Den handelsrejsendes problem: Teori og Heuristikker.

Sixteenth-century peddler.
Woodcut by J. Amman

Lise Møller, 11. august 1994.
Institut for Matematik og Datalogi.
Odense Universitet.

0.1 Tak for ...

Er sjælens møje tung og striden lang,
hør! fra det fjerne toner sejerssang!
Da vandrer styrket vi vor pilgrimsgang.
Halleluja, halleluja!
(DDS 659)

Oh, I get by with a little help from my friends.
Oh, I get high with a little help from my friends.
Mm, I'm gonna try with a little help from my friends.
(The Beatles)

Tak er kun et fattigt ord, som jeg alligevel må nøjes med at bruge, selv om der er nogle, der fortjener meget mere. Det er nemlig i høj grad mine venners fortjeneste, at jeg overhovedet overvandt mine problemer undervejs og nåede til vejs ende. Specielt nævnes derfor i en slags kronologisk orden de følgende.

Tak til Jens Knudsen, der præsenterede mig for min første computer, og lærte mig at elske faget. Og desuden lærte mig at bruge hovedet i stedet for benene.

Tak til Paw Hermansen for at være supertutor og -instruktor, og generelt være enig i at computere er sagen.

Tak til Edmund Christiansen for ikke at lade det gå ud over mig, at studienævnet gik dig imod, da jeg absolut skulle indvies i DM01.

Tak til Søren Larsen for altid at holde døren til dit kontor åben, også for andet end studievejledning. Og fordi Paris' metroer aldrig bliver helt det samme igen!

Tak til Bjarne Toft for at introducere mig for TSP så overbevisende, at jeg stadig kan huske det.

Tak til Jørgen Bang-Jensen for spændende undervisning, og for at minde mig om, at TSP også kunne være spændende at skrive speciale om. Du havde ret!

Tak til Sergey Merkulov, der oversatte "den russiske kilde" for mig.

Tak til Jens Clausen for at være mere end bare censor.

Tak til min mand, min familie, mine venner af alle de sædvanlige årsager. Foruden korrekturlæsning.

0.2 Indledning.

Et af de mest fascinerende graf-teori problemer er Den Handelsrejsendes Problem (Traveling Salesman Problem / TSP). Fascinerende for teoretikere fordi en polynomiel algoritme til løsning af det generelle problem ville medføre polynomielle algoritmer for mange andre problemer. Og fascinerende fordi en sådan polynomiel algoritme efter over 50 år ikke er blevet fundet endnu, så man i praksis må give afkald enten på hurtighed (ved at anvende algoritmer med exponetiel køretid) eller på kvalitet (ved at anvende heuristikker, der giver sub-optimale løsninger). Så både i teori og praksis er der blevet foretaget megen forskning på området.

I dette speciale belyser jeg både teori og praksis med eksempler. Mit udgangspunkt er de specielle polynomielle algoritmer. Specielt har jeg taget mange algoritmer med, der kan finde gode eller optimale løsninger til specialtilfælde af TSP. To af de nyere heuristikker (simuleret udglødning og den genetiske algoritme) har jeg implementeret og testet, og sammenlignet med andre algoritmer (bl.a. MST-algoritmen).

Kapitel 1 ser først på TSP's historie og dets relation til andre nærtbeslægtede problemer, både praktiske og grafteoretiske. Da det er vigtigt at kunne vurdere en algoritme, der giver sub-optimale løsninger, gennemgås og demonstreres analysemetoder, bl.a. på algoritmer for specialtilfælde.

Kapitel 2 gennemgår en del algoritmer, der giver gode resultater, men enten ikke kan garantere optimalitet, eller kræver ret specielle forhold for problemet.

Kapitel 3 gennemgår kort historien for og teorien bag simuleret udglødning, og viser så resultater af forskellige forsøg med algoritmen. Simuleret udglødning bygger på en teori, der i starten udelukkende blev brugt på størknende metaller og lignende. Først senere blev algoritmen også brugt på helt andre ting, og jeg viser både den generelle simulerede udglødning, og dens tilpasning til TSP.

I kapitel 4 ses den genetiske algoritme for TSP. Den genetiske algoritme bygger på, at hvis befolkninger kan producere bedre individer for hver generation, ved tilfældig avling, kan man måske også opbygge befolkninger af løsninger og så lade dem "avle" indbyrdes, forhåbentlig med gode løsninger til følge. Jeg viser kort den generelle genetiske algoritme, og så dens tilpasning til TSP. Herefter følger forsøg med algoritmen.

Jeg oversætter generelt alle engelske udtryk til dansk, med ganske få undtagelser, da hele resten af teksten jo er på dansk.

Hvad jeg har udført af forsøg, er på forskellige sparc-maskiner, for tidsmålinger altid på sparcSLC. Mine programmer er udført i c og findes i directory Specforsog.

Indholdsfortegnelse

Kapitel 1

Generelt om TSP.

Herre, i dit navn tager jeg nu af sted. Fri mig og alle andre fra al fare. Giv min rejse lykke og held og før mig trygt tilbage. For Jesu skyld. Amen.
(En rejsendes bøn, DDS)

I dette kapitel behandles den handelsrejsendes problem (TSP) generelt.

Der findes så meget materiale om TSP, at man kun kan håbe at se toppen af isbjerget. Jeg går bevidst udenom algoritmer, der tillader exponentiel køretid. (Disse algoritmer bygger typisk på lineær programmering, et "gammelt" felt indenfor datalogien.) Blandt de polynomielle algoritmer er der nemlig mange spændende og for de fleste mennesker næsten ukendte at vælge imellem. En vigtig grund til, at TSP bliver behandlet så meget, er at algoritme-teknikker, der kan løse forskellige svære problemer, testes på TSP. Jeg behandler bl.a. simuleret udglødning, der kan angribe mange forskellige problemer indenfor kombinatorisk optimering. Jeg fokuserer kun på, hvad disse algoritmer gør for TSP.

Først gennemgår jeg kort problemets historie, fra formuleringen af problemet ca. 1940 til i dag, hvor stadig større instanser af problemet løses.

Nogle algoritmer er skræddersyede til specialtilfælde af TSP, derfor gennemgås de almindeligste specialtilfælde.

Da man normalt ikke kan håbe på at finde en optimal løsning, gennemgås forskellige metoder til at vurdere de løsninger man får.

Disse metoder bliver anvendt på nogle heuristikker, der er meget lette at formulere, men svære at vurdere og sammenligne. Jeg gennemgår nogle af disse heuristikker, og konklusionerne om deres kvalitet.

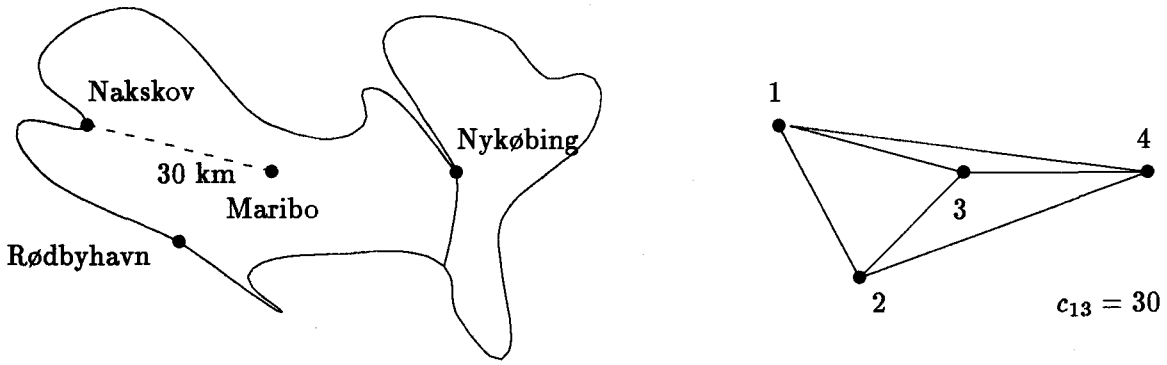
Der antages kendskab hos læseren til gængs grafteori og algoritme-analyse, som fx. i *Corm 90*, dvs. fx. finde korteste udspændende graf og bestemme kompleksitet af algoritmer. At TSP er NP-komplet antages kendt. Det antages bl.a. kendt, at problemet korteste tildeling kan løses i polynomiel tid, men metoden gennemgås ikke. Ellers antages alt direkte om TSP nyt. Se en opsamling af den vigtigste gennemgående notation i næste afsnit.

1.1 Formulering, definitioner.

Min notation følger stort set de almindelige konventioner, en oversigt følger her.

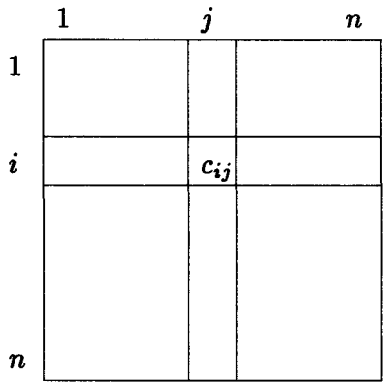
Problemet, der har givet navn til det mere generelle TSP, er det rent fysiske problem, givet n byer, og en matrix, der beskriver afstanden mellem dem (som den trekantede vejlængdetabel der er bag i de fleste landkort), find da den korteste rundtur, der starter og slutter i samme by, og undervejs kommer gennem hver by netop en gang.

Problemet omdannes til det generelle TSP vha. matricen, der beskriver afstandene. Landkortet, der viser byernes beliggenhed i forhold til hinanden, danner grundlag for en evt. graf, som på figur 1.1.



Figur 1.1: Et landkort (Lolland-Falster) og den tilsvarende graf.

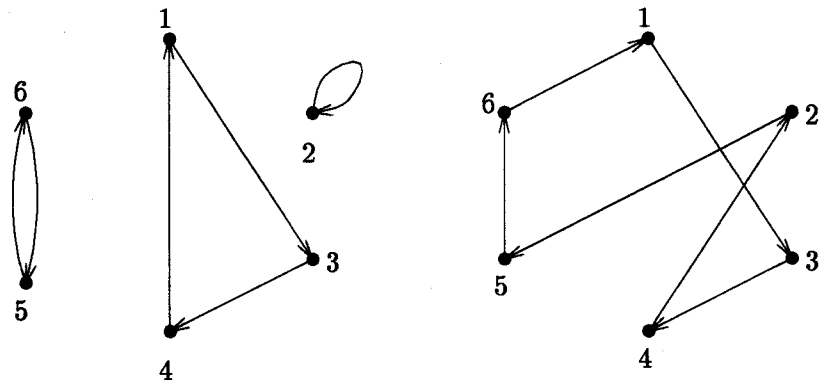
Den opståede matrix (der også kunne være fremkommet på mange andre måder) beskriver en komplet graf med n punkter, dvs. vægten af enhver kant mellem 2 af disse punkter er kendt. Evt. har kanter længden uendelig (dvs. grafen egentlig ikke er komplet, men som regel bruges denne oplysning ikke).



Figur 1.2: En matrix, og c_{ij}.

Kanten (i, j) fra punkt i til punkt j har vægten c_{ij}, se figur 1.2.

Hvad der på landkortet kaldes byer, veje og afstande, kaldes på grafen punkter, kanter og vægte. Betegnelserne bruges i flæng.



Figur 1.3: Her sammenlignes en permutation og en tur.

En tildeling (assignment) tildeler hvert punkt "en næste", enentydigt, dvs. hvert punkt er "næste" for ét andet punkt. Det kan fortolkes som, at man fra et givet punkt altid ved, hvad det næste er, hvor man skal gå hen. En tildeling kan skrives op som en permutation φ af de n punkter/numre, fx. ved at skrive (de ombyttede) numre op.

Ex: (3,2,4,1,6,5) er en permutation af tallene fra 1 til 6.

Da φ opfattes som en funktion, fortolkes ombytningerne således, at det første tal er $\varphi(1)$, det næste $\varphi(2)$, osv. op til $\varphi(n)$.

Ex: $\varphi(1) = 3, \varphi(2) = 2, \ldots \varphi(6) = 5$

Man kan også skrive permutationen, så 1 følges af $\varphi(1)$, der følges af $\varphi(\varphi(1))$ osv., indtil man er tilbage ved 1 igen. Her kræves muligvis mere end en parentes.

Ovenstående kan skrives som (1,3,4)(2)(5,6).

Man skriver ikke altid parenteser med kun et tal op, men for at kunne tælle hvor mange parenteser der er, skriver jeg dem allesammen op.

En tur kan beskrives som en permutation, hvor man kommer gennem alle punkter ved at følge angivelserne af "næsten" fra punkt til punkt. Det svarer til at tegne en streg gennem alle punkter uden at løfte blyanten fra papiret, og så man kommer tilbage til hvor man startede. Hvert punkt passeres netop én gang, og et punkt er med i netop to kanter. En tur τ kan skrives op med kun én parentes på den nederste af de to ovenstående skrivemåder, dvs. hvis man starter i 1 og følger kanterne rundt: $(1, \varphi(1)), (\varphi(1), \varphi(\varphi(1)))$ osv., vil man først nå tilbage til 1 efter n kanter. Omvendt kan en vilkårlig permutation på denne måde opfattes som et antal delture, der er disjunkte, og dækker alle punkter. En tur kaldes også en Hamilton-kreds.

Vores (1,3,4)(2)(5,6) kan opfattes som en deltur gennem 1, 3 og 4 (i den rækkefølge), en anden deltur gennem 5 og 6, og en deltur, der kun indeholder punktet 2 (se til venstre på figur 1.3). Dette er ikke en tur, men et antal delture, der tilsammen indeholder hvert punkt netop en gang. En tur ville fx. være (1,3,4,2,5,6), se til højre på figur 1.3.

Bemærk at der ofte ikke er retning på ture, så turen (1,3,4,2,5,6) er den samme som (1,6,5,2,4,3).

En Hamilton–sti σ er en tur minus den sidste kant. Derfor får man ikke en kreds, men bare en sti, der går gennem alle punkter. Der er også andre typer stier, der ikke nødvendigvis går gennem alle punkter.

En Euler-tur ν gennem en graf går også fra punkt til punkt via kanterne i graferne, men dækker alle kanterne netop én gang (i modsætning til at dække punkterne), og kan derfor ikke altid skrives op som en permutation, fordi et punkt passeres mere end en gang, hvis der er mere end 2 kanter til punktet. En Euler-tur er mulig, hvis alle punkter har lige valens, dvs. et lige antal kanter ender i punktet, og grafen er sammenhængende. (Jeg antager her, at kanterne ikke har retning, jeg beskæftiger mig nemlig kun med denne type grafer i denne forbindelse.)

Et træ er en kantmængde, der ikke er kredse i, dvs. der er ikke nogen delmængde af træet, der er en deltur gennem nogle punkter. Træet er også sammenhængende, således at der altid er (netop én) sti mellem 2 punkter i træet.

Et udspændende træ har alle n punkter dækket, og består af $n-1$ kanter.

En omkostning er defineret fx. for en permutation, som summen af vægtene på de kanter, permutationen anviser en at bruge. $c(\varphi) = \sum_{i=1}^{n} c_{i,\varphi(i)}$ Ofte taler man om omkostningen af fx. en tur, som længden af turen. Man søger at minimere omkostningen, dvs. leder efter den korteste tur. Omkostningen kan defineres for alle kantmængder, dvs. også træer, stier osv. Et minimum udspændende træ (Minimum Spanning Tree) kaldes også et MST. Det er også det udspændende træ med den mindste omkostning.

Da stier nogle gange kun kan beskrives ved deres start– og slutpunkt, skrives længden af stien fra i til j som $c(i,j)$.

Med disse definitioner bliver nogle bestemte problemer, der vil blive betragtet ofte i det følgende, meget nemme at beskrive.

Problem 1 *Der er givet en komplet graf og en angivelse af vægtene på kanterne i grafen, fx. i en matrix. Den handelsrejsendes problem (Travelling Salesman Problem / TSP) er da at finde den korteste tur. Findes den optimale tur har vi en optimal løsning, findes en sub-optimal tur (en længere tur) har vi en brugbar løsning.*

Problem 2 *Der er givet en komplet graf og en angivelse af vægtene på kanterne i grafen, fx. i en matrix. Problemet at finde en minimal tildeling er da at finde en permutation med mindst mulig omkostning.*

En heuristik er en algoritme, der ikke kan garanteres at finde den optimale løsning.

En approximation til en løsning opnås, hvis man kan garantere at heuristikkens løsning maksimalt er en konstant, en konstant faktor eller lignende, for lang, og evt. kan variere algoritmen til at opfylde et ønsket krav, fx. selv bestemme om man vil have en tur der er højst faktor 1,5 eller faktor 1,2 lang.

1.1.1 Problemtyper.

Da visse algoritmer er skræddersyede til kun at virke (eller virke bedre) under visse bibetingelser, gennemgås her nogle specialtilfælde, der ofte betragtes.

Symmetri.

- $c_{ij} = c_{ji}$

Der er lige langt frem og tilbage. Dette medfører bl.a. at man ikke behøver tage hensyn til hvilken retning man gennemfører turen med, længden vil alligevel være den samme. For 2-opt og dermed simuleret udglødning medfører dette, at man kan vende et langt stykke af en tur om, og alligevel kun skulle tage hensyn til 2 nye kanter.

Trekantuligheden.

- $c_{ij} \leq c_{ik} + c_{kj}$

Dvs. at det altid er lige så godt eller bedre at gå den lige vej mellem 2 punkter, fremfor at gå igennem et tredje først.

Positiv matrix.

- $c_{ij} \geq 0$

I dette tilfælde er alle afstande altså positive, og dette er forudsætningen for at nogle algoritmer overhovedet virker, specielt dem der antager, at en tur ikke bliver kortere hvis man tilføjer en by.

Euklidiske TSP.

- Punkterne i problemet kan betragtes som punkter i et d-dimensionalt euklidisk rum, typisk 2- eller 3-dimensionalt, hvor afstandene mellem punkterne er de normale euklidiske afstande.

Dette medfører trekantulighed, symmetri og positiv matrix (det gør alle normer). Det medfører også at problemer bliver lette at tegne, i 2 dimensioner specielt, fordi grafen faktisk kan antages at findes på et almindeligt stykke papir, og punkterne dermed bare afsættes i et almindeligt koordinatsystem. Denne problemtype er så vidt jeg ved den hyppigst undersøgte, bl.a. i det bibliotek for benchmark problemer, jeg har brugt.

Bemærk at nogle problemer også opstår som punkter i et rum, under benyttelse af andre normer. Disse omtaler jeg ikke yderligere.

1.2 Historie.

Kilderne til disse oplysninger er primært *Hoff 85* og *Mill 91*, den første giver et meget detaljeret referat af TSP's historie op til '85, og den anden supplerer med hvad der skete i de næste par år.

TSP er meget vigtigt at kunne løse, bl.a. fordi det kan formuleres på mange forskellige nyttige måder, og stadig essentielt være det samme problem. I næste afsnit præsenterer jeg andre ting TSP kan løse, end lige problemer med veje og biler. TSP er også blevet en målestok for hvor godt algoritmer til generelle problemer indenfor kombinatorisk optimering virker. Ofte refereres der til og sammenlignes der med TSP, når der tales om svære problemer, og problemet er blevet en slags "konge" over denne problemtype. Og i USA ligger en del af problemets tiltrækning i, at Den Handelsrejsende er en mytisk person, der er med i mange vittigheder osv. Men faktisk har problemet ikke været kendt specielt længe, og her følger en oversigt over, i hvilke årstal forskellige begivenheder fandt sted.

1759 Euler diskuterer "Springerens problem", at få flyttet en springer gennem alle et skakbræts felter med en springers bevægelser. 1771 gør Vandermonde det samme. Dette svarer faktisk til at lave en Hamilton-tur rundt på skakbrættet via felterne, så man kun kan komme fra et felt til et andet, hvis man kan gøre et springertræk (et felt midt på brættet har således 8 naboer, og det er ikke de nærmeste af felterne.) Dette kan igen omsættes til en ukomplet graf. *Eu 1759, Va 1771*

1831 En handelsrejsende udgiver en bog beregnet for sine kolleger: *Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein. Von einem alten Commis-Voyageur.* Bogen strejfer også, at den gode handelsrejsende skal tilrettelægge sin rute godt. *Vo 1831*

1858 Hamilton definerer "The Icosian Game", et spil hvor man skal lave Hamilton-kredse mellem hjørnerne på et icosaeder, dvs. en 12-sidet terning, via dens kanter. En variant af dette er "Traveller's Dodecahedron", der dog nok ikke har nogen direkte forbindelse til TSP. Se også figur 1.4. *Ha 1858*

1930 I anden forbindelse beskæftiger Menger sig med problemet korteste Hamilton-sti, og skriver bl.a., at den eneste mulighed, han kan få øje på er at afsøge de mulige permutationer, for derved at finde den korteste, og at algoritmen "nærmeste nabo" i hvert fald ikke virker. *Meng 30*

1931 TSP bliver så småt diskuteret i matematiske kredse på Princeton, bl.a. af Merrill Flood. Og bliver navngivet som TSP. *Floo 56, Tuck 83*

1940'erne Merrill Flood begynder at tale meget om TSP, i artikler osv., støttet af RAND, der ser et spændende problem, der falder udenfor spilteori, hvor de fleste spændende problemer ellers er. I denne periode indser man også, at det nye lineær programmering kan løse flere problemer, der ligner TSP, men tilsyneladende ikke TSP selv, selv om TSP kan formuleres som et ILP-problem. Se nedenfor.

1954 Dantzig, Fulkerson & Johnson løser et problem med 49 byer med en såkaldt branch & cut agtig metode, men først 15 år senere bliver deres opdagelser rigtig udnyttet. *Dant 54* (Dette er en LP-metode, der kræver TSP formuleret som ILP. For at få mere at vide om metoden, se fx. *Mill 91.*)

1964 For første gang findes et specialtilfælde af TSP, der kan løses med en polynomiel algoritme. Denne form for analyse slår dog kun rigtig igennem senere i Sovjetunionen. *Gilm 64, Demi 79.*

1970'erne NP-komplethed "opfindes", og man opdager at TSP falder i denne kategori.

I denne situation ved man altså (og har allerede længe haft en anelse om), at TSP er et problem, der nok ikke kan løses med en polynomiel algoritme, medmindre $P = NP$, dvs. alle de NP-komplette problemer også kan løses med polynomielle algoritmer. Men det skal jo stadig løses, og hvad kan man så gøre? Enten kan man som ovenfor nævnt undersøge, om det aktuelle TSP er et specialtilfælde, der lader sig løse, eller også må man affinde sig med, at der ikke er nogen garanti for både løsning og køretid. I det sidste tilfælde kan man enten vælge at løse problemet, uanset hvor lang tid det tager, eller bruge en heuristik, der producerer en sub-optimal løsning, men gør det i polynomiel tid.

Hvis man accepterer høje køretider er algoritmen branch & cut som sagt en mulighed, og den anvendes da også meget. Hvis man accepterer sub-optimalitet, kan algoritmerne typisk deles op efter, om de konstruerer en tur, eller om de forbedrer på en tur. Dette kommer jeg meget tilbage til i afsnittet med Simple Heuristikker. Desuden kan algoritmerne/heuristikkerne vurderes på flere forskellige måder, når man gerne vil vide nøjagtigt hvor sub-optimale de er.

1965 og før Empirisk analyse, dvs. at finde nogle passende test-problemer at køre algoritmerne på, og så sammenligne resultater og køretider. Problemet er selvfølgelig at finde nogle "typiske" test-problemer.

1960'ernes slutning Worst-case analyse, hvor man fx. beviser at MST-algoritmen højst skyder 100 % for højt. Her er problemet, at analysen nogle gange er for pessimistisk i forhold til de fleste anvendelser. Til gengæld holder båndet på løsningens kvalitet altid. (Bånd: grænse for hvor dårlig en løsning er.) *Grah 66, Grah 69, Chri 76, Sahn 76*

1975 Probabilistisk analyse, dvs. at beregne sandsynligheden for at algoritmen kommer med et resultat af en given kvalitet. Dette indebærer igen, at man faktisk kan udtale sig om, at problemerne, man vil løse, opfylder visse sandsynlighedskrav, såsom at punkterne faktisk er tilfældigt fordelt. *Karp 77*

Disse 3 klasser af analyse vil senere blive beskrevet yderligere, og brugt på forskellige algoritmer.

1980 Crowder & Padberg rapporterer at have løst et 318-bys problem på 3,4 timer med udnyttelse af bl.a. Lin-Kernighans algoritme (som jeg omtaler senere blandt de simple heuristikker) og af branch & cut og andre LP-teknikker. *Crow 80*

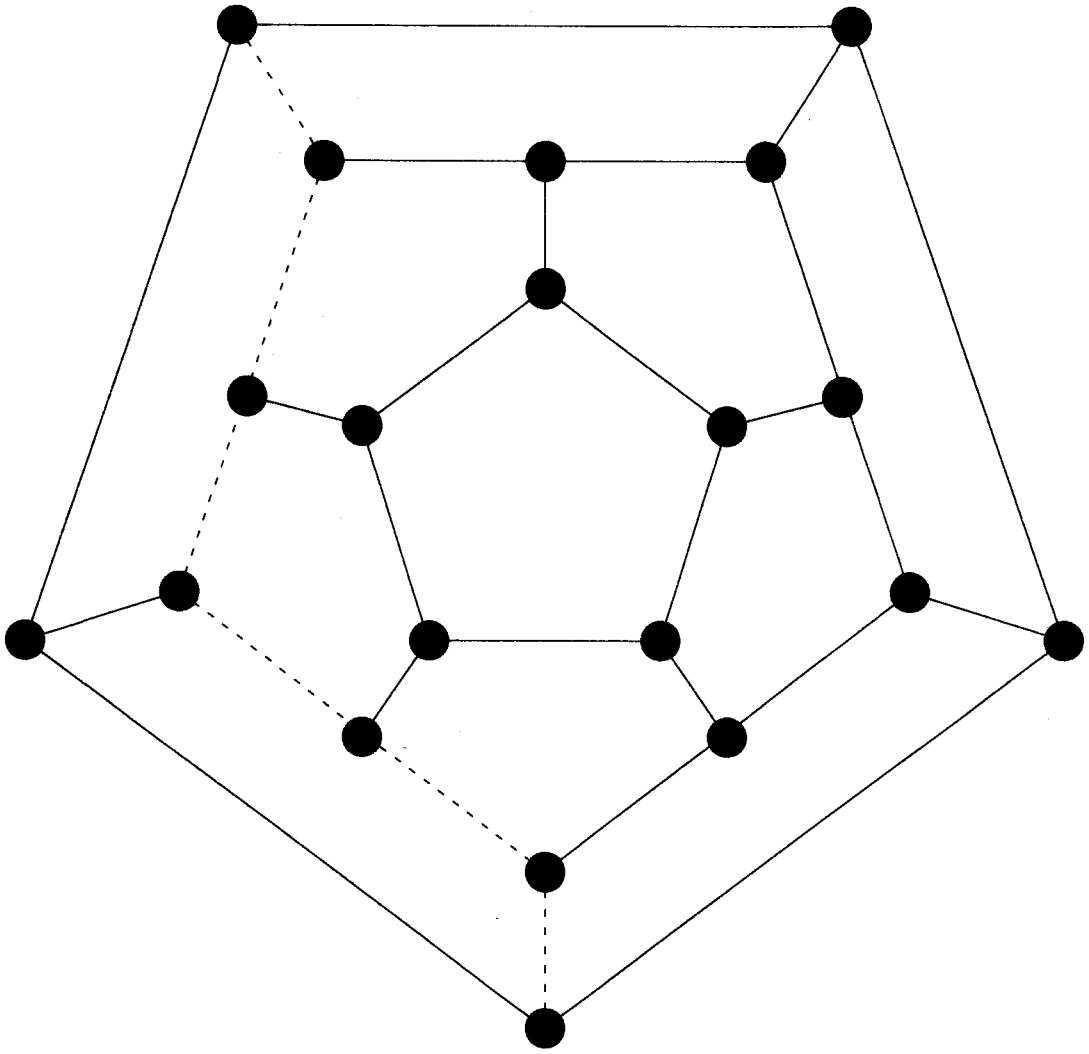
1987-88 Et virkeligt problem på 2392 byer løses på 27,3 timer (og siden 2,6 timer) af Padberg & Rinaldi. *Padb 87, Padb 88*

1991 Miller & Pekny rapporterer at have løst mange store problemer (bemærk: asymmetriske, tilfældigt genererede matricer), deriblandt et på 500.000 byer. En variant af den såkaldte branch & bound metode benyttes. *Mill 91*

Dvs. man kan løse store problemer, flere 1000 byer for symmetriske problemer, og flere 100.000 for asymmetriske problemer. Dvs. at nogle af de store problemer, såsom VLSI-design med 1,2 mio byer, er inden for rækkevidde, hvis man har tid nok. På den anden side vil nogle algoritmer levere næsten optimale løsninger på meget kortere tid, og de er derfor ikke blevet helt uinteressante.

Et af de sjovere forslag, til en algoritme jeg stødte på, var at kombinere et menneskes intuition med en maskines hurtighed til at regne. Et menneske skulle da vurdere bl.a. hvilke punkter, der nok skulle ligge tæt på hinanden i en tur, eller hvilke kanter der nok skulle med i turen. Ud af dette kom *Krol 71*, "A Man-Machine Approach ...".

The icosian game, figur 1.4, bestod af en spilleplade med 20 huller, og 20 nummererede brikker. Et af spillene bestod i at lægge 5 brikker som de 5 første brikker i en tur, og så prøve at fuldføre turen. Bemærk at man kan lave en umulig start på en tur med 7 brikker (starten af denne tur vist med den stiplede linje), ved at spærre et hul inde, så man ikke kan få både dette hul og resten med i samme tur. Spillet havde bemærkelsesværdigt nok en artikel vedlagt med matematiske overvejelser. Tænk hvis man i dag havde udgivet en artikel om algebra sammen med hver Rubiks kube!



Figur 1.4: The icosian game.

Integer Linear Programming

1.2.1 Heltals lineær programmering.

TSP kan formuleres som et problem indenfor heltals lineær programering, eller ILP.

Lad X være en matrix, der svarer til den sædvanlige $n \times n$ matrix, ved at beskrive hvilke af indgangene der skal medtages i summen, dvs. x_{ij} er 0, hvis kanten (i, j) ikke er med i turen, og 1 hvis kanten er med. x_{ij} kan kun antage disse 2 værdier, derfor er dette et heltals problem.

$c(\tau) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$, dvs. en anden måde at beregne τ's længde på, og den størrelse der skal minimeres. Så mangler der bare ligningerne til LP-problemet.

$x_{ij} \in \{0,1\}$, som nævnt.

$\sum_{j=1}^{n} x_{ij} = 1, \forall i$, dvs. reglen for en permutation skal være opfyldt, kun en kant ud fra hvert punkt.

$\sum_{i=1}^{n} x_{ij} = 1, \forall j$, og kun en kant ind i hvert punkt.

Dette beskriver indtil videre problemet korteste tildeling, og dette problem kan faktisk løses vha. LP, bl.a. fordi alle de ligninger, der sikrer heltallige løsninger, kan droppes.

Der mangler altså kun ligninger, der sikrer, at der ikke er mere end en tur i permutationen. Dette kan formuleres på flere måder.

$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S : \emptyset \subset S \subset N = \{1, \ldots, n\}$, altså for alle delmængder af punkter skal der være mindst én kant for lidt til at lave en kreds. Dette giver $2^n - 2$ ligninger! Kravet kan formuleres på andre, mere indviklede måder, der giver færre ligninger.

For at opsummere:

$$\text{Minimér } c(\tau) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$x_{ij} \in \{0,1\}; \sum_{j=1}^{n} x_{ij} = 1, \forall i; \sum_{i=1}^{n} x_{ij} = 1, \forall j; \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S : \emptyset \subset S \subset N = \{1, \ldots, n\}$$

Med denne formulering kan metoderne branch & cut og branch & bound benyttes, da de i hver iteration løser et mindre LP.

1.2.2 NP-komplethed.

Det er almindeligt kendt, at TSP er NP-komplet, så derfor vil jeg kun give en kort gennemgang af, hvordan dette og andre resultater, siger at TSP "er svært".

Et afgørelsesproblem er et der består i at stille et spørgsmål, der kun kræver svaret ja eller nej. TSP er ikke et afgørelsesproblem, men kan laves om til et: "Findes der en tur i grafen, der er højst x lang?" Ved at stille nok af denne type spørgmål (et logaritmisk antal, fx. 2-logaritmen af det søgte svar, er nok), kan man ovenikøbet løse problemet, fordi man vil få "ja" så længe x er lig eller større end længden af den optimale tur, og "nej" hvis x er mindre.

Et problem kan verificeres hvis en algoritme der svarer "Ja, der findes en tur, der højst er x lang." også kan komme med en tur, der opfylder kravet, og denne turs længde kan checkes i polynomiel tid. Dette kan gøres i $O(n)$, så dette krav er også opfyldt.

Et problem er NP-komplet hvis det er et afgørelsesproblem der kan verificeres, foruden at andre NP-komplette problemer skal kunne transformeres til problemet. For de NP-komplette problemer gælder, at man ikke regner med at finde polynomielle algoritmer til at løse dem. Et bevis for, at denne transformation kan ske, findes bl.a. i *John 85a.*

Ovenfor ses en forbindelse mellem afgørelsesproblemet for TSP, der er NP-komplet, og selve TSP, der kaldes NP-hårdt.

Af resultater man derudover har fået for TSP er, at det også er et NP-komplet problem at afgøre, om man har fundet en optimal tur, dvs. om der findes en kortere tur. Dette gælder også for symmetriske problemer med trekantulighed. Ifølge *John 85a* bevises dette i *Papa 77.* Også andre TSP-relaterede problemer viser sig at være NP-hårde.

Som om det ikke var nok, at problemet ikke kan løses, og at man ikke kan bevise at en given løsning er optimal, kan man generelt heller ikke approximere det. Givet en konstant r, vil eksistensen af en polynomiel algoritme, der altid giver resultater højst faktor r større end den optimale løsning, medføre at $P = NP$. *Sahn 76*

En heuristik kan også opstå hvis man tager en tilfældig tur og bliver ved med at forbedre på den, indtil en ny forbedring ikke er mulig mere. Hermed opstår begrebet nabolag til en tur, dvs. de ture man kan komme frem til fra en givet tur, ved at foretage en bestemt type forbedring. Senere bliver forbedringen 2-opt defineret i forbindelse med simuleret udglødning, og dermed også dette nabolag. En anden mulig forandring er at betragte en tur som en liste af byer, og så prøve at bytte rundt på 2 byer i denne liste. Dette giver et nabolag, af alle de ture man kan komme frem til ved en sådan ombytning.

Lad en algoritme bruge polynomiel tid på at udforske sit nabolag og vælge en af de betragtede ture, således at den nye tur er kortere end den gamle. Da kan algoritmen ikke garanteres at finde en tur af længde højst $r \cdot OPT$, selv med et eksponentielt antal nabolag-søgninger, hvis ikke $P = NP$. *Papa 77*

1.3 TSP's familie.

Den handelsrejsendes problem i sig selv (bil, veje osv.) er et yderst nyttigt problem at kunne løse, men måske ikke det man hyppigst støder på. Der er dog andre problemer, der kan løses via TSP, eller måske meget ligner TSP. Da dette bare er eksempler på, hvor varieret TSP er, skitserer jeg kun, hvad man kan gøre og hvordan. De mere detaljerede eksempler, og andre, kan findes i *Garf 85*.

1.3.1 Ruteplanlægning.

Vehicle routing eller ruteplanlægning minder meget om TSP. Ingredienserne er:

- en flåde af biler
- varer
- et enkelt depot, alle biler starter i (en by)
- kunder (byer)
- hvor mange varer kan en givet bil have
- hvor mange varer vil en given kunde have
- hvornår kan kunden modtage sine varer

Et simpelt problem af denne type kunne være et postfirma med 3 biler, der hver kan transportere et ton eller 12 m^3 pakker. På en given dag skal Hansen have en cykel efter middag, Petersen skal have et lille, værdifuldt ur mellem 11 og 13, Sørensen skal have en kassefuld dyner osv. Alle kunders adresser er selvfølgelig givet. Hvis der i øvrigt ikke er flere pakker, end der kan være i de 3 biler tilsammen, er problemet at fordele kunder/pakker til bilerne og planlægge bilernes ruter.

En mulig algoritme for dette problem fordeler kunderne ud på bilerne, og løser så et TSP for hver bil. Dette er ikke en garanti for at finde den optimale fordeling. Problemet er NP-hårdt.

1.3.2 Computer wiring.

Computer wiring eller at forbinde fx. chips med ledninger er et andet problem, der minder meget om TSP. Her findes:

- chips med ben
- hvilke ben (byer) der skal forbindes
- der må højst være to ledninger til et ben

For en given mængde af ben, der skal forbindes, er problemet altså at finde den korteste Hamilton-sti. Dette kommer vi tilbage til. Se i øvrigt figur 1.5.

1.3.3 Tapetklipning.

I dette tilfælde skal der klippes stykker af et mønstret tapet. Der er:

- tapetstykker (byer)
- hvert stykke har givet hvor i mønstret det skal starte og slutte.

Man skal så vælge i hvilken rækkefølge, stykkerne skal klippes, for at så lidt tapet som muligt går til spilde. Igen nærmere et korteste Hamilton-sti problem. Se figur 1.5.

For 2 stykker tapet er givet, at det stykke tapet, man er nødt til at klippe ud mellem bunden af det ene stykke tapet og toppen af det andet, kun afhænger af hvor i mønstret disse 2 stykker slutter og begynder.

> Man kan regne ud, at spildet ved at klippe stykket til bag radiatoren ud efter stykket til over døren er 0,4 m.

Ved at finde spildet for alle par af stykker (asymmetrisk problem) opbygges matricen, og dermed et TSP. Her antages desuden, at det er givet hvor i mønsteret tapetet er klippet før første stykke og efter sidste stykke; hvis dette er samme sted, fås et problem, hvor man klipper ud af en loopet tapetrulle, og dermed et rigtigt TSP, hvor det ikke har betydning, hvilket stykke, der klippes ud først.

1.3.4 Jobs rækkefølge.

Job sequencing, eller at bestemme et antal jobs rækkefølge er endnu en variant. Her er:

- jobs (byer)
- en enkelt maskine
- den tid det tager at omstille maskinen til et givet job efter et andet givet job

> Maskinen kan have en temperatur, der skal hæves eller sænkes mellem jobs, eller maskinens dele kan af/på-monteres afhængigt af den nøjagtige opgave, så dette tager tid.

Det, der skal minimeres er køretiden. Tiden mellem 2 jobs overføres direkte til matricen, og der opstår et TSP, igen kan start- og sluttilstand være givet på forhånd som med tapetklipningen, så problemet kommer til at "køre i ring".

1.3.5 Korteste Hamilton-sti.

Det mere matematiske problem (der altså ligger bagved nogle af de ovennævnte problemer) kan også betragtes som et TSP.

Der findes flere varianter, afhængigt af om man kender stiens start- og slutpunkt eller ej. Lad os sige, at startpunktet er givet, og slutpunktet er ligegyldigt (som om den handelsrejsende er i den by, hvor han skal starte, og ikke behøver skynde sig hjem fra den sidste by). Dvs. kanten fra et givet punkt til startpunktet aldrig bliver brugt. Hvis derfor søjlen med længderne af disse kanter erstattes med en søjle af nuller (startpunktet j, i vilkårligt punkt, $c_{ij} = 0$) og TSP i stedet løses, bliver den resulterende tur, minus kanten til startpunktet den ønskede Hamilton-sti.

Lignende transformationer kan gennemføres i de øvrige mulige situationer. Problemet er NP-hårdt.

1.3.6 Multi-TSP.

Hvis man har m handelsrejsende, med samme hjemby, og disse bare skal dække de øvrige byer tilsammen, har man også en slags TSP, og problemet kan omformes til TSP, ved at tilføje $m-1$ ekstra byer. Først har man en graf, og hvilken by der er hjemby. Denne graf kan ved tilføjelse af $m-1$ kopier af hjembyen og passende nye kanter, omformes til et TSP. TSP i den nye graf svarer til at komme igennem hjembyen m gange, eller lade m handelsrejsende følge hver sin deltur. Se figur 1.5.

Dette er en forenkling af problemet ruteplanlægning. Men stadig NP-hårdt.

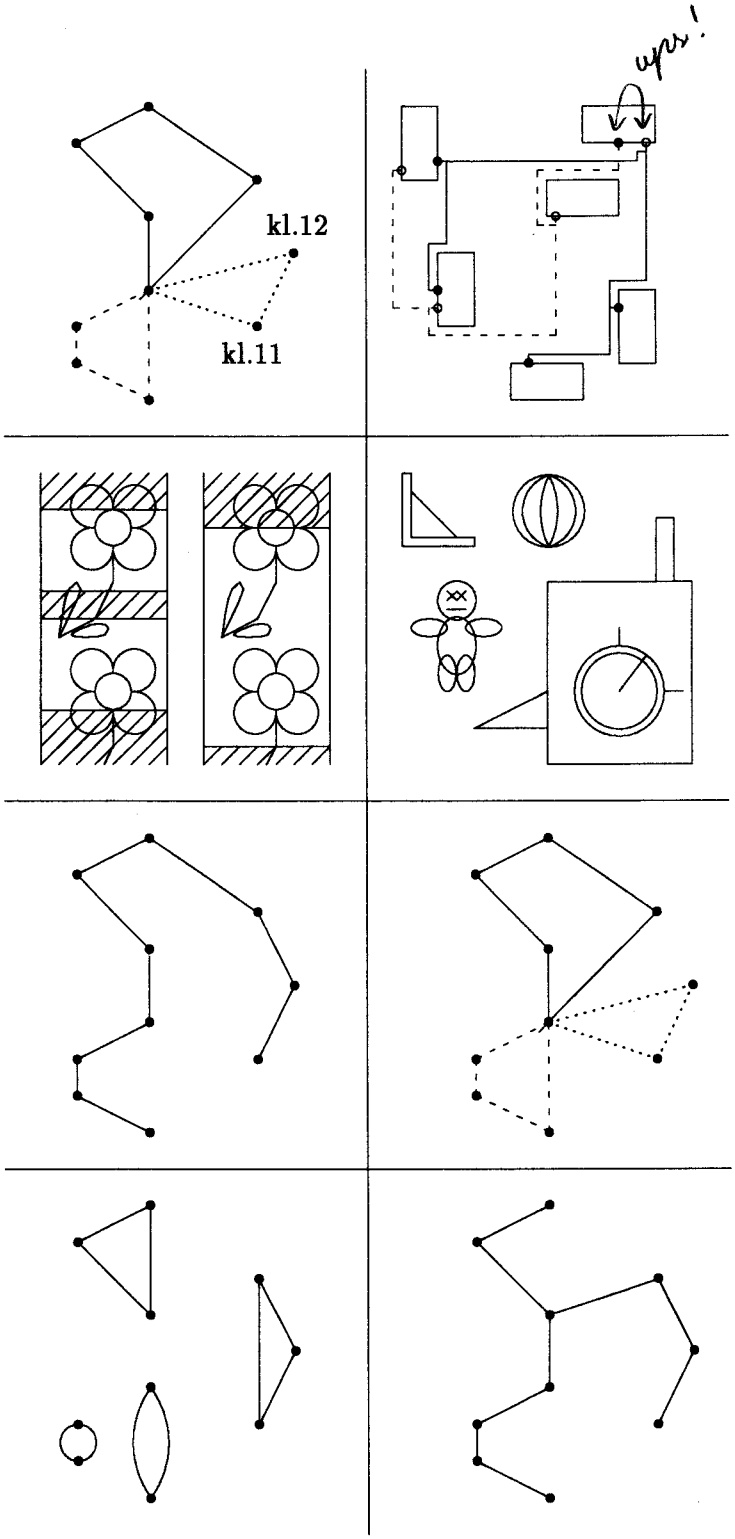
1.3.7 Korteste tildeling.

Dette er en variation af TSP, på den måde at det ene problem er lidt simplere end det andet, dette kan indses på flere måder. I både afsnittet om notation og afsnittet om LP, sås at korteste tildeling kan beskrives ved at beskrive TSP, og derefter smide et krav væk. Kravet til TSP er jo at finde en korteste tildeling, der også er en tur. Man kan faktisk finde den korteste tildeling i polynomiel tid.

1.3.8 MST.

Det minimum udspændende træ (MST) er en anden variant af TSP, fordi en tur minus en kant er en Hamilton-sti, og en Hamilton-sti er et udspændende træ. Man kan finde MST i polynomiel tid.

For lettere at kunne forestille sig sammenhængen mellem de nævnte problemer og TSP, har jeg i figur 1.5 prøvet at tegne dem.



Figur 1.5: Eksempler på de 8 nævnte problemer.

1.4 Simple heuristikker.

De nedenfor nævnte metoder har vist, at de her nævnte heuristikker giver gode resultater, selv om deres worst case bånd (eller mangel på samme) ikke lover dette, så jeg vil kort skitsere dem. Senere vil jeg selv lave empirisk analyse på mine 2 implementerede heuristikker, simuleret udglødning og genetisk algoritme for TSP.

De diskuterede algoritmer blev testet på euklidiske problemer og var enten heuristikker til at konstruere en tur, eller til at forbedre en given tur, eller begge dele. Jeg har hentet dem fra *John 85b* og *Gold 85b*, der opsummerer disse algoritmer og resultaterne af empirisk analyse på dem.

1.4.1 Tur-konstruerende heuristikker.

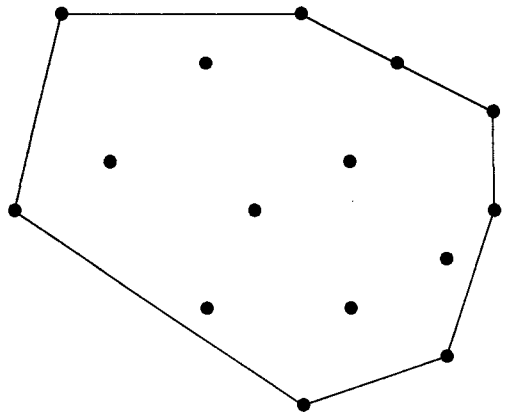
For en heuristik, der konstruerer en tur, skal tre ting være givet: en start-"tur", hvordan en ny by vælges, og hvor den indsættes. Derefter har algoritmen denne generelle struktur.

Algoritme 1 *Tur konstruktion(vægtet graf): tur*

initialisér en tur, der kun har få af byerne med, dvs. m ud af n byer
for ($i = m + 1$ to n) do
    vælg en ny by at indsætte
    indsæt denne by

Algoritme slut

Det viser sig *Flood 56*, at en god start-"tur" er det konvekse hylster, dvs. den der for et euklidisk problem svarer til at slippe en elastik rundt om søm, der repræsenterer punkterne, se også figur 1.6. Der findes nemlig en optimal tur den gennemløber de fælles byer i samme rækkefølge som det konvekse hylster.

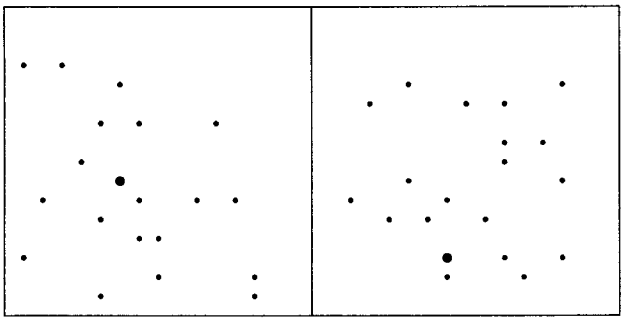


Figur 1.6: Det konvekse hylster.

For ikke at skulle sige det samme mange gange senere, her nogle definitioner af notation.

Definition 1 *For en tur T er $Tur(T,k)$ tilføjelsen af punktet k til turen, så visse krav opfyldes.*
$T = (i)$ (bare den enkelte deltur af byen i) $\Rightarrow Tur(T,k) = (i,k)$, altså delturen af i og k.
T mere end en by, vælg da i og j, så $c_{ik} + c_{kj} - c_{ij}$ minimeres, dvs. det billigste sted i T at indsætte k, da er $T = (\ldots,i,j,\ldots)$ og $Tur(T,k) = (\ldots,i,k,j,\ldots)$.
Desuden defineres $Kost(T,k) = c(Tur(T,k)) - c(T)$, altså udgiften forbundet med at udvide delturen til også at indeholde k.

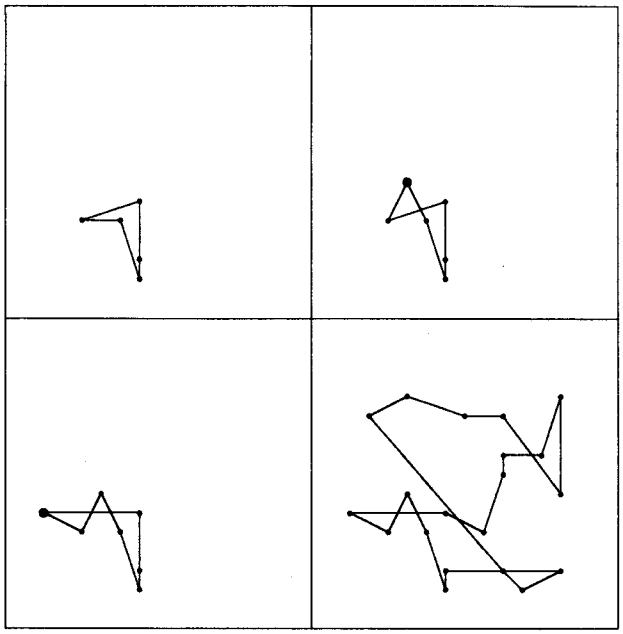
De viste algoritmer skrives op med navn, kompleksitet, de tre ting, der skal fastsættes i forhold til den generelle algoritme, og worst case bånd, hvis kendt. De er afprøvet på 2 grafer med 20 tilfældigt genererede punkter, hvor den bedste fundne tur for hver er illustreret i hhv. fjerneste indsættelse og tilfældig indsættelse. Punkt 1 er anmærket, fordi det altid bruges som det tilfældige punkt, hvis en algoritme starter med et sådant.



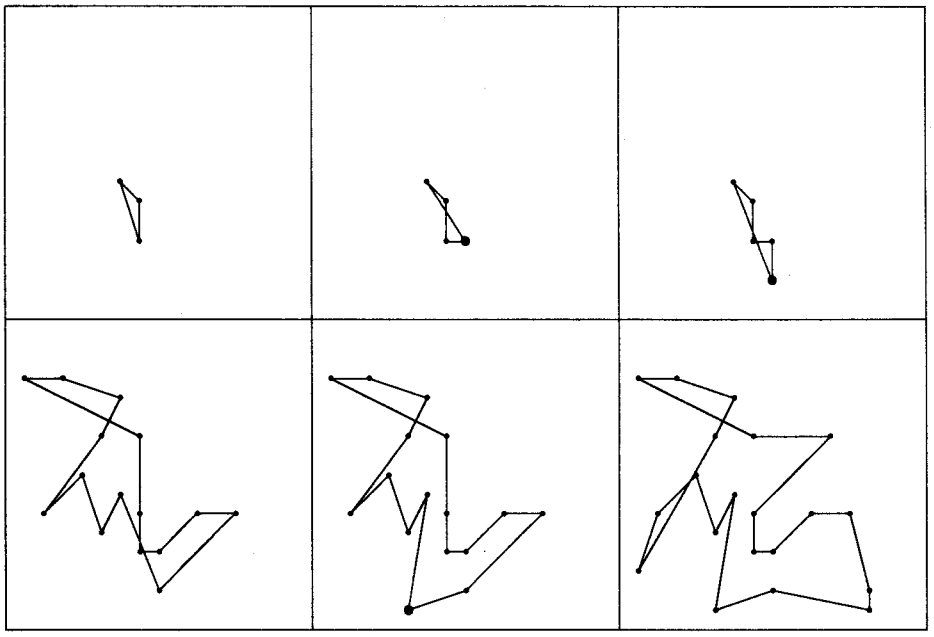
Figur 1.7: Graf 2 til venstre, og graf 1.

