstærket af, at ekstra fællesbesøg bliver tilføjet efter gruppeinddelingen er foretaget og efter tilføjelse af fællesbesøgene, der gør problemet sammenhængende. Dette gør, at de grupper med flest fællesbesøg også bliver de grupper med flest besøg ialt. Der kan derfor være en masse grupper med meget overskudstid og få grupper med meget lidt overskudstid. Om problemet er svært/umuligt at løse, er derfor primært domineret af disse få grupper, hvilket også ses, hvis mængden af ekstra fællesbesøg øges. Det kan til gengæld godt tænkes, at de mange mindre grupper med meget overskudstid er netop det, der gør det muligt at have enkelte større grupperinger med ekstra fællesbesøg og samtidig finde gyldige løsninger.

En mere hensigtsmæssig gruppeinddeling ville nok være, at det var de mindre grupper, der blev tildelt flest fællesbesøg, og de større grupper der havde mindre fællesbesøg Men som skrevet tidligere er gruppeinddelingen ikke dette projekts fokus. Gruppeinddelingen og ruteplanlægningen kan dog ikke skilles fuldstændigt ad hvis der skal genereres realistiske og løselige testinstanser.

Det har desværre ikke været muligt at løse ruteplanlægningen for grupperinger større end elleve besøg til optimalitet, da de optimale løsninger findes vha. simpel brute force metode. Dette har lagt en naturlig begrænsning på, hvilke problemer der kan laves nedre grænser for. Denne begrænsning gør, at gruppestørrelser i testinstanserne også er begrænset, hvis de skal sammenlignes med den type nedre grænse.

Problemet er, at ruterne bliver genereret med en form for grådig heuristik. Kvaliteten af løsninger genereret med en grådig heuristik vil formentlig forringes markant, efterhånden som gruppestørrelsen stiger. Kvaliteten af løsningerne, der findes på testinstanser med større grupperinger (se tabel 22), kan derfor risikere at være knap så god som resten løsningerne fundet i resultatafsnittet. Problemet er, at dette kan hverken be- eller afkræftes, og det kan desværre ikke fastslås ved hvilken gruppestørrelse, der opstår evt. problemer.

Det skal nævnes, at for problemer med større grupperinger vil antallet af grupperinger falde og derved vil mængden af fællesbesøg tilsvarende blive reduceret. Da antallet af fællesbesøg er mindre bliver det nemmere at finde et gyldigt rutesæt. På trods af at der evt. ikke bliver genereret lige så gode

ruter ved større grupperinger, er muligheden, for at de bedste ruter der genereres bliver valgt til rutesættet, formentlig større. Det kan derfor være, at ruterne skal blive virkelig dårlige, før det kan ses på løsningskvaliteten.

Betragtes kørselstiden for at opnå løsningerne i resultatafsnittet, må den betegnes som god. For problemer op til 400 borgere ligger alle køretider under et minut. Med denne køretid vil metoden kunne bruges i forbindelse med sidste minuts ændringer i ruteplaner. Dette behov kan opstå, i forbindelse med sygdom eller lignede i plejerstaben, der gør, at ruterne skal ændres, når der mødes ind om morgenen. Kørselstiden for problemer med mellem 600-800 borgere ligger mellem 1-3 minutter hvilket stadig må betegnes som acceptabel, da der er tale om væsentlig større problemer. De større problemer er dog primært taget med for at teste metoden til grænsen, og det skal også tilføjes at der ikke er parametertunet for så store problemer.

Resultaterne viser, at for stort set alle testinstanser indeholdende 100 borgere findes der både gode og gyldige løsninger. Dette er interessant, fordi at et problem indeholdende f.eks. 400 borgere kan med en anden gruppeinddeling, istedet indeles i fire mindre ukoblede problemer, der kan løses hver for sig. Dette kan f.eks. svare til, at et instans med 400 borgere og $\beta = 0$ bliver delt i fire instanser med 100 borgere og $\beta = 0.1$. Grunden til, at andelen af ekstra fællesbesøg, β, er øget i de fire mindre problemer, er, at de fællesbesøg, der koblede grupperingerne i det store problem, nu er tildelt som ekstra fællesbesøg i de fire mindre problemer. På denne måde er der mulighed for ændre problemstrukturen med gruppeinddelingen og derved evt. opnå bedre og/eller gyldige løsninger.