Nærmeste tilføjelse
$O(n^2)$
Starter med delturen (i), hvor i er valgt tilfældigt. (1a) $\forall k \notin T, \forall j \in T$, vælg k og j, så c_{jk} minimeres. (2a) Indsæt k mellem j og en af dens naboer. (3a)
$2 \cdot OPT$

Nærmeste tilføjelse er nok den mest naive af de næste heuristikker. Den bruger en god information, fordi den indsætter det punkt, der er tættest på den eksisterende tur, men bruger informationen dårligt, fordi punktet næsten indsættes tilfældigt. I figur 1.8 ses hvordan en overkrydsning opstår, fordi punktet bliver indsat på "den forkerte side" af det punkt, det var tæt på i turen. Denne overkrydsning er med i den færdige tur også. I figur 1.9 ses en overkrydsning (fra 3 til 4 punkter) der forsvinder igen (mellem 15 og 16 punkter), men da der er opstået andre overkrydsninger, er dette ikke så glædeligt. Så det er smart at tage den korteste kant med, der ikke er i turen, men der tages ikke hensyn til hvilken kant, man må smide væk samtidig, og hvilken anden kant man er nødt til at tilføje.



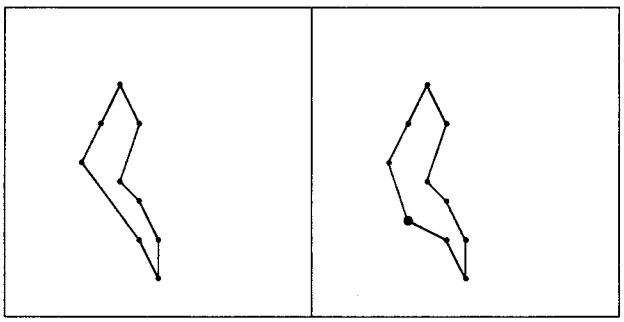
Figur 1.8: Nærmeste tilføjelse, graf 1, med 5, 6, 7 og 20 punkter.



Figur 1.9: Nærmeste tilføjelse, graf 2, med 3, 4, 5, 15, 16 og 20 punkter.

Nærmeste indsættelse
$O(n^2)/O(nlog^2n)$
Som (1a) Som (2a) $T_{ny} = Tur(T,k)$ (3b)
$2 \cdot OPT$

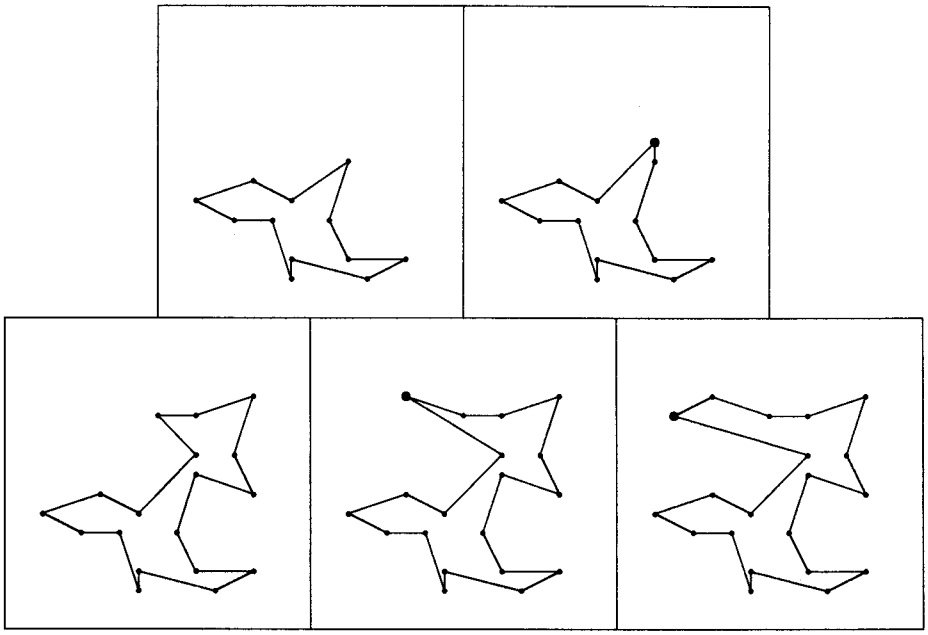
Af de ture, jeg så udviklet med denne algoritme, så jeg ikke nogle med overkrydsninger, og jeg kiggede spændt efter noget der ligner det i figur 1.10: at et punkt tages med, fordi det er tæt på et givet punkt, men indsættes mellem 2 andre. I dette tilfælde er det nye punkt lige tæt på 2 af de gamle punkter, så det er ikke til at sige, om det indsættes ved siden af det punkt, det blev konstateret tæt på.



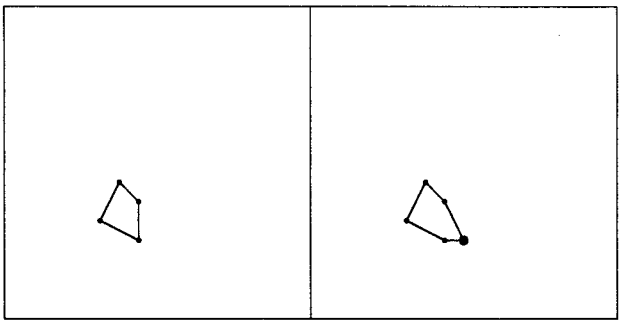
Figur 1.10: Nærmeste indsættelse, graf 1, med 9 og 10 punkter.

Billigste indsættelse
$O(n^2\ logn)$
Som (1a) $\forall k \notin T$ vælg k så $Kost(T,k)$ er minimal. (2b) Som (3b)
$2 \cdot OPT$

Hvis man betragter hvordan turen udvikles af denne algoritme, er det meget tydeligt, at anvendelsen af Tur og $Kost$ gør at turen fra skridt til skridt bare bliver trukket lidt ud, som fra 12 til 13 punkter i figur 1.11 og fra 4 til 5 punkter i figur 1.12. Men dette kan altså også gå galt, som i figur 1.11 fra 18 til 20 punkter, hvor den første tur da er meget rimelig, men den sidste sagtens kunne være pænere, de nye punkter kunne have ligget anderledes, for at give en kortere tur. Så det virker ikke altid bedst bare at "trække ud i elastikken", så den dækker flere og flere punkter.



Figur 1.11: Billigste indsættelse, graf 1, med 12, 13, 18, 19 og 20 punkter.



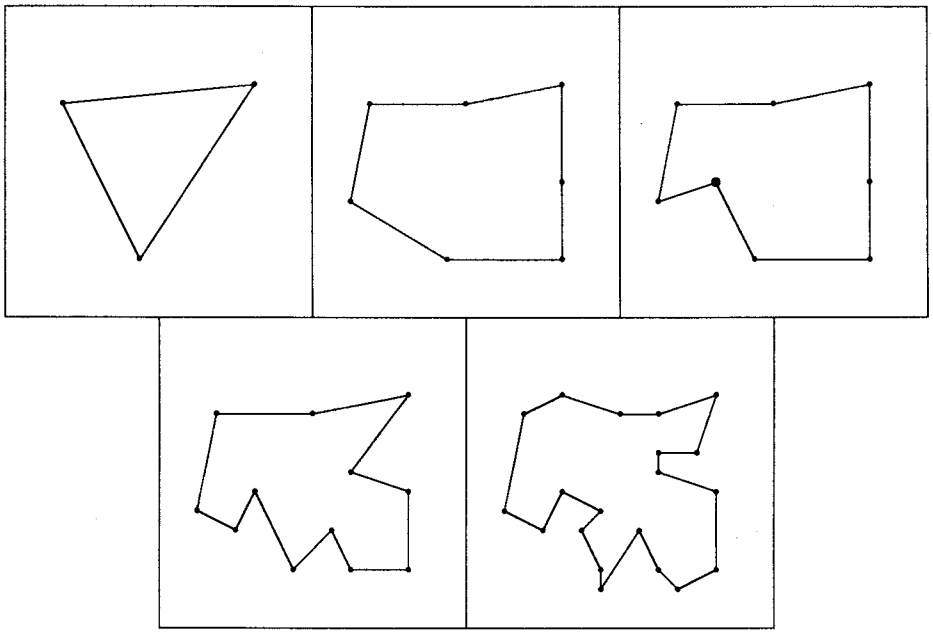
Figur 1.12: Billigste indsættelse, graf 2, med 4 og 5 punkter.

For de 3 ovenstående heuristikker er worst-case at få en tur der er faktor 2 for lang, fordi disse heuristikker, ligesom MST-algoritmen, bygger på at tilføje byer på samme måde som man føjer punkter til et MST. Og for alle disse heuristikker findes grafer, der vil resultere i ture, der svarer til worst case.

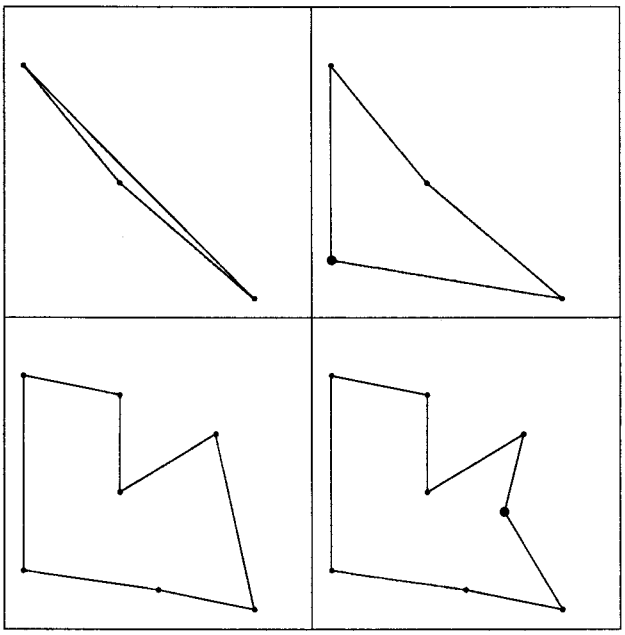
Fjerneste indsættelse
$O(n^2)/O(nlog^2n)$
Som (1a) $\forall k \notin T$ vælg k så $min\{c_{jk} : j \in T\}$ maximeres, dvs. vælg det punkt der er længst fra hele delturen T. (2c) Som (3b)

I disse tilfælde, og også i mange der ligner, konstrueres der altså først noget, der minder om et konvekst hylster. Den facon, turen får tidligt, ændrer sig ikke meget undervejs. Først bliver

det dækkede areal bare større og større, men hurtigt begynder turen at "bukke indad", som ved 8 punkter i figur 1.13 og figur 1.14. Faconen opnået efter 12 punkter i figur 1.13 ændrer sig næsten ikke frem til de 20 punkter. Denne tur er den bedst fundne for disse 5 algoritmer.



Figur 1.13: Fjerneste indsættelse, graf 1, med 3, 7, 8, 12 og 20 punkter.



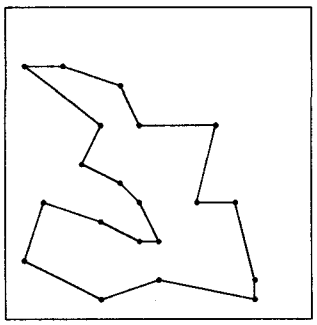
Figur 1.14: Fjerneste indsættelse, graf 2, med 3, 4, 7 og 8 punkter.

Tilsyneladende produceres der ture, der er op til faktor 1.5 for lange, ligesom Chrisofides' algoritme, der vides at have denne garanti, men det lader sig ikke lige gøre, at bevise det samme

for denne heuristik. Det virker dog rimeligt, at denne algoritme skulle virke bedre end fx. nærmeste indsættelse, fordi de punkter, der ligger langt fra hinanden, hurtigt får indflydelse på den "unge", flexible tur.

Tilfældig indsættelse
$O(n^2)$
i vælges tilfældigt, og k vælges, så c_{ik} minimeres. Start-turen er (i,k). (1b) $\forall k \notin T$ vælg k tilfældigt. (2d) Som (3b)

Her iagttages en blanding af de tre foregående algoritmers egenskaber, og hvis det går godt virker det som et lykketræf. På den anden side er denne algoritme hurtig at køre, evt. også flere gange, så den er en god konkurrent. Her ses kun den færdige tur for graf 2 i figur 1.15, fordi denne var den bedste fundne tur blandt de 5 algoritmer.



Figur 1.15: Tilfældig indsættelse, graf 2, med 20 punkter.

Som afrunding her en tabel over hvad længden af de fundne ture for hver kombination af graf og heuristik var. Bemærk at jeg kun har kørt hver algoritme en gang på hver graf (jeg var kun interesseret i at få korrekte eksempler på algoritmernes opførsel), og at jeg ikke har afbildet alle de resulterende ture, men kun dem jeg syntes illustrerede noget interessant. Den kortest fundne tur er afmærket. I de to søjler til højre vises procent over bedst fundne tur, dvs. den bedste fundne tur af de 5 resultater.

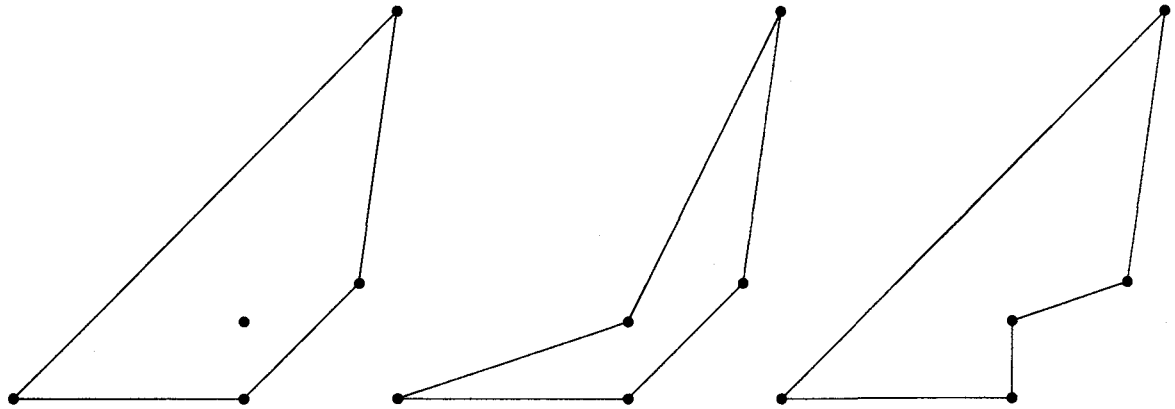
Heuristik	Graf 1	Graf 2	Graf 1	Graf 2
Nærmeste tilføjelse	65,1	70,5	31	21
Nærmeste indsættelse	56,0	62,9	12	8
Billigste indsættelse	59,2	62,3	19	7
Fjerneste indsættelse	*49,8*	61,3	0	6
Tilfældig indsættelse	51,1	*58,1*	3	0

Som det ses på procenterne (eller i hvert fald på deres indbyrdes forhold, da jeg jo ikke kender den optimale tur), svarer disse resultater meget godt til den senere tabel, hvor jeg opsummerer andres resultater med de samme algoritmer.

Herefter betragtes heuristikker, der starter med den konvekse omkreds. Disse heuristikker antager dermed, at problemet er euklidisk.

Største vinkel indsættelse
$O(n^2\ logn)$
T er grafens konvekse hylster. (1c) $\forall k \notin T = (\ldots, i, j, \ldots), \forall i, j \in T$, vælg i, j, k så vinklen mellem de to kanter fra k til i og j maximeres. (2f) $T_{ny} = (\ldots, i, k, j, \ldots)$ (3c)

Som illustration af denne indsætningsmetode, se figur 1.16. Her ses et konvekst hylster, og kun det sidste punkt mangler at blive indsat. Midt i figuren ses indsættelsen, hvor største vinkel bruges, og til højre ses den optimale tur, hvor indsættelsen var den billigste.



Figur 1.16: Indsættelse i konvekse hylster.

Konveks hylster indsættelse
$O(n^2\ logn)$
Som (1c) $\forall k \notin T$, find det par i og j, som k ville blive indsat imellem ifølge $Tur(T, k)$, og vælg k så $\frac{c_{ik}+c_{kj}}{c_{ij}}$ minimal. (2e) Som (3b)

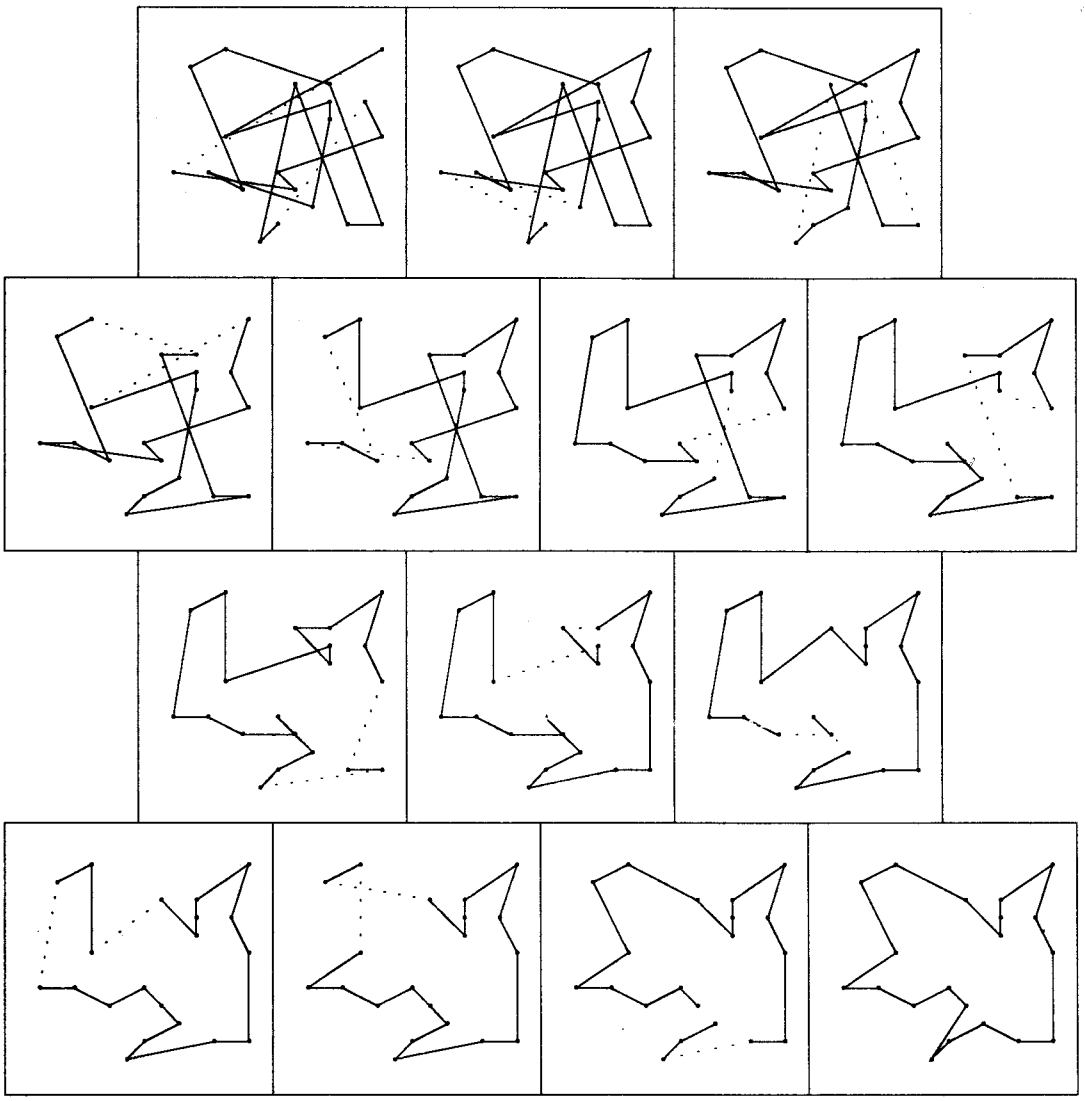
Ratio gange differens indsættelse
$O(n^2\ logn)$
Som (1c) $\forall k \notin T, \forall i, j \in T = (\ldots, i, j, \ldots)$, vælg i, j, k så $(c_{ik} + c_{kj} - c_{ij})\frac{c_{ik}+c_{kj}}{c_{ij}}$ minimeres. (2g) Som (3c)

De to sidste heuristikker er lidt sværere at illustrere uden at kombinere matematik og figurer, og jeg har valgt ikke at gøre mere ud af disse to.

1.4.2 Tur-forbedrende heuristikker.

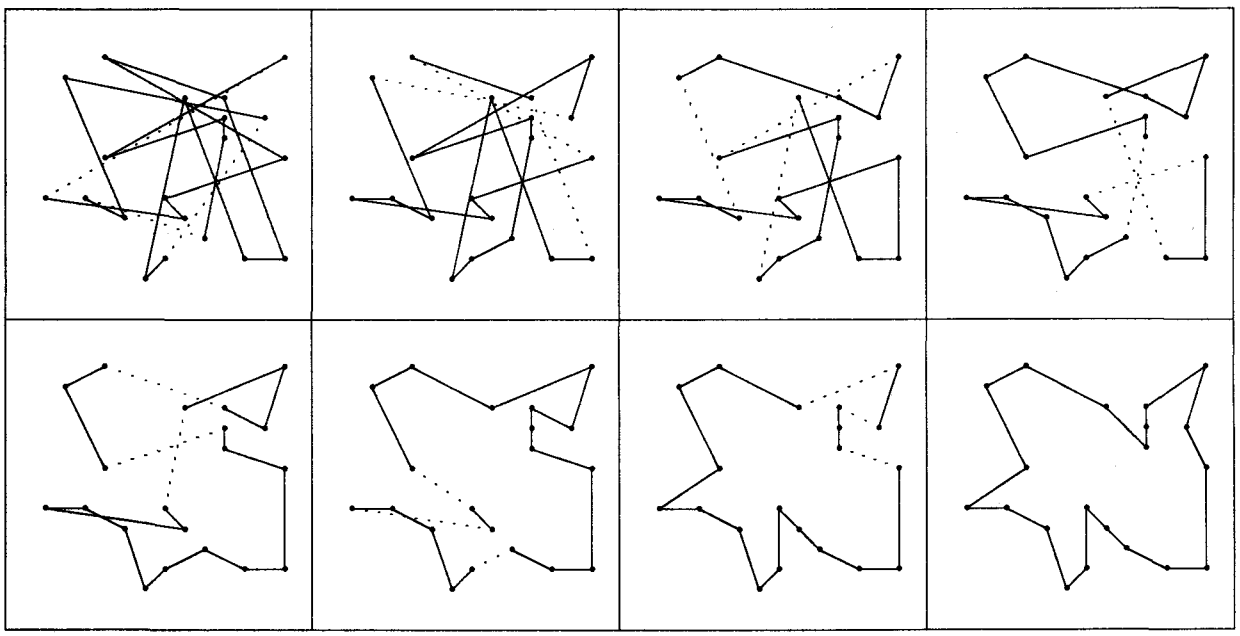
Den mest kendte form for tur-forbedring er kantbytning. Hver algoritme starter med en tilfældig tur, fx. slutresultatet af en anden algoritme, eller en tilfældigt genereret tur. For disse heuristikker er det sværere at sige noget om tidsforbrug og worst case bånd. Bemærk at sætningen om heuristikker (nævnt i afsnittet om NP-komplethed), der afsøger nabolag og dog ikke har nogen garanti for at finde en tur kortere end $r \cdot OPT$, kan bruges her.

En r-opt algoritme erstatter r kanter i en given tur med r andre, så resultatet er en kortere tur. Hvis det er umuligt at få en kortere tur ud af en r-opt er turen r-optimal. Hvis r stiger, stiger køretiden også, men resultatet bliver også en bedre tur. I praksis bruges kun $r \leq 3$. For $r = n$ ville resultatet være den optimale tur. 2-opt bruges i forbindelse med simuleret udglødning og også nogle gange genetiske algoritme. Hvis man taler om algoritmen 2-opt, menes den algoritme, hvor man går fra en given tur til den kortest mulige tur der kan opnås med en enkelt 2-opt, og bliver ved til en 2-optimal tur er fundet.



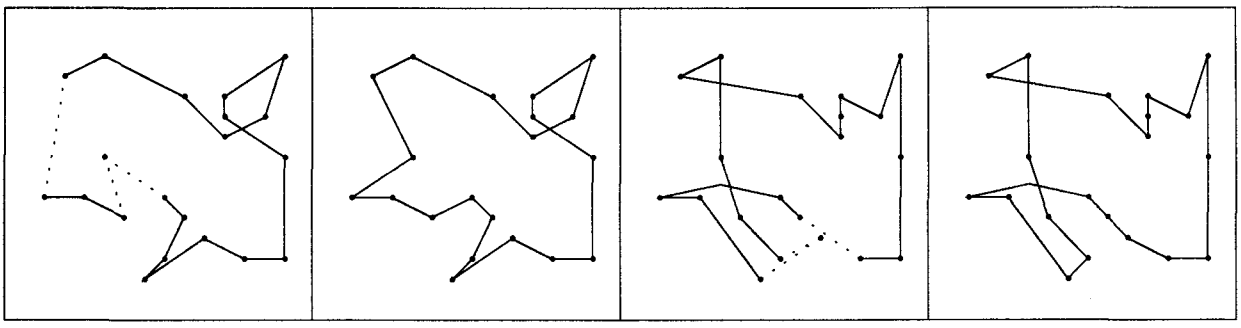
Figur 1.17: Tur før, under og efter 2-opt.

Som illustration af 2-opt ses på figur 1.17 en gennemgang af en 2-opt, fra første tilfældige tur, til den resulterende 2-optimale tur, i dette tilfælde efter 13 iterationer.



Figur 1.18: Tur før, under og efter 3-opt.

Som illustration af 3-opt ses figur 1.18, hvor en gennemgang af 3-opt ses, fra første tur, der er en tilfældigt fundet tur, til sidste tur, der så vidt jeg ved er den optimale, og i alt 7 3-opt vises, hver gang ved at stiple de kanter, der er ved at blive smidt væk, fordi de kan erstattes med 3 andre kanter, så der bliver den størst mulige forskel i tur-længde. Her ses ikke den specielle 3-opt, der egentlig består i at flytte et enkelt punkt et andet sted hen i turen, fordi 2 af de gamle kanter er naboer. Men denne illustreres i udklip fra to andre forløb, i figur 1.19.



Figur 1.19: Ture under 3-opt.

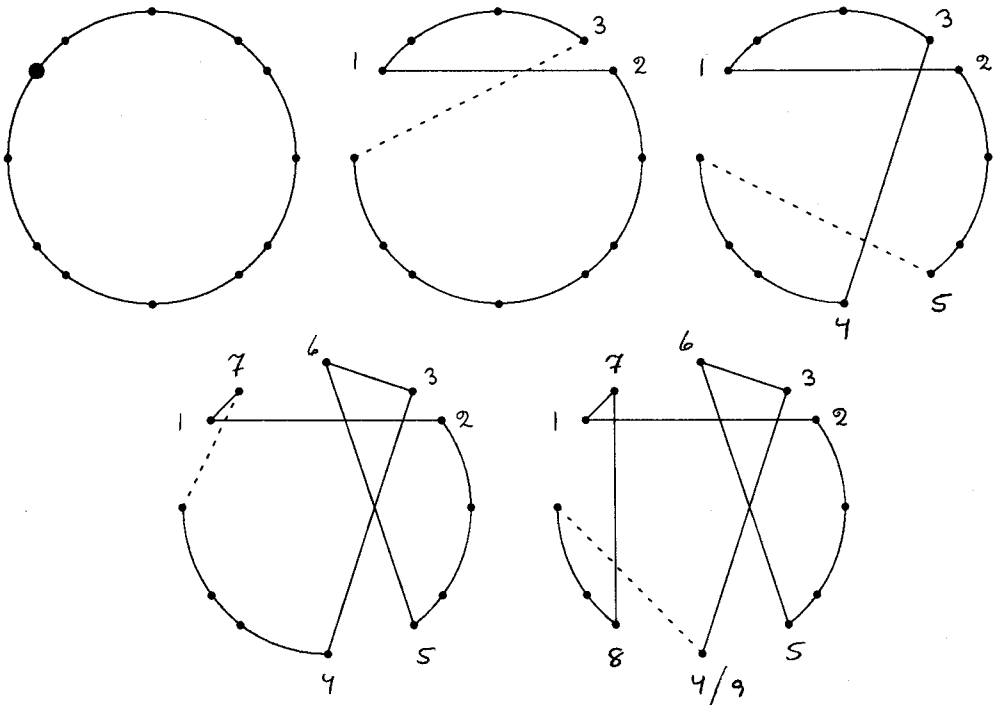
I begge tilfælde flyttes altså kun et enkelt punkt, i 3-opt'en til højre har jeg "bukket" en af kanterne lidt, for at man bedre kan se, hvor alle punkter befinder sig i turen.

Lin-Kernighan's variable r-opt er lidt sværere at kode og at beskrive, men den giver bedre

resultater, og bruges typisk til at skaffe et godt bånd på den optimale tur, og en god tur for andre algoritmer at forbedre på, fx. branch & bound. Den lægger sig ikke fast på at undersøge alle r-opt for et givet r, men konstruerer grådigt en r-opt, indtil det ikke ville kunne betale sig at lade r stige mere. Undervejs er der blevet undersøgt både en 2-opt, en 3-opt, osv. op til en r-opt. Den bedste vælges og udføres, og der vælges et nyt punkt at starte med for en ny søgning.

Som illustration af Lin-Kernighan ses på figur 1.20 en tur, der tages under behandling af Lin-Kernighan. Et punkt vælges, og grådigt vælges de to første par af punkter, der skal have nye kanter ud fra sig. Dette giver en mulig 2-opt som midt på figuren. Det undersøges om der grådigt kan vælges et nyt par punkter, så den dermed tilknyttede 3-opt er kortere end den oprindelige tur. Den mulige 3-opt ses i næste figur. Hele tiden er den stiplede linje, den kant der skal tilføjes til de allerede valgte kanter, for at der kommer en r-opt ud af det. På de næste figurer ses så en mulig 4-opt og 5-opt, og så kan det antages, at der ikke fandtes en 6-opt (med 4 af kanterne og to punkter allerede givet), der gav en tur kortere end den oprindelige. Således er der altså 4 forbedrende r-opt'er at vælge imellem. Den bedste vælges, og heuristikken kan køres igen, med et nyt startpunkt.

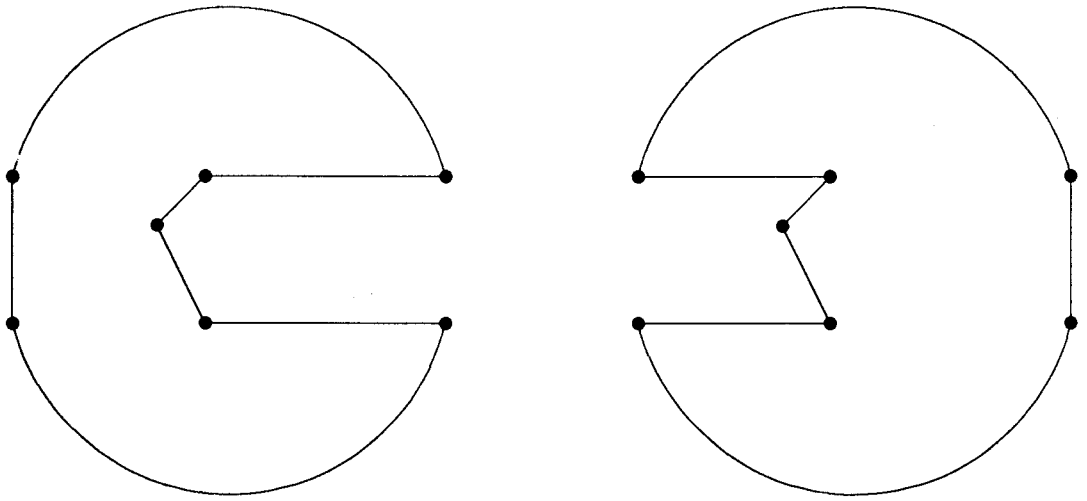
Bemærk at Lin-Kernighan er et specialtilfælde af en algoritme, der kan forsøge at løse andet end TSP.



Figur 1.20: Tur før, under og efter Lin-Kernighan.

En Or-opt er egentlig en 3-opt, men 2 af kanterne vælges, så der kun er 0-2 kanter mellem dem. Se figur 1.21. Den er opkaldt efter I.Or.

Først betragtes alle sæt af 3 kanter, hvor 2 af kanterne ligger 2 kanter fra hinanden (som på figuren). Med dette ekstra krav opnås 3-optimalitet via det nødvendige antal 3-opt. Dette gentages hvor 2 af kanterne har én kant imellem sig. Endelig gentages hvor de har 0 punkter imellem sig, dvs. de har en by fælles. Dette svarer til at tage en enkelt by ud af turen og sætte den ind et nyt sted.



Figur 1.21: Tur før og efter Or-opt.

For at have nogle resultater at sammenligne med senere, implementerede jeg både 2-opt og 3-opt, og kørte dem på graferne eil51 og lin318. Det gav disse tider og resultater.

eil51's optimale tur har længde 426. Ved 200 kørsler af 2-opt på eil51 fandt jeg ture af længder fra 429 til 477, i snit 451,64. Køretiden lå omkring et sekund. Ved 200 kørsler af 3-opt på eil51 fandt jeg ture af længder fra 428 til 456, i snit 439,59. Køretiden lå fra 24 til 36 sekunder, i snit 29,545.

lin318's optimale tur har længde 42.029. Ved 20 kørsler af 2-opt på lin318 fandt jeg ture af længden 44.316 til 47.232, i snit 45.556,25. Køretiden lå fra 352 til 378 sekunder, i snit 366,75. Ved 5 kørsler af 3-opt på lin318 fandt jeg ture af længder fra 43.468 til 44.264, i snit 44.042,4. Køretiden lå fra 60.535 til 65.273 sekunder, eller mellem 17,5 og 18 timer.

1.4.3 Analyseresultater.

Med de givne analysemetoder er man bl.a. kommet frem til:

Gold 80 På 5 problemer med 100 byer, alle euklidiske, kørtes de i skemaet nævnte algoritmer. Hvis algoritmen starter med et tilfældigt punkt, kørtes algoritmen med alle de mulige startpunkter, og det bedste resultat nævnes. For 2-opt er algoritmen kørt 25 gange, med forskellige startture, og bedste resultat nævnes. Blandede algoritmer er både konstruerende og forbedrende, en af disse tager 10 forskellige ture fra tilfældig indsættelse, beholder den bedste og kører 3-opt på den. Resultaterne nævnes som % fra bedst kendte løsning, og flere resultater er resummeret sammen.

(Algoritmerne nærmeste nabo og Christofides behandles senere i større detalje.)

Konklusionen var at de heuristikker, der konstruerer ture bedst, præsterer 5 til 7 % for lange ture, der er gode at arbejde videre på, og hurtige at fremstille.

2-opt og 3-opt kan i kvalitet sammenlignes med de nævnte tur-konstruerende heuristikker.

Blandede algoritmer giver jævnligt 2 til 3 %, men tager også længere tid. Gentag algoritmen for at få 1 til 2 %.

Algoritme(r)	Resultat
Nærmeste nabo Nærmeste indsættelse Billigste indsættelse	11,67 - 22,96
Christofides	3,93 - 14,44
Fjerneste indsættelse Tilfældig indsættelse Konveks omkreds	1,99 - 7,42
2-opt 3-opt	0,51 - 7,82
Blandede algoritmer	0,14 - 3,22

I *Arab 83* forsøges 12 ikke-euklidiske problemer med 10-120 byer løst.

Der konkluderes at, af nærmeste nabo, nærmeste indsættelse, fjerneste indsættelse og nærmeste tilføjelse giver fjerneste indsættelse den bedste tur for Lin-Kernighan at arbejde videre på.

Endnu en analyse fra *Gold 85b* giver at det konvekse hylster altid er en god tur at starte med, også for de algoritmer ovenfor der starter med noget andet. Desuden at 2-opt er dårligere end Or-opt, mens 3-opt og Or-opt kan sammenlignes, og Or-opt har bedre køretid.

En lidt anden type analyse foretages i *John 90*, hvor nogle af de samme algoritmer testes, men på meget store grafer, med lidt overraskende resultater. Se den næste tabel. Tallet foroven i hver søjle angiver antallet af byer i problemet. Først angives køretider, i sekunder (begge de følgende tabeller er tilnærmelsesvis sorteret med den bedste øverst). Den nævnte Christofides-variant erstatter den minimale pardannelse med en grådigt fundet, ikke nødvendigvis minimal pardannelse, og sparer på denne måde tid, men kan give dårligere resultater. Graferne er dannet ved at vælge n punkter tilfældigt i et enhedskvadrat, og bruge de euklidiske afstande. Der gælder at jo kortere køretid, jo flere forsøg for at checke om resultatet opnået er et gennemsnitsresultat. Dette gjaldt i høj grad ved de forsøg, der blev gentaget mange gange, at det var let at finde en god repræsentant for de observerede resultater.

For 2-opt og 3-opt var start-turen et resultat fra nærmeste nabo algoritmen, dette gav både bedre tider og resultater. Noget tilsvarende blev ikke gjort for Lin-Kernighan, fordi det ikke ville have samme effekt.

Algoritme	10^2	10^3	10^4	10^5
2-opt	0,4	5,0	55,0	746
3-opt	0,5	6,6	71,5	980
Nærmeste nabo	0,6	9,8	110,2	1031,2
Lin-Kernighan	1,1	16,7	340,2	17776
MST	2,4	32,1	356,8	9168,0
Christofides-var.	3,1	45,8	483,4	8151,0
Nærmeste indsættelse	3,4	44,3	543,6	10869,3
Fjerneste indsættelse	3,4	56,0	805,5	14925,3
Christofides	7,1	4053,1		

Der ses at nogle af algoritmerne er urealistisk langsomme, ja Christofides er slet ikke kørt på de største grafer, fordi køretiden er så lang, og så meget længere end de andres. Der er 2 grupper, hvor 2-opt, 3-opt og Nærmeste nabo tydeligvis er hurtigere end de andre.

Så angives de opnåede resultater i procent over bedst kendte nedre bånd for den optimale turs længde. Grænsen er et gæt på, hvad en evt. grænseværdi for kvalitet er.

Algoritme	10^2	10^3	10^4	10^5	grænse?
MST	36,9	39,0	39,5	39,9	40
Nærmeste indsættelse	23,0	26,5	26,5	27,1	27
Nærmeste nabo	29,3	25,1	24,2	23,8	24
Christofides-var.	11,6	19,0	19,1	19,4	19
Fjerneste indsættelse	8,3	12,5	13,3	13,6	13
Christofides	8,8	9,9			10
2-opt	6,2	6,4	6,4	6,5	6,5
3-opt	2,6	3,5	3,6	3,6	4
Lin-Kernighan	1,5	2,1	2,2	2,2	2

Her viser resultaterne, at der ikke nødvendigvis er sammenhæng mellem hvor lang tid en algoritme bruger, og hvor godt et resultat den kommer med. Nærmeste nabo er hurtig, men præsterer ikke strålende resultater. Det ses også, at opførslen på små grafer og store grafer er så forskellig, at man i hvert fald også bør undersøge de store grafer, for det er jo her man er nødt til at bruge heuristikker fremfor eksakte løsningsmetoder. 2-opt og 3-opt er en god kombination af hurtighed og kvalitet, mens Lin-Kernighan godt nok er god, men måske ikke tiden værd.

1.5 Analysemetoder.

Som før nævnt, kan man analysere heuristikker på flere forskellige måder. Jeg nævner her de 3 metoder, som de historisk dukkede frem, nemlig empirisk analyse, worst case analyse og probabilistisk analyse. Af disse 3 er worst case analyse lettest at demonstrere, derfor henvises til analysen af fx. MST senere. Probabilistisk (sandsynligheds-) analyse anvendes i forbindelse med både simuleret udglødning og genetiske algoritme, men jeg har også et par generelle resultater med her. Empirisk analyse belyses først, fordi den bruges på nogle heuristikker, der virker godt, men endnu ikke har givet specielt gode garantier med de 2 andre former for analyse. *Karp 85*

1.5.1 Empirisk analyse.

Denne metode kan altid bruges til at vurdere og sammenligne heuristikker, og kræver "bare" et rimeligt antal gode testproblemer (nedenfor bruges 15 forskellige). Hvis forskellige personer tester forskellige heuristikker, vil de typisk være interesseret i, at de har foretaget deres tests på samme problemer, derfor findes der også et bibliotek for såkaldte benchmark problemer, som jeg selv har hentet problemer fra.

Der er forskellige kriterier, man kan teste heuristikker efter, og kvaliteten af løsningerne er selvfølgelig den ene. Men også spøgsmål som Hvad er køretiden? Er den let at implementere? Er den flexibel? (dvs. duer til den mange typer TSP) og Er den simpel? (dvs. er den let at analysere, og er den indbydende at prøve at implementere) skal besvares. Typisk sammenligner man to eller flere heuristikker, for at finde den bedste eller konstatere at de måske er lige gode. Hertil anvendes statistiske metoder, der kun kigger på resultaterne.

Fortegnstesten er en meget simpel metode, der givet to heuristikker og deres løsning til n forskellige problemer, regner differencen mellem resultaterne ud (positivt fortegn når den ene heuristik er bedst, negativt når den anden er bedst), tæller hvor mange positive og negative fortegn der er, og regner ud, hvad sandsynligheden for denne fordeling af fortegn er, hvis metoderne skulle være lige gode. Som eksempel på denne analysemetode, vises her hvordan den virker på resultaterne fra tabel 1.5.1 (fra *Gold85b*), der også bruges senere. Her opremses længden af de løsninger, hver heuristik har fundet til hvert problem. Således er det bedst kendte resultat for problem nr.1 en længde på 310, mens heuristik x fandt en tur af længde 316.

Problem nr.	Bedst kendte resultat	Heur. x	Heur. y	Heur. z	Problem nr.	Bedst kendte resultat	Heur. x	Heur. y	Heur. z
1	310	316	326	331	9	150	172	162	194
2	339	367	367	418	10	258	416	356	407
3	275	289	291	313	11	253	355	339	364
4	274	320	290	350	12	275	302	302	364
5	370	417	383	475	13	395	424	430	501
6	295	316	324	356	14	424	560	564	655
7	312	357	325	355	15	544	592	560	560
8	144	171	164	186					

Tabel 1.5.1: Resultater for 3 heuristikker.

Hvis heuristik x og z skal sammenlignes kan man altså tælle hvor mange gange x giver bedre resultat end z. For problem 7, 10 og 15 giver z bedre resultat, de resterende 12 problemer løser x bedre. Sandsynligheden for at de med dette resultat skulle være lige gode (dvs. der er 50 % chance for at x er bedre end z i det enkelte problem) er 1.8 %, regnet ud med almindelig

sandsynlighedsregning:

$$\sum_{k=12}^{15} \binom{15}{k} (0.5)^k (0.5)^{15-k}$$

Eller med andre ord, med meget simpel matematik kan det afvises på signifikansniveau 98,2 %, at x og z skulle være lige gode. Her testes der for hypotesen at der kunne komme 12 eller flere bedste resultater ($12 \leq k \leq 15$) for den ene heuristik, når de er lige gode.

Der findes en mere avanceret version af denne test, hvor størrelsen af differencen også bruges, men hvor man stadig tester den simple hypotese: er disse 2 heuristikker lige gode. Denne test hedder Wilcoxon testen, og anvendes på data af formen, at alle resultater er givet som antal % ~~procent~~ over bedste kendte resultat, og resultaterne er x_i og y_i for heuristikkerne x og y på problem i.

Udregn for alle problemerne $|x_i - y_i|$. Hvis dette giver 0 for i glemmes dette problem. Sorter $|x_i - y_i|$ efter størrelse, og nummerer dem, så laveste $|x_i - y_i|$ får $|R_i|$=1, næstlaveste $|R_i|$=2 osv. Hvis nogle har samme $|x_i - y_i|$, nummerer dem da først tilfældigt, fortløbende, og ret det siden, så de alle får snittet af de først givne numre. (Kunne fx. give at 2 fik nr.1,5, mens den tredje fik nr.3.)

Hvis $x_i - y_i$ er positiv er R_i positiv, ellers negativ. $W = \sum_i R_i$.

Test hypotesen x og y lige gode, på signifikansniveau α:

med udtrykket	når alternativ hypotese er
$W > W_{1-\frac{\alpha}{2}} \vee W < W_{\frac{\alpha}{2}}$	x og y ikke lige gode
$W > W_{1-\alpha}$	x dårligere end y
$W < W_{\alpha}$	x bedre end y

Endvidere: $n \geq 10 \rightarrow W_\alpha \approx Z(\alpha)\sqrt{n(n+1)(2n+1)/6}$ hvor Z er standardnormalfraktilen, der giver et areal af størrelse α til venstre for $Z(\alpha)$. For de givne resultater fås at $R_3 = -1, R_2$ er slet ikke med, og $R_1 0 = 13$ som det højeste. $W = 61$. Da $W_{1-0,05} = 47{,}08$ udledes det at heuristik x er dårligere end heuristik y på et signifikansniveau 0,95, eller W ville kun være større end 47,08 1 ud af 20 gange.

Hvis man gerne vil sammenligne mere end to heuristikker og finde den bedste findes der en anden test, der egner sig bedre, nemlig Friedman testen, der igen er mere avanceret end Wilcoxon, men bygger på det samme. Den konstaterer primært, om der er forskel på heuristikkernes kvalitet, og dermed om der måske er en bedste heuristik.

Som sidebemærkning kan det siges, at disse forskellige tests ikke havde problemer med at fastslå, at af de ovenstående heuristikker er y bedre end de 2 andre og x bedre end z.

Der foreslås endnu en måde at vurdere heuristikker på. Denne giver et enkelt tal for hver heuristik, nemlig den forventede anvendelighed, og siden kan man så sammenligne tallene, og vælge heuristikken med det højeste tal som den bedste. Der kan man altså let tilføje en ny heuristik, og udregne et tal for den, for at se om den kan sammenlignes med de gamle. Metoden fokuserer på, at gode metoder giver lavt gennemsnit og sjældent giver meget dårlige resultater, jo bedre disse krav opfyldes, jo højere tal.

Der udregnes $b = \frac{s^2}{\bar{x}}$ og $c = (\frac{\bar{x}}{s})^2$ og med β, α og t givet ($t < \frac{1}{b}$) er den forventede ~~sandsynlighed~~ $\alpha - \beta(1-bt)^{-c}$. Her er s^2 den almindelige varians, og $\bar{x}$ middelværdien, dvs. gennemsnittet. For heuristik x udregnes $\bar{x}$ til 17,29, $s = 15{,}31$, $b = 13{,}56$ og $c = 1{,}28$. Her vælges $\alpha = 600, \beta = 100$ og $t = 0{,}05$. Dermed bliver den forventede anvendelighed 173,5. For hhv. y og z bliver den 354,0 og 11,7, altså en bekræftelse af de fundne resultater.

1.5.2 Probabilistisk analyse.

Probabilistisk analyse eller sandsynlighedsanalyse antager at et givet problem opfylder visse sandsynlighedskrav og kommer derfra frem til visse resultater, nogle med sandsynlighed 1, andre

med en sandsynlighed der går mod 1 når antallet af byer går mod uendelig.

Et typisk problem at analysere er det, hvor alle byer er placeret i en d-dimensional enhedsterning, dvs. alle koordinater er mellem 0 og 1, og der er d af disse kordinater.

Et af resultaterne man kan nå frem til er, at $lim_{n\to\infty} OPT(I)\ n^{\frac{d-1}{d}} = c_d$, dvs. der er en grænseværdi, der kun afhænger af d. Her er I et givet problem hvor alle punkterne opfylder sandsynlighedskravet, at punkterne er jævnt fordelt i enhedsterningen.

Et andet resultat er, at med dette krav opfyldt er $OPT(I) \leq d\ n^{\frac{d-1}{d}} + \delta_d\ n^{\frac{d-2}{d-1}}$, hvor δ_d kun afhænger af d.

Et andet resultat opnås i planen, altså for d=2, nemlig at $OPT(I) \leq \sqrt{2n} + 1.75$. Der findes tilsvarende værdier for andre d'er. *Few 55* Da jeg bruger et kvadrat med sidelængde $3\sqrt{n}$ bliver hele formlen for mig $(\sqrt{2n} + 1.75)3\sqrt{n}$.

For 2-opt (3-opt og andre k-opt) findes der meget få resultater, der forudsiger noget om antallet af iterationer, så det ikke rigtig er til at sige, om disse algoritmer kører i polynomiel tid. I *Chan 94* findes der dog en opsamling af nye og gamle resultater, hvor jeg her nævner nogle.

For 2-opt gælder, at en 2-optimal tur for et symmetrisk problem højst er lige så lang som snittet af alle ture. Et nyere resultat siger desuden, at hvis trekantuligheden gælder, er en 2-optimal tur højst en faktor $4\sqrt{n}$ længere end den optimale. (n er antallet af punkter i grafen.) Begge disse resultater er worst-case opførsel.

Der eksisterer par af grafer og startture, der kræver et eksponentielt antal iterationer af 2-opt for at finde en 2-optimal tur. Tilsvarende for 3-opt. For et tilfældigt genereret euklidisk problem, i enhedskvadratet, forventes $O(n^{10}logn)$ iterationer.

Der findes mere om probabilistisk analyse under simuleret udglødning, nemlig den del af det der analyserer heuristikker. For en algoritme, der bygger på tilfældigheder (tilfældig start-tur, sandsynlighed for forandring osv.) kan man ofte sige noget om med hvilken sandsynlighed der opnås gode resultater.

Kapitel 2

TSP for grafer med pæne egenskaber.

Velan, vi ved, hvorhen
den korte vej os leder.
(DDS 584)

Indtil nu har vi set analyse-metoder, meget simple heuristikker og meget teoretiske resultater. Nu vil jeg gerne vise nogle mere interessante resultater, for algoritmer der arbejder på mindre klasser af TSP-problemer, men så også kan komme med mere tætsluttende bånd. Her en oversigt over hvilke bibetingelser der er opfyldt eller hvilke algoritmer der bruges, kombineret med hvilke resultater der så opnås.

Bemærk at nærmeste nabo reglen bruges flere gange, og den første gang ikke giver noget specielt pænt resultat.

Heriblandt er Demidenko-betingelserne, hvis analyse så vidt vides for første gang er så omfattende oversat fra russisk, og derfor suppleres med et fyldigt appendix, hvor hovedsætningen (at det er muligt at finde den optimale tur i en Demidenko-matrix) gennemgås i sin helhed, suppleret med andre resultater fra kilden.

Algoritme/bibetingelser	Worst case
Nærmeste nabo reglen, trekantulighed	∞ !
MST-algoritmen, trekantulighed, symmetri	$2 \cdot OPT$
Chrisofides' algoritme, trekantulighed, symmetri	$1,5 \cdot OPT$
Halin-graf	OPT
Øvre trekantsmatrice, hjælp fra minimal tildeling	OPT
Søjlesorteret, positiv matrix, hjælp fra minimal tildeling	$OPT + max_j\{c_{1j}\}$
Cirkulær matrix, hjælp fra nærmeste nabo reglen	$OPT + max\{c_{ij}\} - min\{c_{ij}\}$
Korteste pyramidetur, distributionsmatrix eller Demidenkobetingelserne	OPT

2.1 Nærmeste nabo reglen.

Blandt de grådige algoritmer, man kunne forestille sig for TSP, er nærmeste nabo reglen nok den første man kommer på. Denne fremstilling stammer fra *John 85b*, der igen støtter sig til *Rose 77*.

2.1.1 Bibetingelser.

- Trekantulighed.

2.1.2 Resultater.

Nærmeste nabo reglen er en grådig algoritme, der er meget simpel:

Algoritme 2 *Nærmeste nabo reglen(graf): tur*

Vælg en tilfældig by, a_1, til stien (a_1).
Så længe stien er $(a_1, a_2, \ldots a_k)$, hvor $k < n$
    vælg da en by, der er tættest på a_k, og ikke allerede er med i stien, som a_{k+1}
Turen er da $(a_1, a_2, \ldots a_n)$.

Algoritme slut

Intuitivt er denne algoritme ikke ret god, fordi den ikke ser fremad, og ikke kan forhindre at man bliver nødt til at vælge nogle dyre kanter til sidst, måske også meget tidligt.

Hvor andre algoritmer gør god brug af, at trekantuligheden gælder, kan man nemlig stadig snyde nærmeste nabo reglen. Der gælder nemlig følgende, når $NN(I)$ er længden af den tur, nærmeste nabo reglen konstruerer, og $OPT(I)$ er længden af en optimal tur.

Sætning 1 *For alle $r > 1$ eksisterer der en instans af TSP med (vilkårligt stort) n byer, hvor trekantuligheden er opfyldt, og $NN(I) \geq r\ OPT(I)$, dvs, nærmeste nabo reglen kan komme med en tur, der er r gange længere end den optimale.*

Bevis: Fremgår af appendix.

Dvs. nærmeste nabo reglen ikke har nogen specielt god garanti, med kun den angivne bibetingelse. Senere anføres der andre bibetingelser, under hvilke den giver bedre resultater.

2.1.3 Komplexitetsanalyse.

Algoritmen vælger ved hvert punkt en kant ud af n, og bliver dermed i alt $O(n^2)$. Dog kan algoritmen i det euklidiske tilfælde implementeres hurtigere, i $O(n\ logn)$ eller $O(n\ log^2 n)$.

2.2 MST–algoritmen.

Ideen i denne algoritme er meget simpel, og bruger undervejs bl.a. en nederste grænse for, hvor lang den optimale tur er. Denne algoritme er blevet opdaget og genopdaget tit nok til at man godt kan sige, at den er folklore. I dette tilfælde har jeg støttet mig til *John 85b*.

2.2.1 Bibetingelser.

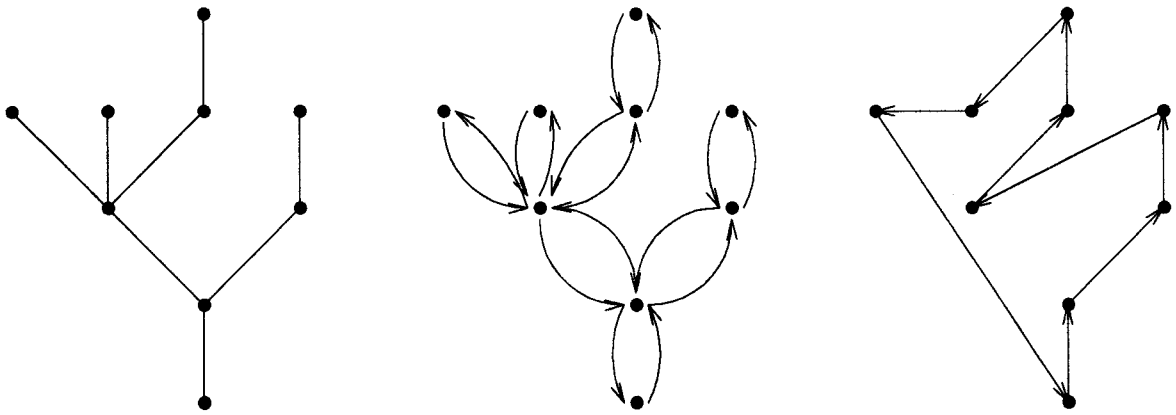
- Positiv.
- Trekantulighed.
- Symmetri.

2.2.2 Resultater.

Et udspændende træ i en graf med n punkter har $n-1$ kanter, for at kunne dække alle n punkter, og være sammenhængende. Det mindste udspændende træ (det med lavest omkostning) kan findes i $O(n^2)$, og kaldes MST / minimum spanning tree.

Hvis man fordobler alle kanter i μ, kan man lave en Euler-tur ν, vha. dybde-først-søgning, der angiver hvilken rækkefølge kanterne kan tages i. ν skal altså modificeres for at der kommer en rigtig tur ud af det.

Hvis man gennemløber ν, og hver gang man kommer til en allerede besøgt by skyder genvej til den næste ubesøgte by, vil hver by kun blive besøgt en gang, og resultatet bliver en tur.



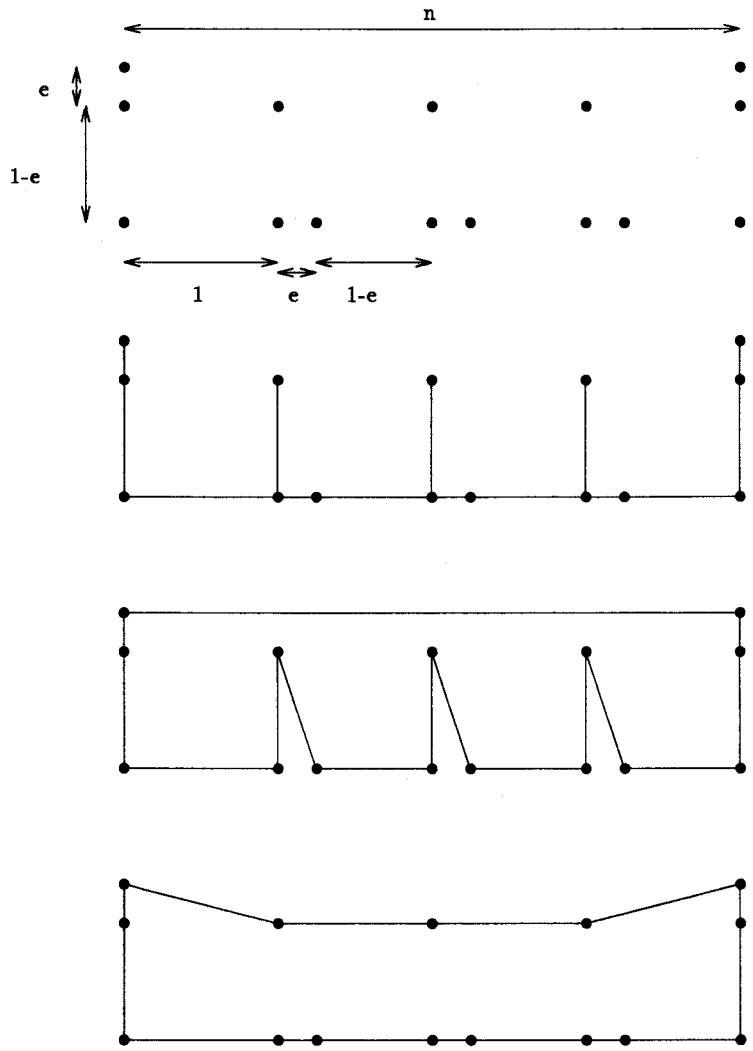
Figur 2.1: Eksempel på MST-algoritmen.

Algoritme 3 *MST-algoritmen*

Find MST μ
Fordobl kanter, og find Euler-turen ν
Gå genveje i ν, og find turen τ

Algoritme slut

Som eksempel på algoritmen i funktion ses det euklidiske problem i figur 2.1. Der vises hhv. μ, ν og τ, hvor roden er det nederste punkt, og søgningen foregår mod uret rundt. Hvis den nederste kant i træet regnes til at være af længde 1, er længden af τ ca. 14,08. Dette tal bruges senere til sammenligning. μ har en samlet længde på ca. 9,66.



Figur 2.2: Eksempel på MST-algoritmen.

Et andet eksempel på algoritmen er også euklidisk, og viser, at båndet på turens længde faktisk kan opfyldes, dvs. den fundne tur er dobbelt så lang som den optimale. Se figur 2.2.

Af denne figur fremgår et problem, der kan varieres i n og ϵ. Fra oven og ned ses problemet (euklidisk), et MST, en tur, der dannes derudfra, og den optimale tur. MST-turen har en længde af ca. $2n$ vandret og $2n$ lodret, i alt $4n$. Den optimale tur har ca. en længde af $2n+2$. Jo tættere ϵ er på 0, og jo højere n er, jo bedre passer de angivne længder, og jo bedre passer påstanden om, at MST-turen er dobbelt så lang som den optimale.

Sætning 2 *MST-algoritmen giver en tur, der højst er dobbelt så lang som den optimale.*

Bevis: For alle ture gælder, at hvis man smider en vilkårlig kant væk, har man et udspændende træ, med lavere eller samme omkostning. Derfor må der specielt gælde $c(\mu) \leq OPT(I)$.

Der gælder $2c(\mu) = c(\nu)$.

Da τ findes ved at gå genveje i ν, må der gælde $c(\tau) \leq c(\nu)$. Genvejene ved at springe punkter over er jo rent faktisk kortere, pga. trekantuligheden.

Kædes dette sammen fås $c(\tau) \leq 2OPT(I)$.

QED

2.2.3 Komplexitetsanalyse.

Algoritmen domineres af første trin, der afvikles i højst $O(n^2)$. (Denne del kan afvikles i $O(m + n\ logn)$, hvor m er antallet af kanter i grafen. For at m ikke bare skal være n^2 kræves der en udtynding af grafen, dette kommer jeg ikke yderligere ind på.) De to øvrige trin er hver $O(n)$.

2.3 Christofides algoritme.

I MST-algoritmen benyttes en meget simpel ide, her forfines metoden lidt. Kilderne er (ifølge *John 85b*) *Chri 76* og *Corn 78*.

2.3.1 Bibetingelser.

- Positiv.
- Trekantulighed.
- Symmetri.

2.3.2 Resultater.

Det fordoblede træ i MST-algoritmen er en Euler-graf, og ud fra denne konstrueres en tur. Hvis man kunne finde en bedre Euler-graf at modificere, ville man kunne få en bedre garanti for turens længde. Dette gør Christofides' algoritme.

En minimum perfekt pardannelse er en mængde af kanter, dannet ud fra en mængde af punkter, så hvert punkt er endepunkt for nøjagtig en kant, og så summen af kanternes vægt er mindst mulig.

Algoritme 4 *Christofides' algoritme*

Find MST μ i grafen
Alle punkter i μ med lige grad "smides væk" og:
For resten af punkterne konstrueres en minimum perfekt pardannelse λ
Kantmængden af μ og λ tilsammen er en Euler-graf
Find Euler-tur i Euler-grafen
Lav genveje, resultat tur τ

Algoritme slut

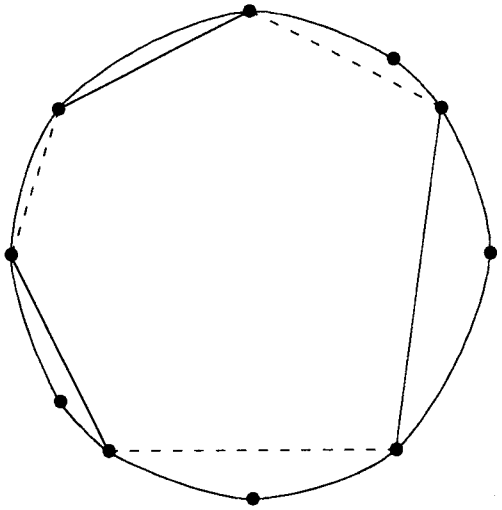
Sætning 3 $c(\tau) < \frac{3}{2}OPT(I)$.

Bevis: Af figur 2.3 fremgår 2 mulige pardannelser med alle punkter med ulige grad i træet. Punkterne på cirklen er grafens punkter, og cirklen er en optimal tur gennem dem. Dem med ulige grad er forbundet yderligere, den ene pardannelse er stiplet, den anden fuldt optrukket.

De to pardannelser kaldes λ_1 og λ_2. De kan tilsammen betragtes som fremkommet ved at have skudt genveje i turen.

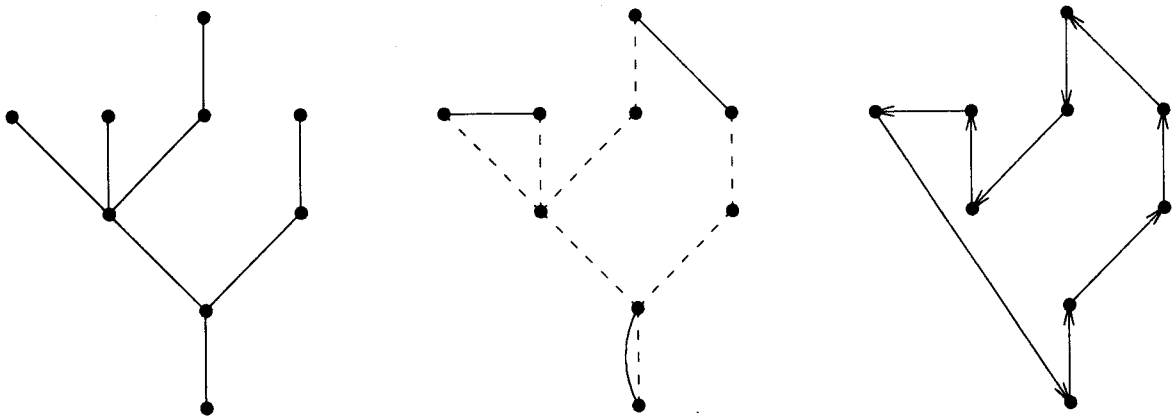
$c(\lambda_1) + c(\lambda_2) \leq OPT(I)$
$\Rightarrow c(\lambda_1) \leq \frac{1}{2}OPT(I) \vee c(\lambda_2) \leq \frac{1}{2}OPT(I)$
$\Rightarrow c(\lambda) \leq \frac{1}{2}OPT(I)$

$c(\mu) < OPT(I)$
$\Rightarrow c(\mu + \lambda) < \frac{3}{2}OPT(I)$
$\Rightarrow c(\tau) < \frac{3}{2}OPT(I)$
QED



Figur 2.3: 2 mulige pardannelser, og den optimale tur.

Som eksempel på hvordan Christofides algoritme virker, ser vi på samme graf som for MST-algoritmen, se figur 2.4. Først ses MST for grafen, så træet plus den minimale pardannelse for punkter med ulige grad, og endelig en tur, konstrueret som for MST-algoritmen, dvs. den starter forneden. Til sammenligning har denne tur længde 12,84.

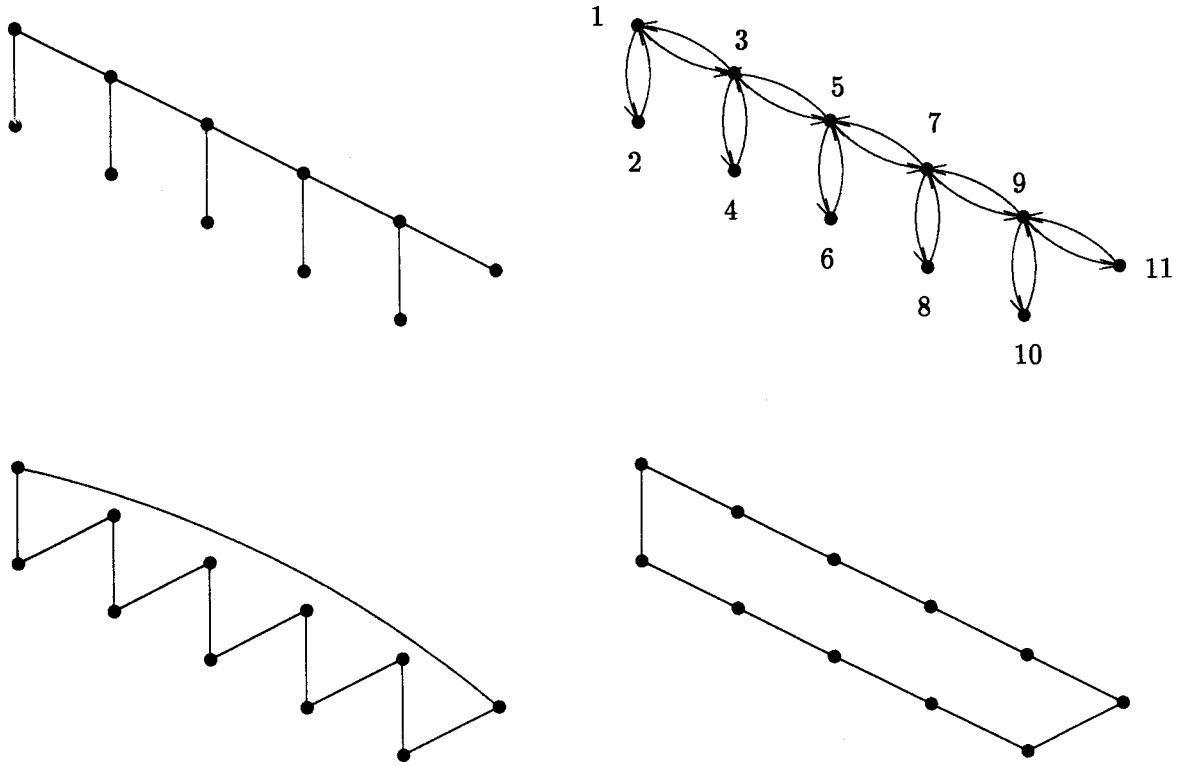


Figur 2.4: Eksempel på Christofides algoritme.

På figur 2.5 ses kanterne af en graf, hvor alle kanter bortset fra den længste har længde 1, og den længste er 5 lang. Først ses et MST for grafen, og så ses en dybde-først søgning (den deraf følgende nummerering af punkterne angives, så det kan ses i hvilken rækkefølge punkterne mødes), startende i øverste venstre punkt og gående mod uret rundt. Heraf konstrueres den til venstre viste tur, hvis længde er 15, hvor den til højre viste tur er optimal, og kun har længde 11. Og $\frac{3}{2}11$ rundes ned til 16.

Generelt kan grafen i figur 2.5 laves vilkårligt lang, således at man med $2n+1$ punkter får en tur på $3n$ (en kant på n og $2n$ kanter på 1) sammenlignet med en optimal tur på $2n+1$. Dermed kan man komme en vilkårlig brøkdel tæt på båndet på en faktor $\frac{3}{2}$, men aldrig ramme

båndet helt.



Figur 2.5: Eksempel på dårlig Christofides-tur.

2.3.3 Komplexitetsanalyse.

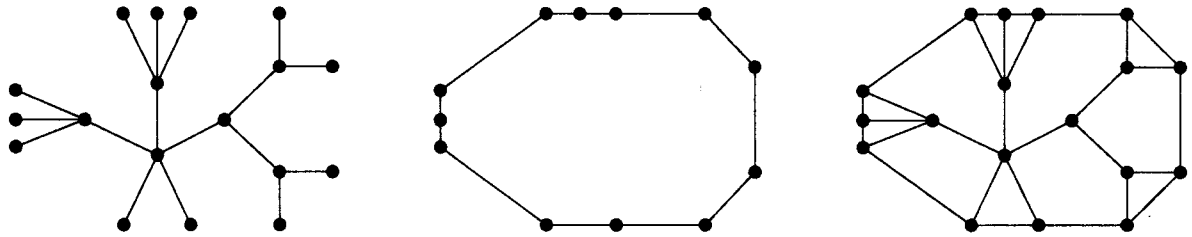
Algoritmen domineres af, at den minimale pardannelse findes i $O(n^3)$. (Accepteres en grådig fundet pardannelse, af dårligere kvalitet, fås en køretid på $O(n\ logn)$ eller $O(n\ log^2 n)$. MST findes i $O(n^2)$, og resten udføres i $O(n)$.

2.4 Halin–grafer.

Halin-grafen blev opfundet af R.Halin som et eksempel på en kant-minimal plan 3-sammenhængende graf. Ifølge *Gilm 85* er disse resultater først set i *Corn 85*.

2.4.1 Bibetingelser.

En Halin–graf er en kombination af et træ, der omfatter alle punkter, og en kreds (egentlig et hylster) gennem alle bladene i træet (punkter med kun en kant ind til sig), efter at træet blevet lagt i planen. Denne graf kan altid tegnes i planen, uden at kanter krydser hinanden. Bemærk at i den færdige graf skal alle punkter have mindst grad 3, således at specielt alle ikke–blade i træet har mindst grad 3.



Figur 2.6: Et træ, en kreds af bladene, og den resulterende Halin–graf.

Figur 2.6 er et eksempel på en Halin–graf, først ses træet, så kredsen gennem alle bladene, og så kombinationen.

- $c_{ij} = c_{ji}$ Kanterne er uden retning.
- $c_{ij} = \infty$ er muligt, det er ikke en komplet graf.

2.4.2 Resultater.

Det nævnes uden bevis, at det er muligt at finde en tur gennem grafen, også selvom en enkelt kant fjernes. Ligeledes er det muligt at lave en tur, en given kant indgår i.

Definition 2 *En Halin–graf er et hjul hvis der kun er et punkt i grafens træ, der ikke er et blad.*

Sætning 4 *Det er nemt at finde en minimal tur i et hjul.*

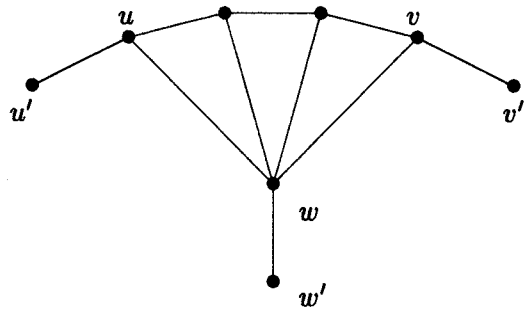
Bevis: En vilkårlig tur i et hjul vil bestå af alle kredsens kanter bortset fra en enkelt, da denne kant er erstattet af to kanter ind til hjulets midte. Find derfor den kant, der billigst kan erstattes. Hvis midten kaldes w, erstattes (u, v) i kredsen af (u, w) og (v, w), således at $c_{uw} + c_{vw} - c_{uv}$ minimeres.
QED

Definition 3 *En 3-kants-snit-mængde er en mængde bestående af 3 kanter, således at hvis alle 3 kanter blev fjernet fra grafen, ville denne gå i to stykker.*

Sætning 5 *Hvis ikke Halin-grafen er et hjul, kan den reduceres til en mindre graf, og i sidste ende til et hjul. En reduceret graf vil indikere samme løsning for de fælles kanter, som den oprindelige graf ville.*

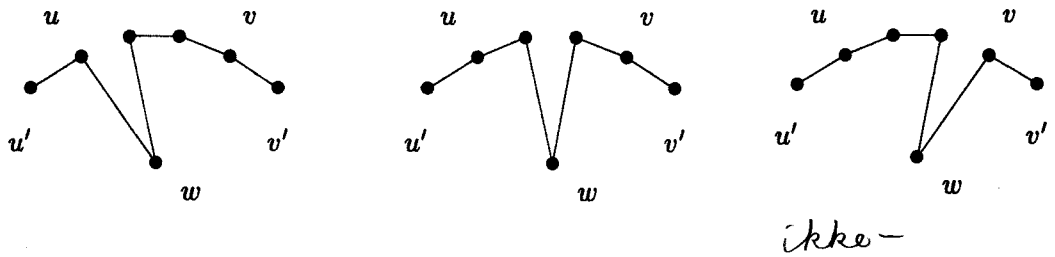
Dette betyder altså at man kan finde den optimale tur i en reduceret graf, og så konkludere at de fælles kanter er med i den optimale tur for den ureducerede graf også.

Bevis: Hvis Halin-grafen ikke er et hjul, er der en 3-kants-snit-mængde i den. Da en tur skal gennem begge komponenterne, der ville blive tilbage, hvis de 3 kanter blev fjernet, må turen benytte sig af netop 2 af de 3 kanter. Og hvor er denne mængde så? Lad os et øjeblik smide alle kanter og punkter i kredsen væk, og se på den tilbageværende graf. Denne graf vil stadig have mindst 2 blade (invariant for træer med kanter i), vælg et af disse blade. Lad os kalde dette blad w. I Halin-grafen er dette punkt et punkt, der dels står i forbindelse med kredsen via mindst 2 kanter (så graden 3 opnås), dels i forbindelse med netop ét andet ikke-blad i træet. Se på alle de punkter, w er forbundet til. Et af dem (w') ligger inde i grafen, resten (lad os kalde de to yderste, der ligger i kredsen u og v) ligger mellem u' (forbundet til u) og v' (forbundet til v). Et eksempel herpå er figur 2.7.



Figur 2.7: En vifte.

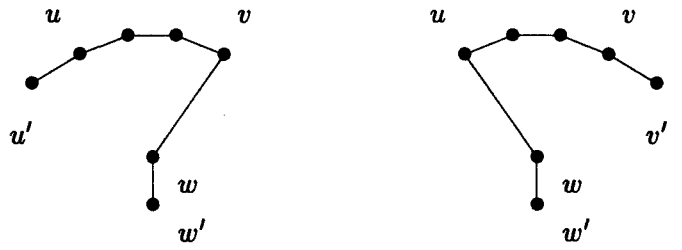
Lad os kalde mængden af punkter i kredsen forbundet til w, og w selv, viften udspændt af w. Heri medregnes også kanterne mellem disse punkter. De 3 kanter, der forbinder en vifte til resten af grafen, er netop en 3-kants-snit-mængde.



Figur 2.8: De mulige Hamilton-stier i viften, trivielle.

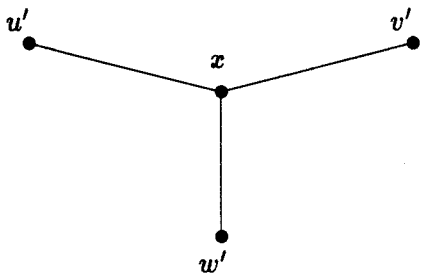
Som nævnt ovenfor må en tur netop benytte sig af 2 af de 3 kanter $(u,u'),(v,v'),(w,w')$. Indenfor viften kan man for hver af disse kombinationer finde den korteste Hamilton-sti mellem de givne kanter, og for to af disse er denne ovenikøbet triviel. Hamilton-stierne i figur 2.7 fremgår af figur 2.8 og figur 2.9.

Ligesom med hjulet, drejer det sig for det ikke-trivielle tilfælde om at finde ud, hvilken kant det er billigst at erstatte med to kanter ind til w.



Figur 2.9: De mulige Hamilton-stier i viften, ikke-trivielle.

Nu kan vi erstatte viften med et enkelt punkt, lige som på figur 2.10, og ændre længderne på kanterne i 3-kants-snit-mængden, så længderne af de passende Hamilton-stier fremgår, så man i grafen både med viften og med det enkelte punkt ville vælge samme vej.



Figur 2.10: Det nye punkt, efter reduceringen.

Lad $c(u,v), c(u,w)$ og $c(v,w)$ være udgifterne forbundet med de korteste Hamiltonstier, der starter og slutter i disse punkter.

$c_{xu'} + c_{xv'} = c_{uu'} + c_{vv'} + c(u,v)$

$c_{xu'} + c_{xw'} = c_{uu'} + c_{ww'} + c(u,w)$

$c_{xv'} + c_{xw'} = c_{vv'} + c_{ww'} + c(v,w)$

Dette ligningssystem af 3 ligninger med 3 ubekendte lader sig nemt løse, så vi kan konstruere den nye graf.

QED

Sætning 6 *Det er muligt at finde den korteste tur i en Halin-graf.*

Bevis: Benyt ovenstående reducering, indtil grafen er et hjul. Find så den korteste tur. Gå baglæns gennem reduceringerne, og find hver gang et punkt erstattes med en vifte, den sti gennem viften, der forbinder de allerede valgte kanter. Da man hele tiden har valgt den korteste vej, og startede med en korteste tur, vil slutresultatet blive den korteste tur i den oprindelige graf.

Grafen bliver faktisk reduceret til et hjul, fordi man ved hver reducering mister nogle punkter, deriblandt altid ét i træet. Til sidst vil træet kun bestå af et punkt.

QED

2.4.3 Komplexitetsanalyse.

Det er muligt at finde den optimale tur i en Halin–graf.

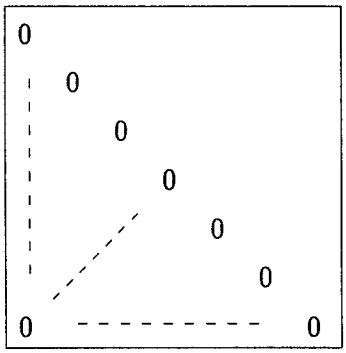
Da det i løbet af reduceringen er muligt at finde en vifte (løb fx. kredsen igennem, $O(n)$) og reducere denne ($O(n)$ at finde $c(u,v)$, lineært at løse de tre ligninger) i $O(n)$, og dette skal gøres $O(n)$ gange (en gang for hvert ikke–blad i træet omtrent), kan det gøres i $O(n^2)$.

2.5 Øvre trekantsmatricer.

Her gives et eksempel på en matrix-klasse, der kan løses ved at kigge på det beslægtede problem, at finde korteste tildeling. Ifølge *Gilm 85* stammer resulaterne fra *Lawl 71*. Topologisk sortering kommer fra *Corm 90*, s.477-79 og 485-87.

2.5.1 Bibetingelser.

- $i \geq j \Rightarrow c_{ij} = 0$. På denne måde bliver hele nederste venstre trekant af matricen nuller, som på figur 2.11.



Figur 2.11: En øvre trekantsmatrix.

2.5.2 Resultater.

Lemma 7 *Lad C være en øvre trekantsmatrix, og φ opfylde at $\varphi(n) = 1$, og er optimal under denne ekstra betingelse. Da er $c(\varphi)$ en nedre grænse for $c(\tau)$, hvor τ er en optimal tur.*

Bevis: Vi kalder en kant (i, j) bagvendt, hvis $i \geq j$. Dermed er $c_{ij} = 0$. I den optimale tur τ fjernes alle bagvendte kanter fra den del af turen, der går fra n til 1. Specielt bliver der så en sti fra 1 til n, men derudover opstår der en masse små stier, der går fra j til i, med $j \leq i$. Hver af disse stier laves nu om til en deltur ved at tilføje den bagvendte kant (i, j). Disse delture er en tildeling, vi kalder φ'. Da alle de bagvendte kanter, der blev fjernet og tilføjet, havde længden 0, er $c(\varphi') = c(\tau)$. Desuden er φ' bare en mulig tildeling, hvor $\varphi'(n) = 1$, og muligvis længere end den optimale. Derfor bliver den optimale tildeling kortere end den optimale tur, som der også står i sætningen.

QED

Sætning 8 *Hvis C er en øvre trekantsmatrix, og φ er en optimal tildeling, under betingelsen $\varphi(n) = 1$. Da er $c(\varphi)$ også længden af en optimal tur τ, der kan konstrueres fra φ.*

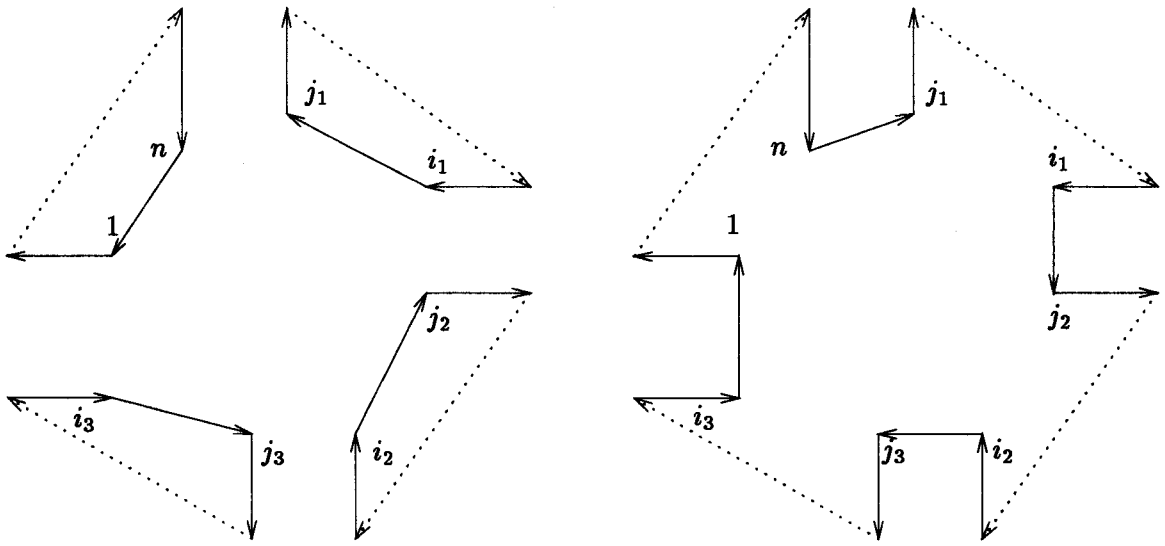
Bevis: Vi ved allerede, at en optimal tur vil være mindst lige så lang som tildelingen, så hvis vi kan konstruere en tur, der er lige så lang som en optimal tildeling, må det være en optimal tildeling.

Hvis φ er en tur, så lad $\tau = \varphi$, og sætningen er trivielt opfyldt.

Ellers består φ af $s \geq 2$ delture, der hver indeholder mindst en bagvendt kant. Fjern kanten $(n, 1)$ fra den relevante deltur, og en tilfældig bagvendt kant fra hver af de andre delture. Resultatet er et antal stier, en fra 1 til n, de andre sorteres, så de går fra j_1 til i_1, j_2 til i_2, ..., j_{s-1} til i_{s-1}, hvor $j_1 > j_2 > \cdots > j_{s-1}$ og $i_k \geq j_k > j_{k+1} (1 \leq k \leq s-2)$ samt $i_{s-1} \geq j_{s-1} > 1$. Tilføj nu de bagvendte kanter $(n, j_1), (i_1, j_2), \ldots, (i_{s-2}, j_{s-1}), (i_{s-1}, 1)$. Resultatet bliver en tur, τ, med samme længde som φ.

QED

Hvis vi på figur 2.12 har en optimal tildeling til venstre, konstrueres af denne turen til højre.



Figur 2.12: En tildeling, og en deraf konstrueret tur.

2.5.3 Permuterede øvre trekantsmatricer.

Hvis der iøvrigt er nuller nok i en matrice, er det muligt at den kan permuteres til en øvre trekantsmatrice ved at omnummerere byerne, og dermed bytte om på rækker og søjler tilsvarende. Det kan testes, om dette kan lade sig gøre, som jeg vil demonstrere nedenfor.

En matrix repræsenterer en komplet graf, men hvis kanter med længde 0 smides væk er det muligt, at punkterne kan sorteres således, at de lagt på en linje kun vil have kanter mellem sig der vender den ene vej. Dette hedder at lave en topologisk sortering. (Dette svarer til, at vi kan lade den nederste venstre trekant i matricen fylde med kanter med længde 0, efter omnummerering.) Hvis dette ikke kan lade sig gøre, er en topologisk sortering ikke mulig, dvs. der vil blive bagvendte kanter.

(Nedenstående som *Corm90*, s.477-79 og 485-87.)

Algoritme 5 *Topologisk sortering(graf): liste af punkter*

Kald Dybde–først–søgning på grafen for at beregne "sluttiden" for de forskellige punkter.
Her er sluttiden det samme som f(u) for punktet u, som defineres nedenfor.
Sorter punkterne efter sluttid, det senest sluttende punkt først.

Algoritme slut

Et punkts sluttid betegner hvornår man sidste gang støder på punktet ved en dybde–først–søgning.

Algoritme 6 *Dybde–først–søgning(graf)*

$\forall u$, punkter i grafen
    afmærk punktet som ubesøgt
    $\pi(u)$ = NIL (* punktet har ikke nogen forælder i dybde–først–søgnings–træet *)
tid = 0
$\forall u$, punkter i grafen
    hvis u er ubesøgt
        kald Dybde–først–søgnings–besøg(u)

Algoritme slut

Algoritme 7 *Dybde–først–søgnings–besøg(punkt u)*

afmærk punktet som under besøg
tid = tid + 1
o(u) = tid (* dette er tidspunktet for punktets opdagelse *)
$\forall v$, u's naboer (* der er en kant (u, v) *)
    hvis v er ubesøgt
        $\pi(v) = u$
        kald Dybde–først–søgnings–besøg(v)
afmærk u som besøgt
tid = tid + 1
f(u) = tid (* dette er tidspunktet, hvor punktet forlades *)

Algoritme slut

Lemma 9 *En graf med retning på kanterne er acyklisk hviss en topologisk sortering ikke giver nogle bagvendte kanter.*

Bevis: Det bevises nedenfor, at i en topologisk sortering bliver punkterne nummereret, så en ane har et højere nummer end sit afkom. Hvis der er en bagvendt kant skaber denne en cykel, fordi den går tilbage til en ane, og derfor sammen med stien fra anen til afkommet skaber en cykel. Så vi skal bare se, at aner virkelig får højere nummer end afkom.

Mens et afkom er under besøg, kan anen enten være ubesøgt, under besøg eller besøgt. Hvis anen er besøgt, er vi af en eller anden grund ikke kommet helt ned til afkommet under DFS–besøg, modstrid med dennes opbygning. Hvis anen er under besøg, vil afkommet blive besøgt (og dermed få sin sluttid) før vi kommer tilbage til anen, så her passer det. Og hvis anen er ubesøgt, vil afkommet blive besøgt (og få sin sluttid) før anen bliver under besøg, og får sin starttid, så her passer det også.

QED

Sætning 10 *Topologisk sortering algoritmen giver faktisk det ønskede resultat.*

Beviset herfor udelades, det kan findes s.487 i *Corm 90.*

2.5.4 Transformerede øvre trekantsmatricer.

En matrice kan også transformeres på den måde, at man givet to vektorer, $\mathbf{a} = (a_1, \ldots, a_n)$ og $\mathbf{b} = (b_1, \ldots, b_n)$ til alle elementer c_{ij} lægger a_i og b_j. Hvis vi har en matrix, der ikke umiddelbart er en øvre trekantsmatrix, kan det være interessant at undersøge, om en transformation af denne art, kan lave en øvre trekantsmatrix ud af den. De to matricer giver nemlig samme optimale tur, da forskellen i omkostning for alle permutationer er $\sum_{i=1}^{n} a_i + b_i$.

Hvis det er muligt at lave en permuteret øvre trekantsmatrix, skal en af byerne nummereres 1, og en anden nummereres n. I by 1's søjle skal der være nuller, fordi der ingen kanter må gå til 1 med længde forskellig fra 0. Tilsvarende må der i den n'te række kun være nuller. Med disse to informationer kan vi finde $\mathbf{a}$ og $\mathbf{b}$ (afgør fx. at når $a_n + b_1 = c_{n1}$ er $a_n = 0$ og $b_1 = c_{n1}$), og trække dem fra overalt. Slutresultatet er entydigt, fordi i C' (den nye matrix) er $c'_{ij} = c_{ij} - a_i - b_j = c_{ij} - (c_{i1} - b_1) - (c_{nj} - a_n) = c_{ij} + c_{n1} - c_{i1} - c_{nj}$. Herefter tester vi, om matricen nu er en permuteret øvre trekantsmatrix.

Da vi ikke på forhånd ved, hvilken by der er nummer 1, og hvilken nummer n, er der $n(n-1)$ forskellige muligheder at gætte på, og vi må blive ved med at gætte, enten indtil vi gætter rigtigt (der kan godt være flere gode gæt) eller der er blevet gættet på alt, uden positivt resultat.

2.5.5 Eksempel på at genfinde en øvre trekantsmatrix.

For at demonstrere de ovenstående resultater vises her forsøget på at få en øvre trekantsmatrix ud af en tilfældig matrix. Se start-matricen i figur 2.13.

8	15	7	13
10	12	9	11
8	12	6	10
9	17	8	10

Figur 2.13: En tilfældig matrix.

Det kan lade sig gøre at gætte på at fire forskellige byer er nr.1, og derefter at en af de tre andre byer er nr.4. En af disse 12 kombinationer er at by nr.3 i virkeligheden er nr.1, og at nr.2 i virkeligheden er nr.4. Derudover gættes på, at $a_4 = 0$ og $b_1 = 9$, dvs. med den gamle nummerering er $a_3 = 0$ og $b_2 = 9$. Dette giver nemlig at $c_{32} = 9 = 0 + a_3 + b_2$. Med dette yderligere gæt kan resten af de to vektorer findes. Fx. er $a_1 + b_2 + 0 = c_{12} = 10$, dermed er $a_1 = 1$. Det findes at $\mathbf{a} = (1,3,0,2)$ og $\mathbf{b} = (7,9,6,8)$ (dvs. den 3.søjle). Se mellemresultatet i figur 2.14.

	1	3	0	2
7	8	15	7	13
9	10	12	9	11
6	8	12	6	10
8	9	17	8	10

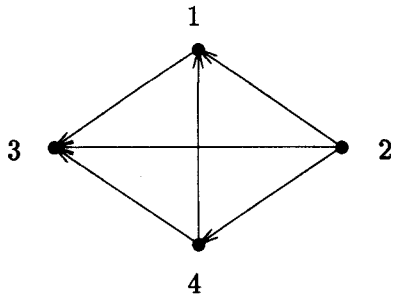
Figur 2.14: Et mellemresultat.

For at komme tilbage til den oprindelige permuterede trekantsmatrix skal der altså fra alle c_{ij} trækkes a_i og b_j, dette resultat ses i figur 2.15.

0	5	0	4
0	0	0	0
1	3	0	2
0	6	0	0

Figur 2.15: Et mellemresultat.

I figur 2.15 ses flere nuller, end bare dem der skulle dukke op i 3.søjle og 2.række, så det ser ikke umuligt ud, at dette kunne være en permuteret øvre trekantsmatrix, men det mangler endnu at blive checket, om en topologisk sortering er mulig. Se umiddelbart figur 2.16 for det hidtidige resultat.



Figur 2.16: Bud på topologisk sortering.

I figur 2.16 ses den foreslåede by nr.1 til venstre, og den foreslåede by nr.4 til højre. Hvis det skal være opfyldt, at der kun er kanter af længde forskellig fra nul i øverste højre hjørne af matricen, så må der kun være kanter der går fra højre mod venstre (fra højt nummer til lavt nummer) i den topologiske sortering. Her mangler det kun at blive fastlagt, hvor de to midterste byer i figuren skal lægges, for at dette er opfyldt, og dette kan klares ved at lægge by 1 ned

mellem 3 og 4, eller rettere lade by 1 blive omnummeret til by nr.2. Dermed har alle byer fået nye numre, de er i rækkefølge 2, 4, 1 og 3. Matricen fra før kommer med disse nye numre til at se således ud, som i figur 2.17.

0	1	2	3
0	0	4	5
0	0	0	6
0	0	0	0

Figur 2.17: Den permuterede, transformerede matrix.

Og som det ses kunne denne matrix altså transformeres og permuteres til en øvre trekantsmatrix.

2.5.6 Komplexitetsanalyse.

For en øvre trekantsmatrix er det nemt ($O(n)$) at konstruere en optimal tur af en optimal tildeling. $O(n)$ fordi man løber tildelingens kanter igennem, og for hver konstateret kreds registrerer en bagvendt kant. Da der er n kanter at løbe igennem bliver gennemløbet $O(n)$, og også selve konstruktionen, der afhænger af antallet af registrerede kanter.

Så i dette tilfælde domineres køretiden af at det er $O(n^3)$ at finde den optimale tildeling. Bemærk at der skal rettes i matricen, så allerede givne kant fra n til 1 vil komme med.

2.6 Søjle–sorterede, positive matricer.

Dette problem er også NP-hårdt (alle matricer kan transformeres om til søjlesorterede matricer). Men det er muligt at komme tæt på en løsning. *Gilm 85*

2.6.1 Bibetingelser.

- $c_{ij} \geq c_{i+1,j}$. Dvs. i søjlerne er tallene sorteret, med det største øverst.
- $c_{ij} \geq 0$. Alle tal er positive.

2.6.2 Resultater.

Sætning 11 *Lad C være en søjle–sorteret, positiv matrix. Givet en optimal tildeling φ er det nemt at konstruere en tur τ, så $c(\tau) \leq c(\varphi) + max_j\{c_{1j}\}$.*

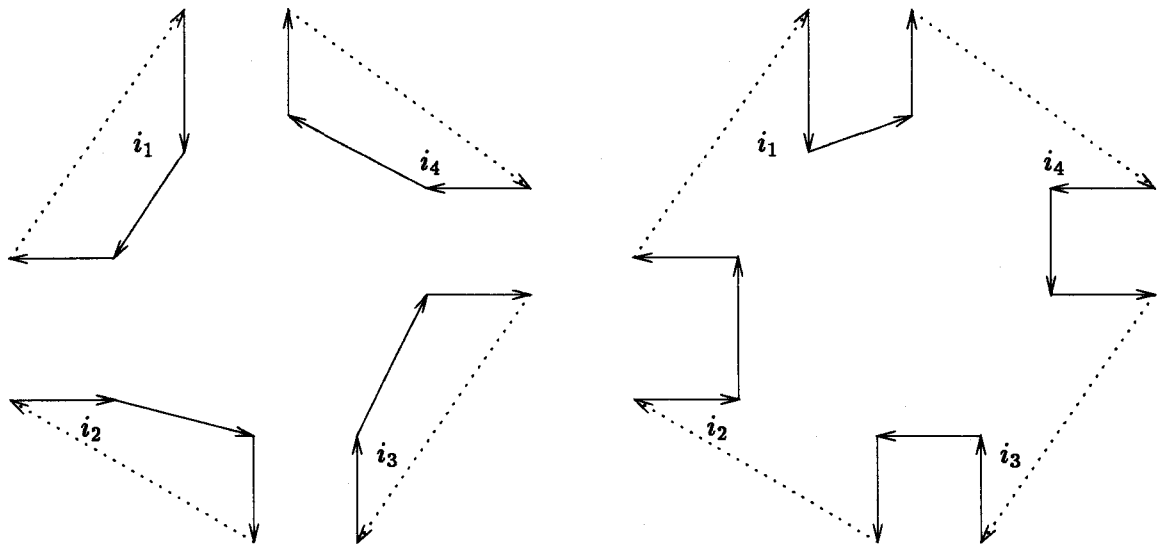
Bevis: Lad φ være den optimale tildeling. Hvis den allerede er en tur, så lad $\tau = \varphi$.

Ellers består φ af $m \geq 2$ delture. Vælg m byer i_k, en fra hver deltur, og sorter disse, så $i_1 < i_2 < \ldots < i_m$. Nu sætter vi delturene sammen til τ ved at beholde de fleste af kanterne, men smide $(i_1, \varphi(i_1)), \ldots, (i_m, \varphi(i_m))$ væk, og inkludere $(i_2, \varphi(i_1)), \ldots, (i_m, \varphi(i_{m-1}))$ og $(i_1, \varphi(i_m))$.

I summen, der giver omkostningen for φ indgår bl.a. leddene $c_{i_1\varphi(i_1)} + \cdots + c_{i_{m-1}\varphi(i_{m-1})} + c_{i_m\varphi(i_m)}$. De første $m-1$ led erstattes af $m-1$ nye led, højst lige så store som de gamle, idet fx. $c_{i_2\varphi(i_1)} \leq c_{i_1\varphi(i_1)}$. Det sidste led erstattes med $c_{i_1\varphi(i_m)}$, der muligvis er større end det erstattede led. Deraf fås, at $c(\tau) \leq c(\varphi) + c_{i_1\varphi(i_m)} - c_{i_m\varphi(i_m)} \leq c(\varphi) + max_j\{c_{1j}\}$.

QED

På figur 2.18 ses et eksempel på konstruktionen i sætning 11.



Figur 2.18: En optimal tildeling, og en deraf konstrueret tur.

2.6.3 Komplexitetsanalyse.

For en søjlesorteret positiv matrix kan man altså nemt ($O(n)$) finde fra en optimal tildeling til en tur, hvis længde, sammenlignet med en optimals tur længde, højst er $max_j\{c_{1j}\}$ for lang, dvs. højst er lige så meget længere end den korteste tur, som størrelsen på det største tal i matricen.

Selve den optimale tildeling findes (som for de øvre trekantsmatricer) i højst $O(n^3)$, og dominerer dermed køretiden.

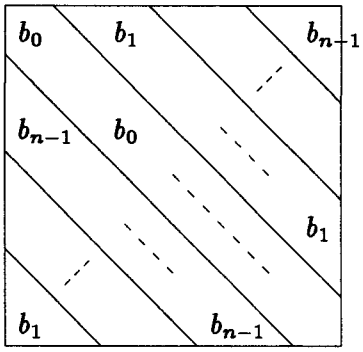
Med en passende repræsentation kan man bare løbe tildelingens kanter igennem, registrere hvilke (ud af n mulige) der skal ændres, og ændre dem, og dermed bliver den resterende køretid $O(n)$.

2.7 Cirkulære matricer.

Denne klasse har ikke nogen kendt polynomiel algoritme endnu, men det kan lade sig gøre at komme tæt på løsningen. Ifølge *Gilm 85* er de nedenstående resultater opnået i *Garf 77* og *Bach 82*. Jeg har selv udvidet på beviserne i forhold til *Gilm 85*.

2.7.1 Bibetingelser.

- $c_{ij} = b_k \Leftrightarrow j - i \equiv k \ (mod\ n)$. Dette giver en matrix med skråstriber af ens værdier, som på figur 2.19.



Figur 2.19: Matrix af skråstriber.

2.7.2 Resultater.

Sætning 12 *Den k'te stribe (alle b_k'erne) svarer til en permutation, der indeholder $gcd(k,n)$ delture.*

Bevis: En skråstribe udgør en permutation, fordi den vælger netop et element i hver søjle og i hver række.

To byer i og j er i samme deltur hviss man via kanterne i den k'te stribe kan komme fra i til j. Hver kant bringer en til en by med et nummer, der er k større. Dvs. efter m kanter er man i by $i + mk$, evt. fratrukket n et antal gange, så resultatet bliver mellem 1 og n. Eller med andre ord: $j - i \equiv mk \ (mod\ n)$ når i og j er i samme deltur. Deraf følger igen $j - i \equiv 0 \ (mod\ gcd(k,n))$, eller $i \equiv j \ (mod\ gcd(k,n))$. Da der med denne ækvivalensrelation kun er $gcd(k,n)$ forskellige tal, kan der kun være $gcd(k,n)$ forskellige delture.

QED

Korollar 13 *Hvis $gcd(k,n) = 1$ er den k'te stribe en tur.*

Korollar 14 *Hvis n er et primtal, er alle striberne ture, undtagen striben af b_0, der er identiteten.*

Lad l være en permutation af tallene fra 0 til $n-1$, så $b_{l(0)} \leq b_{l(1)} \leq \cdots \leq b_{l(n-1)}$, og lad $g_0 = gcd(l(0), n), g_{i+1} = gcd(l(i+1), g_i), (i \leq n-2)$.

Lemma 15 *En nederste grænse for længden (omkostningen) for en Hamilton–sti er givet ved* $(n - g_0)b_{l(0)} + (g_0 - g_1)b_{l(1)} + \cdots + (g_{n-2} - g_{n-1})b_{l(n-1)}$.

Bevis: En Hamilton–sti er specielt et udspændende træ, så lad os se på det minimum udspændende træ. Yderligere kigger vi på kanterne, som om de ikke har retning.

Ifølge Kruskals algoritme til at finde et minimum udspændende træ, sorteres alle kanter efter længde, og efter tur betragtes den korteste (endnu ikke betragtede) kant, og enten tilføjes den en mængde af kanter, der til enhver tid er en skov, eller også smides den væk, fordi den tilføjet til skoven ville danne en kreds. Til sidst vil skoven være blevet et minimum udspændende træ. Lad os sortere alle kanterne, fx. ved at skrive først den $l(0)$'te stribe af kanter op, og så den $l(1)$'te, osv.