10 Fremtidigt arbejde

Et oplagt sted at fortsætte i forbindelse med dette projekt er at tage hul på gruppeinddelingsproblematikken. Der ligger stadig rigtig meget arbejde i at få inddelt en mængde besøg således, at der tages højde for diverse kurser, ønsker, geografisk placering, type service, tidsvinduer og fællesbesøg. Det oplagte ville være at lave en form dynamisk gruppeinddeling, der kunne justere på grupperne løbende, som er skitseret helt tilbage på figur 1. Dette kan både være i tilfælde af, at der ikke bliver fundet gyldige løsninger under ruteplanlægningen, men også være med henblik på at opnå endnu bedre rutesæt. En metode kan f.eks. være ved at vurdere, hvor meget overskudstid der er i de enkelte grupperinger og flytte besøg fra grupper med lidt overskudstid til nabogrupper med mere overskudstid. En sådan metode er selvfølgelig meget afhænging af lav kørselstid både for ruteplanlægningsdelen og gruppeinddelingsdelen. Med en ruteplanlægningsmetode, som præsenteret i dette projekt, vil kørselstiden for problemer med over 200 borgere formentlig blive for stor, hvis ruteplanlægningen skal gentages mange gange.

Det kan være interessant at kigge på muligheder, når der ikke kan findes nogen gyldige løsninger med hovedprogrammet. Dette kan f.eks. være en metode, der lokaliserer, hvilke grupperinger der er flaskehalse i problemet, og genererer nye ruter for disse grupperinger. De nye ruter skal derefter, hvis muligt, blive påtvunget løsningerne fundet med genstartsheuristikken. Disse flaskehalse kan evt. lokaliseres ved at undersøge, hvor stor andel kompatibilitet de enkelte grupperinger har i det vægtede grupperede klike problem. Grupperinger med lav kompatibilitet kunne meget nemt være, hvor flaskehalsene er lokaliseret. En sådan metode kan evt. kombineres med en dynamisk gruppeindeling forslået ovenfor.

Der ligger stadig et stort stykke arbejde i at generere realistiske testinstanser med tilhørende optimale løsninger eller andre sammenligningsgrundlag. Dette kan evt. omgås ved at lave projektet i samarbejde med en kommune eller virksomhed, der kan stille data til rådighed. På denne måde kan en eksisterende ruteplan eller lignende bruges som sammenligningsgrundlag for løsninger fundet i løbet af projektet.

11 Konklusion

I dette projekt er problematikken forbundet med ruteplanlægning i hjemmeplejen blevet betragtet. Projektet tager udgangspunkt i en to-fasedeling af problemet, hvor gruppeinddeling og ruteplanlægning af besøg er blevet behandlet separat. I denne to-fasedeling er der blevet fokuseret på ruteplanlægningsdelen, da denne del lettere kan behandles matematisk. Ruteplanlægningsdelen blev delt op i tre delproblemer, og hvert delproblem er blevet behandlet hver for sig.

Til planlægning af ruter er der blevet udviklet en metode til at generere forskelligartede ruter for små TSPTW. Test af ruterne viste, at kvaliteten vurderet på rutelængde var god indenfor de gruppestørrelser, der kunne laves nedre grænser til. Metodens evne til at generere forskelligartede ruter er i denne forbindelse også blevet illustreret.