Se først på de første n kanter, dem fra den $l(0)$'te stribe. Disse kanter danner tilsammen g_0 delture, eller komponenter, der hver indeholder n/g_0 punkter. Fra hver komponent kan der vælges $n/g_0 - 1$ kanter, uden at danne en kreds. Kruskals algoritme vælger derfor $g_0(n/g_0 - 1) = n - g_0$ kanter fra $l(0)$'te stribe.

Se nu på de første $2n$ kanter. Dan af disse en graf og se på komponenterne i den.

Ligger i og j i samme komponent er det fordi, man via et antal kanter i $l(0)$'te stribe (der lader en bys nummer stige med $l(0)$) og et antal kanter i $l(1)$'te stribe (der lader en bys nummer stige med $l(1)$ kan komme fra i til j. Som ovenfor giver dette, at man fra i kommer til $i + m_1 l(0) + m_2 l(1)$, eller hvis man på denne måde er kommet til j, $j - i \equiv m_1 l(0) + m_2 l(1)\ (mod\ n) \Leftrightarrow j - i \equiv m_2 l(1)\ (mod\ g_0) \Leftrightarrow j - i \equiv 0\ (mod\ g_1) \Leftrightarrow i \equiv j\ (mod\ g_1)$. Dette giver igen, at der er g_1 forskellige komponenter.

Der er altså g_1 komponenter i den nye graf, hver med n/g_1 punkter i. De nye komponenter er opnået ved at tage g_0/g_1 gamle komponenter af gangen og sætte sammen til nye komponenter. I de nye komponenter er der altså allerede udvalgt $(g_0/g_1)(n/g_0 - 1) = n/g_1 - g_0/g_1$ kanter af Kruskals algoritme. Da der er n/g_1 punkter kan Kruskals algoritme yderligere vælge $g_0/g_1 - 1$ kanter, uden at skabe kredse, så der vælges $g_1(g_0/g_1 - 1) = g_0 - g_1$ nye kanter i alt, fra den $l(1)$'te stribe.

Således bliver vi ved, indtil der er valgt $n - 1$ kanter, og således opnås hele summen ovenfor, idet der altså er valgt $n - g_0$ kanter af længde $b_{l(0)}$, $g_0 - g_1$ kanter af længde $b_{l(1)}$ osv.

QED

Sætning 16 *Nærmeste–nabo–reglen vil give en kortest mulig Hamilton–sti.*

Bevis: Her vil jeg faktisk bevise, at det er muligt at have en Hamilton–sti af samme længde som det minimum udspændende træ konstrueret ovenfor, via nærmeste–nabo–reglen.

Ovenfor blev der tilføjet klumper af kanter, til de n punkter, der i starten var skoven. Lad os kalde hver tilføjelse af en klump kanter for en fase. Der er tale om en tilføjelse af kanter fra $l(0)$'te stribe hvis $n - g_0 > 0$, og tilføjelse af kanter fra $l(k)$'te stribe, hvis $g_{k-1} - g_k > 0$. Lad os nummerere faserne kronologisk. I fase 1 tilføjes de første kanter til skoven, osv. og efter fase 1 er der h_1 komponenter, med n/h_1 punkter i.

Lad os nu vælge en tilfældig by, og konstruere en Hamilton–sti. Fra den udvalgte by vælges en af de kanter, der kunne vælges i fase 1, og det bliver vi ved med, indtil en kreds kunne opstå. Så vælges en kant, der kunne have været valgt i fase 2, og man fortsætter så igen med fase 1 kanter. På et tidspunkt vil alle punkter i komponenten, skabt efter fase 2, være med i stien, og nu vælges en fase 3 kant. Herefter gennemløber man næste fase 2 komponent med fase 1 og 2 kanter, osv. På denne måde vælger man faktisk samme antal kanter i de forskellige faser, som blev gjort i konstruktionen af det minimum udspændende træ, og alle kanter vælges grådigt, dvs. man vælger en af de korteste kanter, der ikke danner en kreds.

QED

Korollar 17 *Nærmeste-nabo-reglen vil give en tur, der højst er $b_{l(n-1)} - b_{l(0)}$ længere end den korteste.*

Bevis: Efter at have fundet en Hamilton-sti σ som i sætning 17, lukker man stien med en sidste kant (af længde $b_{l(k)}$), og får dermed turen τ'. Lad τ være en kortest mulig tur. τ er også bare en kant (af længde c_{ij})og en Hamilton-sti (af længde $c(\sigma')$), der hver for sig muligvis er længere end de korteste muligheder:

$b_{l(0)} + c(\sigma) \leq c_{ij} + c(\sigma') = c(\tau)$.

$c(\sigma) + b_{l(n-1)} \geq c(\sigma)b_{l(k)} = c(\tau)$, fordi man kan vælge en kant af højst længde $b_{l(n-1)}$ at tilføje til den fundne Hamilton-sti, og så have dannet en tur.

$c(\sigma) + b_{l(0)} \leq c(\tau) \leq c(\sigma) + b_{l(n-1)}$, kan disse to resultater sættes sammen til. Dvs. man med nærmeste-nabo-reglen danner en tur, der højst kan være $b_{l(n-1)} - b_{l(0)}$ længere end den korteste.

QED

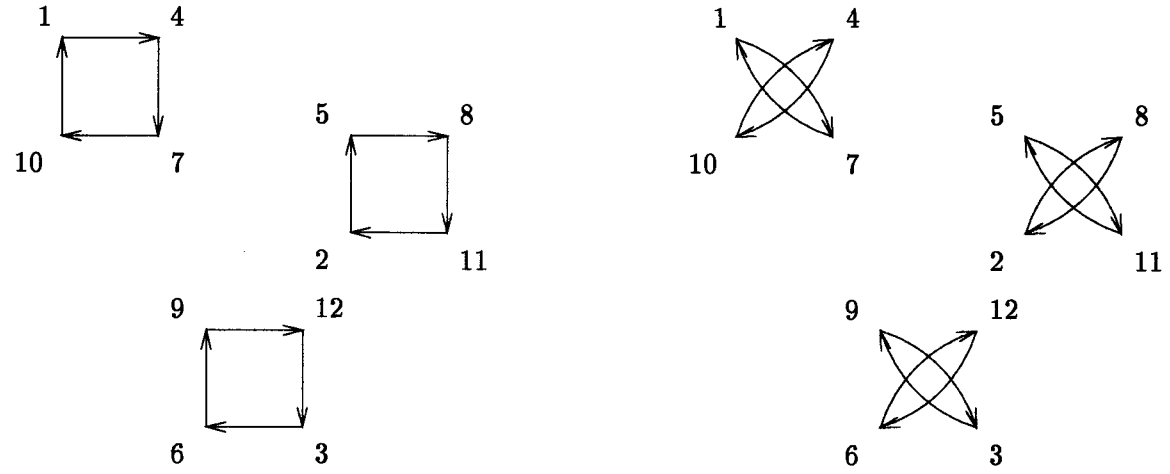
Som et eksempel på hvad alt dette betyder kommer her en tegnet gennemgang af et tilfælde med 12 punkter. De værdier, jeg anvender kunne se ud som i figur 2.20. Stjernen angiver en værdi, der ikke kendes, men den er som alle de andre ikke-angivne mindst 3, og benyttes derudover da turen laves.

			1	3			2				
				1	3			2			
					1	3			2		
						1	3			2	
							1	3			2
2						*		1	3		
	2								1	3	
		2								1	3
3			2								1
1	3			2							
	1	3			2						
		1	3			2					

Figur 2.20: En cirkulær matrix, et eksempel.

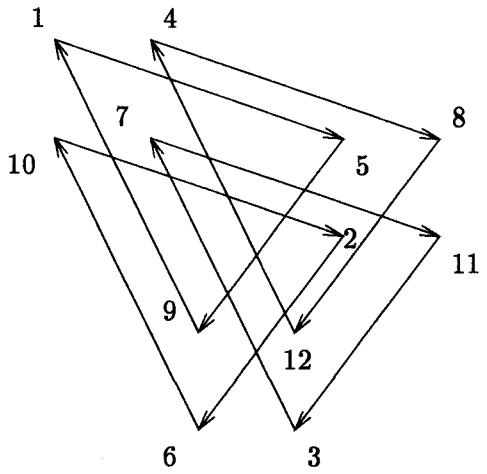
I dette tilfælde er de korteste kanter dem med længde b_3, så der bliver 3 komponenter med 4 kanter hver. Kruskals algoritme ville vælge 3 fra hver komponent i fase 1.

De næstkorteste kanter er dem med længde b_6, og der bliver 6 komponenter, med 2 kanter hver. Lægges disse kanter i samme graf som dem ovenfor vil der stadig være 3 komponenter, og ingen af disse kanter ville blive valgt. Se figur 2.21.



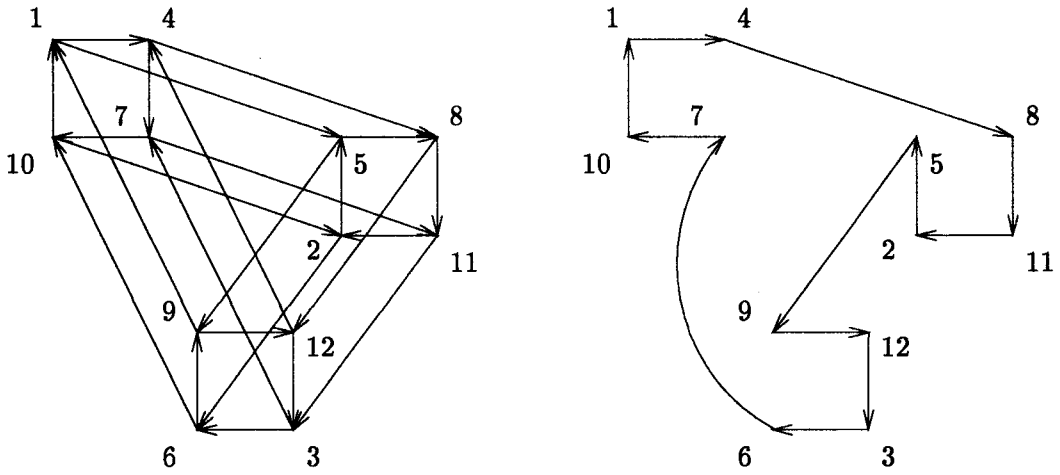
Figur 2.21: Komponenterne af kanter fra 3. og 6. skråstribe.

I fase 2 vælges nogle af kanterne med længde b_4. Figur 2.22 er tegnet, så man nemt kan se komponenterne fra fase 1.



Figur 2.22: Komponenterne af kanter fra 4. skråstribe.

En mulig Hamilton–sti kan følges gennem denne graf, med kanter fra både fase 1 og 2. Der ses på figur 2.23 også et forslag til en Hamilton–sti, og den deraf følgende tur, der er nødt til at benytte sig af den buede kant, vi endnu ikke ved noget om.



Figur 2.23: Kanterne fra fase 1 og 2, og en mulig tur.

2.7.3 Komplexitetsanalyse.

For en cirkulær matrix gælder det, at vi nemt ($O(n^2)$) kan finde en korteste Hamilton–sti, og dermed en tur, der højst er så meget længere end den korteste tur, som den længste kant i grafen er længere end den korteste kant.

$O(n^2)$ fordi vi n gange leder efter den korteste kant blandt n mulige.

Derudover er der mange tilfælde hvor den korteste tur er endnu nemmere at finde. Hvis k er $l(0)$ (eller $l(1)$ hvis $l(0) = 0$), vil alle tilfælde hvor $gcd(k, n) = 1$ give, at den k'te stribe giver den korteste tur. Dette sker fx. for n eller k primtal.

2.8 Pyramide–ture.

Denne klasse af matricer udmærker sig ved, at man kan bruge en algoritme til at gennemsøge et begrænset antal ture, og så bruge den korteste man finder, fordi denne faktisk også er optimal. Disse resultater er ikke ret kendt eller benyttet, bl.a. fordi de delvis er fundet i ex-Sovjetunionen. Kilderne er *Gilm 85* og *Demi 79.*

2.8.1 Bibetingelser.

Hvis vi skriver en tur op med by 1 først, og så hele turen, og til sidst 1 igen, vil en pyramide–tur skrevet op på denne måde først være en række tal, der bliver større og større, indtil man kommer til n, og så en række tal, der bliver mindre og mindre, indtil man igen kommer til 1.

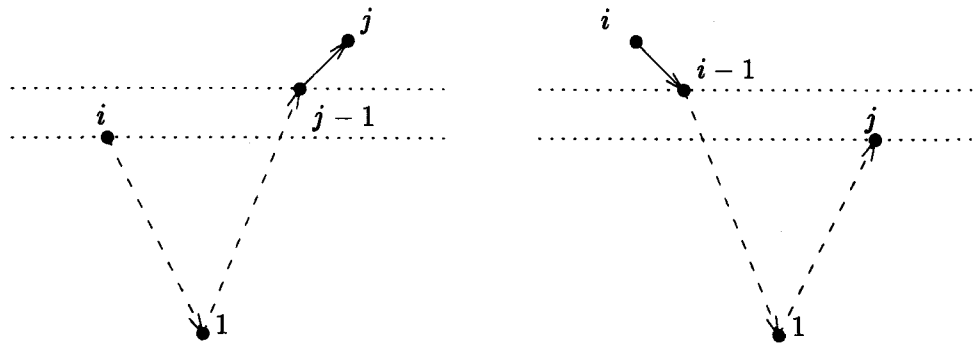
Ex: (1,2,4,7,8,6,5,3) er en pyramide–tur.

Dette kan også skrives på flere andre måder. Fx. skal der gælde for $1 < i < n$, at enten er $\tau^{-1}(i) < i < \tau(i)$ eller også er $\tau^{-1}(i) > i > \tau(i)$. Der gælder også for en pyramide–tur, at når man først ved for alle i, om man kommer gennem by i før eller efter by n, så ved man også præcis hvor by i skal være i turen.

2.8.2 Resultater.

Vi kan altid finde den korteste pyramide–tur for en matrix, og som vi skal se senere, vil denne nogle gange også være en optimal tur.

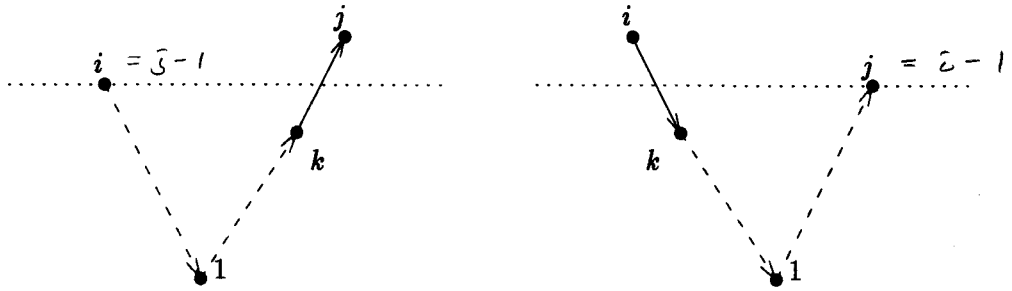
Algoritmen til at finde en korteste pyramide–tur bygger på dynamisk programmering. Hvis vi kender længden (og opbygningen) af den korteste sti fra i, via 1, til j, for alle $i, j < k$, kan vi også finde dem for $i, j \leq k$.



Figur 2.24: Stier, der let forlænges.

Der er to tilfælde, hvor vi nødvendigvis må bruge en bestemt kant til at forlænge den sti, vi kender til, se figur 2.24. Det ene tilfælde er når vi skal slutte i j, og allerede kender stien fra i til $j-1$. Dette indtræffer når $i < j-1$. Tilsvarende kan vi kun vælge en kant når vi laver stien udfra $i-1$ til j, fordi $i-1 > j$.

Der er to andre tilfælde, hvor et antal tilfælde må søges igennem, se figur 2.25. Når stien fra $j-1$ til j skal laves, kan vi ikke betragte en enkelt sti, der er jo ikke nogen sti fra $j-1$ til $j-1$ at betragte. Altså må vi kigge på alle stier fra $j-1$ til k, suppleret med kanten fra k til j. Vi kigger på alle de resulterende stier og vælger den korteste. Der er et tilsvarende eksempel for stien fra i til $i-1$.



Figur 2.25: Stier, der "vanskeligt" forlænges.

Med andre ord kan vi finde længden af den korteste sti fra i til j, $c(i,j)$, ved at betragte et af følgende 4 tilfælde:

Når $i < j$:

1) $i < j-1 : c(i,j-1) + c_{j-1,j}$

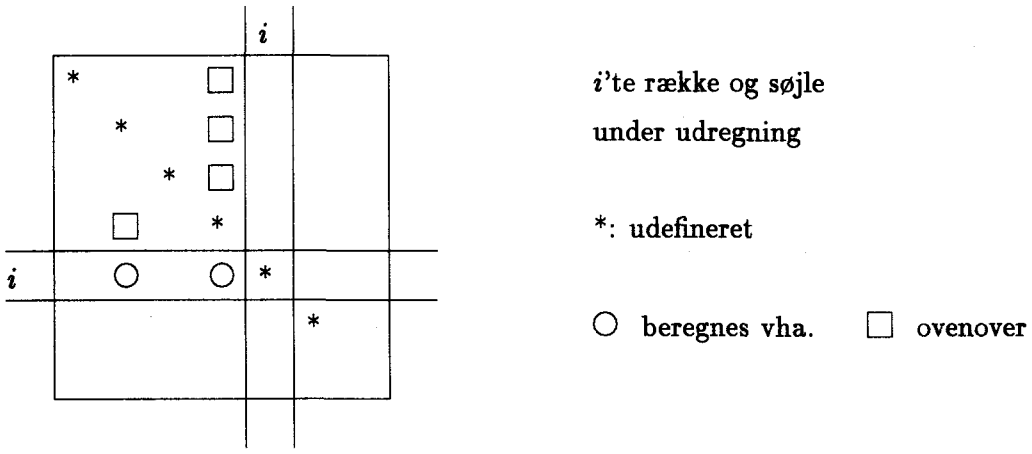
2) $i = j-1 : min_{k<i}(c(i,k) + c_{kj})$

Når $i > j$:

3) $i-1 > j : c_{i,i-1} + c(i-1,j)$

4) $i-1 = j : min_{k<j}(c_{ik} + c(k,j))$

Dette svarer til: når vi er i færd med at udfylde en matrix med værdierne for $c(i,j)$, starter vi i øverste, venstre hjørne, og lægger i hvert skridt endnu en række og søjle til det beregnede. Diagonalen udfyldes ikke. En ny værdi i en række fås enten ved at betragte værdien umiddelbart over eller (hvis diagonalen er umiddelbart over) hele søjlen ovenover. Se figur 2.26. Her ses eksempel på regel 3 og 4, begge for $i = 5$. For $j = 2$ bruges værdien lige henover, fra $i = 4$, for $j = 4$ bruges hele resten af søjlen ovenover. Tilsvarende for nye søjle–værdier.



Figur 2.26: De elementer, der betragtes, når en minimal sti findes.

Når vi skal finde alle disse værdier, starter vi med $c(1,2) = c_{12}, c(2,1) = c_{21}$. Når alle værdier udenfor diagonalen er beregnet, kan vi finde den korteste tur, ved at sammenligne $c_{n,n-1} + c(n-1,n)$ med $c(n,n-1) + c_{n-1,n}$. Byen $n-1$ må jo nødvendigvis ligge enten lige før eller lige efter byen n. Efter sammenligningen beholdes selvfølgelig den korteste mulighed,

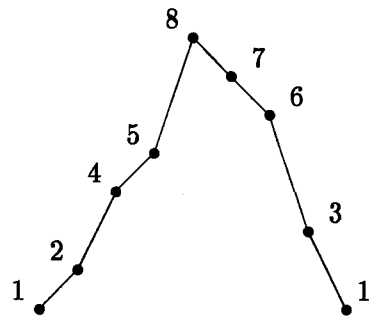
og vi har vores tur. Der skal holdes styr på valgene undervejs, så man kan gå baglæns i den dynamiske programmering og finde ud af, hvordan den korteste tur blev konstrueret.

Algoritme 8 *Korteste pyramide-tur(vægtet graf): tur*

$c(1,2) = c_{12}$
$c(2,1) = c_{21}$
for ($r = 3$ to n) do
  for ($s = 1$ to $r-1$) do
    (* $i = s, j = r, i < j$ *)
    if ($s = r-1$) then
      $c(s,r) = min_{t<s}(c(s,t) + c_{tr})$
      (* husk det t, der blev brugt *)
    else
      $c(r,s) = c(r,s-1) + c_{s-1,s}$
    (* $i = r, j = s, i > j$ *)
    if ($r - 1 = s$) then
      $c(r,s) = min_{t<s}(c_{rt} + c(t,s))$
      (* husk det t, der blev brugt *)
    else
      $c(r,s) = c_{r,r-1} + c(r-1,s)$
  (* end for s *)
(* end for r *)
$c(n,n) = min(c_{n,n-1} + c(n-1,n), c(n,n-1) + c_{n-1,n}))$
(* husk hvad der blev valgt *)

Algoritme slut

For at finde ud af, hvornår denne algoritme er noget værd, må det checkes om visse krav er opfyldt, og eksempler på fyldestgørende krav ses i de næste par afsnit. Inden må vi dog lige kigge lidt nærmere på definitionen af en pyramide-tur. Den her viste alternative definition på en pyramide-tur, skal bruges senere til at vise egenskaber ved Demidenko-matricerne.



Figur 2.27: En pyramide-tur.

Definition 4 *Lad os kigge på turen τ.*
En tinde i τ er karakteriseret ved, at $\tau^{-1}(i) < i > \tau(i)$. Mængden af alle tinder kaldes $T(\tau)$.
En dal i τ er karakteriseret ved, at $\tau^{-1}(i) > i < \tau(i)$. Mængden af alle dale kaldes $D(\tau)$.
Hvis et punkt hverken er en tinde eller en dal, kaldes det en skråning. Mængden af alle skråninger kaldes $S(\tau)$.
En tinde ligger mellem to dale. Alle punkter mellem disse to dale kaldes bjerget om punktet. Bjerget om i kaldes $B(\tau, i)$.

For en pyramide–tur er $B(\tau, n) = \{2, 3, \ldots, n\}$. Se et eksempel på figur 2.27.

Lemma 18 *τ er en pyramide–tur* $\Leftrightarrow$
$\forall k \in T(\tau), 3 \leq k \leq n, \forall i, 0 \leq i \leq k-3$:
$\{k-i, \ldots, k\} \subseteq B(\tau, k) \Rightarrow (k-i-1 \in T(\tau) \vee k-i-1 \in B(\tau, k))$

Eller med andre ord: hvis τ er en tur, og k er en tinde i τ, og alle $k's$ nærmeste, mindre naboer ($k-1, k-2$ osv.) ned til $k-i$ ligger i k's bjerg, antag da at $k-i-1$ (af værdi højst $k-1$, mindst 2) enten selv en tinde, eller også er med i k's bjerg. Hvis dette kan siges for alle par af tinder og mulige værdier af i, da er det ensbetydende med, at τ er en pyramide-tur.

Bevis: Hvis den nederste betingelse ikke gælder må det være fordi der er et "dårligt par", (k, i), der er et modeksempel. Derfor er vi egentlig ude på at bevise, at τ pyramide–tur $\Leftrightarrow$ ingen dårlige par i τ.

Antag τ er en pyramide–tur. Da er $B(\tau, n) = \{2, 3, \ldots, n\}$. Vælg et k.

Da k skal være en tinde, kan kun n vælges. Kun 1 er ikke i $B(\tau, n)$, og da $i \leq n-3$ er $n-i = 3$ den laveste mulighed for et problem, kan 1 ikke blive part i et dårligt par.

Antag der ikke er nogle dårlige par.

Find det mindste $k \in T(\tau)$.

Se på $B(\tau, k)$. Find mindste i, så $k-i \in B(\tau, k), k-i-1 \notin B(\tau, k)$. Det kan lade sig gøre, for $k \in B(\tau, k)$, men $1 \notin B(\tau, k)$. Da desuden k mindste tal i $T(\tau)$, må $k-i-1 < k \Rightarrow k-i-1 \notin T(\tau)$.

i blev fundet mindst mulig, så $\{k-i, \ldots, k\} \subseteq B(\tau, k)$ og $k-i-1 \notin B(\tau, k)$ og $k-i-1 \notin T(\tau)$. Derfor kan der ikke gælde $0 \leq i \leq k-3$ (ellers havde vi jo et dårligt par), så $i \geq k-2 \Leftrightarrow 2 \geq k-i$. Da $k-i \in B(\tau, k), k-i-1 \notin B(\tau, k)$ må $k-i-1 = 1$, og da dalene til k må være mindre end $k-i$ (ellers var de jo ikke dale / $k-i$ ikke en skråning) må de være $k-i-1 = 1$ (eller mindre, men dette er heller ikke muligt), dvs. begge k's dale er 1, dvs. der er kun et bjerg, og $k = n$, dvs. τ er en pyramide–tur.

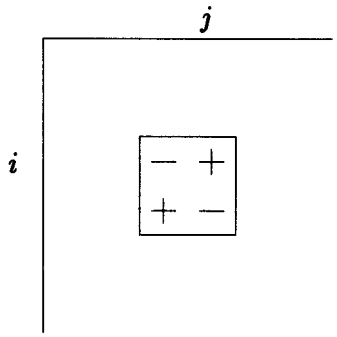
QED

2.8.3 Distributions–matricer.

Som eksempel på en klasse af matricer, der har en optimal pyramidetur ses her klassen af distributionsmatricer. Den viser sig senere at være et specialtilfælde af Demidenko-matricer, men der vises også et seperat bevis for denne klasse. *Gilm 85*

Bibetingelser.

Lad os, givet matricen C, som fx. i figur 2.28, kigge på matricen D, hvor $d_{ij} = c_{ij} + c_{i+1,j-1} - c_{i,j-1} - c_{i+1,j}(c_{n+1,j} = c_{i0} = 0)$.



Figur 2.28: Elementer der bruges til beregning af d_{ij}.

- $d_{ij} \geq 0$. Dvs. i D er alle indgange ikke–negative. Da er C en distributions-matrix.

Resultater.

Lemma 19 $c_{ij} = \sum_{k=i}^{n} \sum_{l=1}^{j} d_{kl}$.

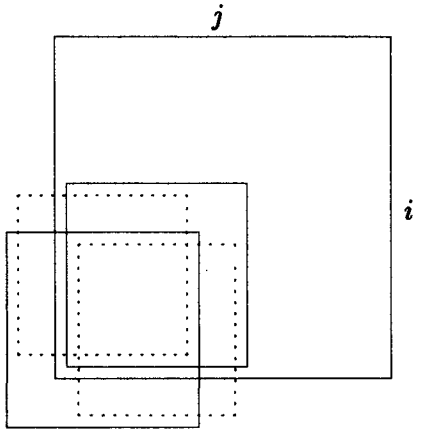
Bevis:

$$\begin{aligned}
\sum_{k=i}^{n}\sum_{l=1}^{j} d_{kl} &= \sum_{k=i}^{n}\sum_{l=1}^{j} c_{kl} + c_{k+1,l-1} - c_{k,l-1} - c_{k+1,l} \\
&= \sum_{k=i}^{n}\sum_{l=1}^{j} c_{kl} + \sum_{k=i}^{n}\sum_{l=1}^{j} c_{k+1,l-1} - \sum_{k=i}^{n}\sum_{l=1}^{j} c_{k,l-1} - \sum_{k=i}^{n}\sum_{l=1}^{j} c_{k+1,l} \\
&= \sum_{k=i}^{n}\sum_{l=1}^{j} c_{kl} + \sum_{k=i+1}^{n+1}\sum_{l=0}^{j-1} c_{kl} - \sum_{k=i}^{n}\sum_{l=0}^{j-1} c_{kl} - \sum_{k=i+1}^{n+1}\sum_{l=1}^{j} c_{kl} \\
&= \sum_{k=i}^{n}\sum_{l=1}^{j} c_{kl} + \sum_{k=i+1}^{n}\sum_{l=1}^{j-1} c_{kl} - \sum_{k=i}^{n}\sum_{l=1}^{j-1} c_{kl} - \sum_{k=i+1}^{n}\sum_{l=1}^{j} c_{kl} \\
&= \left(c_{ij} + \sum_{l=1}^{j-1} c_{il} + \sum_{k=i+1}^{n} c_{kj} + \sum_{k=i+1}^{n}\sum_{l=1}^{j-1} c_{kl}\right) + \sum_{k=i+1}^{n}\sum_{l=1}^{j-1} c_{kl} \\
&\quad + \left(-\sum_{l=1}^{j-1} c_{il} - \sum_{k=i+1}^{n}\sum_{l=1}^{j-1} c_{kl}\right) + \left(-\sum_{k=i+1}^{n} c_{kj} - \sum_{k=i+1}^{n}\sum_{l=1}^{j-1} c_{kl}\right)
\end{aligned}$$

$$= c_{ij}$$

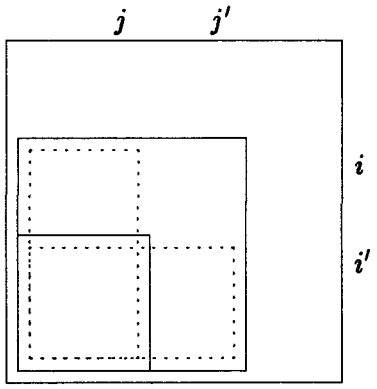
QED

Betragt også en tegning af hvad der foregår, figur 2.29.



Figur 2.29: Beregning af $\sum_{k=i}^{n} \sum_{l=1}^{j} d_{kl}$, i C.

Indgangene i de to fuldt optrukne kvadrater adderes, mens indgangene i de prikkede kvadrater trækkes fra. Tilbage bliver kun c_{ij}.



Figur 2.30: Beregning af $c_{ij'} + c_{i'j} - c_{ij} - c_{i'j'}$, i D.

Lemma 20 $i < i', j < j' \Rightarrow c_{ij'} + c_{i'j} \geq c_{ij} + c_{i'j'}$.

Bevis:

Betragt tegningen, figur 2.30. I denne tegning svarer de to fuldt optrukne kvadrater til venstresiden, og de to prikkede rektangler til højresiden. Eller se appendiks B.

QED

korteste

Korollar 21 *I en distributionsmatrix er identitets-permutationen en optimal tildeling.*

Bevis: Lad φ være en optimal tildeling. Hvis den ikke er identiteten er der byer i, i', så $i < i'$ og $\varphi(i) > \varphi(i')$. Vælg den mindste i mulig, for at få sådan et par. Brug lemma 20 ovenfor, til at lave en tildeling med lavere omkostning, dvs. en med samme omkostning, da φ jo er optimal. Lad nemlig $j' = \varphi(i)$ og $j = \varphi(i')$, og lad i den nye tildeling φ' gælde $\varphi'(i) = j, \varphi'(i') = j', \varphi'(k) = \varphi(k)$ for $k \neq i, i'$. Da man nu ikke kan lave sådanne par med i eller et lavere tal, vil man efter et endeligt antal af omarrangeringer i tildelingen være nået frem til identiteten, der har samme omkostning.

QED

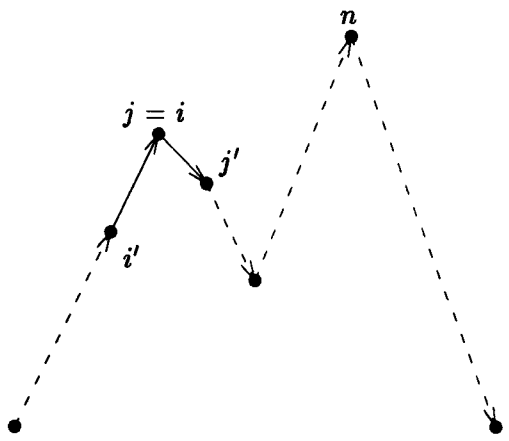
Sætning 22 *I en distributionsmatrix kan man finde en optimal pyramide-tur.*

Bevis: Induktionsbevis.

Basis: det holder for 2 byer, trivielt, fordi den eneste mulige tur er en pyramide-tur. (Faktisk også trivielt for 3 byer, for begge mulige ture.)

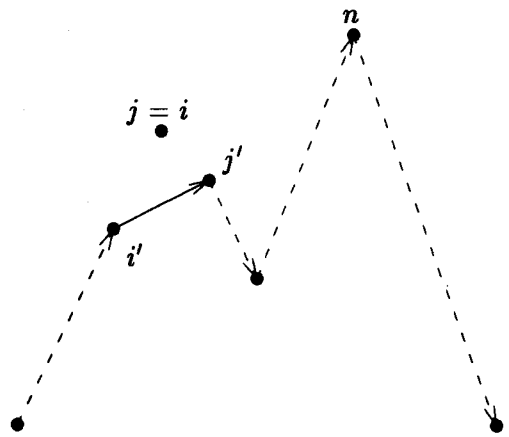
Induktion: antag det holder for $n-1$ byer, dvs. der findes en pyramide-tur på $n-1$ byer, der er optimal.

Se nu på tilfældet med n byer, og den optimale tur τ. Hvis τ ikke er en pyramide-tur er der en tinde (en lokal top) i $j \neq n$, så $\tau^{-1}(j) < j > \tau(j)$. Et eksempel ses i figur 2.31.



Figur 2.31: En ikke-pyramide-tur.

Ved hjælp af lemma 20 kan vi lave en kortere tildeling. Lad $i = j, i' = \tau^{-1}(j), j' = \tau(j)$. Resultatet bliver tildelingen τ', med $\tau'(j) = j, \tau'(i') = j', \tau'(k) = \tau(k)$ for $k \neq i', j$. Tildelingen indeholder to delture, den ene kun med punktet j, se figur 2.32.



Figur 2.32: Den mellemliggende tildeling.

Den lange deltur indeholder $n-1$ byer, og kan således laves om til en pyramide–tur, hvis den ikke allerede er det. (En delmatrix af en distributionsmatrix er også en distributionsmatrix.) Så sætter vi j ind i pyramide–turen τ'. Vi indsætter den inden n, og finder et i, så $i < j < \tau'(i)$. Igen bruges lemma 20, med $i' = j, j' = \tau(i)$. Resultatet bliver en pyramide-tur der er kortere end den optimale tur τ, eller rettere har samme længde.

QED

For alternative beviser for, at distributionsmatricer har optimale pyramideture, se også appendiks B.

2.8.4 Demidenko–matricer.

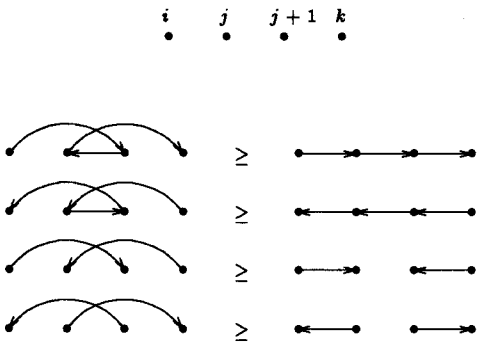
Kilden til disse resultater er *Demi 79*, som før kun fandtes på russisk. Med disse resultater, inklusive definition 4, lemma 18 og appendiks C er denne artikel nu oversat og dækker alle tilfælde. Desværre er beviset for sætning 25 meget langt, og man kunne ønske sig et mere elegant bevis.

Bibetingelser.

Disse betingelser kaldes Demidenko–betingelserne, deraf navnet på matrix–klassen.

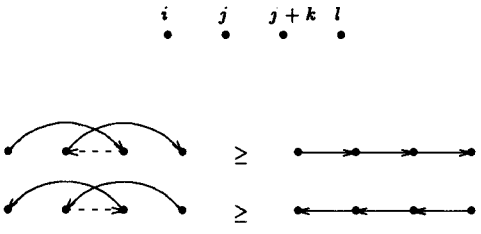
Den største sætning i dette emne (nemlig at i Demidenko-matricer findes der optimale pyramide-ture) er min egen fremstiling af 2 kilder, hvorfra jeg dels har hentet fremstillingsformer jeg godt kunne lide, dels har fundet huller og fejl, der gerne skulle være rettet.

Betingelserne handler i første omgang om at vælge vilkårlige $i < j, l > j+1$, så man får 4 byer med $i < j < j+1 < l$. Da skal de i figur 2.33 tegnede forhold gælde.



Figur 2.33: Demidenko–betingelserne.

Disse betingelser kan også udtrykkes med ligninger:
$\forall i, j, k : i < j, j+1 < l$ gælder:
$c_{i,j+1} + c_{j+1,j} + c_{jl} \geq c_{ij} + c_{j,j+1} + c_{j+1,l}$ (A1)
$c_{lj} + c_{j,j+1} + c_{j+1,i} \geq c_{l,j+1} + c_{j+1,j} + c_{ji}$ (A2)
$c_{i,j+1} + c_{lj} \geq c_{ij} + c_{l,j+1}$ (B1)
$c_{j+1,i} + c_{jl} \geq c_{ji} + c_{j+1,l}$ (B2)



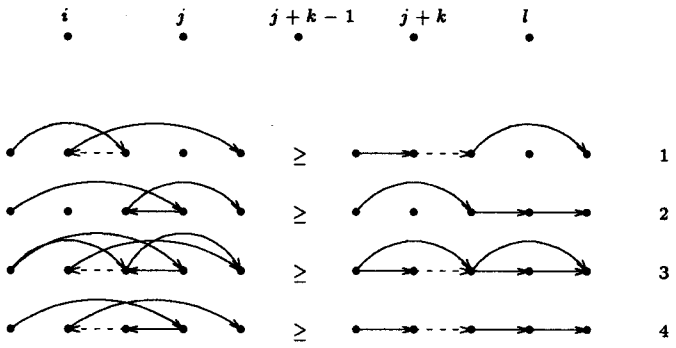
Figur 2.34: Betingelserne opfyldt i lemmaet.

Lemma 23 *For en Demidenko-matrix gælder også, at hvis vi ved $c(i,j)$ forstår omkostningen ved stien $(i,i+1),(i+1,i+2),\ldots,(j-1,j)$ hvis $i<j$, og omvendt for $i>j$, gælder det for alle valg af i,j,k,l så $i<j, l>j+k$, at $i<j<j+k<l$ og:*
$c_{i,j+k}+c(j+k,j)+c_{jl} \geq c_{ij}+c(j,j+k)+c_{j+k,l}$ *(A3)*
$c_{lj}+c(j,j+k)+c_{j+k,i} \geq c_{l,j+k}+c(j+k,j)+c_{ji}$ *(A4)*

De nye betingelser A3 og A4 ses i figur 2.34. En stiplet linje betyder i denne forbindelse en sti, der går gennem alle punkter med numre mellem endepunkternes numre.

Bevis: Induktionsbevis. Basis: for $k=1$ gælder det trivielt.

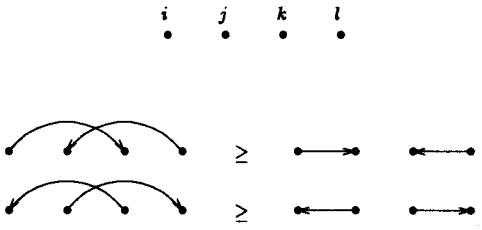
Induktion: givet i,j,k,l, så $i<j, j+k<l$. Antag da sætningen gælder for $k-1$, lad os da vise den for k.



Figur 2.35: Skitse til beviset.

I den første linje af figur 2.35 vises situationen, vi kender pga. induktionsantagelsen. I den anden vises, hvad vi ved ifølge definitionen af A1. I tredje linje er de to ovenfor stående linjer adderet. Og i fjerde linje er kanter, der var på begge sider af ulighedstegnet, fjernet. Slutresultatet er det ønskede, A3. Det symmetriske tilfælde kan bevises på samme måde.

QED



Figur 2.36: Betingelserne gældende ifølge lemmaet.

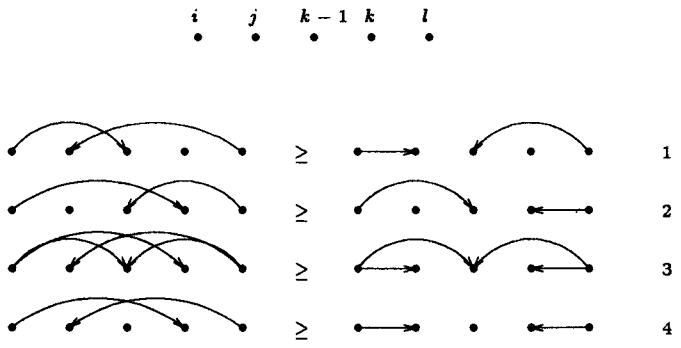
Lemma 24 *B-betingelserne i definitionen af Demidenko-betingelserne holder hviss for $i<j<k<l$*
$c_{ik}+c_{lj} \geq c_{ij}+c_{lk}$ *(B3)*
$c_{ki}+c_{jl} \geq c_{ji}+c_{kl}$ *(B4)*

B3 og B4 illustreres i figur 2.36.

Bevis: Hvis B3 og B4 gælder, følger specialtilfældene B1 og B2 trivielt.

Det omvendte kan ses ved et induktionsbevis. Basis er $j + 1 = k$, hvor det følger af B1 og B2.

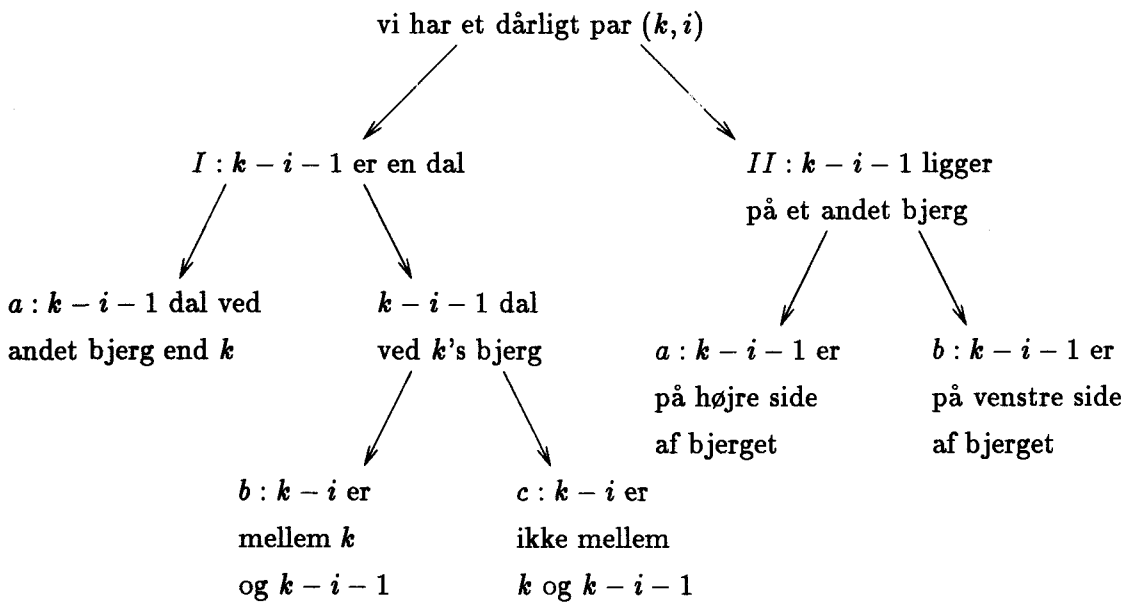
Induktionsantagelse: B3 og B4 gælder for $i < j < k - 1 < l$. Vis nu for $i < j < k < l$.



Figur 2.37: Skitse til beviset.

I første linje af figur 2.37 ses induktionsantagelsen i brug. I anden linje bruges B1. I tredje linje er de to ovenstående adderet. Endelig er er i fjerde linje ens kanter fjernet. Slutresultatet er det ønskede. Det symmetriske tilfælde vises på samme måde.

QED



Figur 2.38: Skema over specialtilfælde i beviset.

Sætning 25 *For enhver tur i en Demidenko-matrix, findes der en pyramide-tur der ikke er længere.*

Bevis: Ifølge lemma 18 er en tur forhindret i at være en pyramide–tur, hvis vi kan finde et dårligt par. Derfor vil vi angribe alle ikke–pyramide–ture ved at fjerne deres dårlige par, uden at øge længden af turen, og så alle dårlige par til sidst er fjernet. Da dette må gøres på forskellig vis i forskellige situationer, deles der op i en del specialtilfælde.

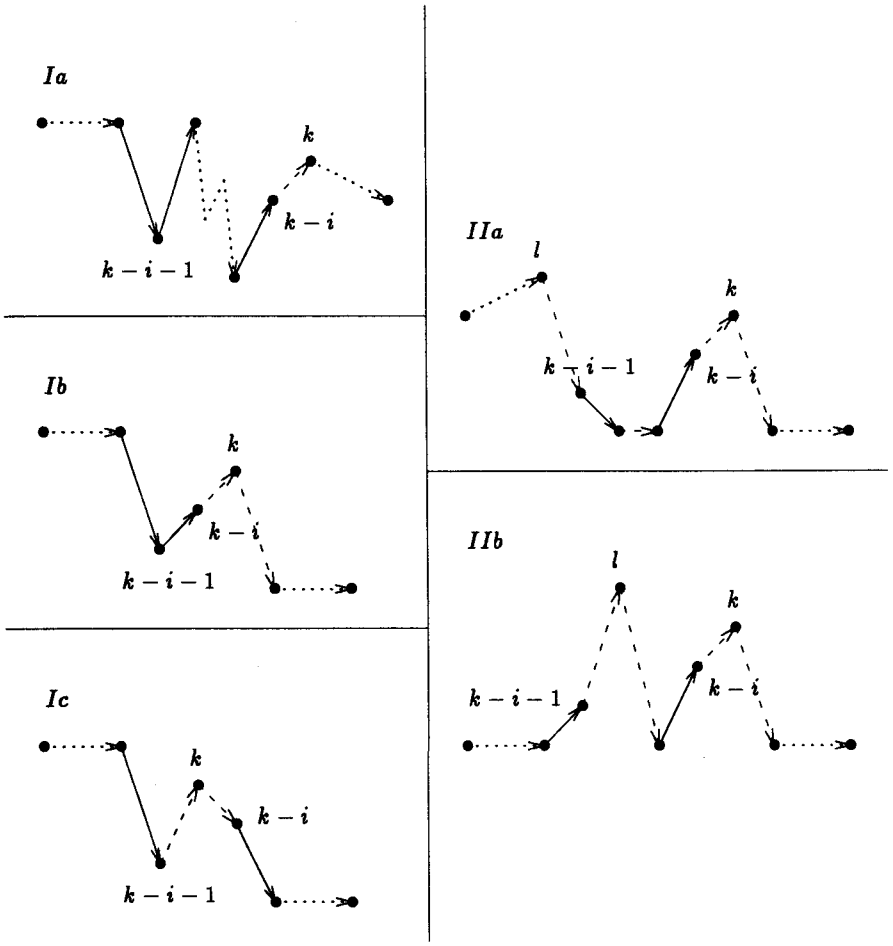
Der gøres brug af elementære transformationer, der fjerner det dårlige par, uden at øge turens længde. Af et antal elementære transformationer laves en sammensat transformation, der foruden at fjerne det dårlige par (k,i) ikke skaber et nyt dårligt par (k',i'), hvor $k'-i' \geq k-i$. Ved gentagne gange at fjerne det dårlige par med højest $k-i$ kan vi fjerne alle dårlige par.

Da et par er dårligt når vi har alt opfyldt i lemma 18, undtagen enten $k-i-1 \in T(\tau)$ eller $k-i-1 \in B(\tau,k)$, må der gælde $I : k-i-1 \in D(\tau)$ eller $II : k-i-1 \in B(\tau,l)$, hvor $l > k$. (Da alle punkter mellem $k-i$ og $k-1$ er i k's bjerg, kan $k-i-1$ ikke have en af disse som tinde. Men l skal jo være større end $k-i-1$, og dermed også større end k.)

Disse tilfælde kan igen spaltes op. Hvis I er opfyldt har vi enten $a : k-i-1$ er dal ved et andet bjerg end k, eller $k-i-1$ er dal ved k's bjerg, der igen deler op i to tilfælde: $b : k-i$ ligger på τ mellem $k-i-1$ og k, $c : k-i$ ligger på den anden side af bjerget.

Tilsvarende kan II spaltes op: $a : k-i-1$ ligger på højre side af l's bjerg, eller $b : k-i-1$ ligger på venstre side af l's bjerg.

På figur 2.38 ses et skema over disse tilfælde, der hver for sig senere vil blive vist.



Figur 2.39: Tegninger til enkelttilfælde i beviset.

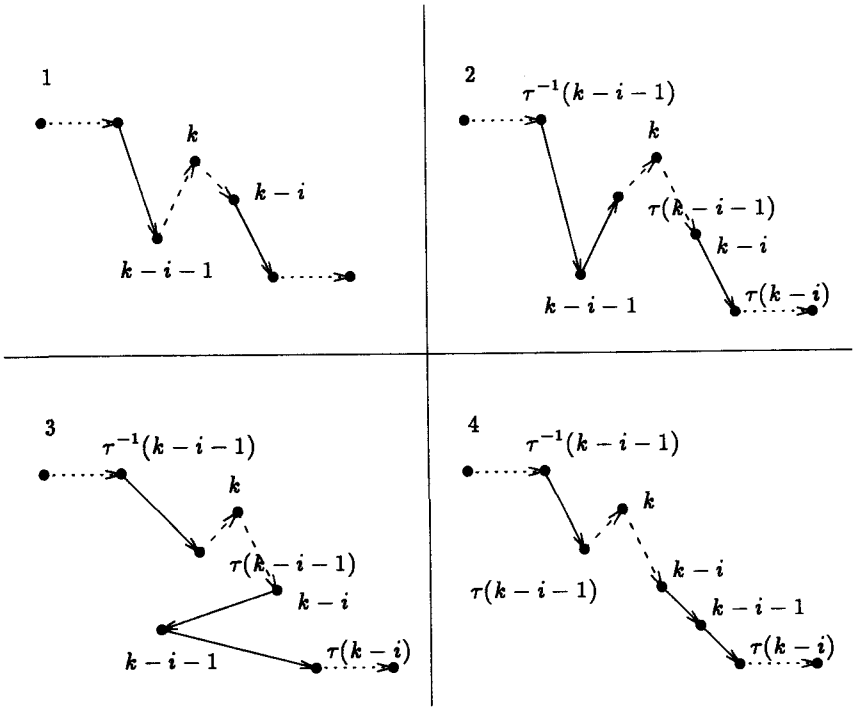
Alle tilfældene findes desuden i 2 versioner, der er hinandens spejlbilleder. For Ia betyder dette, at vi ser på tilfældet hvor man fra $k-i-1$ kommer til $k-i$ før k. For Ib og Ic betyder dette, at man fra 1 kommer til $k-i-1$ før k. For II betyder det at man fra 1 kommer først til $k-i-1$, så til $k-i$ og så til k. Nogle af tilfældene kan yderligere underinddeles, dette gøres efter behov.

For at tydeliggøre dette, se da figur 2.39, hvor hver situation nævnt i figur 2.38 er tegnet, i den version beviset bruger.

En pil er en kant. En stiplet pil er et antal kanter, måske 0, der forbinder de angivne punkter. En prikket kant er det samme, men typisk er vi enten ikke interesseret i disse kanter, eller ved ikke ret meget om hvordan de forløber. Hvis det vides, kan det ses af punkternes højde på tegningen, hvilke punkter der har højest nummer, fx. vil $k-i$ kunne ses liggende over $k-i-1$. På tegningerne i figur 2.39 er der angivet mange flere punkter, end der er navngivet. Dette er fordi, de senere vil blive navngivet, for at beviset kan gennemføres, men deres navne er ikke nødvendige for at overskue situationen.