Et vægtet grupperet klike problem er blevet opstillet til at finde gyldige rutesæt, der minimere den samlede rutelængde. Problemet er derefter opstillet som en matematisk model. Modellen og dens LP-relaksering er vha. GAMS blevet løst til optimalitet. En genstartsheuristik er blevet udviklet til hurtigt at finde løsninger til det vægtede grupperede klike problem. Heuristikkens løsninger er blevet vurderet i forhold til løsningerne fundet med GAMS.

Til at teste gyldigheden af rutesættet, når der tages højde for evt. ventetid ved fællesbesøg, er der blevet opstillet et gyldighedsproblem. Gyldighedsproblemet er opstillet som et lineært programmeringsproblem og er blevet løst ved at kalde en Cplex-solver. Et uløseligt gyldighedsproblem er blevet behandlet ved at prøve nye rutesæt fundet med genstartsheuristikken under dens tidligere søgning.

Det er i dette projekt blevet påvist, at en række TSPTW problemer koblet gennem fællesbesøg kan løses ved først at finde en mængde ruter for hver TSPTW og dernæst udvælge en rute for hver problem, således at der findes et rutesæt, hvor tidsvinduerne for fællesbesøgene overholdes. Løsningerne er fundet under forudsætning af, at der er tilstrækkelig overskudstid i de enkelte TSPTW, hvilket bla. de store tidsvinduer, der typisk arbejdes med i hjemmeplejen, kan sikre.

Vha. den optimale TSPTW løsning for hver gruppering er der blevet lavet en nedre grænse for testinstanserne anvendt i dette projekt. Alle gyldige løsninger fundet med hovedprogrammet ligger mellem 0-10 % og heraf de fleste mellem 0-5 % fra denne nedre grænse. Løsningerne bliver fundet inden-

for en fornuftig kørselstid og programmet kan derfor bruges i sammenhæng med at planlægge nye ruter fra dag til dag.

Udfra resultaterne kan der konkluderes, at en to-fasedeling af problemet virker yderst interessant, og en metode, der kombinerer de to faser, bestemt er værd at arbejde videre med.

Dato Clement Lessel

A Data fra parametertuning

A.1 TI-400-0.1-7-30-3

Ruter	Restarts			
	20	25	30	35
10	6.1	6.1	6.1	6.1
30	1.39	1.39	2.0	2.99
50	1.23	2.41	1.23	1.23
70	1.2	2.66	3.45	2.52

Tabel 23: *Størrelse på kandidatsættet* $u_f = 5$

Ruter	Restarts			
	20	25	30	35
10	8.47	8.77	6.1	6.14
30	1.39	1.39	3.08	1.39
50	1.23	1.23	1.23	1.23
70	1.2	1.2	3.0	1.2

Tabel 24: *Størrelse på kandidatsættet* $u_f = 10$

Ruter	Restarts			
	20	25	30	35
10	6.25	6.25	6.1	6.1
30	1.39	3.38	3.18	1.39
50	1.23	1.23	1.23	1.23
70	1.2	1.2	2.76	2.53

Tabel 25: *Størrelse på kandidatsættet* $u_f = 15$

A.2 TI-300-0.1-8-30-3

Ruter	Restarts			
	20	25	30	35
10	8.91	8.91	9.63	8.91
30	5.36	5.36	5.72	5.36
50	5.3	5.69	4.31	4.71
70	5.53	6.44	4.42	4.65

Tabel 26: *Størrelse på kandidatsættet* $u_f = 5$

Ruter	Restarts			
	20	25	30	35
10	10.01	9.87	8.91	8.91
30	5.36	5.36	5.36	5.36
50	6.77	4.77	4.86	4.36
70	6.38	4.66	5.0	5.45

Tabel 27: *Størrelse på kandidatsættet* $u_f = 10$

Ruter	Restarts			
	20	25	30	35
10	10.02	8.91	8.91	8.91
30	5.36	5.36	6.38	5.36
50	5.87	5.5	4.73	5.4
70	5.69	4.92	6.47	4.97