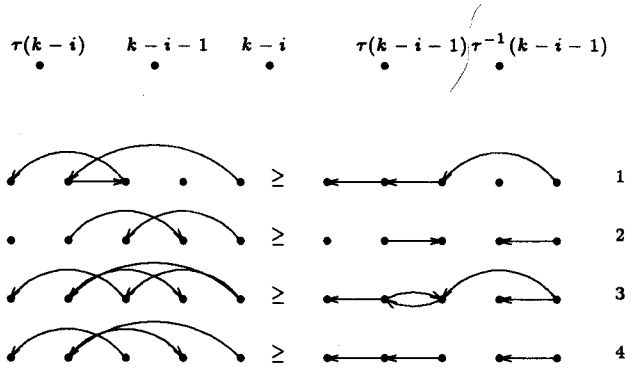
Måden hvert bevis gennemføres på er ved hjælp af en tegning at vise, hvordan transformationen virker ("før og efter"), og så bevise at denne transformation opfylder de fastsatte krav, og at man kan bygge en sammensat transformation vha. den.

Således vises på figur 2.40 transformation Ic, som de andre navngivet efter den situation, den forsøger at reparere.



Figur 2.40: Skitse af transformation Ic.

At denne transformation virker, bevises bl.a. vha. figur 2.41.



Figur 2.41: Bevis til transformation Ic.

De til figur 2.41 svarende formler er:

$c_{\tau^{-1}(k-i-1),k-i-1} + c_{k-i-1,k-i} + c_{k-i,\tau(k-i)} \geq c_{\tau^{-1}(k-i-1),k-i} + c_{k-i,k-i-1} + c_{k-i-1,\tau(k-i)}$, eller A4.

$c_{k-i-1,\tau(k-i-1)} + c_{\tau^{-1}(k-i-1),k-i} \geq c_{k-i-1,k-i} + c_{\tau^{-1}(k-i-1),\tau(k-i-1)}$, eller B3.

Den næste linje i figuren svarer til at lægge disse to uligheder sammen, og resultatet af at gøre det, og fjerne størrelser forekommende på begge sider af ulighedstegnet er:

$c_{\tau^{-1}(k-i-1),k-i-1} + c_{k-i,\tau(k-i)} + c_{k-i-1,\tau(k-i-1)} \geq c_{k-i,k-i-1} + c_{k-i-1,\tau(k-i)} + c_{\tau^{-1}(k-i-1),\tau(k-i-1)}$

Sammenligning med denne sidste ulighed og situation 2 og 3/4 i figur 2.40 viser, at det netop er de ønskede kanter, der forekommer i uligheden.

(Dette er en tilføjelse i forhold til begge kilder.)

For øvrige bevis: se appendiks C.

QED

2.8.5 Komplexitetsanalyse.

Dermed er vist at også for Demidenko-matricer er det muligt at finde en optimal tur, ved hjælp af algoritmen der finder den optimale pyramide-tur. Og denne algoritme kører i $O(n^2)$ (indre for-løkke i $O(r)$, $O(n)$ gange, dermed i alt $O(n^2)$).

Confidence man Simon Suggs of Alabama outmaneuvering Colonel Bryan, confidence man. Engraving by O. C. Darley in Johnson J. Hooper's *Adventures of Simon Suggs* (1845). (*Library of Congress*)

Kapitel 3

Simuleret udglødning.

Han sidder ved smeltediglen
og lutrer sølvet med flid.
(DDS 544)

Blandt de nyere algoritmer til at klare vanskelige problemer indenfor kombinatorisk optimering er simuleret udglødning. Denne algoritme er også blevet tilpasset til TSP.

I dette kapitel vil jeg kort beskrive forløbet fra fysisk proces til algoritme, videre til algoritme for TSP. Der argumenteres for, hvorfor algoritmen skulle give gode resultater. Herefter følger en række forsøg, der udforsker algoritmens opførsel og resultater, for euklidiske 2-dimensionale grafer.

3.1 Fra fysisk fænomen til algoritme.

Her beskriver jeg hvorledes udviklingen er gået, fra Metropolis' artikel i 1953 beskrev at simulere udglødning af fx. metaller, til i dag hvor den generelle teknik bruges til "udglødning" af mange andre "materialer", bl.a. at placere byerne i et TSP i en hensigtsmæssig rækkefølge.

3.1.1 Udglødning, det fysiske fænomen.

Hvis man ønsker at et stof fx. skal krystallisere pænt når det størkner, kan det godt betale sig at nedkøle det langsomt. Dette kaldes på engelsk annealing, på dansk udglødning. Ved et opslag i "Ordbog over Det Danske Sprog", Nordisk Forlag, 1950, fandt jeg bl.a. ud af at udglødning er:

> [at] opvarme og derefter langsomt afkøle metal for at udligne egenspændinger, ophæve virkningen af koldstrækning.

Både ordene afkøling og krystallisering bliver brugt om denne proces på dansk, men jeg synes udglødning er den mest præcise oversættelse af navnet på en proces, der har som mål (og ikke som garanti) at krystallisere, og som kræver en omhyggelig afkøling.

Ordet udglødning havde jeg aldrig selv hørt før, men min far, der har været skibsværftsarbejder den første snes år, kunne fortælle om en proces, hvor man netop blødgør kobbertråd ved at opvarme den og så lade den afkøle i luft, fremfor dyppet i vand. Kobbertråden bliver så lettere at bøje, indtil strukturen bliver ødelagt, når den har været bøjet nogle gange. Derfor tyder det på, at det alligevel er et godt dansk ord, jeg herefter vil bruge om processen.

Med forbehold for hvad ordet smeltet præcist betyder er udglødning altså en proces, hvor man ønsker at

- hæve temperaturen, indtil materialet er smeltet
- sænke temperaturen langsomt, mens materialet stivner i den "energifattigste" tilstand, fx. en perfekt krystal

Teorien bag processen er, at atomerne under nedkølingen bliver langsommere og langsommere / har mindre og mindre energi. De vil bevæge sig rundt og undervejs falde ned i en rar position (i krystalgitteret) og vil ikke have energi nok til at komme op igen, fordi krystalgitteret netop er den energifattigste måde at arrangere atomerne på. Ved for hurtig nedkøling når alle huller i gitteret måske ikke blive udfyldt, eller der kan ske andre ting, så den perfekte krystal ikke opnås.

3.1.2 Simuleret udglødning, oprindelige version.

Med ovenstående beskrivelse af atomernes færden, kan man på computer lave en simulering af processen. Metropolis–algoritmen fra 1953 (*Metr 53*, opkaldt efter en af personerne, der opfandt den) gør netop dette på følgende måde: det smeltede materiale er i en eller anden tilstand, beskrevet ved alle atomernes positioner. Nu foreslås en lille ændring i denne tilstand, fx. at flytte et atom lidt. Energi–ændringen ved at skifte tilstand udregnes, og en forbedret tilstand (med lavere energi) accepteres. Hvis energien stiger accepteres den nye tilstand med sandsynlighed $e^{\Delta E/k_B T}$. Her er ΔE energiændringen, T den aktuelle temperatur, og k_B Boltzmann's konstant. Ifølge fysikere er dette en god beskrivelse af, hvordan atomer opfører sig. Og ifølge forskellige teorier er det nødvendigt, at der netop indgår en eksponentialfunktion i denne beskrivelse. (s.26, *Laar 87*)

Nu gennemføres en simulering. Man starter med en høj temperatur, og gennemfører et antal tilstands–ændringer ved denne temperatur. Så sænkes temperaturen lidt, flere ændringer foretages osv., og forhåbentlig ligger atomerne til sidst i det perfekte krystalgitter.

Et krav til temperatursænkningen er, at den først finder sted, når der er nået ligevægt ved den aktuelle temperatur. Dette er defineret som følger:

Lad $Z(T) = \sum_j e^{-E_j/k_BT}$, hvor j er nummeret på en tilstand, og der summeres over alle de mulige tilstande.

Lad $P_T\{X = i\} = e^{-E_i/k_BT}/Z(T)$, der er sandsynligheden for at systemet er i tilstand nr. i. Divisionen med $Z(T)$ sikrer at summen af sandsynlighederne bliver 1, som den bør.

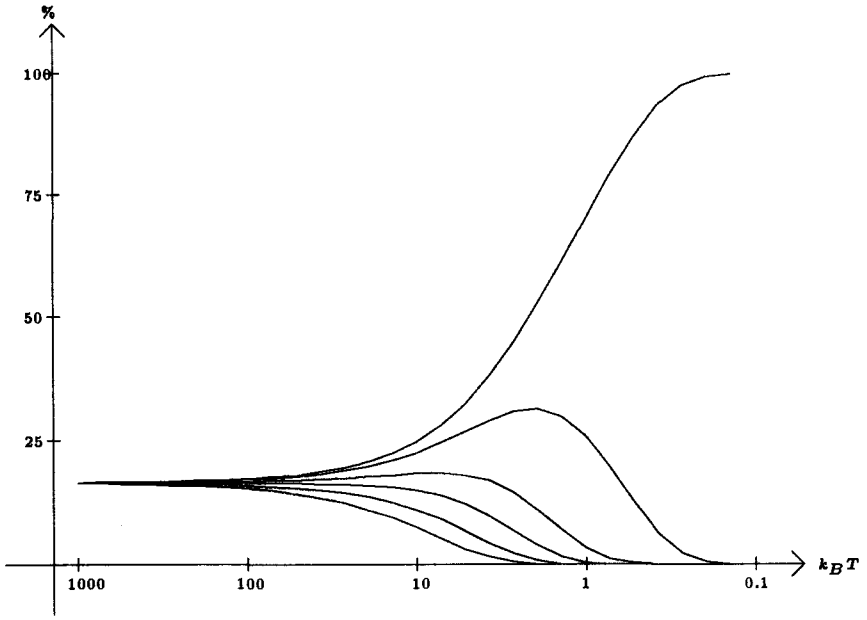
Hvis der blev foretaget et antal nedkølinger, og sluttilstanden ved denne temperatur blev noteret, skulle sandsynligheden for en given tilstand empirisk stemme overens med den ovenfor givne sandsynlighed, for at vi siger at ligevægt er opnået. Da vi her taler om sandsynligheder osv., er ligevægt også bare, at vi skal være tilpas tæt på ligevægtstilstanden, så tæt at med fx. et signifikansniveau på 95 % kunne der faktisk godt være ligevægt.

Et andet krav til nedkølingen er, at vi starter ved en høj temperatur, dette fortolkes som, at alle tilstande er lige mulige. En anden måde at sige dette på, er at hvis der er N tilstande at vælge imellem, er alle sandsynligheder $1/N$, fordi alle tilstande har samme energi.

For at se hvad disse formler betyder, kigger vi på et system med 6 forskellige tilstande, der har disse energier:

j	1	2	3	4	5	6
E_j	1	2	4	6	9	13

Nu lader jeg temperaturen falde, ved at lade k_BT dale. Ved de forskellige temperaturer regnes sandsynlighederne $P_T\{X = i\}$ ud .



Figur 3.1: Sandsynlighederne ved temperaturfald.

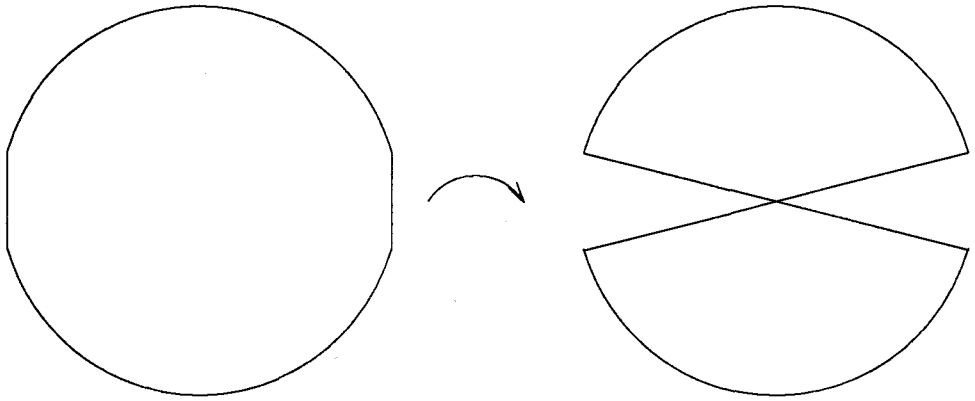
Det ses på figur 3.1, at når T er høj nok, er sandsynligheden for de forskellige tilstande stort set den samme, og det er det vi altid vil forsøge at opnå ved starten. Som T falder, falder sandsynligheden for de energirige tilstande også, så der til sidst kun er en forsvindende sandsynlighed for alle andre end den energifattigste tilstand. Undervejs vil sandsynligheden for alle tilstande stige, undtagen for den energirigeste. Jo energirigere tilstanden er, jo hurtigere vil

dens sandsynlighed dog begynde at falde igen, og til sidst vil sandsynligheden for den bedste være 1, mens sandsynligheden for de øvrige er 0.

3.1.3 Simuleret udglødning, generelle version.

Senere (først i *Kirk 83*) fandt man på, at simulere andet end fysiske systemers nedkøling. Man taler stadig om temperaturer, nedkøling, energi af tilstande osv., men der menes nu noget andet.

Hvis man prøver at angribe TSP med simuleret udglødning, vil en tilstand være en mulig tur, og energien af tilstanden længden af turen. Et forslag til en ændring af tilstanden er typisk at klippe turen τ over to steder, og lime disse to stykker sammen til en tur igen ved at bruge de to "andre" kanter, til turen τ'. Denne operation kaldes en 2-opt, og anvendes af mig selv og flertallet. Med et symmetrisk problem beregnes ændringen i energi nemt. $k_B T$ kaldes blot T, dens værdi har alligevel ikke ret meget med rigtige temperaturer at gøre mere. Så en energi–forøgende ændring accepteres altså med sandsynlighed $e^{(c(\tau)-c(\tau'))/T}$.



Figur 3.2: Ændring af tur, vha. 2-opt.

Gloser.

Tilstand: τ (tur).

Energi af tilstand: $c(\tau)$(længde af tur).

Energi–ændring: forskellen på to tilstandes energier.

Tilstands–ændring: en mulig ændring fra en tilstand til en anden (for ture, se figur 3.2).

Nabo: tilstand der kan findes, ved at foretage en tilstands–ændring på den aktuelle tilstand.

Temperatur: T, en parameter, der afgør sandsynligheden for, at en energi–forøgende tilstands–ændring udføres.

Transformation: omfatter at foreslå en tilstands–ændring, og evt. at acceptere/udføre den.

Kæde: alle transformationer foretaget ved samme temperatur.

Længde af kæde: L_k, antallet af transformationer i den k'te kæde.

Algoritmen simuleret udglødning.

I meget løse gloser ses her den generelle udgave af algoritmen simuleret udglødning.

Algoritme 9 *Simuleret udglødning 1*

find på tilfældig τ
vælg høj T
gentag
    gentag
        find en tilfældig nabo τ' til τ
        hvis $c(\tau') \leq c(\tau)$
            $\tau = \tau'$
        ellers
            med sandsynlighed $e^{(c(\tau)-c(\tau'))/T}, \tau = \tau'$
    indtil ligevægt ved denne T
    find ny, lavere T
indtil systemet er frosset

Algoritme slut

I et lidt mere datalogisk sprog ser algoritmen således ud.

Algoritme 10 *Simuleret udglødning 2*

initialisering(τ, T_0, L_0)
$k := 0$
repeat
    for $l := 1$ to L_k do
        $\tau' =$ tilfældig nabo(τ)
        if $c(\tau') \leq c(\tau)$ then
            $\tau = \tau'$
        else
            if $e^{(c(\tau)-c(\tau'))/T} >$ random[0,1) then
            (* tilfældigt tal mellem 0 og 1, 1 ikke inklusive *)
                $\tau = \tau'$
    (* end for *)
    (* ligevægt opnået efter L_k transformationer *)
    $k := k + 1$
    udregn(L_k, T_k)
until stopkriterie

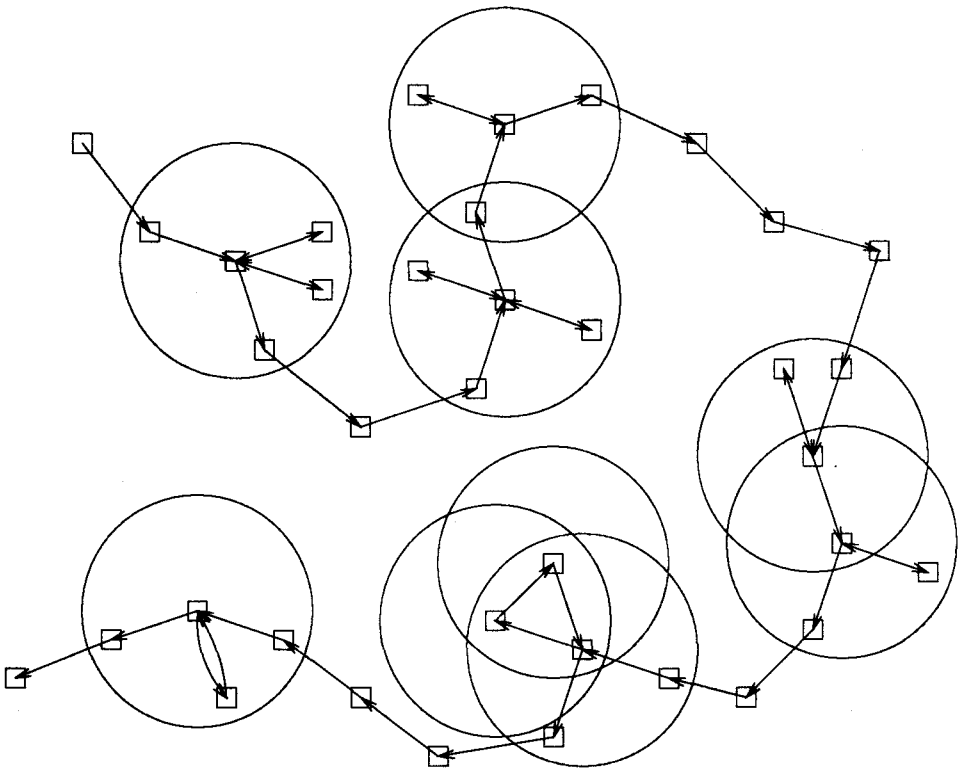
Algoritme slut

Som med det fysiske fænomen defineres nogle regler for ligevægt:
$Z(T) = \sum_\tau e^{c(\tau)/T}$.
$P_T\{X = \tau\} = e^{c(\tau)/T}/Z(T)$.
Kravene der stilles til algoritmen er følgende:

- Find T_0, så $P_T\{X = \tau\}$ giver den uniforme fordeling.
- Find L_k, så ligevægt ved T_k opnås.

- Vælg ikke nye T_k, så temperaturen falder for hurtigt. (Hvis temperaturen falder for hurtigt bliver det måske nødvendigt at tilbringe længere tid ved hver temperatur, for at opnå ligevægt inden temperaturskift. Alt i alt bruges der måske så længere tid i alt også.)
- Stopkriterie: høj sandsynlighed for at den aktuelle τ er en optimal løsning, eller i hvert fald tæt på. (Opnås når systemet er "frosset".)
- En nabo er defineret således, at nabolagene er "gode". Dette betyder stort set bare, at man fra alle løsninger via et antal nabolag skal kunne komme til alle andre.

3.1.4 En vandring i bjerglandskabet.



Figur 3.3: Vandring i simuleret udglødning.

I figur 3.3 ses et eksempel på en vandring i "bjerglandskabet" simuleret udglødning udforsker. Det fremgår ikke hvilke af de udforskede punkter (dvs. ture, illustreret med firkanter) der ligger højt i landskabet (har høj energi), fordi det ses ovenfra. Hvis mere end 2 af et punkts nabopunkter udforskes, angiver en cirkel størrelsen af nabolaget, og hvilke af de udforskede punkter, der er med i nabolaget. Bemærk at naboer er hinandens naboer i dette tilfælde, så man altid kan komme tilbage hvor man lige er kommet fra, pga. at definitionen for nabolag er symmetrisk.

Der kan ske mange ting i en vandring. Med i tegningen er der bl.a. alle de gange, hvor et nyt punkt udforskes, og omgående forkastes til fordel for det gamle (angivet med dobbeltpil). En enkelt gang tages det nye punkt, hvorefter det gamle punkt, der er med i det nyes nabolag, tages igen. Nabolag overlapper selvfølgelig, og det kan lade sig gøre at støde på det gamle punkt mere end en gang. For ikke at forvirre har jeg ikke taget med, at man kan komme tilbage til et punkt, der blev besøgt for lang tid siden.

3.2 Hvorfor virker simuleret udglødning?

Her vises hvordan man med probabilistisk analyse kan sandsynliggøre, at man ved simuleret udglødning får nær-optimale løsninger.

3.2.1 Markov–kæder.

Et af kravene ovenfor var, at en kæde skulle være lang nok, til at ligevægt opnås. Nu prøver vi om teorien for Markov–kæder kan bruges på vores kæder, til at fortælle hvad "lang nok" er. I dette afsnit er tilstande udfald med sandsynligheder, der tilhører et udfaldsrum, der omfatter alle mulige udfald. En transformation kaldes et forsøg.

Definition 5 *En Markov–kæde er en serie af forsøg, hvor sandsynlighederne for udfaldet af hvert forsøg kun afhænger af udfaldet af det foregående forsøg.*
En Markov–kæde kaldes endelig, hvis antallet af forsøg i den er endeligt.
En Markov–kæde kaldes inhomogen, hvis sandsynlighederne afhænger af tidspunktet for forsøget / hvilket nummer forsøget har i kæden. Ellers kaldes den homogen.

I vores tilfælde afhænger hvert udfald netop af det foregående, fordi vi kan få en nabo til den aktuelle tilstand, og dermed har nogle veldefinerede udfaldsrum, et for hvert muligt udfald. Vi ønsker at vores Markov–kæder skal være endelige, og da vores udfald afhænger af temperaturen, der varieres undervejs, er den samlede Markov–kæde inhomogen, eller en serie af homogene Markov–kæder.

Definition 6 *En Markov–kæde kaldes irreducibel hvis man kan komme fra et vilkårligt udfald til et vilkårligt andet udfald i et endeligt antal skridt.*
En Markov–kæde kaldes aperiodisk hvis et vilkårligt udfald kan indtræffe flere gange i træk.

I vores definition af gode nabolag var vi netop inde på, at man skal kunne komme fra alle steder til alle andre steder, så vi får irreducible Markov–kæder. Og en transformation giver mulighed for ikke at ændre noget, så vi får aperiodiske kæder.

Derudover skal vi kende noget mere til Markov–kæden, nemlig de sandsynligheder der bliver talt om. I alt bliver der en $N \times N$–matrix, der for N forskellige mulige udfald beskriver sandsynligheden P_{ij} for at komme fra udfald τ_i til udfald τ_j. I vores tilfælde afhænger disse sandsynligheder af to ting, G_{ij}, der er sandsynligheden for at generere udfaldet τ_j (0 hvis τ_j ikke er en nabo til τ_i) og A_{ij}, der er sandsynligheden for, at τ_j bliver accepteret som ny udfald, hvis τ_j blev genereret. Så har vi

(1a) $P_{ij} = G_{ij}A_{ij}$, når $i \neq j$.

(1b) $P_{ij} = 1 - \sum_{k=1,k\neq i}^{N} G_{ij}A_{ij}$, når $i = j$.

Faktisk har vi altså 3 matricer, en for A_{ij}, en for G_{ij} og en for P_{ij}. Formelt:

(2a) $G_{ij} = 1/M$, hvis τ_j er en af τ_i's naboer, hvor M er antallet af naboer til τ_i.

(2b) $G_{ij} = 0$ hvis τ_j ikke er en af τ_i's naboer.

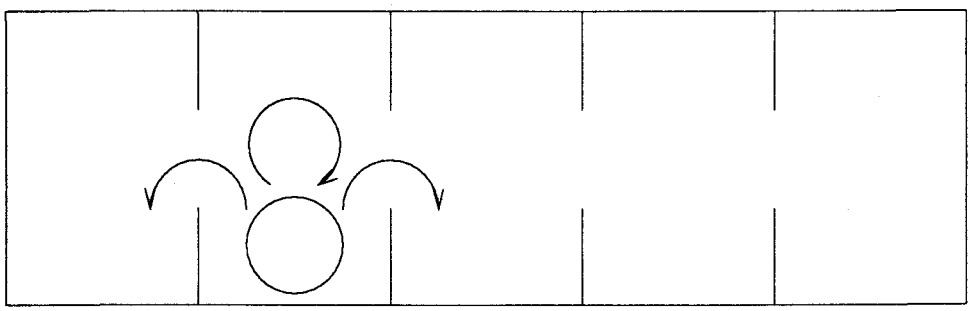
$A_{ij} = e^{(c(\tau_i)-c(\tau_j))/T}$ hvis der er tale om energiforøgelse, ellers 1. Dette kan også skrives som

(3) $A_{ij} = min\{1, e^{(c(\tau_i)-c(\tau_j))/T}\}$.

Lad os sige, at vi har en N–vektor, q, der beskriver hvilken tilstand vi er i. Vi er i tilstand τ_i, derfor er det i'te element i vektoren 1, og resten 0. Nu kan vi udregne sandsynligheden for at være i en bestemt tilstand efter L_k transformationer. Efter 1 transformation er sandsynlighederne givet ved q_1 der findes ved at gange P på q_0. Vi kan gøre endnu et skridt ved at gange P på en gang til. Og vi kan gøre L_k forsøg ved at gange P på L_k gange. Vi skal bare have en start–vektor q_0 og P. Den resulterende vektor er da forhåbentlig uafhængig af den vektor vi begyndte med, som tegn på ligevægt.

3.2.2 Et eksempel.

Et simpelt eksempel med få tilstande er en kasse med 5 rum, og en bold. (fra s.38-42, *Torr 88*) Se figur 3.4.



Figur 3.4: Kasse med 5 rum og 1 bold.

Lad os sige, at hvis kassen bliver rystet, vil bolden enten hoppe ud af sit rum til et naborum med sandsynlighed 0.5, eller efter fx. et sekund stadig være i samme rum, også med sandsynlighed 0.5. Med lige stor sandsynlighed for at gå til højre og venstre får vi en matrix P, der ser således ud:

$$P = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix}$$

Hvis bolden starter i det andet rum er vektoren, der beskriver start–situationen $q_0 = (0,1,0,0,0)$.

Efter at have ganget P på én gang er vektoren $q_1 = (0.25,0.5,0.25,0,0)$, dvs. vi får nogle sandsynligheder der svarer til beskrivelsen af hvad der sker efter at have rystet kassen én gang.

Efter at have ganget P på én gang til får vi $q_2 = (0.25,0.4375,0.25,0.0625,0)$, der beskriver hvordan bolden med en lille sandsynlighed efter to hop kan være nået to rum væk, men med størst sandsynlighed stadig er i et af de tre første rum.

P ganges på endnu en gang, og vi får $q_3 = (0.2343,0.4062,0.25,0.0937,0.0156)$, og at bolden nu kunne være nået over i det femte rum.

Skrevet op i et lille skema, sammen med en angivelse af hvor mange gange P er ganget på opsummeres dette til:

0	0	1	0	0	0
1	0.25	0.5	0.25	0	0
2	0.25	0.4375	0.25	0.0625	0
3	0.2343	0.4062	0.25	0.0937	0.0156
13	0.1475	0.2819	0.25	0.2181	0.1024
18	0.1352	0.2645	0.25	0.2355	0.1148
∞	0.125	0.25	0.25	0.25	0.125

Det bemærkes, at hvis der er konvergens, er det netop den nederste vektor, der vil blive konvergeret imod. Der er nemlig brug for en vektor (a,b,c,d,e) så $P(a,b,c,d,e) = (a,b,c,d,e)$, og dermed:

$$\begin{aligned}
1: \quad 0.5a + 0.25b &= a \\
2: \quad 0.5a + 0.5b + 0.25c &= b \\
3: \quad 0.25b + 0.5c + 0.25d &= c \\
4: \quad 0.25c + 0.5d + 0.5e &= d \\
5: \quad 0.25d + 0.5e &= e
\end{aligned}$$

$$\begin{aligned}
1: \quad 0.25b &= 0.5a \\
5: \quad 0.25d &= 0.5e \\
2: \quad 0.5a + 0.5b + 0.25c &= b \\
3: \quad 0.25b + 0.5c + 0.25d &= c \\
4: \quad 0.25c + 0.5d + 0.5e &= d
\end{aligned}$$

$$\begin{aligned}
1: \quad b &= 2a \\
5: \quad d &= 2e \\
2: \quad 0.5a + a + 0.25c &= 2a \\
3: \quad 0.5a + 0.5c + 0.5e &= c \\
4: \quad 0.25c + e + 0.5e &= e
\end{aligned}$$

$$\begin{aligned}
1: \quad b &= 2a \\
5: \quad d &= 2e \\
2: \quad 0.25c &= 0.5a \\
4: \quad 0.25c &= 0.5e \\
3: \quad 0.5a + 0.5e &= 0.5c
\end{aligned}$$

$$\begin{aligned}
0.5a = 0.25c = 0.5e &\Rightarrow a = e \\
b = 2a = 2e = d &\Rightarrow b = d = 2a \\
0.5a + 0.5e = 0.5c \Leftrightarrow a = 0.5c &\Rightarrow c = 2a
\end{aligned}$$

$a + b + c + d + e = 1 \Rightarrow a + 2a + 2a + 2a + a = 1 \Rightarrow 8a = 1 \Leftrightarrow a = 0.125$
$a = 0.125, b = 0.25, c = 0.25, d = 0.25, e = 0.125$

Hvis bolden i stedet var startet i det femte rum havde skemaet set sådan ud:

0	0	0	0	0	1
1	0	0	0	0.5	0.5
2	0	0	0.125	0.5	0.375
3	0	0.0312	0.1875	0.4687	0.3125
15	0.1018	0.2171	0.2500	0.2829	0.1483
20	0.1145	0.2351	0.2500	0.2649	0.1355
∞	0.125	0.25	0.25	0.25	0.125

Som det ses er konvergensen rimelig hurtig, så i dette tilfælde skulle længden af Markov-kæden / antallet af ryst/forsøg ikke være ret stort (måske 30) før vi er tæt nok på ligevægten, til at en ny Markov-kæde kan begynde.

En anden kæde kunne laves med kassen hældet lidt, så bolden hellere vil til højre end til venstre. Den nye matrix ville så fx. blive:

$$P = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.1 & 0.5 & 0.4 & 0 & 0 \\ 0 & 0.1 & 0.5 & 0.4 & 0 \\ 0 & 0 & 0.1 & 0.5 & 0.4 \\ 0 & 0 & 0 & 0.1 & 0.9 \end{pmatrix}$$

Med denne P bliver udviklingen naturligvis en anden. Start med bolden i det andet rum, og skemaet bliver således:

0	0	1	0	0	0
1	0.1	0.5	0.4	0	0
2	0.1	0.34	0.4	0.16	0
3	0.084	0.26	0.352	0.24	0.064
8	0.0293	0.0940	0.1678	0.2453	0.4636
12	0.0138	0.0474	0.1009	0.2151	0.6227
17	0.0064	0.0244	0.0662	0.1977	0.7053
∞	0.0023	0.0117	0.0469	0.1877	0.7511

3.2.3 Mere om Markov-kæder.

Det nævnes uden bevis, at (1) fra definitionen af P_{ij} (og "ergodicity", som jeg ikke definerer) medfører: (s.42-43, *Torr 88*)

Sætning 26 *For en homogen Markov-kæde eksisterer grænseværdien* $lim_{k\to\infty} q_k = q_\infty$.

Sætning 27 *For en inhomogen Markov-kæde, og med P defineret som ovenfor i (1), (2) og (3), eksisterer der en grænseværdi* $lim_{k\to\infty} q_k = q_\infty$.

Lad τ^* være en optimal løsning, og lad (2) og (3) være opfyldt, da gælder:

Sætning 28 *På i'te plads i* q_∞ *står* $e^{-(c(\tau_i)-c(\tau^*))/T}/Z(T)$, *hvor N er antallet af tilstande og* $Z(T) = \sum_{r=1}^{N} e^{-(c(\tau_r)-c(\tau^*))/T}$.

Dette er også vores tidligere mål for opnået ligevægt, så vi har nu set, at tilstandene i løbet af Markov-kæden faktisk konvergerer mod denne ligevægt.

Hvis $T \to 0$ gælder der desuden, af der på i'te plads står 0, hvis $c(\tau_i) > c(\tau^*)$ og hvis K er antallet at optimale løsninger står der $1/K$ hvis $c(\tau_i) = c(\tau^*)$.

3.3 Beskrivelse af mine forsøg.

Lad os endnu en gang kigge på algoritmen, og indføre et muligt nedkølingsskema.

3.3.1 Et simpelt nedkølingsskema.

Algoritme 11 *Simuleret udglødning A*

$L = n(n-3)/2$
initialisering(τ, χ_0, α)
find T_0 ved hvilken procentdelen χ_0 af transformationerne accepteres
$k := 0$
repeat
  for $l := 1$ to L do
    $\tau' =$ tilfældig nabo(τ)
    if $c(\tau') \leq c(\tau)$ then
      $\tau = \tau'$
    else
      if $e^{(c(\tau)-c(\tau'))/T} >$ random[0,1) then
      (* tilfældig tal mellem 0 og 1, 1 ikke inklusive *)
        $\tau = \tau'$
  (* end for *)
  (* ligevægt opnået efter L_k transformationer *)
  $k := k + 1$
  $T_k = \alpha T_{k-1}$
until stopkriterie

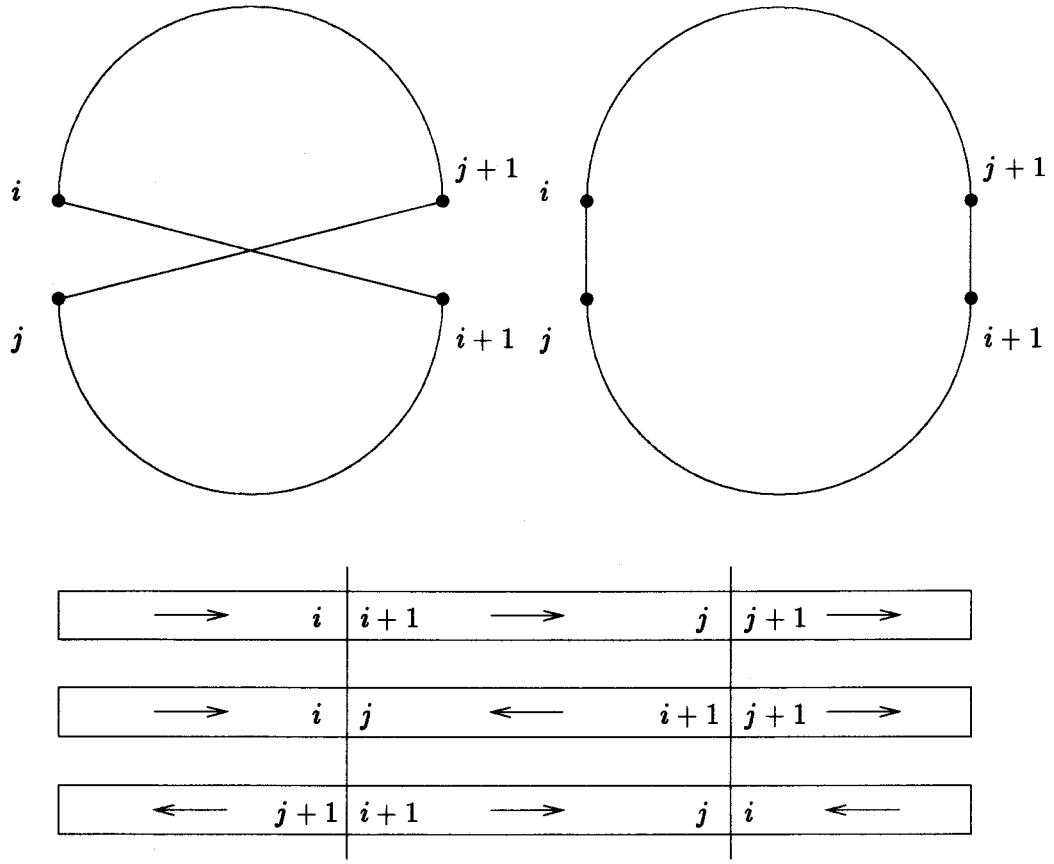
Algoritme slut

For at opsummere: et nedkølingsskema fastlægger starttemperaturen, længden af Markov-kæderne, måden en ny temperatur findes på, og hvornår algoritmen skal standse.

I stedet for den variable L_k udregnes nu en gang for alle længden af Markov-kæderne, og den sættes til samme størrelse som størrelsen af et nabolag. (Dette er en version af en de lette metoder i *Laar 87*, tilskrevet *John 86*, hvor der bruges et multiplum af nabolagets størrelse.)

Jeg finder en tilfældig nabo til τ ved at finde en tilfældig 2-opt, og denne findes igen ved at finde 2 byer. Den ene findes helt tilfældigt blandt de n mulige, den anden må ikke være den først fundne, eller de 2 lige før. (På figur 3.5 svarer dette til, at den første valgte by er nr. i, og $j+1$ så ikke må være hverken $i, i-1$ eller $i-2$.) Jeg fortolker nemlig de 2 byer således, at stykket mellem disse 2 byer, byerne inklusive, nu skal vendes, og at der skal være mindst én kant tilbage, der ikke bliver vendt. I praksis implementerer jeg i øvrigt 2-opt således, at det stykke af mit array, der ligger mellem byerne, bliver vendt, dvs. hvis byerne ligger forkert i forhold til hinanden (dvs. både 1. og n.by skal vendes), vender jeg det modsatte stykke, med samme effekt. Se figur 3.5 for de to forskellige måder at bytte byer på, når stykket mellem i og $j+1$ vendes. Her er det øverste af de tre arrays det oprindelige, mens begge de to nedenunder repræsenterer samme nye tur, og jeg altså anvender den metode, det midterste array viser: at vende midterstykket.

Da der således ikke er $n(n-3)$ forskellige 2-opt, men kun halvt så mange, bliver $L = n(n-3)/2$.



Figur 3.5: 2 måder at lave en 2-opt på.

χ_0 er et reelt tal under, men tæt på 1. En måde at finde T_0 på er at komme med et forslag til T_0, og teste hvor mange tilstandsændringer der bliver accepteret. (Jeg antager, at jeg som under selve simuleringen faktisk udfører de accepterede tilstandsændringer. Dette medfører bl.a. at jeg ikke er bundet til forholdene omkring min starttur, og følger iøvrigt konklusionen om dette fra *Ebse 92*). Forhåbentlig bliver der accepteret for få, således at et nyt forslag til T_0, der er højere, nødvendigvis vil være et bedre forslag, eller i hvert fald lige så skidt. Hvis der ikke blev accepteret en procentdel af transformationerne på χ_0, foreslås et nyt, højere T_0, fx. ved at gange det gamle med 2, og man prøver igen. Denne metode anvender jeg selv, med et spring på en faktor 2 mellem forslagene, således at man rimeligt hurtigt finder et anvendeligt T_0, og kan komme i gang med selve algoritmen. (Simpleste metode nævnt af *Laar 87*, fra *Kirk 83*.) Jeg udfører 100 transformationer, og vurderer så om forslaget kan bruges. Hvis jeg kører algoritmen flere gange, under samme forhold, finder jeg kun T_0 en enkelt gang.

α bruges når et nyt T_k regnes ud, som fremgår. (Denne metode foreslås ifølge *Laar 87* første gang af *Kirk 83*, men bruges også af andre, med værdier mellem 0.5 og 0.99.)

Algoritmens stopkriterie er i mit tilfælde at tælle hvor mange tilstandsændringer der er. Den hidtil bedste tur, algoritmen er stødt på, opbevares. Hvis der ikke er forbedret på den bedste tur i et antal iterationer (dette antal kaldes *niveau*), og der i øvrigt er accepteret færre end 1 % tilstandsændringer i hver af disse iterationer, standser jeg algoritmen, fordi systemet nu er frosset hårdt, og løsningen tilsyneladende ikke bliver bedre. Jeg forestiller mig, at næsten hele den fundne løsnings nabolag bliver afsøgt i disse sidste iterationer, hvis algoritmen i øvrigt er tæt på den bedste løsning hen mod slutningen. (Dette er min egen fortolkning af den metode,

jeg fandt i *Laar 87*, tilskrevet bl.a. *Kirk 83*, forfinet af *John 86*.)

3.3.2 Et mere avanceret nedkølingsskema.

Der er kommet mange bud på hvordan det simple skema kan gøres mere avanceret (fx. i *Otte 84, Aart 85, Rome 85, Huan 86 og Lund 86*, alle nævnt i *Laar 87* og sammenlignet), og dermed mere effektivt mht. køretid og/eller resultater. Et af disse bud indvirker på køretiden ved at gøre Markov-kæderne kortere, hvis det er muligt at opnå ligevægt hurtigt, og skulle dermed give lige så gode resultater på en kortere tid. (*Rome 85, Laar 87*)

For at opnå et globalt minimum, skal der altid være en "tilstrækkelig" sandsynlighed for at forlade enhver tilstand, også et lokalt minimum.

Lad $\tilde{N}_i(T)$ være det forventede antal transformationer inden tilstand i forlades.

$\tilde{N}_i(T) = (1 - P_{ii}(T))^{-1}$.

Lad $|R_i|$ være størrelsen af nabolaget R_i til τ_i.

$P_{ii}(T) = 1 - \frac{1}{|R_i|} \sum_{\tau_j \in R_i} min\{1, e^{\frac{-(c(\tau_j)-c(\tau_i))}{T}}\} = 1 - \frac{1}{|R_i|} \sum_{\tau_j \in R_i} A_{ij}$

Egentlig er man interesseret i $\tilde{N}_i(T)$'s maximumsværdi blandt alle i, og denne kan vurderes ved at erstatte $c(\tau_j) - c(\tau_i)$ med $max_j\ c(\tau_j) - min_i\ c(\tau_i)$, der igen erstattes af de værdier, der kendes, dvs. i og j tages blandt de hidtil mødte tilstande, $\hat{i}$ og $\hat{j}$ vælges. Således opnås:

$\tilde{N}_i(T) = (e^{\frac{-(c(\tau_j)-c(\tau_i))}{T}})^{-1}$ og $L_k \propto \tilde{N}_i(T)$, fx. $L_k = 4 \cdot \tilde{N}_i(T)$

Bortset fra denne modifikation af skemaet benyttes det ovenfor viste, og selv da checkes det, hvor lang hver kæde er, så den ikke bliver meget længere end efter den gamle definition. Faktisk beregnes begge L_k, og den korteste bruges.

3.3.3 Alternative nedkølingsskemaer.

I de fleste kilder stødte jeg på, at der var meget arbejde for at få kortere Markov-kæder, hurtigere nedkøling, eller bedre start- og sluttemperatur. Efter at have foretaget få kørsler af simuleret udglødning så jeg ture, der kunne forbedres af den specielle 3-opt, der bare flytter et enkelt punkt, og jeg overvejede derfor konsekvensen af at supplere nabolagene, fundet vha. 2-opt, med denne 3-opt. (Se kapitel 1 for mere om denne 3-opt.)

Senere fandt jeg ud af (bl.a. i *John 90*), at dette har været prøvet, med forskellige versioner af 3-opt inddraget, og at resultaterne ikke var bedre end hvis kun 2-opt blev brugt.

På den anden side påstår *Yao 91*, at større nabolag er bedre i starten af en kørsel, hvis alt andet holdes fast.

Med andre ord kunne det være spændende at implementere nogle af disse udvidede nabolag og se hvad der sker.

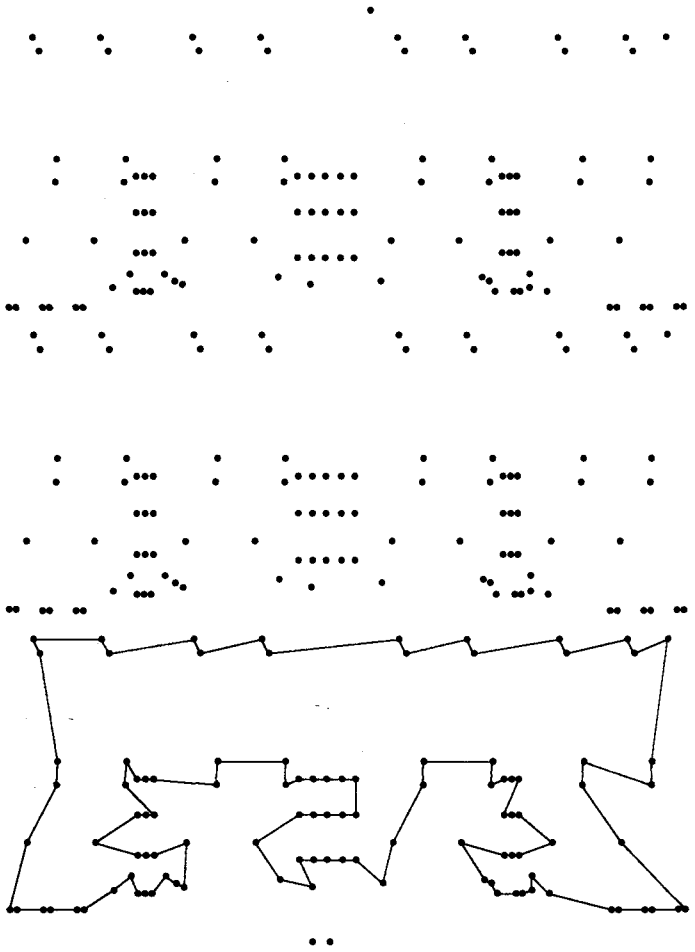
Et andet forslag til at forkorte Markov-kæderne kommer fra *Ebse 92*, at lade en Markov-kæde have længden $L_k = \beta \cdot \frac{n(n-3)}{2}$, hvor β så varieres, og der findes en værdi hvor resultaterne stadig er gode, men køretiden er blevet kortere.

Det kunne også være spændende at bruge backtracking, dvs. registrere de mulige lokale minima undervejs, sammen med den aktuelle temperatur, og så genstarte kørslen her, fremfor helt forfra.

3.3.4 Problemerne.

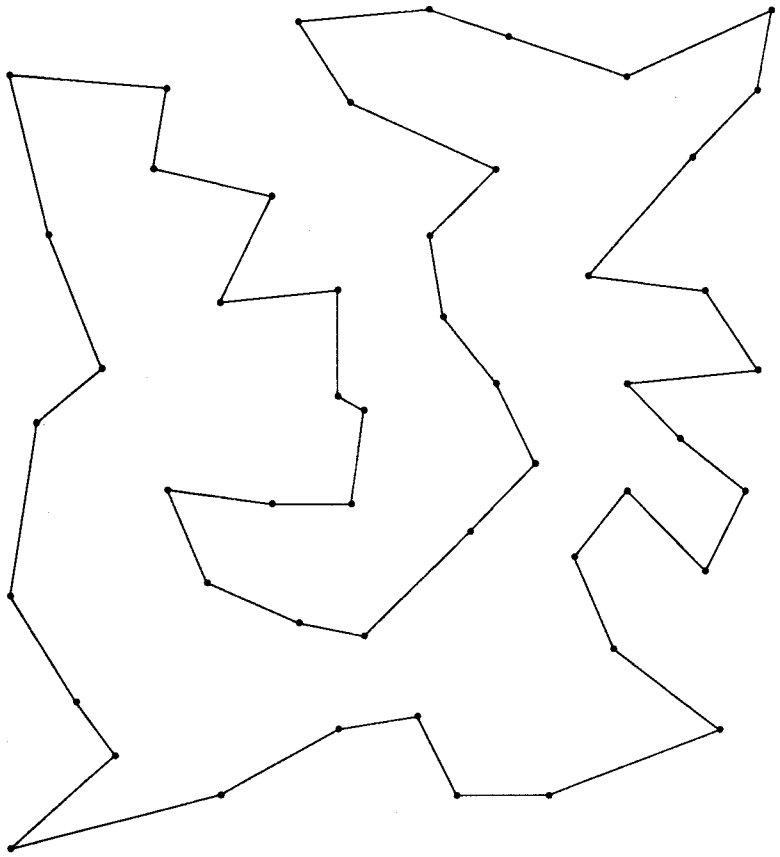
Under afprøvningen af programmet beskrevet ovenfor, prøvede jeg at løse forskellige problemer, jeg selv havde lavet. Disse var 2-dimensionale, euklidiske problemer, hvor jeg fandt n tilfældige punkter med koordinater, hvis værdier lå mellem 1 og $3\sqrt{n}+1$, således at punkterne ligger i et kvadrat. For n=16 er bredden 12, for n 49 er bredden 21, og for n 100 er bredden 30. Det forudsiges at i et enhedskvadrat er den optimale tur højst $\sqrt{2n}+1.75$ lang. For mig bliver denne

formel $(\sqrt{2n} + 1.75)3\sqrt{n}$, der for n=16 giver 88.88, for n 49 244.64 og for n 100 giver 476.76. Punkterne har heltallige koordinater, mens afstandene beregnes som reelle tal. Dobbeltpunkter produceres ikke. For hvert n ville jeg straks producere 5 forskellige problemer.



Figur 3.6: Grafen lin318.

I de følgende forsøg har jeg primært foretaget forskellige forsøg på problemerne eil51 og lin318 fra tsp-biblioteket, hvis optimale løsning vides at have længde hhv. 426 og 42029. Disse problemer har kun heltallige (dvs. afrundede) afstande. I figur 3.6 ses punkterne fra lin318, de er ordnet i 3 grupper på 105 punkter (den optimale tur gennem sådan en gruppe, længde 14.379, er angivet) og 3 ekstra punkter, et foroven, to forneden. I figur 3.7 ses grafen eil51, med den optimale tur angivet.



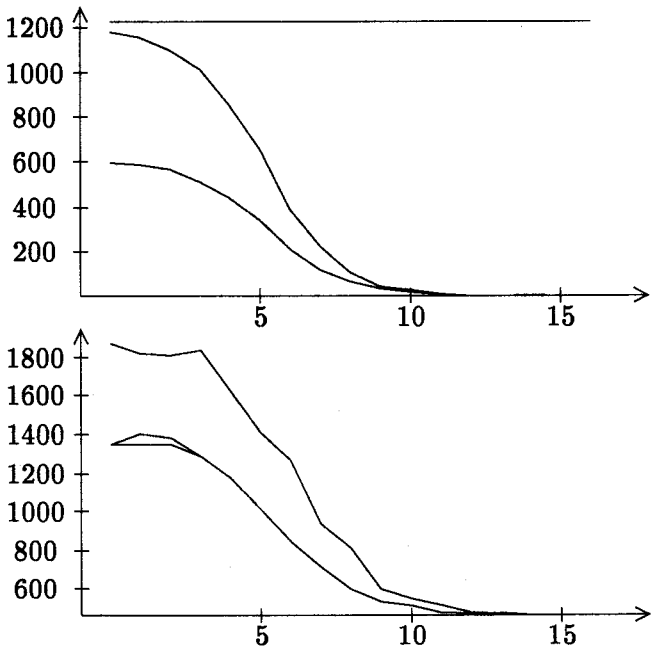
Figur 3.7: Grafen eil51.

3.4 Generelle forsøg, indstiling af parametre.

I dette afsnit udfører jeg forsøg for at besvare forskellige generelle spørgsmål, der helst skal være besvaret før man kører simuleret udglødning for alvor.

Spørgsmål 1 *Hvordan ser en typisk kørsel ud, hvor ofte findes en ny bedste løsning, hvor mange forslag accepteres pr. iteration osv.?*

Forventet svar: teorien passer godt sammen med kurvernes udseende. Og denne teori forudsiger, at hvis algoritmen kører godt, vil lokale minima ofte blive forladt, og bedre blive fundet, og færre tilstandsændringer vil blive accepteret når temperaturen falder, osv.



Figur 3.8: eil51, α=0.6.

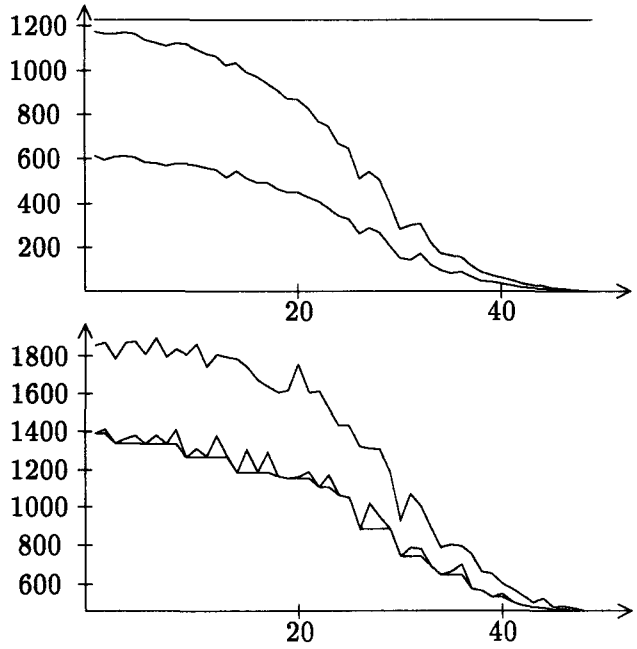
Ved disse og de følgende figurer vises 2 ting ved forløbet af en enkelt kørsel. Foroven i figuren ses hvor mange tilstandsændringer, der er forbedrende (dvs. den resulterende tur er kortere) (nederste kurve), hvor mange der accepteres (mellemste kurve) og hvor mange der foreslås (øverste kurve). Da længden af Markovkæderne for dette nedkølingsskema er konstant, bliver den øverste kurve bare en ret linje. Forneden i figuren ses kvaliteten af de gennemløbne tilstande, nemlig den bedste hidtil sete tur (nederste kurve), den bedste tur i den givne iteration (mellemste kurve) og den dårligste tur i en given iteration (øverste kurve). x-aksen angiver hvilket nummer iterationen har, y-aksen angiver hhv. antallet af transformationer og længden af turen. For eil51 er længden af en Markovkæde 1224 ($\frac{51 \cdot 48}{2}$), og for lin318 50085.

Figur 3.8 viser forløbet ved en rimelig lav α, nemlig 0.6. Dette og de to følgende forsøg afvikles ved $\chi_0 = 0,95$ og *niveau* = 3. Der er 18 iterationer, den bedste tur har længde 455, og den dårligste længde 1868.

Bortset fra lidt i starten følges de 2 nederste kurver forneden i figuren ad, som tegn på at der bliver fundet en ny bedste tur i hver iteration. Hvis kurverne overhovedet skilles ad med denne lave α er det næsten altid i den første halvdel af kørslen. Argumentet for at bruge simuleret

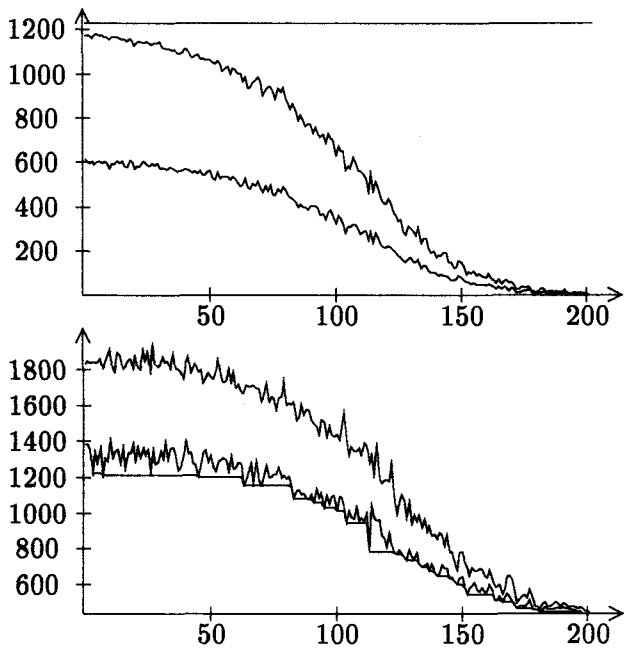
udglødning er netop, at det skal kunne lade sig gøre i meget høj grad at komme væk fra lokale minima, og denne opførsel er ikke ret tydelig ved denne dårlige α.

Bemærk at alle 3 kurver falder, også den der angiver længden af den dårligste tur i en given iteration. Dels bliver den tur, der 2-opt'es altså bedre og bedre, og dermed også den resulterende tur, dels accepteres de meget lange ture (pga. forværringen i forhold til den aktuelle) jo ikke til sidst.



Figur 3.9: eil51, α=0.9.

På figur 3.9 er der 51 iterationer, bedste tur har længden 454 og dårligste 1892. Her ses meget tydeligere, at et fundet minimum forlades i en eller flere iterationer, for derefter at blive erstattet af et lavere minimum. Og dette sker også i sidste halvdel af kørslen.

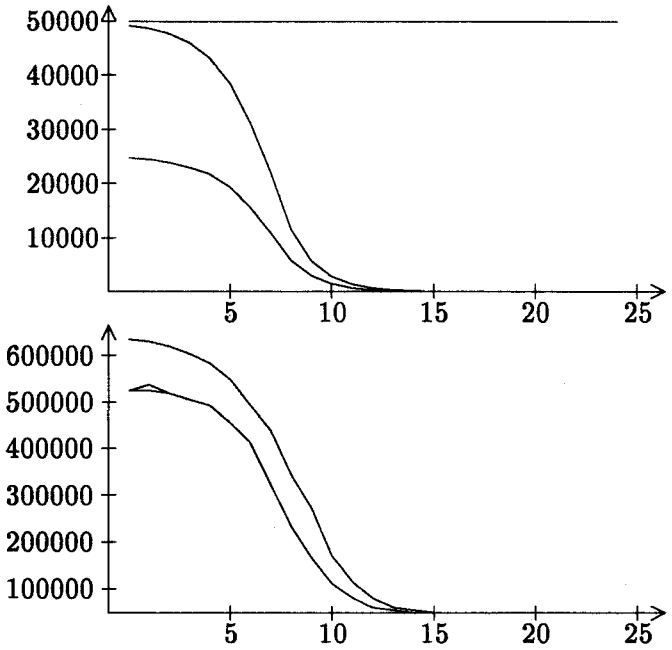


Figur 3.10: eil51, α=0.975.

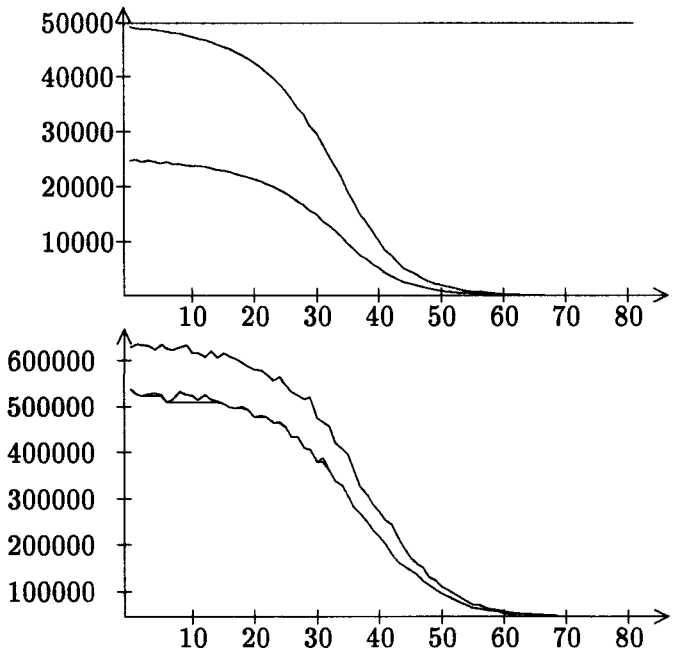
Her på figur 3.10 går den mellemste kurve væk fra den nederste i endnu længere tid af gangen. Det dramatiske fald i den nederste kurve ca. ved 110. iteration følges ikke op, og giver kun anledning til, at der går længere tid, inden dette lokale minimum bliver erstattet af et bedre. I øvrigt er det jo ikke til at vide, om disse lokale minima faktisk er lokale minima, vi ved kun, at de bliver forladt til fordel for ture der er længere, og ikke om man kunne komme fra dem til en kortere tur. For dette forsøg var der 204 iterationer, dårligste resultat var 1949 og bedste 437.

I øvrigt besvarer disse kurver mest min nysgerrighed for noget, jeg ikke fandt andre steder. Tilsyneladende bliver der i en iteration taget ca. lige mange gode og dårlige tilstandsændringer, som om ét dårligt valg giver mulighed for ét godt. Kurverne er uniforme på den måde, at en kørsel, uanset antallet af iterationer, har samme bredde. På denne måde ses let, at kurverne ligner hinanden meget i generel facon. I alle 3 figurer ses ca. samme facon på kurverne foroven, og til slut et fladt stykke af ca. konstant længde, svarende til de *niveau* sidste iterationer.

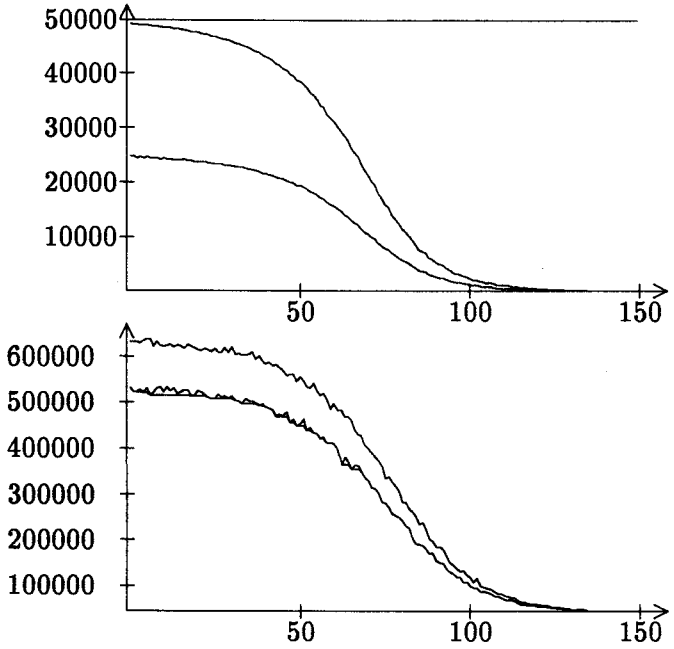
Herefter følger figur 3.11 til figur 3.14, hvor samme forsøgsrække vises på lin318. Samme iagttagelser, bare med glattere kurver fordi målestokken er højere. Igen er $\chi_0 = 0,95$ og *niveau* = 3, hvilket faktisk er for lavt ved de høje α, således at bedre resultater ville produceres ved højere *niveau* og dermed få ekstra iterationer. Den bedste tur er ved hver af de 4 forsøg af længde 46.090, 44.015, 43.361 og 46.146.



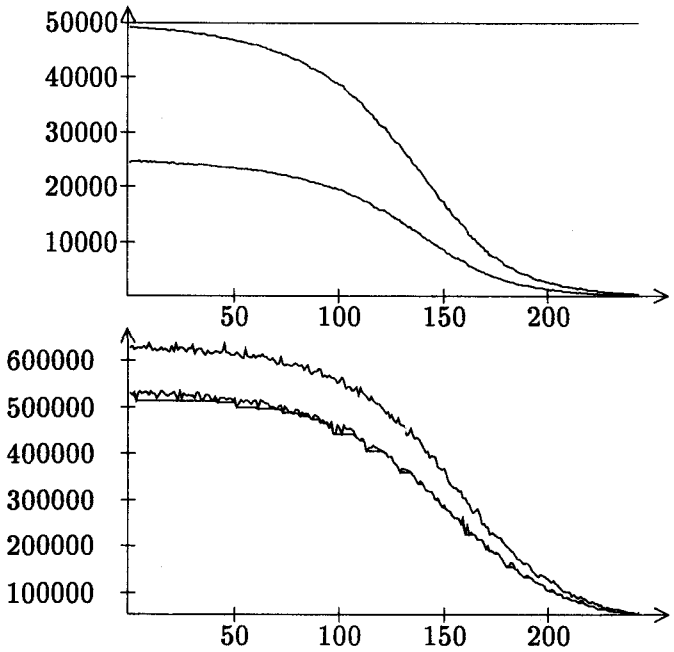
Figur 3.11: lin318, α=0.6.



Figur 3.12: lin318, α=0.9.



Figur 3.13: lin318, α=0.95.

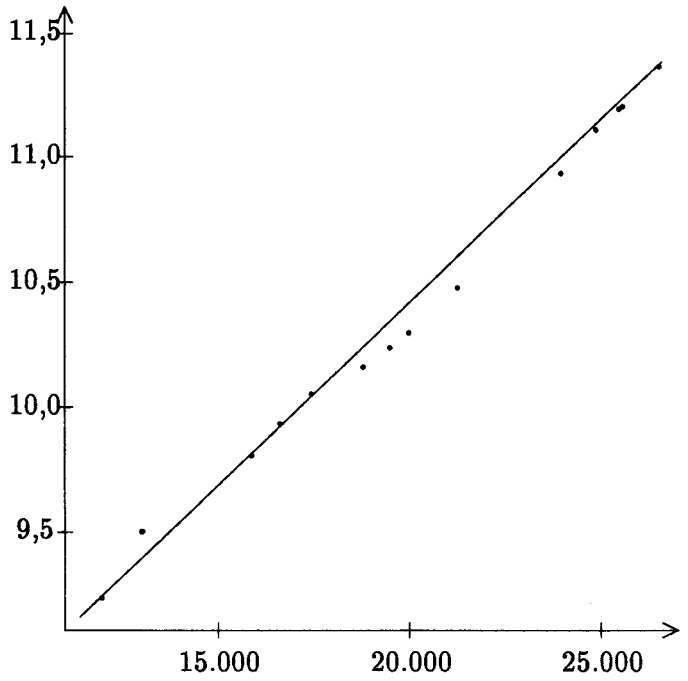


Figur 3.14: lin318, α=0.975.

Svar 1 *Teorien og praksis passer godt sammen, når det drejer sig om beskrivelsen af forløbet af en kørsel.*

Spørgsmål 2 *Kan man direkte sammenligne antallet af iterationer i kørsler med forskellige parametre, i stedet for at sammenligne tidsforbruget? (Er en iteration altid lige lang i snit?)*

Forventet svar: det er ikke for groft at sammenligne forskellige kørsler ved at sammenligne antallet af iterationer, og antage at en iteration i snit altid er lige lang.

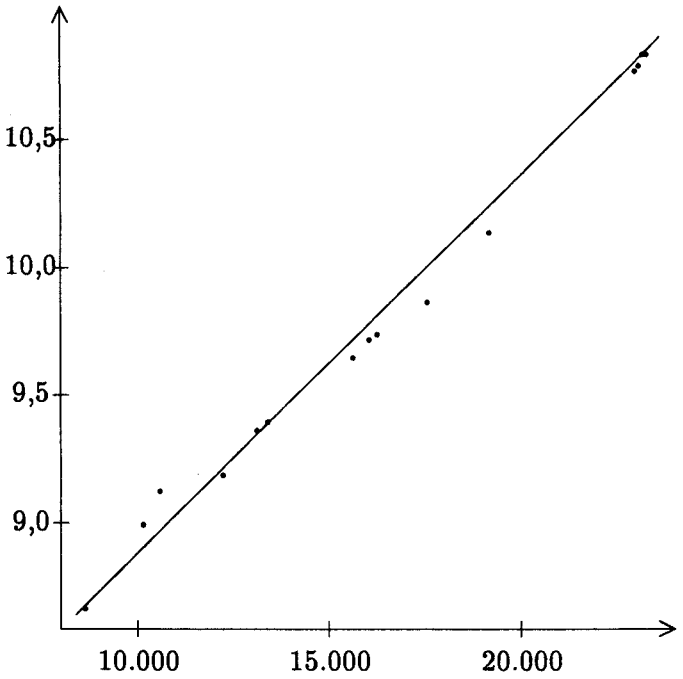


Figur 3.15: Forhold mellem tid og brugte transformationer.

På figur 3.15 ses et bud på hvorfor det ikke er for groft at sammenligne iterationer fremfor tider, nemlig den gode sammenhæng der er mellem antal brugte transformationer i snit pr. iteration (en af de faktorer der påvirker køretiden) og køretid pr. iteration i snit.

Dette forsøg holder styr på dels hvor mange transformationer der accepteres gennem hele forløbet af et forsøg, dels hvor lang tid forsøget tager at køre. På x-aksen afbildes antallet af brugte transformationer i snit pr. iteration (max. 50.085 for lin318 i dette tilfælde), på y-aksen ses gennemsnitstiden pr. iteration. At der bliver en pæn ret linje ud af denne figur skyldes, at man kan skrive en lineær funktion op: $a \cdot \mathit{brugte} + b = \mathit{snittid}$ i sekunder, altså at en iteration tager en vis konstant tid (den tid det tager at foreslå en transformation og afgøre om den skal udføres) og at den derudover tager tid afhængigt af antallet af accepterede transformationer. Det ser ud til, at en accepteret transformation tager konstant tid.

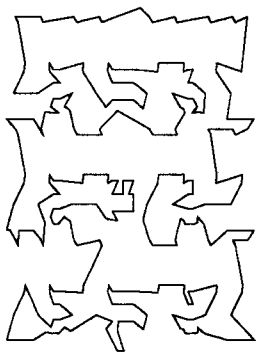
Grafen er lin318, og der udføres 3 forsøg ved hver af α 0,6, 0,8, 0,9, 0,95 og 0,975, ved *niveau* 3 og $\chi_0 = 0,95$. På den næste figur er χ_0 kun 0,9, men ellers er omstændighederne de samme.



Figur 3.16: Forhold mellem tid og brugte transformationer.

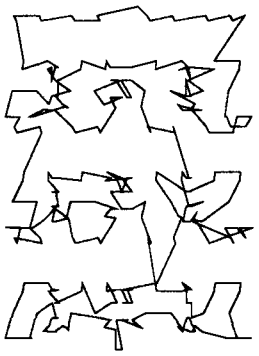
Her ses en tabel over sammenhængen mellem antallet af iterationer, køretid i sekunder og resultat. Dette er for forsøget ved $\chi_0 = 0,9$.

α	1. forsøg	2. forsøg	3. forsøg
0,6	(23,210,44.665)	(28,243,45.401)	(24,216,44.216)
0,8	(47,432,44.369)	(43,404,43.781)	(44,412,44.633)
0,9	(76,740,43.304)	(77,748,43.912)	(79,762,43.608)
0,95	(145,1430,43.374)	(133,1348,44.037)	(110,1190,53.802)
0,975	(224,2414,51.168)	(225,2420,50.292)	(222,2402,51.902)



Figur 3.17: 1.tur ved $\alpha = 0,9$.

På figur 3.17 og figur 3.18 ses to eksempler på ture fra en kørsel af simuleret udglødning. Den første af disse er en god tur, af længde 43.304, kun 3 % for lang. Her har kørslen fået lov til at køre så færdig, som den nu kunne, og så er systemet frosset. I den anden figur kunne der stadig foretages flere forbedringer, men da *niveau* har stået for lavt for denne α er systemet blevet stoppet, i den tro at det nu var frosset, og en masse gode 2-opt er ikke blevet opdaget og udført.



Figur 3.18: 3.tur ved $\alpha = 0{,}95$.

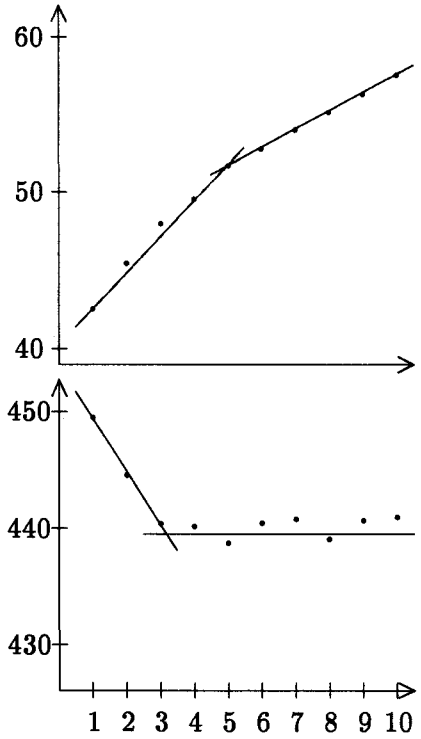
Af begge forsøg fremgår, at det tager længere tid, når der skal bruges flere iterationer, og at det desuden afhænger lidt af antallet af brugte transformationer i den enkelte transformation. De i forsøgene fundne tider viser dog, at det tager ca. 0,14 msek at udføre en transformation, mens det tager 0,15 msek at tage stilling til den. Dvs. det relativt er meget konstant hvor lang tid en iteration tager, mens det giver stor forskel i den forbrugte tid at tage en ekstra iteration.

Umiddelbart kunne jeg næsten ikke forstå, at man også kan antage, at en transformation tager konstant tid. Der udføres jo transformationer af meget forskellig længde, og til slut mange korte. Men jeg antager at dette kan forklares ved, at der netop er så stor overvægt af de korte transformationer, at en transformation i snit ikke tager ret lang tid, og så de få første lange forsvinder lidt i sammenligning med. Ved en lang transformation mener jeg en, der kræver at mange byer skal byttes rundt, og jeg var urolig for, at min "dovne" implementation, hvor jeg altid bytter rundt på byer inde i listen, selv om dette måske kræver flere byt, skulle trække ned på køretiden. Men da der tilsyneladende er mange korte 2-opt, og dette som regel medfører at implementationen udfører byttene effektivt, så gør det ikke noget.

Svar 2 *Det holder ikke helt at en iteration altid tager lige lang tid, men det holder godt, at jo flere iterationer der er, jo længere er den enkelte iteration, og jo længere også den samlede køretid.*

Spørgsmål 3 *Hvad er effekten af at øge* niveau*? Er der en grænse for hvor god* niveau *kan blive?*

Forventet svar: jo højere *niveau*, jo bedre løsning, og jo længere køretid.

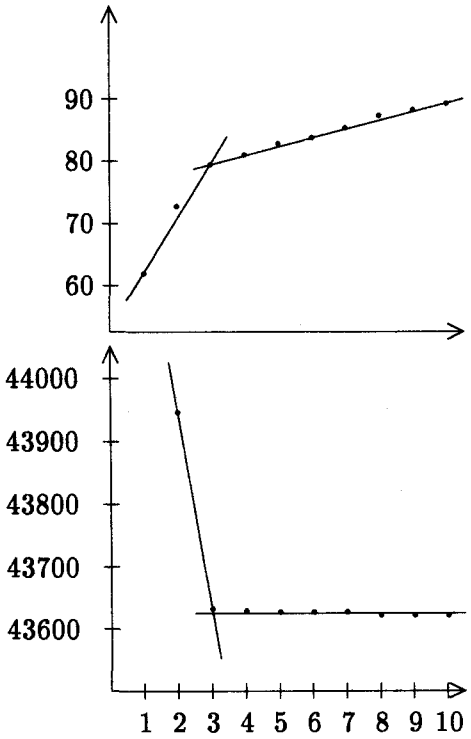


Figur 3.19: Forhold mellem *niveau* og kvalitet/køretid.

I figur 3.19 illustreres hvordan løsningens kvalitet og køretiden ændres når *niveau* forøges. Der udføres 50 forsøg ved hver af 10 forskellige *niveau*, og snittet af resultaterne aftegnes på kurven. Grafen er eil51, ved $\alpha = 0{,}9$ og $\chi_0 = 0{,}9$ (resulterende i $T_0 = 128$).

Køretiden stiger, men for *niveau* 6 eller højere stiger køretiden langsommere, næsten kun med den ene iteration, som algoritmen aligevel skal køre længere for at få bekræftet at *niveau* en højere også er opfyldt. I dette tilfælde vil en svigten af et givet *niveau* slå hårdt, fordi der så skal køres *niveau* iterationer ekstra, og dette sker trods alt også. Med hensyn til køretid vil et *niveau* på 5 altså være passende. Mht. kvalitet er *niveau* 3 godt nok, fordi der ikke sker en forandring i kvalitet højere oppe.

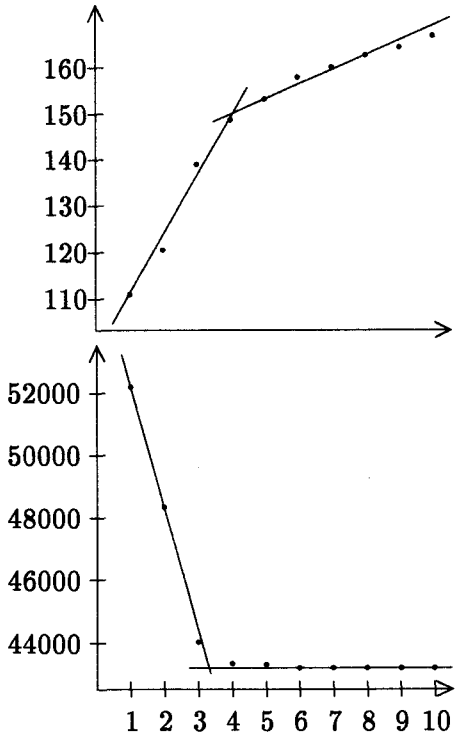
Figur 3.20 er lavet lidt anderledes. I stedet for først at køre 20 forsøg på *niveau* 1, så 20 på *niveau* 2 osv., så blev der egentlig kun kørt 20 forsøg på *niveau* 10, men undervejs blev det registreret hvad resultatet af et *niveau* 1 forsøg ville have været (fordi et *niveau* 10 resultat har passeret sådan et undervejs). Derfor bliver resultaterne under alle omstændigheder bedre ved højere *niveau*. Testen blev kørt på lin318. $\alpha = 0{,}9$ og $\chi_0 = 0{,}9$ (resulterende i $T_0 = 16.384$). Punktet ud for resultatet ved *niveau* 1 er helt forsvundet, det ligger på et snit på 47914, og for at kunne se resten af resultaterne bedre, blev dette punkt smidt væk. Der ses igen et skarpt knæk ved *niveau* 3, denne gang på begge kurver.



Figur 3.20: Forhold mellem *niveau* og kvalitet/køretid.

Da jeg begyndte at lave forsøg på lin318 hvor jeg brugte *niveau* 3, gik noget galt når jeg brugte for høj α, på den måde at de opnåede løsninger ikke blev bedre, men dårligere. Ved yderligere eksperimenteren fik jeg mistanke til, at værdien af *niveau* kunne have betydning for dette, og jeg lavede derfor et nyt forsøg ved højere α, nemlig 0,95. Resultatet ses i figur 3.21, og uden yderligere forklaring af de samme fænomener ses her et højere påkrævet *niveau*. Både kvalitet og køretid flader først ud ved *niveau* 4, og *niveau* 5 bør nok bruges for en sikkerheds skyld, fordi mange af kørslerne viste sig stadig at være ustabile (bedste løsning ikke fundet endnu) på dette *niveau*, mens ændringer efter *niveau* 5 kun var meget små forbedringer.

Nogle steder har jeg set anvendt helt andre stopkriterier, og jeg kunne godt have tænkt mig et stop-kriterie, der specielt tager hensyn til, at ved højere α bør *niveau* nok også være højere, fordi det så tager flere iterationer at nå helt ned til den frosne tilstand.

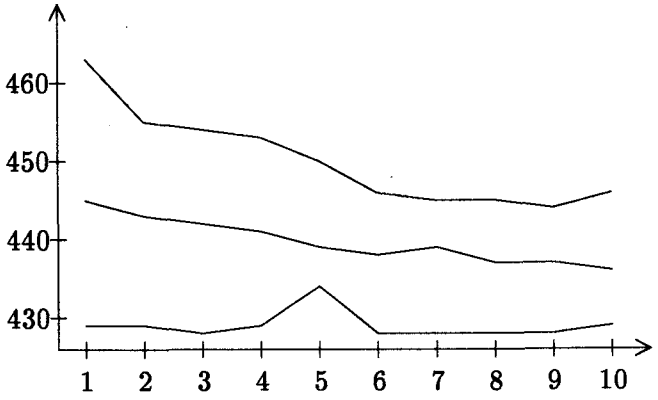


Figur 3.21: Forhold mellem *niveau* og kvalitet/køretid.

Svar 3 *Løsningerne bliver bedre, og køretiden længere. Der findes en grænseværdi for* niveau, *men den er afhængig af de andre parametre.*

Spørgsmål 4 *Hvor god er itereret simuleret udglødning?*

Forventet svar: løsningernes kvalitet bliver bedre, køretiden stiger forudsigeligt.



Figur 3.22: eil51, itereret simuleret udglødning.

Ved itereret simuleret udglødning forstås, at når algoritmen selv stopper, genstarter man den forfra, og dette gentages et fastsat antal gange, inden man checker hvad det bedst opnåede resultat er. Senere skal dette sammenlignes med en "normal" simuleret udglødning, for at se

hvilken form der udnytter tiden bedst, dvs. om man skal bruge en given tid på at køre algoritmen flere gange, eller fx. på at nedkøle langsommere.

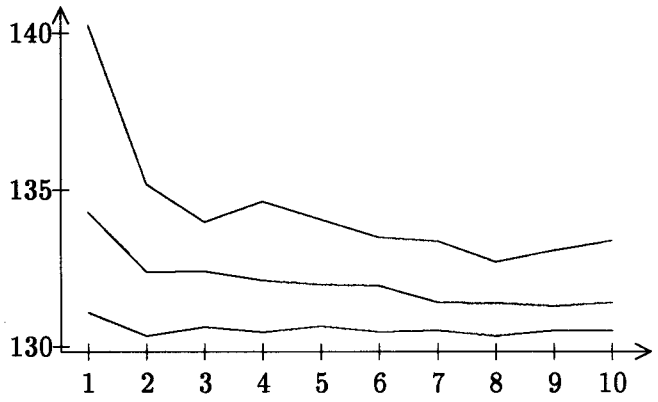
Her præsenteres et typisk resultat, hvor antallet af "superiterationer" er et tal mellem 1 og 10, og jeg kører hver superitereret algoritme 30 gange, for at finde et bedste og et værste resultat, og et snit. Grafen er eil51, $\chi_0 = 0,95$ og $\alpha = 0,95$.

På figur 3.22 er de tre kurver nedefra altså for bedste, snit og værste. Det ses at de 2 øverste kurver falder, mens den nederste kurve viser, at selv de bedste forsøg altså ikke nødvendigvis finder en rigtig god løsning konsekvent. Ved andre forsøg sås et endnu mere markant fald i den øverste kurve, som tegn på at denne form for forbedring af algoritmen fjerner de værste løsninger, simpelthen fordi chancen for at finde dårlige løsninger 2 gange i træk allerede er meget lille.

Den bedste løsning i dette forsøg havde længden 428, 0.5 % værre end den optimale.

På figur 3.23 ses et andet forsøg, hvor grafen denne gang er en af mine egne, indeholdende 50 punkter. Stort set de samme ting iagttages.

Umiddelbart er der ikke nogen grænse for, hvornår det ikke kan betale sig at tage en iteration mere med.



Figur 3.23: Egen graf, itereret simuleret udglødning.

Svar 4 *Itereret simuleret udglødning fjerner de rigtig dårlige løsninger, og forbedrer generelt løsningernes kvalitet.*

Spørgsmål 5 *Hvad er det bedst at bruge tiden til, flere kørsler af algoritmen, eller en enkelt under skærpede parametre?*

Forventet svar: Påstanden er (ifølge *Dodd 90*) at man lige så godt kan køre lange som korte kørsler hvis der bare bruges den samme tid i alt. Dette er et godt argument for at parallellisere simuleret udglødning ved at sprede korte kørsler ud på mange computere til parallel afvikling.

I dette tilfælde præsenteres resultaterne bedst i en tabel. Forsøget er gennemført ved at køre simuleret udglødning på samme graf (lin318). $\chi_0 = 0,9$. Der foretages kørsler ved forskellige α, og tilnærmelsesvis medfører en dobbelt så god α (afstanden til 1 den halve) også den dobbelte køretid. På denne måde kan fx. 2 kørsler ved $\alpha = 0,9$ hvor bedste resultat vælges sammenlignes med en enkelt kørsel ved $\alpha = 0,95$. Aktuelt foretages 48 kørsler ved 0,6, 24 ved 0,8, 12 ved 0,9, 6 ved 0,95 og 3 ved 0,975. En tredjedel af resultaterne ses i den næste tabel.

$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 0,9$	$\alpha = 0,95$	$\alpha = 0,975$
44665	44222	43655	43167	43210
45423				
44880	45465			
45370				
44778	44580	44219		
46287				
44599	43819			
44695				
45201	43359	43473	43279	
44961				
44915	44782			
45167				
46372	44657	43593		
46086				
45955	44258			
45284				

Systemet i tabellen er, at der i er lige så mange resultater i en søjle, som der kan opnås på en given tid. Således kan der på samme køretid opnås 16 resultater ved $\alpha = 0,6$, men kun 8 ved $\alpha = 0,8$, og 1 ved $\alpha = 0,975$. Når resultaterne skal sammenlignes, findes det bedste resultat i hver søjle over en given højde. Den viste del af tabellen kan bruges til at se, at man med den givne køretid kunne finde en tur af længde 44.599 ved $\alpha = 0,6$, en af længde 43.359 ved $\alpha = 0,8$ osv. Resultaterne opsummeres her nedenfor.

α	1 enhed	2 enheder	4 enheder	8 enheder	16 enheder	48 enheder
0,6	45.280	44.973	44.806	44.561	44.396	44.273
0,8		44.287	43.903	43.756	43.542	43.359
0,9			43.578	43.378	43.312	43.164
0,95				43.198	43.087	43.001
0,975					43.021	42.726

Resultaterne ud for fx. 8 enheder viser hvad man får ud af at optælle bedste resultat for hver 8 tidsenheder brugt (en enkelt enhed svarer til tiden for en enkelt kørsel ved $\alpha = 0,6$) for hver af de 6 blokke, der kommer ud af de i alt 48 enheder brugt. Dette giver ikke mening for de kørsler, der enkeltvis tager mere end 8 enheder. De 6 fremkomne resultater tages der så snit af.

Som sagt er påstanden, at man lige så godt kan køre lange som korte kørsler hvis der bare bruges den samme tid i alt. Men jeg synes bestemt ikke dette holder. Ovenikøbet har de svage kørsler (dem ved lav α) fået ekstra tid, dels fordi det ikke helt holdt at en kørsel ved dobbelt så god α tog dobbelt så lang tid (den blev faktisk kun knapt dobbelt så lang), dels fordi jeg kørte hele forsøget ved *niveau* 7, og dette er overdrevet, og nærmest spild af tid ved lave α. Men selv da, er det klart bedre at lave en lang kørsel end at lave flere små. Resultatet ved at anvende 48 enheder ved $\alpha = 0,6$ kan konkurrere med det opnået ved at anvende 2 enheder ved $\alpha = 0,8$. Det allerbedste resultat findes ved $\alpha = 0,975$, og forskellene i kvalitet råder i det hele taget til hellere at bruge en enkelt kørsel ved bedste mulige α, og så bare bruge dette resultat.

En tidsenhed ligger mellem 3,5 og 4,5 minut.

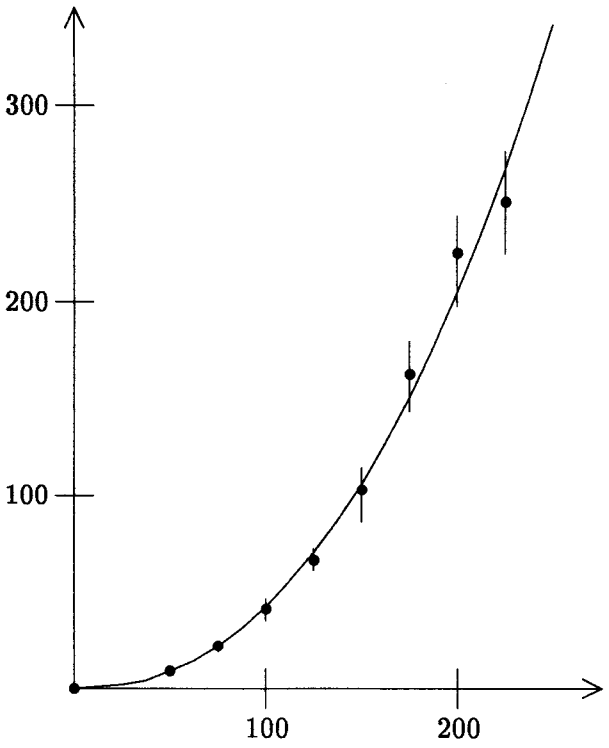
Svar 5 *Hellere høj α end mange iterationer.*

3.5 Eftervisning af resultater.

For det aktuelle nedkølingsskema, og generelt for simuleret udglødning, ønsker jeg nu at vise forskellige resultater.

Spørgsmål 6 *Er køretiden polynomiel i antallet af byer?*

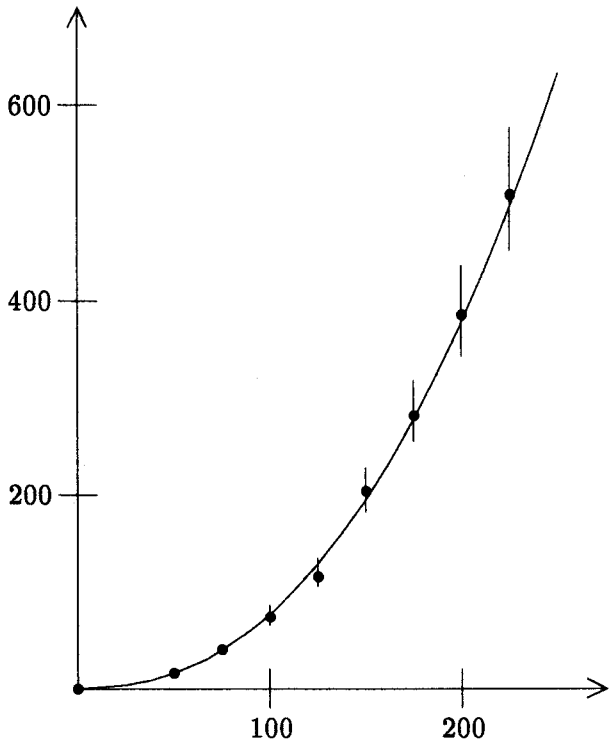
Forventet svar: ja, fx. noget med $n^4 logn$, som i *Aart 89*.



Figur 3.24: Sammenhæng mellem størrelse af graf og køretid.

Data på figur 3.24 og figur 3.25 er fremkommet ved at lade α være hhv. 0,9 og 0,95. $\chi_0 =$ 0,9 i begge forsøg, og *niveau* = 3, hvilket muligvis er for lavt for de største grafer. Dette har jeg ikke checket, jeg var kun interesseret i køretiderne.

De 2 figurer er desuden fremkommet ved at fremstille 8 grafer med mellem 50 og 225 punkter, og med spring i størrelse på 25. Disse fremkom som før ved at finde tilfældige punkter med heltallige koordinater i et kvadrat af stigende størrelse for stigende antal punkter. På hver graf blev der udført 30 forsøg, og køretiderne blev noteret. På figuren ses et punkt ud for antal punkter i graf og gennemsnitskøretid, og desuden en streg for at angive, hvor meget køretiden varierede, dvs. der er en streg mellem længste og korteste køretid. Herefter prøvede jeg at få punkterne til at passe med et polynomium, og dette lykkedes godt begge gange. Polynomiet i figur 3.24 er af grad 2,284, mens det på figur 3.25 har grad 2,304. Og korrelationsfaktoren var i begge tilfælde meget tæt på 1, som tegn på at hypotesen om en polynomiel sammenhæng er god.



Figur 3.25: Sammenhæng mellem størrelse af graf og køretid.

Svar 6 *Der er en god polynomiel sammenhæng mellem størrelse af graf og køretid. Ca.* $O(n^{2,3})$.

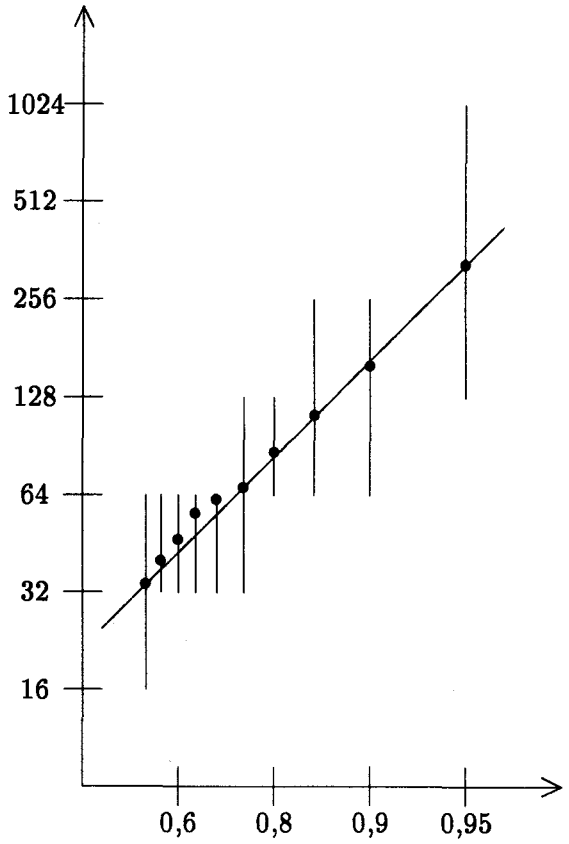
Spørgsmål 7 *Hvad er effekten af at øge* χ_0*?*

Forventet svar: T_0 stiger med χ_0.

På figur 3.26 ses en graf over sammenhængen mellem χ_0 og T_0, disse resultater er opnået ved at holde χ_0 fast og så undersøge hvilke T_0 der kommer ud af det. I dette tilfælde undersøges ved hjælp af eil51. Der foretages 200 forsøg ved hver χ_0.

Stregerne angiver, at der kan komme flere forskellige T_0 ud af det samme χ_0, og stregen angiver da yderværdierne. Prikken angiver gennemsnitsværdien. Bemærk at hvis det er angivet at enten 32 eller 64 er blevet valgt, så har det ikke været muligt at få et T_0 mellem disse to værdier, fordi hvis 32 forkastes fordobles, og dvs. 64 derefter overvejes.

Data forekom mig at passe bedst i et dobbeltlogaritmisk koordinatsystem, så man kan se den pæne linje der kommer frem. Og ud af dette kan konkluderes: jo højere χ_0, jo højere T_0. Og der er en pæn sammenhæng.



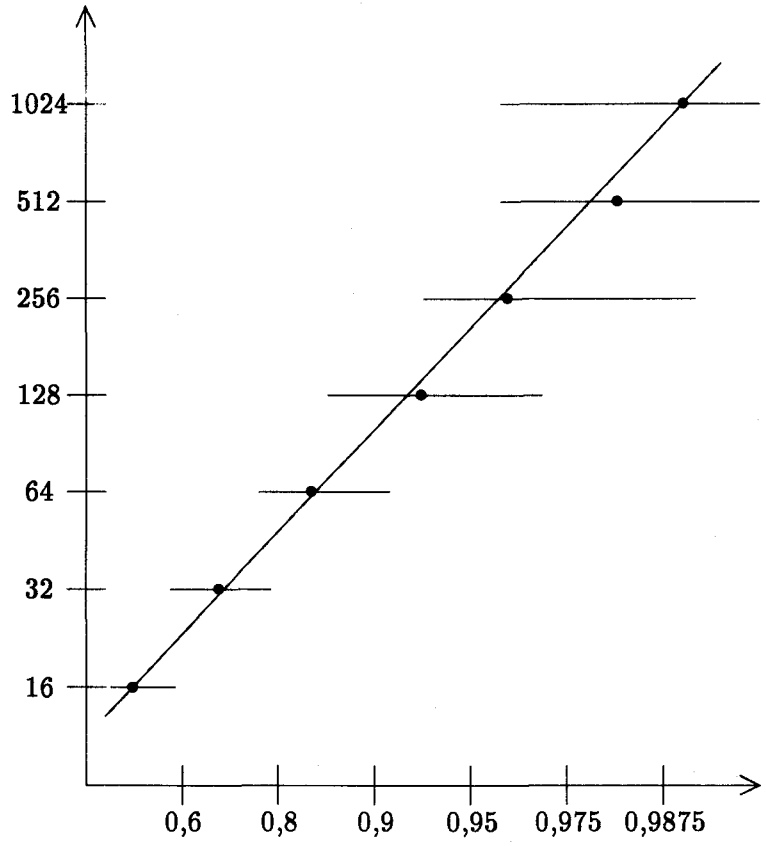
Figur 3.26: Sammenhæng mellem χ_0 og T_0.

På figur 3.27 ses en anden måde at undersøge den samme sammenhæng på. Med den samme graf (eil51) holdes T_0 nu fast, og det undersøges hvilke χ_0 der så kan opfyldes, dvs. hvor mange procent af de foreslåede nye ture ved denne temperatur, der bliver accepteret. Jeg starter med $T_0 = 1$ og kører 20 forsøg med hver 100 forslag. Derefter noterer jeg mig mindste og største mulige χ_0 og gennemsnittet, der svarer til det χ_0 jeg kunne acceptere ved et enkelt forsøg med 2000 forslag. Herefter fordobler jeg T_0 og gentager. Jeg bliver ved op til $T_0 = 1024$.

Det er ikke hele det nævnte forsøg jeg har med på grafen, men kun den del der direkte kan sammenlignes med figur 3.26. For de lave temperaturer sås noget tilsvarende. Jeg bemærkede nogle sideeffekter af at lave forsøget på den nævnte måde. Helt fra start ved $T_0 = 1$ blev en χ_0 på 0,36 mulig, men i de efterfølgende forsøg faldt dette, helt ned til 0. Ved at undersøge længden af de ture, der lå bagved, dvs. de ture, der blev ændret på og foreslået, fandt jeg ud af, at systemet hurtigt blev koldt, dvs. længden af turene blev generelt kort, som ved slutningen af en kørsel af simuleret udglødning, men fra start havde jeg jo en tilfældig tur, der var lang, således at mange af de mulige ændringer ville medføre kortere ture (mange i forhold til hvis det havde været en kort tur) og dermed blev mange ændringer accepteret, og en høj χ_0 var mulig. Men jo længere tid jeg blev ved $T_0 = 1$, jo kortere blev turene, og jo lavere blev χ_0.

I løbet af forsøget blev turene længere, fordi jo højere temperaturen er, jo større er sandsynligheden for en lang tur, så ved hver temperatur fandt jeg egentlig en kombination af en tur og en χ_0, der passede sammen. I hvor høj grad dette sker når jeg første (og eneste) gang bestemmer en T_0 i starten af en kørsel af simuleret udglødning ved jeg ikke, men da jeg i det tilfælde ikke bliver nær så længe på den samme temperatur, og selv da, hvis der sker det samme, ender med

en tilpas høj temperatur og en tilpas blandet/lang tur, så gør det jo ikke så meget. Og selv hvis turen efter at have fundet T_0 ikke er specielt lang, så bliver den det efter den første iteration.



Figur 3.27: Sammenhæng mellem χ_0 og T_0.

Bemærk i øvrigt, at de to linjer i de to figurer ikke viser helt samme værdier, således at man kan gå ind i de to figurer og forudsige samme T_0 ved at kende χ_0. Dette må skyldes de forskellige omstændigheder, altså at en tur, fra et netop overstået forsøg ikke er en tilfældig tur at køre forsøget med igen, og at dette har forskellige konsekvenser i de to tilfælde.

Svar 7 *Jo højere χ_0, jo højere T_0, en forbindelse som $T_0 = k \cdot (\chi_0)^c$. Men k og c variabel.*

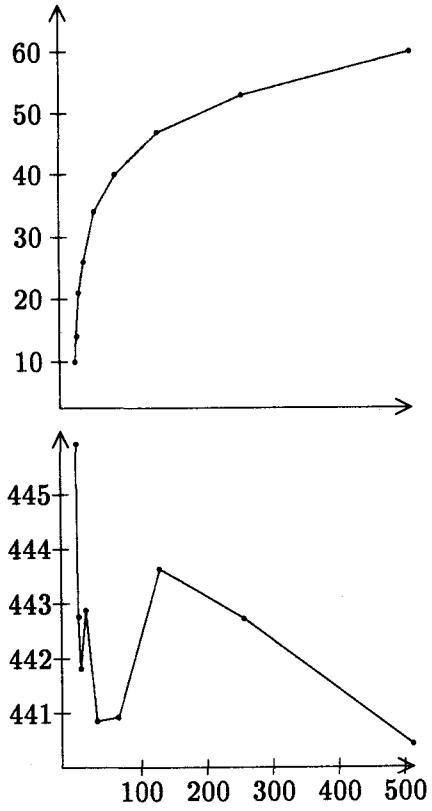
Spørgsmål 8 *Hvad er effekten af at øge T_0?*

Teoretisk burde det give bedre løsninger at øge T_0, selvom man selvfølgelig samtidig bruger mere tid i algoritmen.

Egentlig er det χ_0 der er en parameter for algoritmen, men da man godt kan få forskellige T_0 for samme χ_0, vælger jeg at undersøge konsekvensen af at ændre T_0, og har seperat fundet T_0 som funktion af χ_0 tidligere.

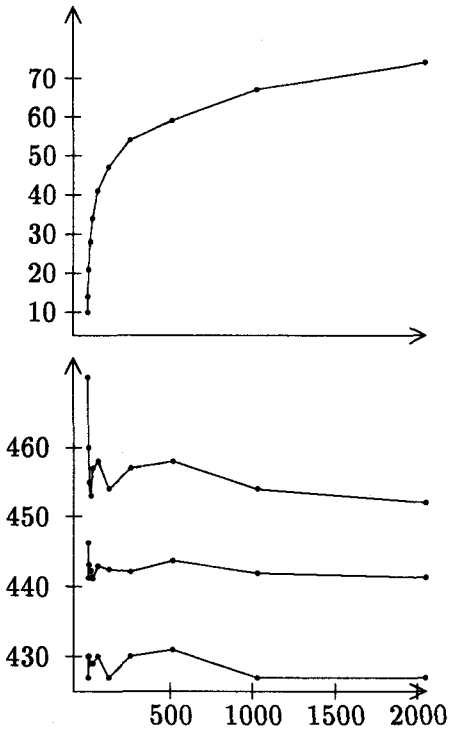
Jeg lavede et forsøg hvor jeg ved forskellige T_0 foretog 50 forsøg, for derefter at bestemme snittet af resultaterne. Dette ses på figur 3.28 og figur 3.29, hvor andet forsøg er en gentagelse af første, ved nogle højere T_0. Ikke alene viser ingen af kurverne et fald der er rigtig til at stole på, de viser heller ikke samme værdier ved samme T_0.

Dette er forsøg på eil51. $\alpha = 0,9$, *niveau* $= 3$.



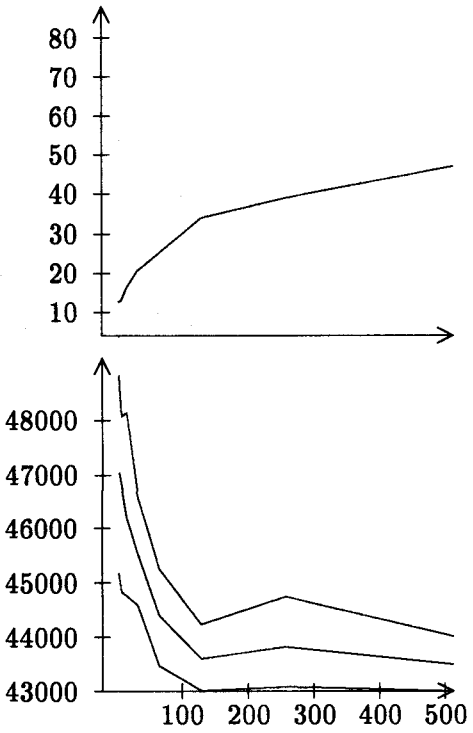
Figur 3.28: Sammenhæng mellem T_0 og resultat / antal iterationer.