Tabel 28: *Størrelse på kandidatsættet* $u_f = 15$

A.3 TI-250-0-7-20-2.5

Ruter	Restarts			
	20	25	30	35
10	2.57	3.63	2.57	2.57
30	0.85	1.05	0.43	0.88
50	0.69	0.61	0.66	0.56
70	1.02	0.7	0.7	0.82

Tabel 29: *Størrelse på kandidatsættet* $u_f = 5$

Ruter	Restarts			
	20	25	30	35
10	2.57	2.57	2.57	2.57
30	0.95	1.52	1.02	1.8
50	1.06	0.92	0.75	0.55
70	1.28	2.02	0.94	0.72

Tabel 30: *Størrelse på kandidatsættet* $u_f = 10$

Ruter	Restarts			
	20	25	30	35
10	2.63	2.57	2.57	2.57
30	1.31	0.86	2.46	0.85
50	0.92	1.09	0.96	1.33
70	0.6	0.99	1.05	1.21

Tabel 31: *Størrelse på kandidatsættet* $u_f = 15$

A.4 TI-200-0-7-40-3

Ruter	Restarts			
	20	25	30	35
10	-	-	10.39	10.39
30	4.43	5.03	5.0	6.31
50	5.02	5.82	4.53	4.87
70	6.1	5.82	5.16	-

Tabel 32: *Størrelse på kandidatsættet* $u_f = 5$

Ruter	Restarts			
	20	25	30	35
10	9.46	12.65	10.98	-
30	4.43	4.43	-	4.6
50	6.22	5.03	5.02	8.02
70	5.83	5.52	5.02	7.29

Tabel 33: *Størrelse på kandidatsættet* $u_f = 10$

Ruter	Restarts			
	20	25	30	35
10	-	-	-	-
30	4.43	6.81	4.43	5.96
50	6.15	4.92	4.64	4.51
70	6.32	6.21	6.74	6.18

Tabel 34: *Størrelse på kandidatsættet* $u_f = 15$

A.5 TI-150-0-8-30-2.5

Ruter	Restarts			
	20	25	30	35
10	11.0	10.76	9.45	9.77
30	7.26	7.67	8.45	6.12
50	4.57	4.57	4.57	4.57
70	4.57	5.97	4.57	5.34

Tabel 35: *Størrelse på kandidatsættet* $u_f = 5$

Ruter	Restarts			
	20	25	30	35
10	11.51	11.03	9.77	10.86
30	6.04	6.04	6.81	6.04
50	4.57	4.57	7.0	4.57
70	4.57	6.48	4.57	4.57

Tabel 36: *Størrelse på kandidatsættet* $u_f = 10$

Ruter	Restarts			
	20	25	30	35
10	9.45	11.51	11.51	9.77
30	6.04	6.04	6.04	6.32
50	4.57	4.57	6.34	4.57
70	4.23	6.48	4.57	4.47

Tabel 37: *Størrelse på kandidatsættet* $u_f = 15$

A.6 TI-50-0.1-7-30-2.5

	Restarts			
Ruter	20	25	30	35
10	6.9	6.9	6.9	6.9
30	3.31	3.93	3.31	3.31
50	2.28	2.28	2.28	2.28
70	2.28	2.28	2.28	2.28

Tabel 38: *Størrelse på kandidatsættet* $u_f = 5$

	Restarts			
Ruter	20	25	30	35
10	6.11	6.9	6.11	6.87
30	3.31	3.31	3.31	4.55
50	2.28	2.28	2.57	2.28
70	2.28	2.28	2.28	2.28

Tabel 39: *Størrelse på kandidatsættet* $u_f = 10$

	Restarts			
Ruter	20	25	30	35
10	6.95	6.9	6.9	6.9
30	3.31	3.31	4.41	3.31
50	2.28	2.28	2.28	3.31
70	2.28	2.28	2.28	2.28