Igen er de 3 kurver i figur 3.29 tegn på bedste, snit og værste. Desværre kan jeg ikke rigtig se at nogle af kurverne falder, dvs. det er nærmest ligegyldigt hvilken temperatur man starter ved! Dette stemmer ikke overens med mine forventninger, eller hvad mine kilder havde stillet mig i udsigt, men jeg har ikke kunnet finde fejl i mine programmer eller lignende til at forklare fænomenet. Eneste mulige konklusion er, at $T_0 = 2$ er dårligere end en højere T_0.



Figur 3.29: Sammenhæng mellem T_0 og resultat / antal iterationer.

I figur 3.30 ses samme forsøg på lin318. Her ses måske en lidt klarere konklusion. Tilsyneladende bliver der et bedre resultat ved at bruge lidt højere T_0, startende ved 128, men derfra er det igen lidt tilfældigt hvad der sker. $\alpha = 0,9$. 21 forsøg. Både bedste, værste og snit angivet.

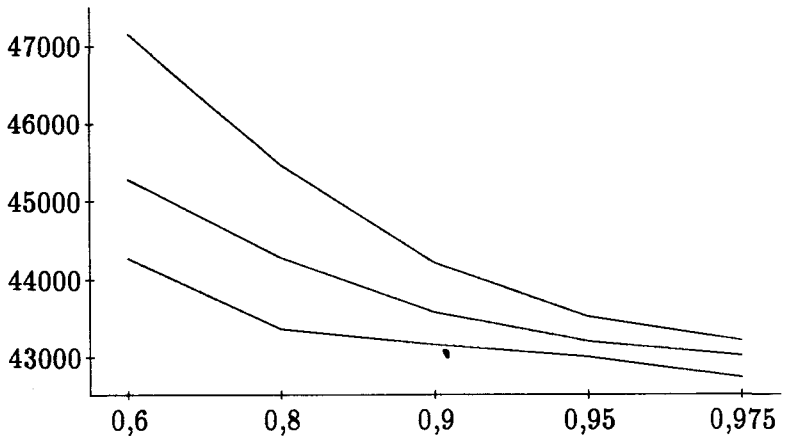


Figur 3.30: Sammenhæng mellem T_0 og resultat / antal iterationer.

Svar 8 *Tilsyneladende hjælper det ikke!*

Spørgsmål 9 *Hvad er effekten af at øge α?*

Forventet svar: løsningerne bliver bedre, køretiden bliver længere.



Figur 3.31: Sammenhæng mellem kvalitet og α.

Her genbruges resultaterne fra spørgsmål 5, hvor der blev kørt forskelligt antal forsøg ved forskellige α. På figur 3.31 bruger jeg dog bare snittet af alle resultaterne, uanset hvor mange der var.

Det ses tydeligt, hvordan løsningerne bliver bedre, jo højere α der bruges. Jeg stopper ved 0,975, men jeg har læst om forsøg hvor 0,99 blev brugt. Jeg starter ved 0,6, hvor andre først er startet ved 0,8 eller endnu højere. Ofte lægger man sig fast på en α fra start, for så at eksperimentere med de øvrige parametre.

Dette er altså en kørsel på lin318, hvor bedste resultat var på 42.726, og dårligste på 47.158.

Svar 9 *Kvaliteten af løsningerne forbedres, og som tidligere rapporteret øges køretiden.*

Spørgsmål 10 *Hvordan er køretiderne og kvaliteten sammenlignet med 2-opt og 3-opt?*

For eil51 er køretiden for både 2-opt og simuleret udglødning meget kort, nede omkring et sekund, så det er svært at foretage sammenligninger.

For lin318 er det kun køretiden for 2-opt der overhovedet kan sammenlignes med køretider for simuleret udglødning, da 3-opt bruger omkring 18 timer for at finde et enkelt resultat. Her bruger 2-opt i snit 367 sekunder på at finde ture af gennemsnitslængde 45.556. Tidligere sås køretid i snit på 223 sekunder og tur i snit på 44.761, og senere køretid i snit på 416 sekunder og tur i snit på 44.261, bedste tur af længde 43.781. Simuleret udglødning kan altså nemt udkonkurrere både 2-opt og 3-opt mht. tid og kvalitet.

Svar 10 *Simuleret udglødning giver bedre resultater på kortere tid, sammenlignet med både 2-opt og 3-opt.*

Kapitel 4

Genetiske algoritme.

> Derpå sagde Gud til [Israel]: "Jeg er Gud den Almægtige! Bliv frugtbar og mangfoldig! Et folk, ja folk i hobetal skal nedstamme fra dig, og konger skal udgå af din lænd; det land, jeg gav Abraham og Isak, giver jeg dig, og dit afkom efter dig giver jeg landet!"
> *(1. Mosebog,35,11-12)*

Den genetiske algoritme er en ny måde at finde en optimal kombination på, fordi den hele tiden har mange kombinationer at se på på én gang, ligesom Moder Natur kan finde gode individer ved at holde en stor befolkning.

I dette kapitel beskriver jeg sammenhængen fra evolutionært fænomen til algoritme, til algoritme for TSP. Jeg begrunder hvorfor den genetiske algoritme skulle finde gode løsninger. Endelig udfører jeg en række forsøg, for at finde ud af hvor godt den genetiske algoritme virker under forskellige valg af parametre osv.

4.1 Baggrunden for den genetiske algoritme.

Som navnet antyder, bygger den genetiske algoritme på fænomener, der kendes fra genetik. Nogle steder bruges navnet evolutionær algoritme/programmering dog også (fx. i *Mich 92*), fordi de gode egenskaber ved evolution også gerne skulle optræde.

I genetik studeres resultaterne af at lade to kendte individer få afkom. Afkommets egenskaber kan forudsiges med en vis sandsynlighed (50 % chance for brune øjne), men da der er så mange forskellige egenskaber at holde styr på i praksis (lad os sige F forskelle på forældrene, og L ligheder) bliver der hurtigt for mange forskellige kombinationsmuligheder (groft sagt 2^F) til, at man kan holde styr på dem alle. På den anden side er der fx. $5 \cdot 10^9$ mennesker, og dermed alligevel mange forskellige individer og forældre/afkom at studere.

Evolutionsteorien bygger på "survival of the fittest", dvs. at de gode/levedygtige individer overlever og overleverer deres gode egenskaber til deres afkom, mens de dårlige individer dør hurtigt, uden at få børn. Dette er selvfølgelig et unuanceret forhold til naturen, og specielt mennesker er gode til at sætte sig ud over reglerne, så fx. fysisk handicappede, der overladt til sig selv hurtigt ville dø (fx. af sult) nu kan overleve og få børn, og måske få lejlighed til at lade "gener for intelligens" gå videre. Men selv i den vilde natur kan fugle, der ikke kan flyve, være heldige ikke at blive fanget af katten, mens de sunde, unge fugle havner i en jægers net. Der er altså højst en vis sandsynlighed for, at et givet individ lever og avler eller dør i forhold til sin levedygtighed. Og ud af dette kommer en sandsynlighed for, at befolkningen bliver bedre og bedre, generation for generation, og at der specielt er gode individer i mellem også.

Den nævnte videregivning af egenskaber sker ved krydsning af to enkeltindivider, og indenfor genetik kendes begrebet overkrydsning, der har som formål at sikre, at der ud af to kromosomer kommer to nye, hver med gener fra begge de to oprindelige. I den genetiske algoritme genbruges overkrydsning.

En anden måde at få nye individer på er ved mutation, fx. opstået ved bestråling af et ubefrugtet æg, så det senere afkom får et originalt gen, ingen af forældrene havde. Også mutation genbruges.

Hermed er begreberne til den genetiske algoritme introduceret, og de er altså: befolkning, levedygtighed, overlevelse/død, overkrydsning og mutation, hvor de to sidste begge sikrer afkom. Og den genetiske algoritme kan dermed også skrives op, så man kan se ligheden med biologisk evolution.

Algoritme 12 *Genetisk algoritme 1*

```
skab en start-befolkning
(* alle individer er udstyret med et tal, der angiver levedygtighed *)
gentag
    beregn relativ levedygtighed
    vælg hvilke individer, der skal leve, avle og dø, udfra levedygtigheden
    lad de udvalgte avle, ved overkrydsning og mutation
    lav en ny befolkning af de overlevende og det nye afkom
indtil slut-betingelse opfyldt
```

Algoritme slut

Et individs levedygtighed er relativ, således at tallet for levedygtigheden er konstant, men skal sammenlignes med den øvrige befolknings. Måske er der et individ, der fødes som superindivid, men dør som en svækling. Derfor udregnes den relative levedygtighed for hver ny

generation, typisk som forholdet mellem den enkeltes levedygtighed og summen af alle individers levedygtighed. Dette forhold kan fortolkes som sandsynligheden for, at det enkelte individ overlever (måske ovenikøbet bliver "klonet", så der bliver flere ens kopier) og avler (måske i flere forskellige "parforhold").

Den ovenstående algoritme er meget generel, mens den der her følger, er mere speciel og har udelukket nogle muligheder.

Algoritme 13 *Genetisk algoritme 2*

skab en start-befolkning på N individer
gentag
    beregn relativ levedygtighed
    lav $N - M$ kopier af individer fra befolkningen over i den nye befolkning
    lav M andre kopier, der får afkom
    lad rM af de sidste kopier blive krydset sammen to og to, til lige så mange afkom
    (* rM rundes ned til lige tal *)
    lad de øvrige $(1 - r)M$ kopier blive muteret
    lad den nye befolkning overtage dens gamle plads
indtil slut-betingelse opfyldt

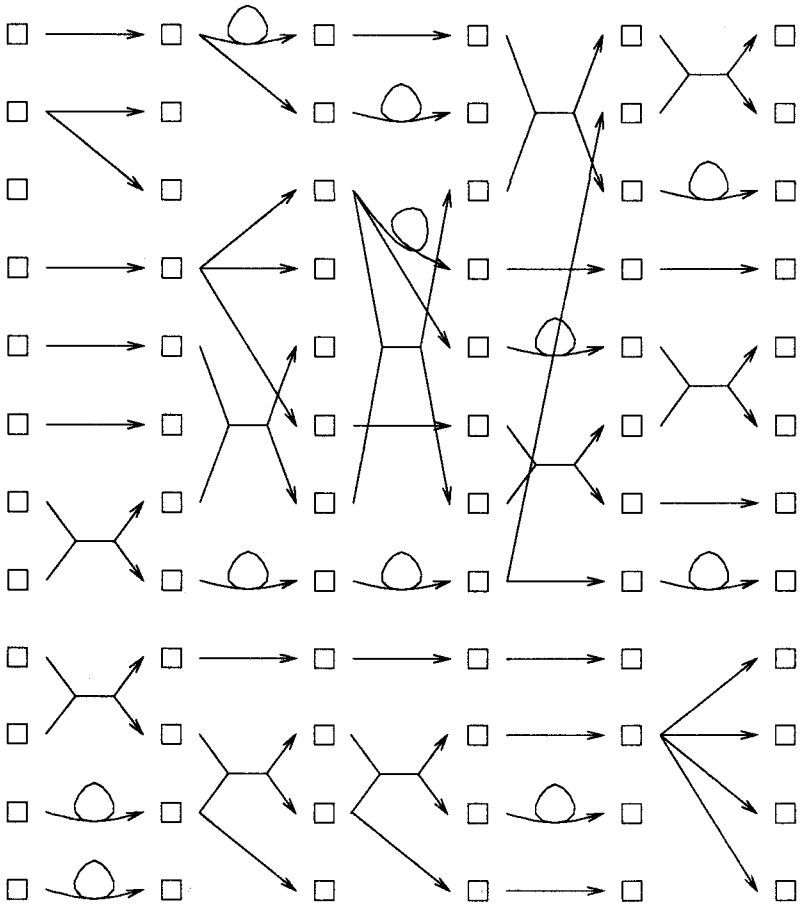
Algoritme slut

I denne proces indgår et "roulettehjul", der er nyt for hver generation. Hvert felt på hjulet svarer til et individ, og bredden svarer til hjulets omkreds med samme forhold som den relative levedygtighed. For hver ny kopiering snurres roulettehjulet for at finde et nyt individ. (I praksis udregnes de enkeltes relative levedygtighed, hvorefter den kumulerede levedygtighed udregnes, givet en rækkefølge af individerne, og derefter findes et tilfældigt tal mellem 0 og 1, 1 inklusive. Dette udpeger et individ, der har kumuleret levedygtighed lig med eller større end det tilfældige tal, og hvor det foregående individ har levedygtighed mindre end det tilfældige tal.) De individer, der dør, udvælges passivt, på den måde at de altså ikke udvælges til kopiering. Et individ kan ikke både blive muteret og krydset med et andet, med mindre det tilfældigvis blev kopieret til begge formål, og da er det altså to forskellige kopier, der undergår de to forskellige forandringer.

Af parametre for denne algoritme er der N (den konstante størrelse af befolkningen), M (antallet af individer udvalgt til at blive erstattet af afkom, $M < N$), heraf følger sandsynligheden for at et individ er nyt afkom M/N, r (hvilken brøkdel af de avlende, der bliver overkrydset, men ikke muteret) og slutbetingelsen, der typisk er et mål for, hvor mange generationer forsøget fortsættes, men også kan afhænge af fx. den aktuelle forbedring af levedygtighed i sidste iteration.

4.1.1 Befolkningernes gang.

I figur 4.1 ses et eksempel på resultaterne af den genetiske algoritme. Hver lodret række er en generation, og pilene angiver hvordan næste generation opbygges ved kopiering (almindelige pile), mutation (krøllede pile) og overkrydsning (pile med dobbelt-ender). Da hvert individ kan udsættes for alle overførsler til næste generation mere end én gang, er det muligt at der optræder både kopier, muterede kopier og resultater af overkrydsning med individet i næste generation. Faktisk er en af farerne i den genetiske algoritme at et superindivid ender med at blive kopieret så mange gange, at kun mutationer får nye individer frem, så resten af den pågældende generation er kopier af superindividet.

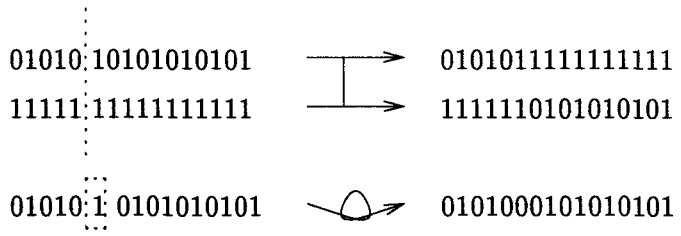


Figur 4.1: Ændring ved genetisk algoritme.

Af typografiske årsager er individerne ikke blevet blandet fra generation til generation, som de faktisk også bliver.

4.2 Genetiske algoritme tilpasses til TSP.

Da den genetiske algoritme blev foreslået, var individerne altid bit-strenge. Hvis man fx. søger at maximere en funktion defineret på heltallene fra 0 til 65.535 kan en 16-bits-streng repræsentere tal (individer) hvor funktionsværdien udregnes. Her defineres også nemt både overkrydsning og mutation, der svarer meget til begreberne for DNA, se figur 4.2.



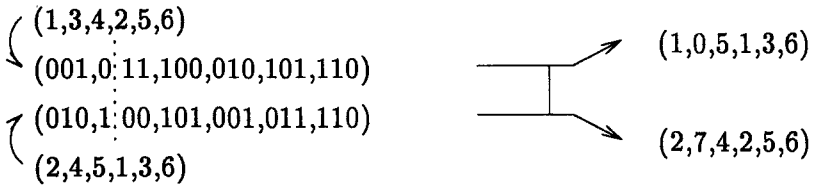
Figur 4.2: Krydsning og mutation af bit-strenge.

Så man kan altså krydse to strenge ved at klippe dem over og "lime" dem sammen omvendt, og man kan mutere ved at vende en enkelt bit. Der er også andre variationer over disse temaer (fx. klippe to strenge i mere end 2 stykker hver, før sammenlimningen). Levedygtigheden er direkte baseret på funktionsværdien (jo højere, jo bedre), og en befolkning er N 16-bits tal.

For en tid var påstanden fremme, at hvis individer ikke kunne repræsenteres ved bit-strenge, der kunne krydses på den viste, ukomplicerede måde, så havde man ikke en rigtig genetisk algoritme (fx. i *Mich 92*, hvor udtrykket evolutionær programmering bruges oftere), men dette er man vist ved at gå bort fra igen, og i hvert fald findes der for både TSP og andre knapt så let repræsenterbare problemer genetiske algoritmer, med lidt mere indviklede former for avl.

For TSP er et individ en tur. Dens levedygtighed afhænger af dens længde, og for at få en høj levedygtighed for en kort tur, anvendes typisk den inverse af længden (tur af længde 2 har levedygtighed 1/2). Turen repræsenteres som en liste: (1,3,4,2,5,6)

Hvis denne krydses sammen med en anden tur på den allerede viste måde, bliver der sandsynligvis problemer, her vises krydsningen hvor listerne er lavet om til bit-strenge, på figur 4.3.



Figur 4.3: Krydsning og mutation af TSP-bit-strenge.

Hvis der krydses på det viste sted, får man i det ene tilfælde en tur, hvor den ikke-eksisterende by nr.0 er med, og hvor by nr.1 er med to gange. Eller rettere sagt, resultatet er slet ikke en tur. Det sidste problem med dubletter af byer, opstår også selvom turene krydses som lister i stedet for bit-strenge, dette svarer til at flytte det viste krydsningssted en tak til venstre, hen over kommaet.

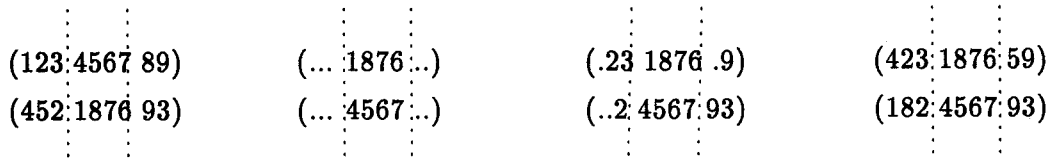
Mutationer vil medføre tilsvarende problemer med dubletter eller ikke-eksisterende byer.

4.2.1 Overkrydsning.

En tur-overkrydsning skal altså være mere avanceret for at opfylde, at afkommet stadig er ture. Her følger 2 (af mange eksisterende) forslag.

PMX.

Den delvis afbildende overkrydsning (partially mapped crossover, PMX, ifølge *Mich 92* fra *Gold 85a*) laver en overkrydsning med to overkrydsningspunkter, bytter rundt på midterstykket i de to ture, og sørger så for, at resultatet stadig er en tur bagefter. Se figur 4.4.

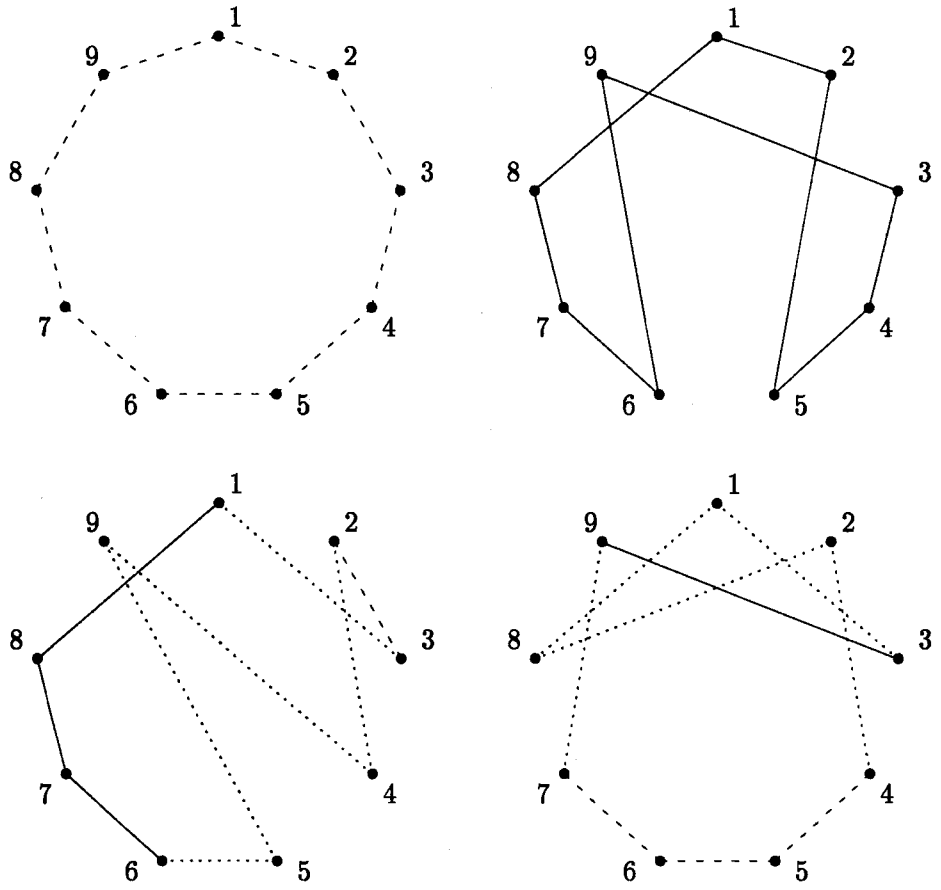


Figur 4.4: Krydsning vha. PMX.

I de 4 situationer ses først de to ture, der skal krydses, og de to overkrydsningspunkter. I næste situation er der byttet rundt på de to midterstykker, denne del af den nye tur kendes hermed, men inden resten af turen laves, skal der forsikres mod gentagelser af byer. Hvis det defineres, at det allerede eksisterende 1-tal i den øverste tur byttes ud med et 4-tal, for at gøre det omvendte af hvad der allerede er sket, burde det gå op. Således definerer de to midterstykker nu en permutation for hver tur. I den øverste tur byttes 1 med 4, 8 med 5, 6 med 7 og 7 med 6 (de to sidste byt falder bort, fordi det der skulle byttes ud allerede er med i midterstykket). Tilsvarende for den nederste tur. Dvs. 1 og 8 ikke bare skal kopieres over i den nye tur, så i den tredje situation ses de to nye ture, hvor resten af byerne er kopieret på plads. I fjerde situation ses endelig de to sidste byer byttet ud som anvist.

Resultatet af PMX er altså to nye ture, og at mindst de to byttede midterstykker svarer til stykker i forældrene. Derudover kan der være stumper rundt omkring, der bliver kopieret direkte over i afkommet, men desuden opstår der mange nye kanter.

På figur 4.5 ses de to i figur 4.4 allerede krydsede ture, og deres to afkom, og her er den ene vist med stiplede kanter, den anden med fuldt optrukne kanter, og de prikkede kanter er nye. Bemærk at antallet af nye kanter afhænger meget af antallet af byer i turen og afstanden mellem krydsningspunkterne. Senere kommer jeg ind på, at det er bedst hvis der ikke opstår alt for mange nye kanter af gangen, og hvis gode brudstykker har en vis chance for at blive hængende sammen. PMX tager mere hensyn til byernes position i listen, end til position i forhold til deres naboer, og dette er ikke helt hensigtmæssigt.

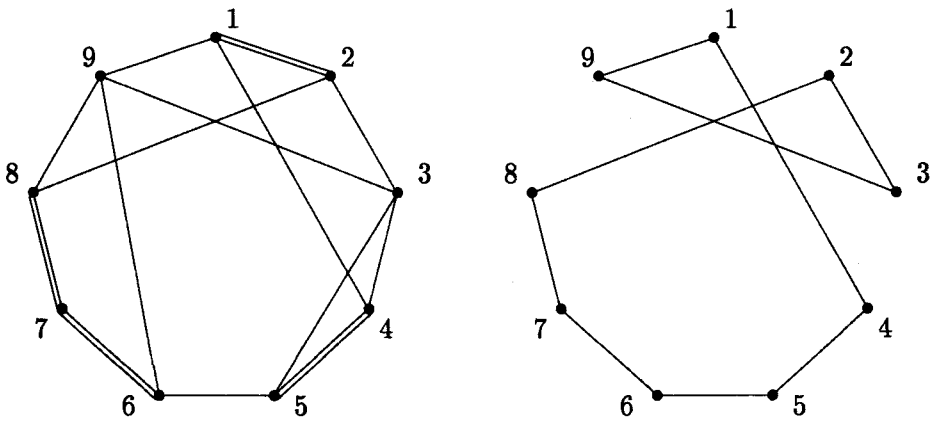


Figur 4.5: Krydsning vha. PMX.

ERC.

Denne kant rekombinerings-overkrydsning (edge recombination crossover, ERC, ifølge *Mich 92* fra *Whit 89*) tager udgangspunkt i at bevare så mange kanter som muligt, ved at vælge kanter der lader flest muligheder stå åbne for kantvalg senere, således at der kun sjældent skal gribes til kanter, ingen af forældrene havde på forhånd.

På figur 4.6 ses turene (1,2,3,4,5,6,7,8,9) og (4,1,2,8,7,6,9,3,5), lagt oveni hinanden, så det kan ses hvor mange forskellige kanter, der er at vælge imellem på forhånd. I denne graf er en Hamilton-tur selvfølgelig mulig (den er jo kombineret af to forskellige) men hvis man først har valgt en kant fra hver tur, er det ikke længere til at sige, om en tur kan konstrueres uden at tage nye kanter.



Figur 4.6: Krydsning vha. ERC.

Til højre på figuren ses en ny tur, der består af kanter fra begge forældrene. Dens opståen forklares bedst i detaljer.

Punkt nr.1 er det første punkt i den første tur, og bliver derfor også det første punkt i det ene afkom.

1's naboer: 2 (3), 4 (3), 9 (4).

I parentes står hvor mange naboer hvert punkt har. For at bevare muligheden for også senere at kunne vælge en kant til et naborigt punkt, vælges nu et nabofattigt punkt, det bliver enten 2 eller 4. Her vælges tilfældigt, i dette tilfælde 4.

4's naboer: 1 (start), 3 (4), 5 (3).

1 er startpunktet, og skal således ikke vælges igen. Af de to andre punkter, har 5 færrest naboer, og vælges derfor.

Herefter synes jeg kilden (*Mich 92*) bliver uklar på punktet om præcis hvor mange naboer punkterne har, for jeg synes ikke at det er interessant at vide, at et punkt har mange naboer, hvis disse naboer alle sammen er med i turen i forvejen. Derfor vælger jeg nu at ignorere naboer, der allerede er med i turen i mine optællinger, undtagen startpunktet, der ikke kan vælges, men som der gerne må føre en kant tilbage til til sidst.

5's naboer: 3 (3), 6 (3). 6 vælges.

6's naboer: 7 (2), 9 (4). 7 vælges entydigt.

7 har kun 8 som nabo, der vælges.

8's naboer: 2 (3), 9 (3). 2 vælges.

Af 2's naboer kan kun 3 vælges.

Af 3's naboer kan kun 9 vælges.

9 har en kant tilbage til 1.

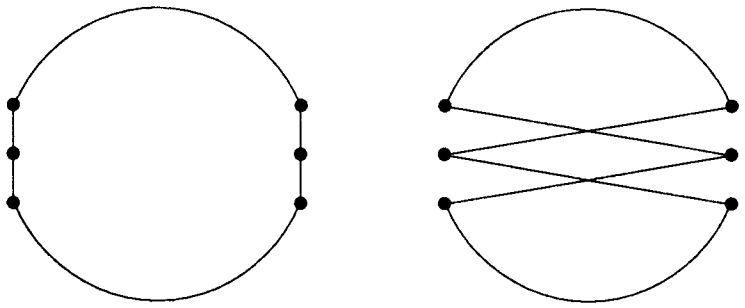
Og således blev der en ny tur ud af det. Denne metode er meget effektiv til at genbruge kanter fra forældrene, og måske knapt så konsekvent til at genbruge lange stykker (selv om det vil ske, hvis de to ture har et stykke tilfælles). Kun omkring en procent af kanterne i afkommet (ifølge *Mich 92* fra *Whit 89*) er nye kanter, og dette er et godt resultat.

I øvrigt forsøgte jeg med dette lille eksempel (kun 9 byer) at lave det andet afkom også, hvor man starter med by nr.4, men ellers gør det samme. I de to mulige ture jeg fandt, var alle 4 af dublet-kanterne med (kun 3 af dem er med i ovenstående eksempel), og faktisk var der kun 2-3 kanter, der var forskellige fra ovenstående afkom. Men dette vil altså ændre sig med længere ture.

forhold

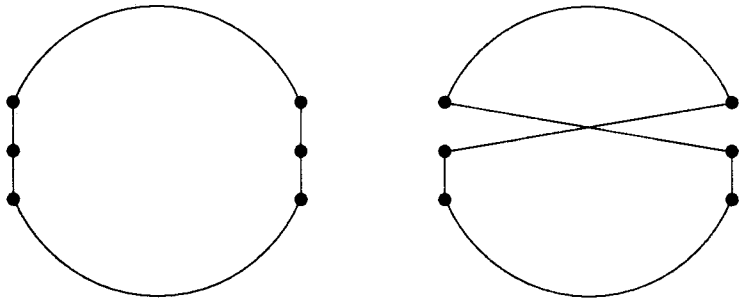
4.2.2 Mutationer.

En mutation skal tilsvarende opfylde, at resultatet stadig er en tur. En nem metode er at bytte om på 2 byer i listen, se figur 4.7.



Figur 4.7: 2-bytning, af punkter.

Da der herved opstår hele 4 nye kanter, er denne metode (empirisk og intuitivt) underlegen overfor 2-opt (fra bl.a. simuleret udglødning), hvor et helt stykke af listen vendes, så kun 2 nye kanter opstår, se figur 4.8.



Figur 4.8: 2-opt.

4.2.3 Mine forsøg.

Jeg vil teste både PMX og ERC sammen med 2-opt. Graferne vil blive de samme som de brugt til simuleret udglødning.

Start-befolkningerne dannes ved at finde tilfældige individer. I *Gref 87* argumenteres der vha. forsøg for, at man kan lade befolkningen bestå af ture konstrueret fx. af nærmeste nabo algoritmen, eller lignende algoritmer, der hurtigt giver ture med visse gode egenskaber. Dog skal der stadig være stor variation i start-befolkningen, for at man ikke kører fast i det samme individ for hurtigt.

Der argumenteres også for at bruge 2-opt algoritmen på de ture, der findes til sidst, og dette synes jeg lyder som en god ide, fordi den fundne tur jo ikke nødvendigvis er 2-optimal, ja faktisk sikkert ikke er det.

Begge disse metoder kunne være spændende at undersøge.

I mine forsøg stødte jeg på, at min brøkudregning ikke er nøjagtig nok, så det er ikke sikkert at den kumulerede levedygtighed for det sidste individ er 1. Derfor indfører jeg en "trimning", hvor de sidste individer reguleres en lille smule hver i relativ levedygtighed, så fejlen fordeles lidt.

4.3 Teori bag genetiske algoritme.

Denne teori redegør for årsagerne til, at den generelle genetiske algoritme vil give gode resultater. Meget af denne teori kommer fra *Mich 92*, men stammer bl.a. fra *Holl 75*.

Definition 7 *For en bit-streng er en skabelon en angivelse af nogle af bit-værdierne. En bit-streng passer sammen med en skabelon hvis den har de samme værdier som skabelonen, på de pladser skabelonen definerer.*
En skabelons orden, $o(S)$, er antallet af på forhånd bestemte positioner.
En skabelons længde, $\delta(S)$, er afstanden mellem første og sidste bit.
En skabelons hyppighed, $\xi(S,t)$, er antallet af individer i befolkningen til tid t, der passer sammen med skabelonen.
En skabelons relative levedygtighed, $l(S,t)$, er snittet af de relative levedygtigheder for alle de individer i befolkningen, der passer sammen med skabelonen.

For de tidligere betragtede 16-bits-strenge, kunne en skabelon være (...1...1.....). Denne angiver et et-tal på 4. og på 8.plads, og en streng passer sammen med skabelonen hvis den har et-taller disse to steder. $o(S) = 2$, fordi netop 2 pladser er defineret. $\delta(S) = 8 - 4 = 4$, forskellen på 4. og 8.plads. (Bemærk $o(S) = 1 \Rightarrow \delta(S) = 0$.)

Det interessante er hvad der sker i den næste generation, altså: hvordan er $\xi(S, t+1)$?

Det første der sker er kopiering, og da den relative levedygtighed af enkeltindivider såvel som af skabeloner afgør antallet i næste generation, fås umiddelbart:

$\xi(S, t+1) = \xi(S,t) \cdot N \cdot l(S,t)$

Dvs. hvis $l(S,t)$ er højere end $\frac{1}{N}$ (gennemsnitslevedygtigheden) bliver der sandsynligvis højere hyppighed af skabelonen og vice versa. Yderligere kan det argumenteres at hyppigheden vil vokse exponentielt henover flere generationer, omtrent med faktor $(N \cdot l(S,t))^k$ over k generationer (omtrent fordi $l(S,t)$ ikke er konstant gennem generationerne).

Med den for bit-strenge viste simple overkrydsning, vil skabelonen stadig passe til et afkom, hvis skabelonen passede til en af forældrene, og overkrydsningen ikke skete mellem skabelonens yderbits. Sandsynligheden for det sidste afhænger af $\delta(S)$. Overkrydsningsstedet kunne være $m-1$ forskellige steder (m antallet af bits, dvs. 16), og sandsynligheden bliver dermed $p_d(S) = \frac{\delta(S)}{m-1}$ for at S destrueres ved overkrydsningen, og $p_b = 1 - \frac{\delta(S)}{m-1}$ for at S bevares.

Da kun de overkrydsede strenge påvirkes bliver sandsynligheden for overlevelse: $1 - \frac{rM}{N}\frac{\delta(S)}{m-1}$ den faktor der skal tilføjes ligningen.

Da der ved overkrydsning godt kan opstå afkom, der passer til skabelonen uden at nogle af forældrene passede, eller netop hvis begge forældre passede, bliver sandsynligheden lidt større. Derfor bliver ligningen en ulighed:

$\xi(S, t+1) \geq \xi(S,t) \cdot N \cdot l(S,t)(1 - \frac{rM}{N}\frac{\delta(S)}{m-1})$

Ved mutation vil en skabelon blive bevaret, hvis ingen af bittene i skabelonen vendes, dvs. med sandsynlighed $\frac{M}{N}(1-r)$ for mutation og sandsynlighed $\frac{o(S)}{m}$ for at en af skabelonens bits bliver vendt, og sandsynlighed for at skabelonen bliver bevaret medregnet fås dette:

$\xi(S, t+1) \geq \xi(S,t) \cdot N \cdot l(S,t)(1 - \frac{rM}{N}\frac{\delta(S)}{m-1} - \frac{M}{N}(1-r)\frac{o(S)}{m})$

Igen kan en skabelon opstå ved en mutation, så ulighedstegnet er dobbelt berettiget.

Sætning 29 *En skabelon af kort længde, lav orden og med levedygtighed over gennemsnittet, vil få en exponentielt voksende hyppighed over flere generationer.*

Beviset er den ovenstående udledning.

Hypotese 1 *Den genetiske algoritme søger nær-optimale løsninger ved at sammensætte korte, lav-ordens, meget levedygtige skabeloner, kaldet byggeklodser.*

I lyset af hvilke overkrydsningsmetoder der virker, og hvordan algoritmen i praksis virker, er dette nok en god hypotese, men altså svær at bevise.

For TSP kan disse beviser ikke direkte genbruges, men der vil stadig kunne defineres skabeloner osv., op til byggeklodser, der bliver kortere eller længere kant-stykker, der vil gå igen ofte, jo bedre de er.

4.4 Forsøg med parametrene.

4.4.1 ERC-forsøgsresultater.

Spørgsmål 11 *Ca. hvordan skal parametrene befolkningsstørrelse (og dermed antal generationer), antal af nyt afkom og sandsynlighed for overkrydsning hvis avl allerede bestemt (altså N, M og r) indstilles, for at algoritmen kører bedst? (Overkrydsning: ERC.)*

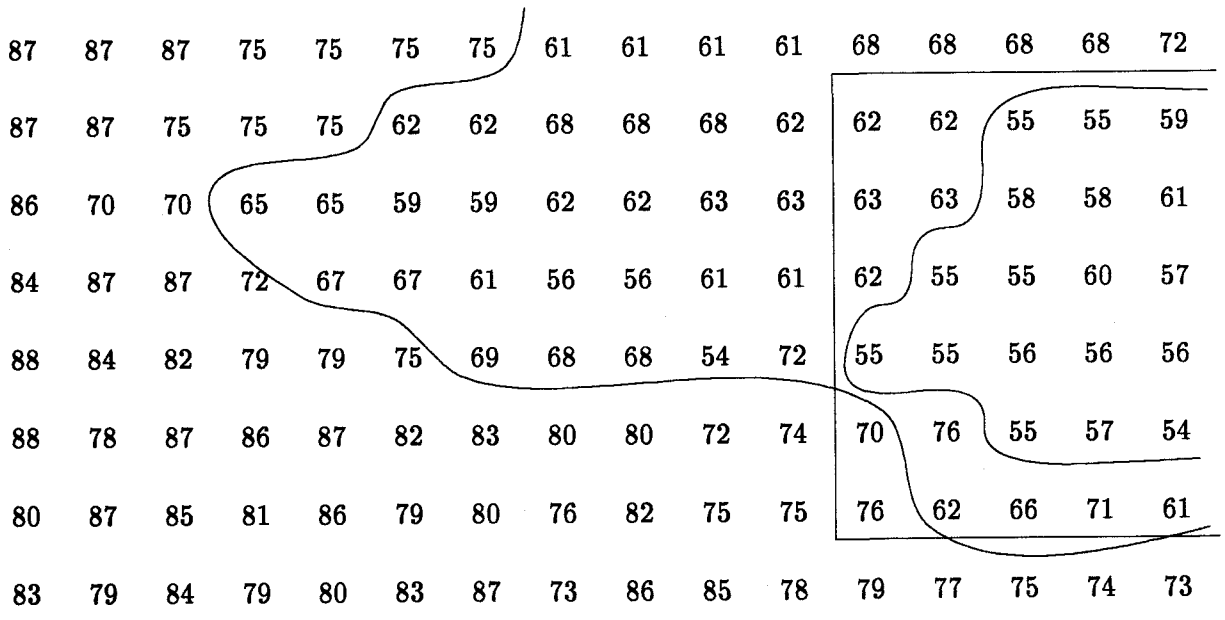
Jeg har set resultater der peger på, at mængden af afkom kan blive både for høj og for lav. Hvis mængden af "nyfødte" er for høj, vil næsten ingen af de gode individer allerede fundet overleve til næste generation, og man kunne lige så godt lade hver ny generation være fundet som den første: ved tilfældig generering. Hvis mængden er for lille spildes tiden med at passe på de gode individer, der godt kunne overleve i en skarpere konkurrence, og køretiden for algoritmen bliver for lang. Befolkningsstørrelsen rapporteres at kunne blive for lille, således at resultatet af algoritmen bliver rimeligt kaotisk, fordi selv det bedste individ har værre chancer i en lille befolkning end i en stor, mens befolkningen kan blive så stor, at en forøgelse af størrelsen kun forøger køretiden og ikke resultaterne (hvis antallet af generationer er konstant). Endelig må mutationssandsynligheden (dvs. det omvendte af sandsynligheden for overkrydsning) ikke blive for stor, igen fordi der så opstår for mange helt nye egenskaber, før de allerede eksisterende rigtigt når at blive testet, mens der er en grænse nedefter for, at mindre mutation bare giver dårligere resultater, fordi der ikke kommer nye egenskaber nok.

Derfor laver jeg et forsøg hvor jeg tester resultaternes kvalitet ved forskellige værdier af de tre parametre. Jeg lader produktet af befolkningsstørrelse og antal generationer være konstant 400.000 (som i *Ebse 92*), så der bliver kopieret 400.000 individer, mens det er variabelt hvor mange af dem der bare bliver kopieret, uden yderligere ændringer. Så varierer jeg M/N mellem 0,1 og 0,45 og r mellem 0,05 til 0,8, begge i spring på 0,05, og tester de to størrelser af befolkning 100 og 200. Resultaterne opsummeres her. Algoritmen får lov til at køre det angivne antal generationer, og længden af den bedste tur undervejs er resultatet af en kørsel. Der køres kun en enkelt gang ved hvert sæt parametre, men resultater fra to "nabo-sæt" kan godt betragtes som resultat af to næsten identiske kørsler.

Den graf, algoritmen blev testet på, var min egen fremstillede med 20 punkter. Det bedste resultat i kørslen var en tur af længde 52 (reelle afstande, turlængder rundet ned). Ved befolkning på 100 var de bedste resultater for M/N mellem 0,15 og 0,4 og r mellem faktorerne 0,6 og 0,8, hvor 0,8 er den ene grænse indbygget i forsøget, og jeg måske derfor ikke ser nogle endnu bedre resultater for højere værdier af r. Disse gode resultater var mellem 54 og 76, med et snit på 60,3 blandt de 30 resultater. Se figur 4.9.

Ved en befolkning på 200 blev resultaterne generelt bedre overalt, men den samme "ø" af gode resultater kunne ses, med resultater mellem 52 og 76, og et snit på 58,8. Jeg konkluderer heraf, at jeg godt vil checke befolkninger på mere end 200 individer også, at M/N vil jeg godt checke igen, med værdier højere end de 0,45, og at jeg opgiver værdier for r mindre end 0,5.

På figuren ses alle resultaterne opgivet, hvor M/N vokser nedad, og r vokser mod højre. Med løs hånd er angivet to områder, hvor de fleste resultater opfylder hver sit kvalitetskrav: resultater bedre end hhv. 60 og 70. Desuden er angivet den kasse, der indeholder en klump gode resultater, og denne kasse kunne jeg altså godt tænke mig at vide mere om.



Figur 4.9: Ved befolkning 100, resultater.

Nedenfor er for fuldstændighedens skyld opgivet alle resultater, bortset fra r på 0,05 og 0,1, der ikke er specielt forskellige fra deres naboer. I øverste højre hjørne angives befolkningens størrelse. I øverste række ses r og i venstre søjle M/N. De bedste resultater er indrammet.

100	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8
0,1	87	75	75	75	75	61	61	61	61	68	68	68	68	72
0,15	75	75	75	62	62	68	68	68	62	62	62	55	55	59
0,2	70	65	65	59	59	62	62	63	63	63	63	58	58	61
0,25	87	72	67	67	61	56	56	61	61	62	55	55	60	57
0,3	82	79	79	75	69	68	68	54	72	55	55	56	56	56
0,35	87	86	87	82	83	80	80	72	74	70	76	55	57	54
0,4	85	81	86	79	80	76	82	75	75	76	62	66	71	61
0,45	84	79	80	83	87	73	86	85	78	79	77	75	74	73

200	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8
0,1	71	69	69	64	64	67	67	67	67	67	67	56	56	69
0,15	75	64	64	69	60	62	62	64	66	55	55	61	63	71
0,2	61	66	64	59	61	57	61	59	61	60	55	57	53	61
0,25	71	65	60	63	57	57	58	57	60	56	58	58	56	55
0,3	81	59	73	78	64	61	56	60	54	55	53	57	56	53
0,35	86	83	85	77	82	69	78	74	74	56	73	53	53	53
0,4	81	83	82	82	79	85	80	78	70	76	73	72	56	52
0,45	80	82	83	80	84	81	80	82	75	67	65	80	66	69

I det næste forsøg undersøger jeg de før nævnte paramterer-indstillinger.

200	0,5	0,6	0,7	0,8	0,9
0,1	67	67	56	69	69
0,2	59	60	57	61	58
0,3	60	55	57	53	56
0,4	78	76	72	52	52
0,5	81	78	76	69	61
0,6	82	81	75	78	68
0,7	81	82	77	77	74
0,8	84	82	79	76	76

400	0,5	0,6	0,7	0,8	0,9
0,1	61	66	65	63	60
0,2	56	57	62	53	52
0,3	53	54	52	52	53
0,4	78	53	74	53	52
0,5	80	75	76	71	61
0,6	74	77	81	75	71
0,7	79	82	82	73	75
0,8	76	75	73	78	75

600	0,5	0,6	0,7	0,8	0,9
0,1	61	63	64	58	59
0,2	58	56	55	57	54
0,3	58	53	52	53	52
0,4	76	70	60	53	52
0,5	83	79	68	71	63
0,6	79	79	77	74	67
0,7	78	80	79	75	72
0,8	77	78	81	77	75

Af disse 3 resultater ser jeg, at den lave M/N faktisk er god nok, og jeg dropper derfor den nederste halvdel af hver tabel. Til gengæld synes jeg ikke det er muligt med rigtig god samvittighed at droppe nogle søjler. Snittet i øverste halvdel af hver tabel er hhv. 61,7, 58,45 og 58,2, altså en svag forbedring så det stadig kunne være interessant at studere større befolkninger. Nu holder jeg M/N fra 0,4 og lavere.

Jeg checker nu "kanten" af disse områder, dvs. hvad der sker for M/N lavere end 0,1, r højere end 0,9, og befolkning højere end 600.

500	0,50	0,60	0,70	0,80	0,90	0,92	0,94	0,96	0,98
0,02	82	82	88	88	82	82	82	82	82
0,04	71	72	70	59	66	66	66	85	64
0,06	68	64	58	62	73	73	73	73	64
0,08	70	59	69	56	63	63	58	58	55
0,10	59	59	57	65	58	58	54	54	58
0.20					55	52	54	52	54
0.30					52	52	54	53	53
0.40					53	52	53	52	53
0.50					56	61	53	56	53
0.60					70	69	66	60	59
0.70					69	66	69	63	63
0.80					72	72	71	66	66

1000	0,50	0,60	0,70	0,80	0,90	0,92	0,94	0,96	0,98
0,02	74	77	63	68	62	62	62	65	65
0,04	72	67	58	69	58	58	62	62	63
0,06	64	61	56	61	55	62	62	55	55
0,08	57	70	57	54	58	56	59	59	55
0,10	56	59	56	52	54	54	56	56	58
0.20					59	52	52	52	52
0.30					53	52	52	52	52
0.40					52	52	52	52	52
0.50					62	59	55	53	53
0.60					71	66	64	65	59
0.70					67	71	70	65	62
0.80					68	73	72	67	62

1500	0,50	0,60	0,70	0,80	0,90	0,92	0,94	0,96	0,98
0,02	64	63	63	73	60	60	60	60	59
0,04	61	55	61	58	56	53	53	53	53
0,06	58	55	63	54	54	54	58	69	55
0,08	54	63	54	53	53	54	53	53	53
0,10	54	59	54	53	53	53	54	53	54
0.20					52	53	52	53	52
0.30					52	52	52	52	53
0.40					53	52	52	52	52
0.50					62	59	55	56	55
0.60					68	66	61	62	54
0.70					75	72	68	65	62
0.80					72	72	72	69	67

2000	0,50	0,60	0,70	0,80	0,90	0,92	0,94	0,96	0,98
0,02	65	63	73	72	64	64	62	62	58
0,04	66	61	62	56	58	55	56	56	56
0,06	61	53	53	55	57	55	58	54	54
0,08	56	55	56	56	54	53	52	53	53
0,10	52	53	57	53	53	54	52	52	53
0.20					52	52	52	53	52
0.30					52	52	52	52	52
0.40					53	56	52	52	54
0.50					63	59	59	58	57
0.60					68	67	65	59	60
0.70					71	69	68	68	62
0.80					74	70	72	70	67

Af disse 4 tabeller ser jeg, at området hvor algoritmen kører bedst er for M/N mellem 0,1 og 0,5, og dette gælder også i hele det nye område, for mutationssandsynlighed fra 0,9 og op. Snittet i denne box er for de 4 tabeller hhv. 54,2, 53,92, 53,52 og 53,92. Endnu kan det godt være et tilfælde, at resultaterne begynder at blive dårlige ved så høj befolkning (og så få generationer), så jeg vil godt checke ved højere befolkning også.

Næste forsøgsrække laver jeg ved 0,1 (laveste gode værdi af M/N, og giver korteste køretid at have få afkom), ved befolkninger på 2000 til 10000, og med mutationssandsynlighed varierende fra 0,95 til 0,999, i spring der halverer afstanden til 1.

0,1	0,9500000	0,9750000	0,9875000	0,9937500	0,9968750	0,9984375
2000	56	53	53	54	54	54
4000	53	54	53	52	52	52
6000	57	55	55	60	56	56
8000	57	57	61	60	60	61
10000	64	67	61	60	62	64

I denne tabel synes jeg, at der kommer et skel ved 6000 eller flere i befolkningen, således at resultaterne igen begynder at blive dårlige. Men der er stadig ikke nogen grænse for, hvor sjældent mutation skal finde sted, for at resultaterne bliver dårlige.

Så nu vil jeg komme med en samlet konklusion.

Svar 11 *Befolkningen bør ligge omkring 1500, men gerne mellem 500 og 4000.*
Afkomssandsynligheden bør ligge omkring 0,2, men gerne mellem 0,1 og 0,5.
Sandsynligheden for krydsning bør ligge omkring 0,9, men gerne fra 0,5 til lige under 1.

Hvad nøjagtigt der vælges afhænger også af hvor lang køretid man kan bruge. Det tager længere tid med flere nye afkom pr. generation, og en krydsning tager længere tid end en mutering. Om man har 500 individer og 800 generationer eller 2000 individer og 200 generationer burde tage den samme tid (bortset fra afrunding af antallene), men det passer ikke helt. Det tager længere tid med mange individer af gangen, og jeg vil tro at dette skyldes, at en større del af maskinens hukomelse så er i sving, når der kopieres individer frem og tilbage. Det kunne være interessant at undersøge nøjagtigt hvilken effekt de forskellige parametre har på køretiden.

4.4.2 PMX-forsøgsresultater.

Spørgsmål 12 *Ca. hvordan skal parametrene N, M og r indstilles, for at algoritmen kører bedst? (Overkrydsning: PMX.)*

500	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0,1	74	72	62	71	71	69	58
0,2	78	68	55	58	72	61	61
0,3	64	66	59	61	67	69	59
0,4	81	81	79	80	80	77	57
0,5	79	81	84	78	82	80	79
0,6	85	77	83	80	84	79	81
0,7	85	76	82	73	71	83	80
0,8	81	85	76	82	82	78	81

1000	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0,1	74	70	71	70	70	68	69
0,2	65	72	66	60	67	63	70
0,3	59	72	58	61	67	59	53
0,4	80	79	81	83	74	64	66
0,5	76	80	81	82	77	82	76
0,6	75	85	81	82	83	78	75
0,7	84	85	83	76	80	76	82
0,8	82	83	80	82	78	77	83

1500	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0,1	70	69	73	65	73	66	60
0,2	68	67	64	54	61	68	66
0,3	69	69	58	52	56	67	60
0,4	86	85	79	76	79	77	63
0,5	79	85	81	83	72	82	77
0,6	80	81	86	81	85	78	78
0,7	83	81	85	82	82	84	79
0,8	86	83	85	86	82	80	80

2000	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0,1	73	76	63	59	71	58	65
0,2	68	69	68	61	54	67	69
0,3	85	59	69	62	69	64	69
0,4	85	80	82	81	79	72	75
0,5	82	81	81	70	83	82	80
0,6	82	77	73	83	81	76	80
0,7	70	84	81	82	76	77	80
0,8	84	81	80	79	80	76	78

Af de fire ovenstående tabeller ses resultater for befolkninger mellem 500 og 2000 individer, M/N mellem 0,1 og 0,8 og r mellem 0,3 og 0,9. Af alle tabeller kan jeg se, at det nok er bedst at undersøge for både større og mindre befolkninger også, at M/N skal holdes på højst 0,3 og at krydsningssandsynligheden skal holdes på mindst 0,4. Men jeg får altså lyst til at se, hvad

der sker ved større og mindre befolkninger, ved meget små værdier af M/N og ved meget store krydsningssandsynligheder.

I tabellerne er indkredset de gode områder, der i de 4 tilfælde har snit på hhv. 64,4, 65,9, 63,8 og 65,1.