Tabel 40: *Størrelse på kandidatsættet* $u_f = 15$

B Data fra test af hovedprogrammet

B.1 $\Delta AB = 3$ timer, $s_v = 30$ min

γ	Υ						β
	50	100	200	400	600	800	
6	1.15	0.73	1.70	0.64	2.42	2.56	0
6	0.27	1.06	2.11	2.82*	2.18**	-	0.1
6	2.58	1.36	2.06	2.54*	-	-	0.2
7	2.43	0.64	2.53	1.84	4.72*	5.31**	0
7	2.14	1.55	3.94	5.31	-	-	0.1
7	0.70	3.48*	2.78*	1.73**	-	-	0.2
8	3.81	3.63	3.56	4.15	5.87*	-	0
8	2.97	3.98	4.50*	7.02	3.39**	-	0.1
8	3.16	4.38	3.84*	-	-	-	0.2

Tabel 41: *Forskel mellem løsning og nedre grænse*

γ	Υ						β
	50	100	200	400	600	800	
6	0	0	3	12	42	65	0
6	0	0	3	12*	44**	-	0.1
6	0	2	5	20*	-	-	0.2
7	1	3	6	23	76*	135**	0
7	0	2	9	34	-	-	0.1
7	1	2*	6*	29**	-	-	0.2
8	1	2	10	45	76*	-	0
8	0	2	8*	49	85**	-	0.1
8	0	3	13*	-	-	-	0.2

Tabel 42: *Køretid*

B.2 $\Delta AB = 3$ timer, $s_v = 40$ min

γ	Υ 50	100	200	400	600	β
7	1.81	4.52	3.12	4.43	6.20**	0
7	3.62	2.47	3.24	-	-	0.1
8	7.06	4.43	5.15	8.19	6.39**	0
8	4.70	6.65	7.83*	7.28**	10.24**	0.1

Tabel 43: *Forskel mellem løsning og nedre grænse*

γ	Υ 50	100	200	400	600	β
7	0	1	7	26	76**	0
7	0	1	6	-	-	0.1
8	0	2	8	38	90**	0
8	0	2	9*	38**	99**	0.1

Tabel 44: *Køretid*

B.3 $\Delta AB = 3$ timer, $s_v = 20$ min

	Υ						
γ	50	100	200	400	600	800	β
7	2.56	2.08	2.00	0.93	1.20	1.95	0
7	1.51	4.17	1.78	2.53	3.85	2.22*	0.1
8	3.58	3.06	2.94	3.11	3.44	2.73	0
8	3.52	2.36	3.24	4.12	3.45*	3.86**	0.1

Tabel 45: *Forskel mellem løsning og nedre grænse*

	Υ						
γ	50	100	200	400	600	800	β
7	0	1	6	22	50	88	0
7	0	1	6	28	71	112*	0.1
8	0	2	8	33	99	161	0
8	0	2	10	41	94*	184**	0.1

Tabel 46: *Køretid*

	Afvigelse			Kørselstid			
	Borgere ialt Υ			Borgere ialt Υ			
γ	1000	1200	2000	1000	1200	2000	β
7	2.63	1.86	4.65*	146	209	738*	0
8	2.41	2.67	4.90	247	341	1343	0

Tabel 47: *Tabellen angiver resultater for instanser uden ekstra fællesbesøg* $\beta = 0$ *med op til 2000 borgere. Der er både angivet afvigelse og køretid.*

B.4 $\Delta AB = 3$ timer, $s_{fb} = 20$ min og $s_v = 40$ min

γ	Υ						β
	50	100	200	400	600	800	
7	3.55	2.98	1.48	2.72	3.25	-	0
7	2.43	1.39	4.30*	6.62**	4.07**	-	0.1
8	2.80	3.25	4.38	5.88	-	4.38**	0
8	2.80	3.12	1.59*	2.22**	6.03**	-	0.1

Tabel 48: *Forskel mellem løsning og nedre grænse*

γ	Υ						β
	50	100	200	400	600	800	
7	0	1	5	23	58	-	0
7	0	1	10*	43**	61**	-	0.1
8	0	2	10	37	-	142**	0
8	0	2	8*	40**	89**	-	0.1

Tabel 49: *Køretid*

B.5 $\Delta AB = 2.5$ timer, $s_v = 30$ min

	Υ					
γ	50	100	200	400	600	β
7	3.38	2.28	3.85	-	5.67**	0
7	3.47	1.59	3.10	2.34**	-	0.1
8	6.48	3.29	5.15	6.38*	-	0
8	5.45	7.09*	6.36*	6.55*	7.51**	0.1

Tabel 50: *Forskel mellem løsning og nedre grænse*

	Υ					
γ	50	100	200	400	600	β
7	0	1	8	-	66**	0
7	0	1	5	27**	-	0.1
8	0	2	10	40*	-	0
8	0	2*	9*	41*	89**	0.1

Tabel 51: *Køretid*

B.6 $\Delta AB = 2.5$ timer, $s_{v_{fb}} = 20$ min og $s_v = 30$ min

γ	Υ 50	100	200	400	β
7	3.75	1.65	2.70	2.75*	0
7	3.25	2.63	2.80	2.47**	0.1
8	3.91	2.35	10.93	5.20*	0
8	2.25	3.04	3.64*	5.30*	0.1

Tabel 52: *Forskel mellem løsning og nedre grænse*

γ	Υ 50	100	200	400	β
7	0	1	5	28*	0
7	0	1	8	25**	0.1
8	0	2	8	47*	0
8	0	2	8*	38*	0.1

Tabel 53: *Køretid*

Litteratur

[1] Laurence A. Wolsey, *Integer Programming.*

[2] A. M. Campbell, M. Savelsbergh, *Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems* Transportation science vol. 38, no. 3, 2004.

[3] P. Eveborn, P. Flisberg, M. Rønnqvist, *LAPS CARE - an operational system for staff planning of home care* European Journal of Operational Research vol. 171, 2006.

[4] E. Cheng, J. L. Rich, *A Home Health Care Routing and Scheduling Problem* Department of Mathematical Sciences Oakland University (Ikke udgivet), 1998

[5] S. Bertels, T. Fahle, *A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem* Computers and Operations Research vol. 33, 2006

[6] A. Grosso, M. Locatelli, F. D. Croce, *Combining Swaps and Node Weights in a Adaptive Greedy Approach for the Maximal Clique Problem* Journal of Heuristics, vol. 10, 2004

[7] M. Gendreau, P. Soriano, L. Salvail, *Sloving the maximal clique problem using a tabu search approach* Annals of Operations Reasearch, vol. 41, 1993

[8] A. M. Campbell, M. Savelsbergh *Efficient Insertion Heuristics for Vehicle and Scheduling Problems* Transportation Science, vol. 38, No.3, 2004

[9] C. Voudouris, E. Tsang *Guided local search and its application to the traveling salesman problem* Annals of Operations Reasearch, vol. 41, 1993

[10] K. Thomsen *Optimization on Home Care* IMM-M.Sc-2006-26

[11] S. V. Begur, D. M. Miller, J. R. Weaver *An Integrated Spatial DSS for Scheduling and Routing Home-Health-Care Nurses* Interfaces, vol. 27, no. 4, 1997

[12] Y. Li, A Lim, B. Rodrigues *Manpower Allocation with Time Windows and Job-Teaming Constraints* Wiley InterScience, www.interscience.wiley.com

[13] M. Gendreau, A. Hertz, G. Laporte, M. Stan *A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows* Operations Research, Vol. 46, No. 3 1998

[14] J. W. Ohlmann, B. W. Thomas *A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows* Informs journal on computing Vol. 19, No. 1, 2007

[15] H. Jula, M. Dessouky, P. Ioannou, R. Hall *Full-Truck-Lóad Assignment and Route Planning in Deterministic and Stochastic Environments* (Ikke udgivet)