Bemærk at i det følgende forsøg giver mange nabo-kombinationer af parametre samme resultat, fordi de faktisk giver samme forsøg. Hvis 2 % af 100 individer skal udvælges til avl, udvælges 2. Hvis mindre end 100 % af disse så bruges til krydsning, eller rettere det eneste par ikke krydses sammen, går de begge til mutation i stedet. Når resten af omstændighederne derudover er de samme (samme initial-befolkning, samme tilfældige tal undervejs), vil forsøget altså blive gentaget. Men for fuldstændighedens skyld er disse forsøg også udført, og deres resultater taget med. Denne effekt gør sig også gældende andre steder. Til gengæld er alle disse forsøg ikke kørt på samme slags maskine, så nogle dublet-resultater alligevel ikke er ens.

100	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	95	95	95	95	95	95	95	95	95	95
0.04	93	93	93	83	83	83	83	83	83	83
0.06	81	81	81	81	87	87	87	87	87	87
0.08	78	78	83	83	83	83	83	83	83	83
0.10	73	73	75	75	79	79	79	79	79	79
0.20	71	59	69	66	70	67	67	67	67	67
0.30	79	64	60	52	63	57	57	65	65	65

200	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	95	95	95	94	94	94	94	94	94	94
0.04	81	81	80	80	80	85	85	85	85	85
0.06	81	81	81	81	78	78	83	83	83	83
0.08	76	76	76	71	85	85	85	85	85	85
0.10	73	74	80	70	73	78	78	78	78	78
0.20	67	62	69	63	72	63	63	63	71	71
0.30	62	61	56	61	65	57	57	65	65	73

300	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	90	90	90	90	94	94	94	94	94	94
0.04	79	79	87	87	80	80	91	91	91	91
0.06	84	79	70	70	85	75	75	75	84	84
0.08	70	74	76	74	78	81	81	81	79	79
0.10	71	68	76	74	76	76	76	70	70	70
0.20	67	71	68	61	66	69	69	73	73	68
0.30	68	63	62	65	67	63	61	64	64	66

400	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	94	94	78	78	78	91	91	91	91	91
0.04	79	79	80	82	77	77	77	83	83	83
0.06	82	82	77	83	80	83	83	83	82	82
0.08	68	76	78	79	73	81	79	79	79	84
0.10	75	78	75	68	75	68	68	68	88	88
0.20	67	71	67	69	71	62	62	56	64	67
0.30	61	73	56	64	68	58	65	52	67	60

500	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	89	92	92	85	85	81	81	81	81	81
0.04	83	82	87	69	82	85	85	85	81	81
0.06	76	74	71	70	74	79	79	79	79	75
0.08	68	73	72	74	75	72	72	71	71	76
0.10	72	62	71	71	69	58	58	74	74	73
0.20	68	55	58	72	61	61	61	73	65	57
0.30	66	59	61	67	69	59	53	63	65	65

For de 5 tabeller ovenfor, hvor de små befolkningsstørrelser er testet, indkredser jeg de to nederste rækker som specielt gode, men kan ikke komme med bud på nogle specielt gode søjler til nogle af siderne. Snittet af de to nederste rækker giver hhv. 64,85, 64,3, 66,4, 64 og 62,9.

2000	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	79	76	73	75	86	73	73	72	72	79
0.04	77	81	72	74	73	63	77	66	66	75
0.06	76	74	73	66	62	73	74	68	62	67
0.08	71	74	61	64	59	64	61	72	70	66
0.10	76	63	59	71	58	65	59	69	72	68
0.20	69	68	61	54	67	69	56	62	54	70
0.30	59	69	62	69	64	69	52	54	63	58

4000	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	87	72	78	71	72	74	82	75	75	82
0.04	73	81	72	76	70	70	73	75	75	72
0.06	78	68	68	68	75	70	70	75	73	65
0.08	70	75	67	66	67	69	71	65	67	70
0.10	72	70	75	70	69	69	66	68	72	70
0.20	65	67	71	67	70	63	56	65	55	65
0.30	74	82	79	73	71	73	70	70	65	73

6000	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	83	84	71	81	78	80	79	72	85	83
0.04	71	80	82	75	71	77	70	77	73	67
0.06	76	77	77	77	75	74	71	73	66	72
0.08	67	77	81	76	69	66	71	74	66	71
0.10	79	74	74	77	72	68	73	66	64	70
0.20	67	72	68	70	66	63	69	66	70	65
0.30	76	81	78	80	66	73	73	72	66	73

8000	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	82	84	81	84	79	80	81	82	72	76
0.04	71	77	75	80	83	74	80	72	77	78
0.06	77	78	71	79	75	80	73	75	75	77
0.08	76	78	75	78	67	78	70	74	74	72
0.10	79	80	72	74	67	80	76	70	70	73
0.20	75	74	73	66	69	70	75	70	73	70
0.30	78	82	80	79	79	74	75	72	75	75

10000	0.4	0.5	0.6	0.7	0.8	0.9	0.92	0.94	0.96	0.98
0.02	78	86	73	81	80	79	81	82	72	76
0.04	87	83	78	85	82	70	84	70	77	73
0.06	81	80	72	77	75	77	69	81	74	78
0.08	76	79	78	80	79	77	76	69	76	71
0.10	82	75	81	72	72	75	76	79	72	75
0.20	79	80	71	80	70	70	71	77	73	71
0.30	71	80	78	72	77	79	78	70	75	81

Igen er de bedste resultater i de to nederste rækker, med en svag effekt af, at der også er gode resultater for r ca. 0,8, og M/N lidt lavere end 0,2. Snit for de nederste rækker er hhv. 62,45, 68,7, 70,7, 74,2 og 75,15.

Svar 12 *Befolkningen bør ligge mellem 500 og 2000.*
Afkomssandsynligheden bør ligge mellem 0,2 og 0,3.
Sandsynligheden for krydsning bør ligge over 0,4.

4.5 Figurer for forsøg.

Mine næste forsøg vil bl.a. foregå på en større graf (eil51) og jeg er derfor spændt på, om ovenstående konklusioner kan genbruges. Specielt kan længden af en tur, og dermed antallet af mulige krydsningssteder og mutationer for samme tur, og antallet af forskellige ture, have betydning for hvor stor en befolkning bør være, fordi der måske skal være en vis procentdel af de mulige ture repræsenteret.

For at få et indtryk af forholdene undersøger jeg nøje en kørsel ved de tre N 500, 1000 og 2000, ved de tre M/N 0,1, 0,2 og 0,3, og ved de tre r 0,8, 0,9 og 0,95. I alt 27 forskellige muligheder.

Forløbet af en kørsel studerer jeg nu ved, for hver generation, at notere mig bedste, værste og gennemsnits-tur. Dette gør jeg både for alle de nye afkom, og for hele den nye befolkning. Dog noterer jeg mig den værste blandt de gamle (den del af befolkningen, der er kopier), for at se om den værste tur altid er blandt de nye. Dette giver mig 6 tal for hver generation, og dermed 6 kurver, når jeg studerer udviklingen over alle generationer. Her viser jeg figurerne med alle 6 kurver på, selv studerede jeg også kurverne hver for sig, men man kan godt skelne, at kurverne nedefra er bedste blandt alle, bedste blandt nye, snit blandt alle, snit blandt nye, og så de to kurver for værste, hvor værste blandt gamle nogle gange er meget lavere end værste blandt nye. Ved alle figurer angives værdierne af $N, M/N$ og r.

4.5.1 ERC eil51.

8 8 8 12 12 13 120 113 117
11 11 12 16 17 17 126 121 123
16 17 17 20 20 21 127 124 125

Køretider

Spørgsmål 13 *Hvordan ser en "god" kørsel ud? Hvordan ser den resulterende tur ud? (Overkrydsning: ERC.)*

500	0,8	0,9	0,95
0,1	716	687	636
0,2	550	612	585
0,3	1061	859	706

1000	0,8	0,9	0,95
0,1	645	705	572
0,2	698	622	644
0,3	1093	758	737

2000	0,8	0,9	0,95
0,1	763	706	753
0,2	826	764	715
0,3	1116	993	882

Snit for de tre N: 712, 719 og 835, umiddelbart er 2000 individer dårligt.

Snit for de tre M/N: 687, 668 og 912, umiddelbart er 0,3 for høj M/N. Blandt de 9 resultater opnået ved 0,3, er de 6 længste ture.

Snit for de tre r: 830, 745 og 692, umiddelbart er 0,95 den bedste r.

Derudover studerer jeg altså forløbet af hver kørsel, for at se hvad der kendetegner en god eller dårlig kørsel, og hvordan sammenhængen er med parametrene.

For $M/N = 0{,}3$ ser jeg en meget lille forskel på de 2 snit-kurver (som på figur 4.14 og figur 4.15), som om de nye og de gamle er lige dårlige. Ved $M/N = 0{,}1$ er der derimod stor forskel på disse 2 kurver (som fx. på figur 4.10). Den gamle del af befolkningen er jo kopier fra den tidligere generation, vægtet således at snittet i den gamle del bliver lavere end det var i hele den tidligere generation (med stor sandsynlighed). Jo færre nye afkom der er, jo bedre slår denne effekt igennem, og jo bedre bliver det endelige resultat.

Det er desuden et godt tegn, hvis afstanden mellem snit-kurverne er stigende længe, og dermed den endelige afstand er stor, og denne effekt ses også ved lav M/N (ses tydeligt på figur 4.13).

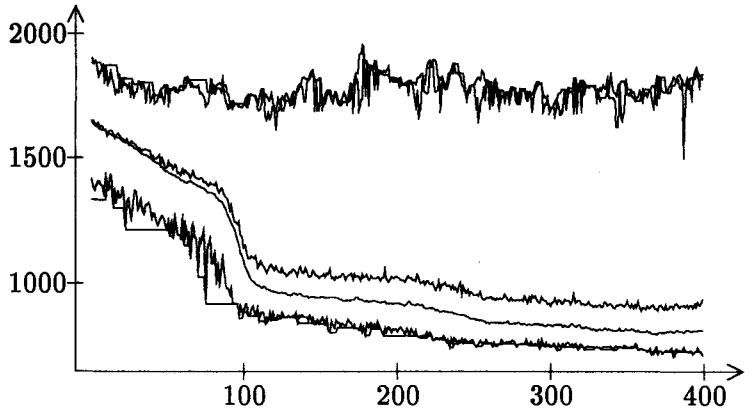
For $r = 0{,}95$, fremfor ved lavere r, ses en større hældning af alle kurverne (se på figur 4.11). En dårlig kørsel kan have, at alle kurverne godt nok svinger, men ellers er flade (se figur 4.14 og figur 4.15). Jo bedre kørsel, jo mere hældning, måske i starten af kørslen, så slutningen bliver

flad (som på figur 4.12), måske over hele kørslen (som på figur 4.11). Og meget hældning er tegn på en hurtig bevægelse fra tilfældige ture med dårlig længde til udvalgte ture med god længde.

Ved stigende r falder afstanden mellem snit-kurverne (sammenlign figur 4.12 og figur 4.13). Og ved stigende r falder antallet af mutationer, der kan tilføje det nye afkom nogle ture, der oftest vil være længere end forældrene.

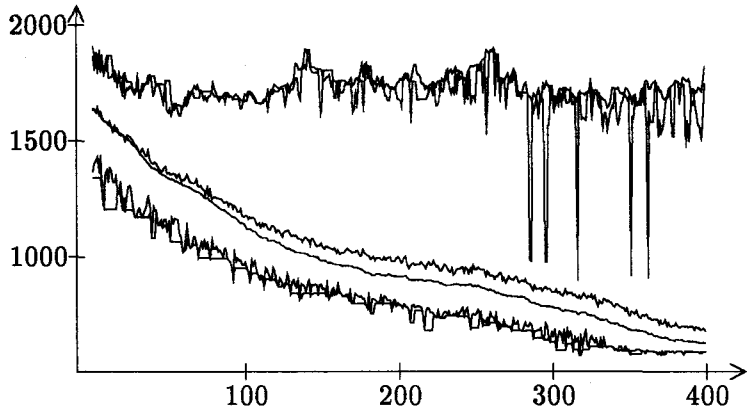
Ved $N = 2000$ ser kurverne ud, som om kørslen ikke når at blive færdig på 200 generationer. En færdig kørsel vil bl.a. se snit-kurverne og bedste-kurverne komme tættere på hinanden til sidst (som på figur 4.11, egentlig som tegn på, at der nu er mange kopier af de samme gode individer) og se alle kurver flade ud (som på figur 4.12), fordi en forbedring ikke mere sker / ikke mere er mulig.

Jeg observerer nogle dybe hak i bedste-blandt-gamle-kurven når der er meget få mutationer (4-6 individer, fx. på figur 4.12).



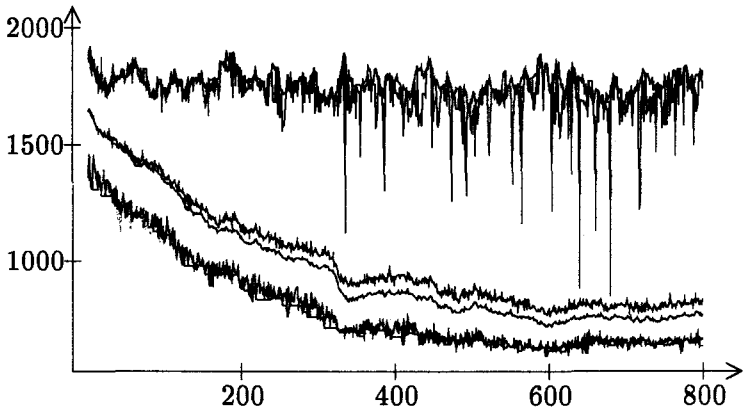
max: 1953, min: 705.

Figur 4.10: 1000, 0,1, 0,9.



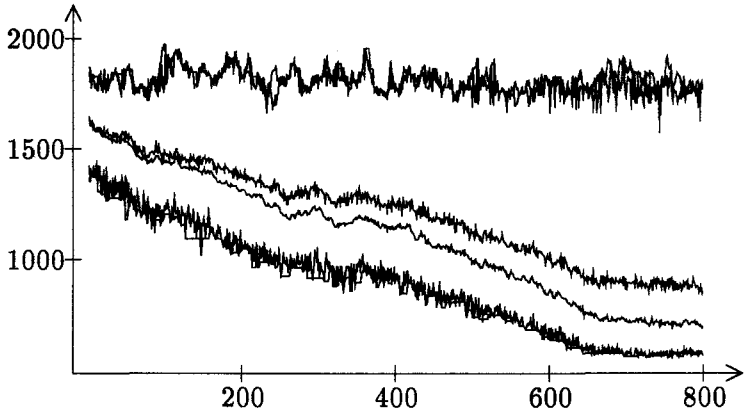
max: 1912, min: 572.

Figur 4.11: 1000, 0,1, 0,95.



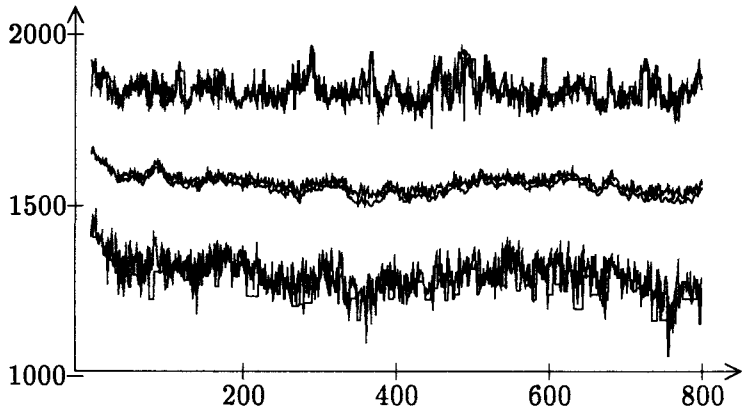
max: 1922, min: 585.

Figur 4.12: 500, 0,2, 0,95.



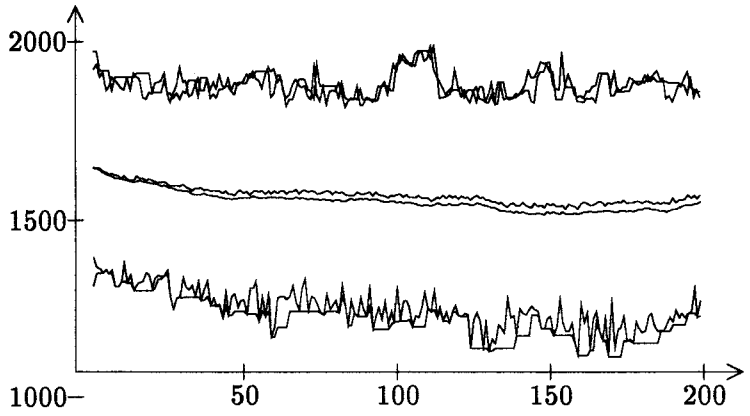
max: 1973, min: 550.

Figur 4.13: 500, 0,2, 0,8.



max: 1966, min: 1061.

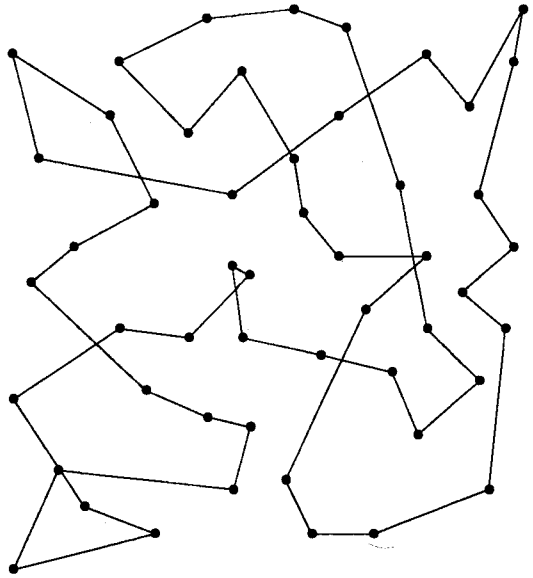
Figur 4.14: 500, 0,3, 0,8.



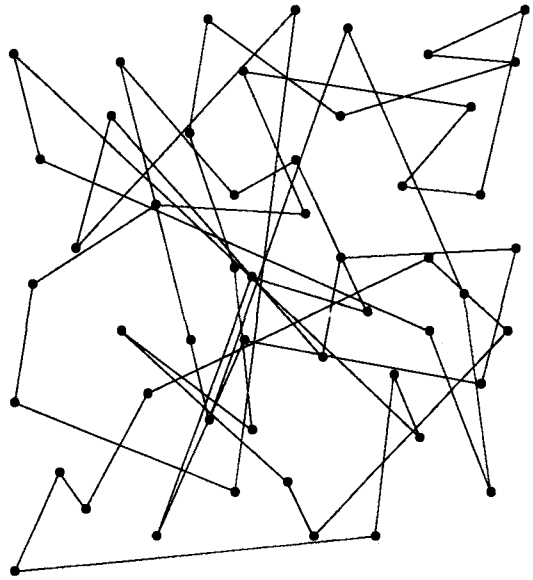
max: 1987, min: 1116.

Figur 4.15: 2000, 0,3, 0,8.

For at vise hvilken slags ture, der kan findes med den genetiske algoritme, vises her den bedste og den dårligste tur fra denne kørsel. Den bedste ville i hvert fald kunne forbedres med 2-opt, men er derudover ret pæn, og har fat i mange korte kanter.



Figur 4.16: 500, 0,2, 0,8, længde 550.



Figur 4.17: 2000, 0,3, 0,8, længde 1116.

4.5.2 PMX, 20 punkter.

Spørgsmål 14 *Hvordan ser en "god" kørsel ud? Hvordan ser den resulterende tur ud? (Overkrydsning: PMX.)*

500	0,8	0,9	0,95
0,1	69	68	76
0,2	63	61	61
0,3	60	66	55

1000	0,8	0,9	0,95
0,1	67	67	64
0,2	57	59	65
0,3	63	61	55

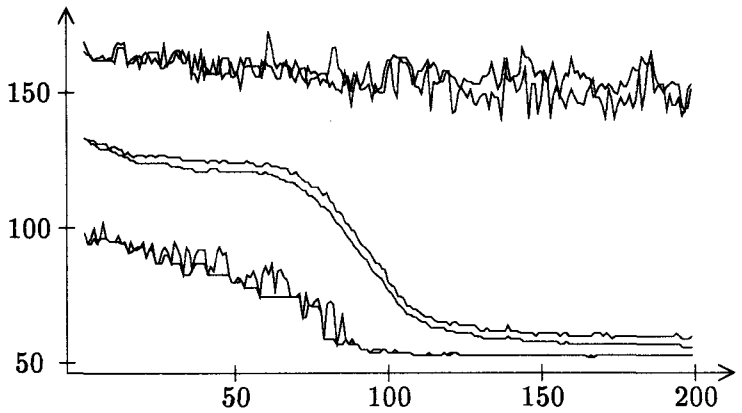
2000	0,8	0,9	0,95
0,1	65	55	69
0,2	64	62	54
0,3	56	53	52

Snit for de tre N: 64, 62 og 59, umiddelbart er det bedst med 2000 individer.
Snit for de tre M/N: 67, 61 og 58, umiddelbart er det bedst med 0,3.
Snit for de tre r: 63, 62 og 62, umiddelbart er der ingen forskel.

Jeg observerer hak nedad i værste-kurven for mange af kørslerne, som på figur 4.19.

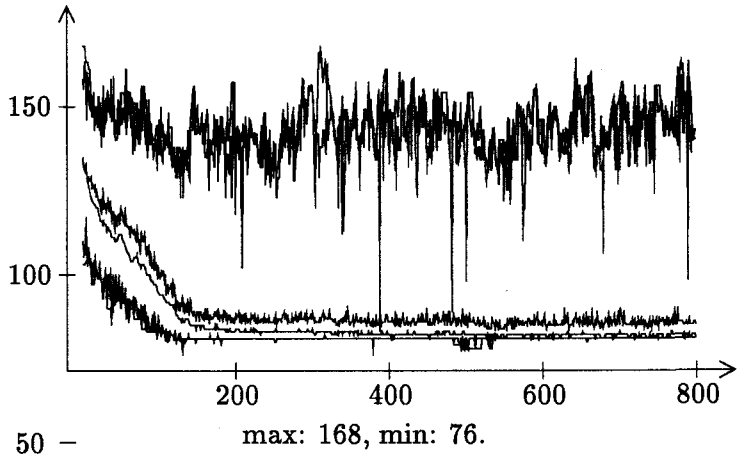
For 500 og 1000 individer er kurverne flade meget længe til sidst (som på figur 4.19), som om hvilket resultat der nu end er fundet, ikke kan blive bedre. Ved 2000 individer udnyttes den store befolkning positivt, og kørslen bliver alligevel færdig.

Ved $M/N = 0{,}3$ er der størst afstand mellem bedste- og snit-kurverne, sammenlign figur 4.19 og figur 4.18.



max: 167, min: 52.

Figur 4.18: 2000, 0,3, 0,95.



Figur 4.19: 500, 0,1, 0,95.

4.5.3 PMX, eil51.

Spørgsmål 15 *Hvordan ser en "god" kørsel ud? Hvordan ser den resulterende tur ud? (Overkrydsning: PMX.)*

500	0,8	0,9	0,95
0,1	955	1039	1107
0,2	846	910	768
0,3	1213	1192	1178

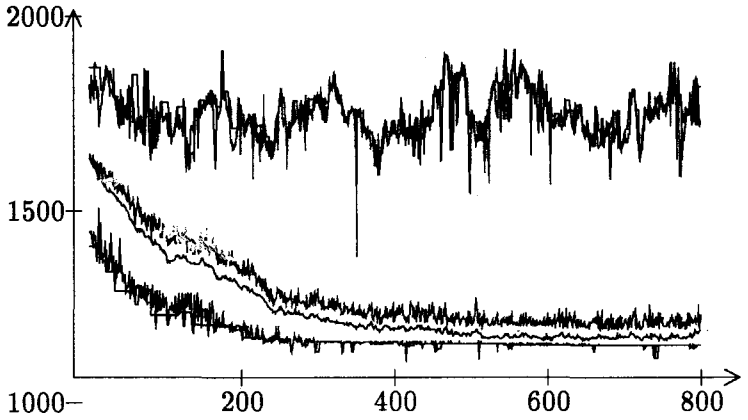
1000	0,8	0,9	0,95
0,1	966	961	970
0,2	993	789	884
0,3	1256	1177	1228

2000	0,8	0,9	0,95
0,1	967	1001	934
0,2	1167	1086	1141
0,3	1230	1199	1121

Snit for de tre N: 1023, 1025 og 1094, umiddelbart er 2000 for mange individer.
Snit for de tre M/N: 989, 954 og 1199, umiddelbart er 0,2 bedst.
Snit for de tre r: 1066, 1039 og 1037, umiddelbart er 0,8 for lav.

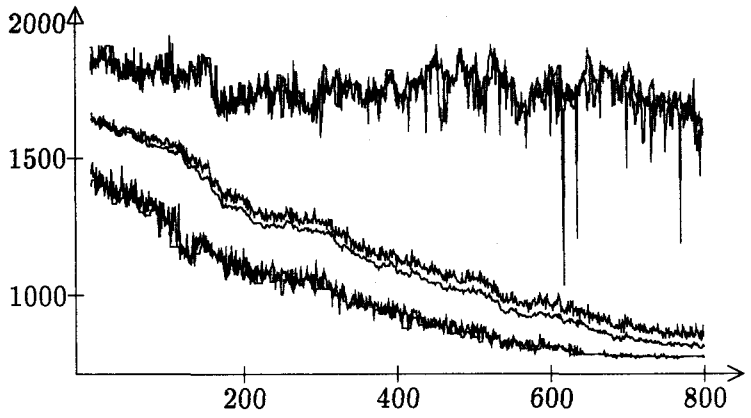
Ved 0,3 er der meget flade kurver (som på figur 4.22), ved 0,1 bliver kurverne først flade til sidst (som på figur 4.20, på figur 4.23 bliver de slet ikke flade). Bemærk hvordan kurverne på figur 4.21 bliver fladere meget senere end på figur 4.20, altså ved 0,2 ses pænere opførsel end ved 0,1 for M/N.

Ved 2000 bliver kørslen ikke færdig.



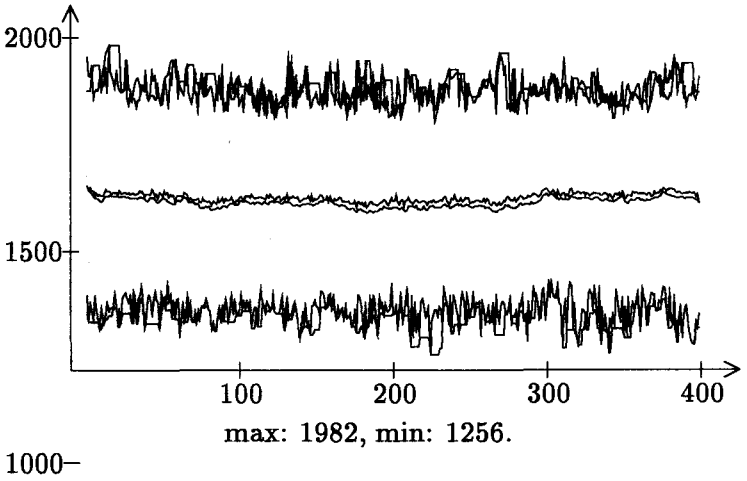
max: 1915, min: 1107.

Figur 4.20: 500, 0,1, 0,95.



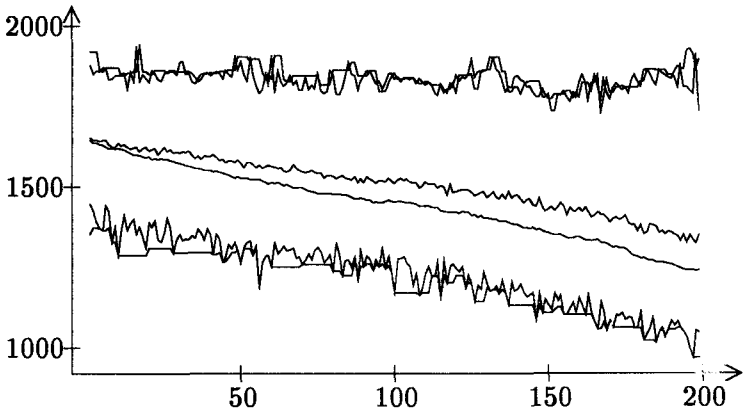
max: 1915, min: 768.

Figur 4.21: 500, 0,2, 0,95.



max: 1982, min: 1256.

Figur 4.22: 1000, 0,3, 0,8.



max: 1939, min: 967.

Figur 4.23: 2000, 0,1, 0,8.

Kapitel 5

Konklusion.

I dette speciale har jeg belyst forskellige sider af den handelsrejsendes problem. Jeg har beskæftiget mig med problemets historie og med forskellige, nye og gamle måder at løse det på. Og forhåbentlig har jeg videregivet, hvor fascinerende jeg har fundet TSP, siden jeg hørte om MST-algoritmen første gang.

Jeg kunne godt have tænkt mig at have lavet flere forsøg med både simuleret udglødning og den genetiske algoritme, der angriber problemet på nye, spændende måder. Jeg kunne også godt have tænkt mig at bruge mere tid på de mere teoretiske aspekter.

Alt i alt tror jeg dog jeg her giver et passende indblik i et meget stort emne, der stadig optager mange forskere rundt omkring, og som jeg selv har været glad for at lære mere om.

Odense, 25/7-'94.

Lise Møller

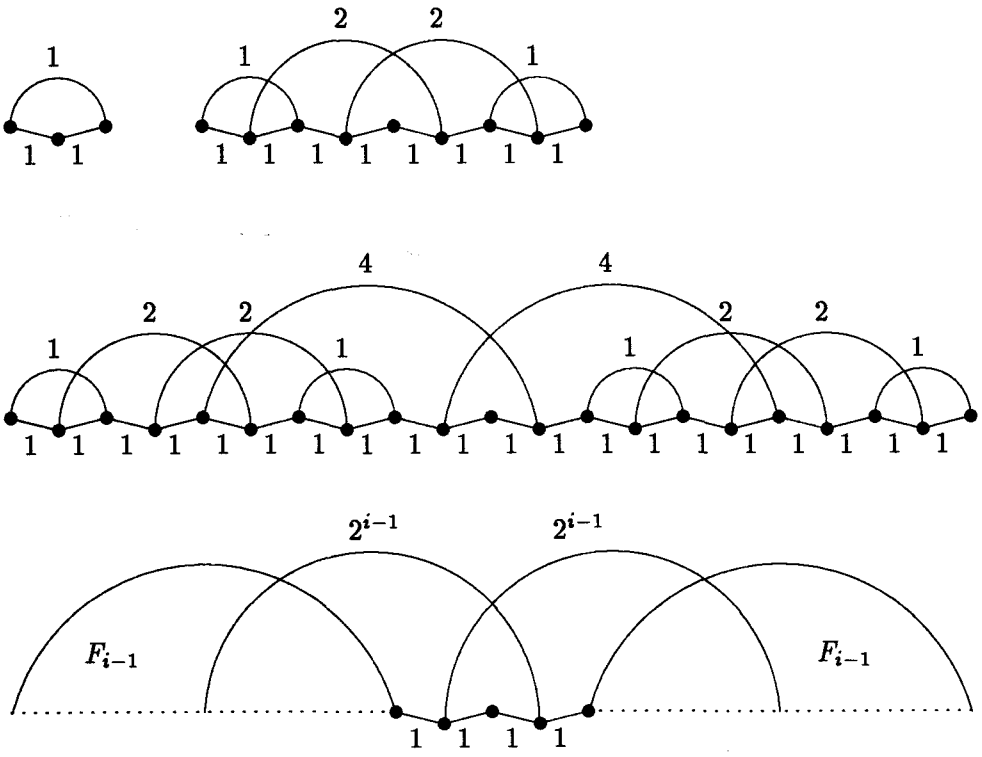
Swindling Yankee (1839)

Appendix A

Bevis for nærmeste nabo reglen.

Som nævnt kan det bevises, at nærmeste nabo reglen ikke er specielt god, bare fordi trekantuligheden gælder. Her følger beviset, lidt udvidet i forhold til *John 85b*.

Sætning 30 *For alle $r > 1$ eksisterer der en instans af TSP med (vilkårligt stort) n byer, hvor trekantuligheden er opfyldt, og $NN(I) \geq r\ OPT(I)$, dvs, nærmeste nabo reglen kan komme med en tur, der er r gange længere end den optimale.*



Figur A.1: En graf, hvor nærmeste nabo reglen ikke er så god.

Bevis: Vi benytter en konstruktion af grafer, hvor en masse kanter ikke fremgår af tegningerne, men da antages at være lige så lange som den korteste kendte anden rute, og hvor

graf nr.i giver nærmeste nabo reglen mulighed for at vælge en tur, der er $(i+2)/6 = r$ gange længere end den optimale.

På figur A.1 ses graferne F_1, F_2, F_3 og den generelle F_i. Hver graf konstrueres af 2 kopier af grafen med en lavere nummer, 4 kanter af længde 1, og 2 kanter af længde 2^{i-1}.

Påstand 1: trekantuligheden overholdes af alle påtegnede kanter, også de nye, der dukker op i hvert trin. Dette undlader jeg at vise.

Påstand 2: i F_i er der $3(2^i - 1)$ byer.

I F_1 er der 3 byer. $3 = 3(2^1 - 1)$.

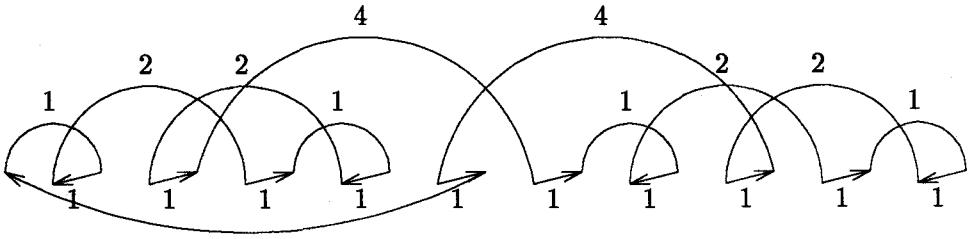
I F_2 er der 2 eksemplarer af F_1, foruden 3 nye byer, dermed i alt 9 byer. $9 = 3(2^2 - 1)$.

I F_i er der 2 eksemplarer af F_{i-1}, foruden 3 nye byer, dermed i alt $2\,3(2^{i-1}-1)+3 = 3(2^i-1)$.

Påstand 3: $OPT(I) \leq 6\ 2^i - 8$.

Det er muligt at gå igennem F_i ved at gå gennem alle byerne fra venstre mod højre, og så tage kanten mellem de to yderste punkter tilbage til startpunktet. Der er $3(2^i - 1) = 3\ 2^i - 3$ byer, og dermed $3\ 2^i - 4$ kanter af længde 1 på vej mod højre. Den sidste kant er i hvert fald ikke længere end alle de andre. Dermed er der en mulig tur af længde $2(3\ 2^i - 4) = 6\ 2^i - 8$.

At det er muligt at gå en anden tur ses af figur A.2.



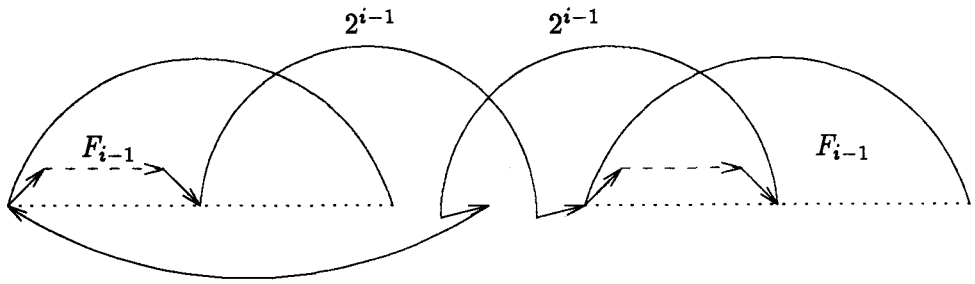
Figur A.2: En lang tur gennem F_3.

På denne måde bruges alle de buede kanter.

Påstand 4: vejen fra venstre yderpunkt til midteste punkt har i F_i længde $(i+1)2^i - 2$, og kanten fra midten tilbage har længden $2^i - 1$, i alt en tur på $(i+2)2^i - 3$.

Denne tur har for F_1 længde 3 (det har den eneste mulige tur), og $3 = (1+2)2^1 - 3$.

For F_2 er turen en gennemgang af de 2 F_1, hver af længde 2, 2 kanter af længde 1, de to nye kanter af længde 2, og den ekstra kant til sidst, af længde 3 (2 rette kanter af længde 1 og en buet kant af længde 1 går samme vej). Denne tur har altså længde 13, og $13 = (2+2)2^2 - 3$.

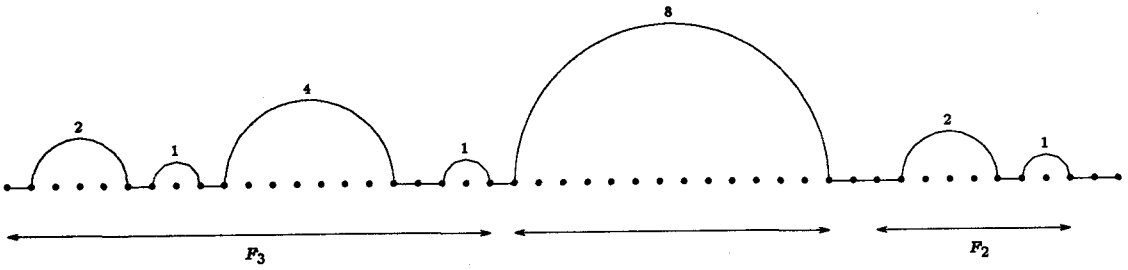


Figur A.3: En lang tur gennem F_i.

For F_i på figur A.3 er turen en gennemgang af 2 F_{i-1}, hver af længde $i\ 2^{i-1} - 2$, 2 nye kanter af længde 2^{i-1}, 2 små kanter af længde 1, og så den sidste kant af længde $2^i - 1$.

$2(i\ 2^{i-1} - 2) + 2^{i-1} + 2 = 2((i+1)2^{i-1} - 1) = (i+1)2^i - 2$.

Dvs. den første del af påstanden (den om vejen ud til midtpunktet) er opfyldt.



Figur A.4: En mulig vej fra midtpunkt til venstre punkt.

Af figur A.4 ses en mulig vej fra F_5's midterste punkt (helt til højre på tegningen) til venstre side at grafen, en vej der benytter sig af en tilsvarende kort vej gennem en hel F_2 og en hel F_3, og en bue af længde 8, eller 2^{5-2}, og i alt udgør en hel vej gennem F_4, og lidt til.

For en vilkårlig F_i er vejen fra midtpunkt tilbage til startpunkt (gennem F_{i-1}) generelt en vej gennem F_{i-2} og F_{i-3}, foruden en bue af længde 2^{i-2}, og tre kanter af længde 1 indenfor F_{i-1}, og 2 kanter af længde 1 hen til F_{i-1}.

Induktionsantagelsen udtaler, at vejen tilbage har længde $2^i - 1$, heraf er de 2 altså udenfor F_{i-1}, så vejen derinde har længde $2^i - 3$. Anvendes dette fås, at hvis induktionsantagelsen holder under i, hvolder den også for i:

$(2^{i-1} - 3) + 1 + 2^{i-2} + 2 + (2^{i-2} - 3) + 2 = 2^i - 1$

Påstand 5: Da $NN(I) = (i+2)2^i - 3$, er $NN(I)/OPT(I) = (i+2)/6$.

$\frac{NN(I)}{OPT(I)} = \frac{(i+2)2^i-3}{6\ 2^i-8} \longrightarrow_{i\to\infty} \frac{(i+2)2^i}{6\ 2^i} = \frac{i+2}{6}$

QED

Appendix B

Mere om distributionsmatricer.

Som supplering til stoffet om distributionsmatricer, følger her yderligere et par beviser. Begge to har jeg selv konstrueret, det første med hjælp fra *Gilm 85*.

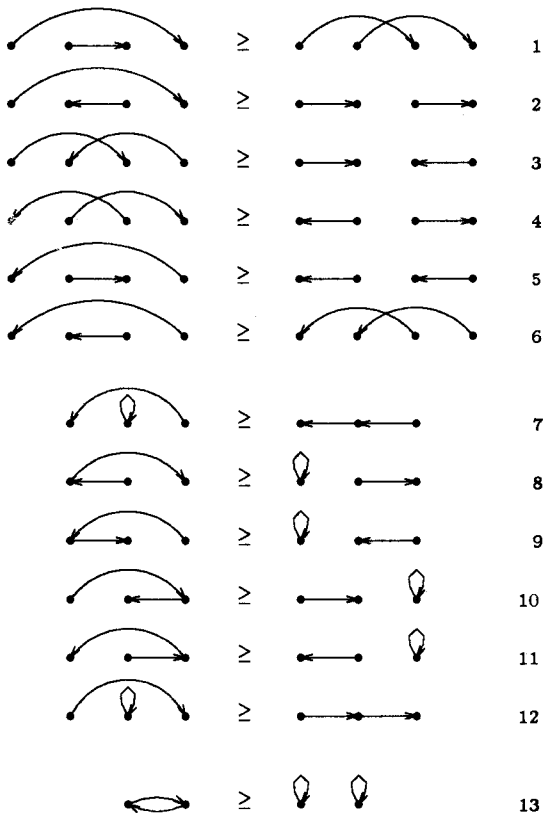
Lemma 31 $i < i', j < j' \Rightarrow c_{ij'} + c_{i'j} \geq c_{ij} + c_{i'j'}$.

Bevis:

$$
\begin{aligned}
c_{ij'} + c_{i'j} &= \sum_{k=i}^{n}\sum_{l=1}^{j'} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl} \\
&= \left(\sum_{k=i}^{i'-1}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=j+1}^{j'} d_{kl} + \sum_{k=i}^{i'-1}\sum_{l=j+1}^{j'} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl}\right) + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl} \\
&= \sum_{k=i}^{i'-1}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=j+1}^{j'} d_{kl} + \sum_{k=i}^{i'-1}\sum_{l=j+1}^{j'} d_{kl} \\
&\geq \left(\sum_{k=i}^{i'-1}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl}\right) + \left(\sum_{k=i'}^{n}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=j+1}^{j'} d_{kl}\right) \\
&= \sum_{k=i}^{n}\sum_{l=1}^{j} d_{kl} + \sum_{k=i'}^{n}\sum_{l=1}^{j'} d_{kl} \\
&= c_{ij} + c_{i'j'}
\end{aligned}
$$

QED

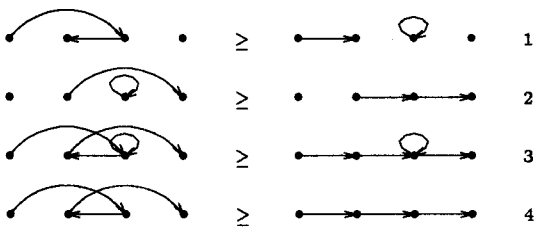
Den oprindelige bibetingelse, og dens følge lemma 31, ser enkel nok ud, men er i forhold til de følgende Demidenko-betingelser meget omfattende. For at kunne sammenligne de to, har jeg i figur B.1 tegnet alle muligheder indenfor distributions-matricernes betingelser.



Figur B.1: Distributions-betingelserne.

Som det ses er der 6 symmetriske tilfælde, hvor de 4 involverede punkter er forskellige, 6 hvor 2 af punkterne af ens, og 1 hvor punkterne er ens to og to. Bemærk at linje 3 og 4 senere vil komme igen som B3 og B4.

Som det fremgår af figur B.2 kan disse kombineres til A1.



Figur B.2: Bevis for A1.

I linje 1 anvendes linje 10 fra figur B.1, i linje 2 anvendes linje 12. I tredje linje er summen, og i linje 4 den reducerede sum, netop A1. Heraf følger denne sætning.

Sætning 32 *Distributionsmatricer er også Demidenkomatricer.*

Appendix C

Sætning om Demidenko-matricer.

Som før nævnt er den primære kilde til denne sætning, og det øvrige indhold i dette appendiks, *Demi 79*. For selve hovedsætningen har jeg dog også støttet mig til *Gilm 85*, der trods alt var på engelsk. Fremstillingsformen er en blanding af disse 2 kilders. Dette er så vidt jeg ved første gang, dette stof findes på dansk, og første gang overhovedet beviset er så fyldigt.

Sætning 33 *For enhver tur i en Demidenko–matrix, findes der en pyramide–tur der ikke er længere.*

Bevis: Ifølge lemma 18 er en tur forhindret i at være en pyramide–tur, hvis vi kan finde et dårligt par. Derfor vil vi angribe alle ikke–pyramide–ture ved at fjerne deres dårlige par, uden at øge længden af turen, og så alle dårlige par til sidst er fjernet. Da dette må gøres på forskellig vis i forskellige situationer, deles der op i en del specialtilfælde.

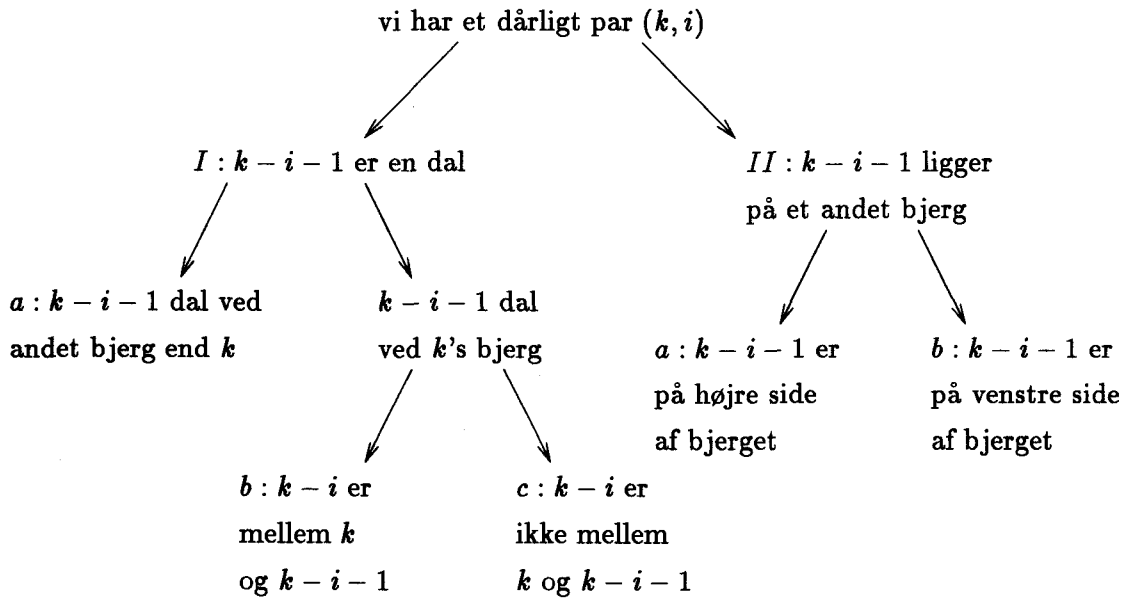
Der gøres brug af elementære transformationer, der fjerner det dårlige par, uden at øge turens længde. Af et antal elementære transformationer laves en sammensat transformation, der foruden at fjerne det dårlige par (k,i) ikke skaber et nyt dårligt par (k',i'), hvor $k'-i' \geq k-i$. Ved gentagne gange at fjerne det dårlige par med højest $k-i$ kan vi fjerne alle dårlige par.

Da et par er dårligt fordi vi har alt opfyldt i lemma 18, undtagen $k-i-1 \in T(\tau)$ og $k-i-1 \in B(\tau,k)$, må der gælde $I : k-i-1 \in D(\tau)$ eller $II : k-i-1 \in B(\tau,l)$, hvor $l > k$. (Da alle punkter mellem $k-i$ og $k-1$ er i k's bjerg, kan $k-i-1$ ikke have en af disse som tinde. Men l skal jo være større end $k-i-1$, og dermed også større end k.)

Disse tilfælde kan igen spaltes op. Hvis I er opfyldt har vi enten $a : k-i-1$ er dal ved et andet bjerg end k, eller $k-i-1$ er dal ved k's bjerg, der igen deler op i to tilfælde: $b : k-i$ ligger på τ mellem $k-i-1$ og k, $c : k-i$ ligger på den anden side af bjerget.

Tilsvarende kan II spaltes op: $a : k-i-1$ ligger på højre side af l's bjerg, eller $b : k-i-1$ ligger på venstre side af l's bjerg.

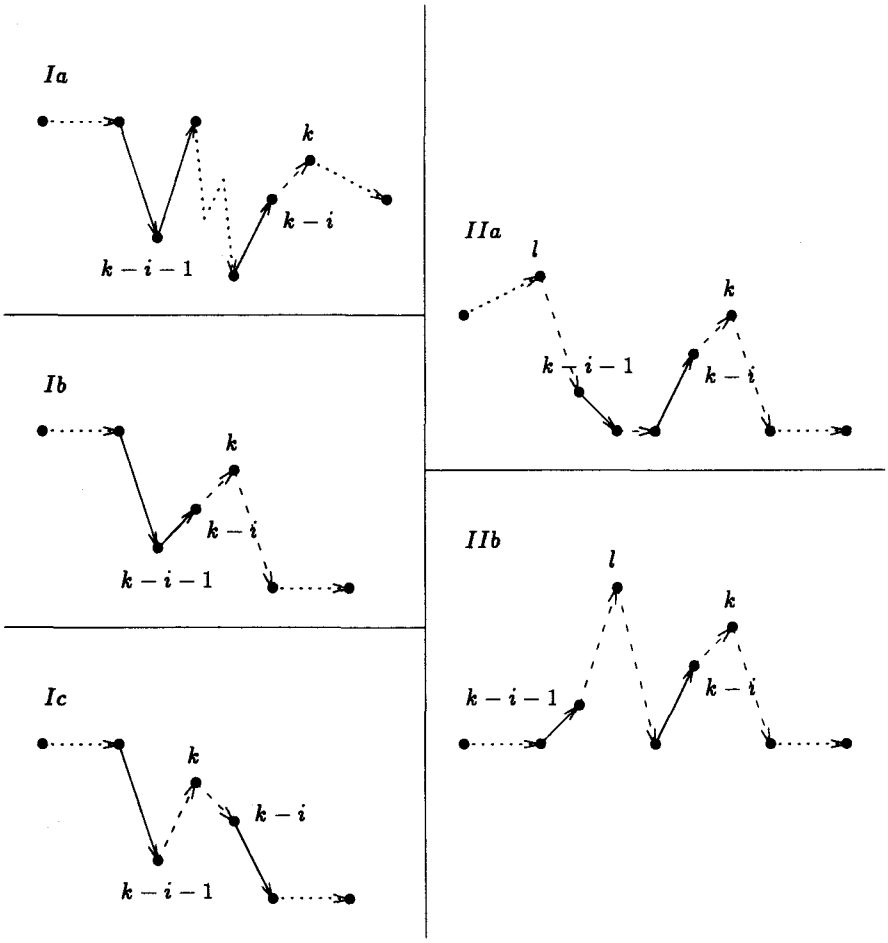
Lad os nu se på et skema over disse tilfælde (figur 2.38), der hver for sig senere vil blive vist.



Figur C.1: Skema over specialtilfælde i beviset.

Alle tilfældene findes desuden i 2 versioner, der er hinandens spejlbilleder. For Ia betyder dette, at vi ser på tilfældet hvor man fra $k - i - 1$ kommer til $k - i$ før k. For Ib og Ic betyder dette, at man fra 1 kommer til $k - i - 1$ før k. For II betyder det at man fra 1 kommer først til $k - i - 1$, så til $k - i$ og så til k. Nogle af tilfældene kan yderligere underinddeles, dette gøres efter behov.

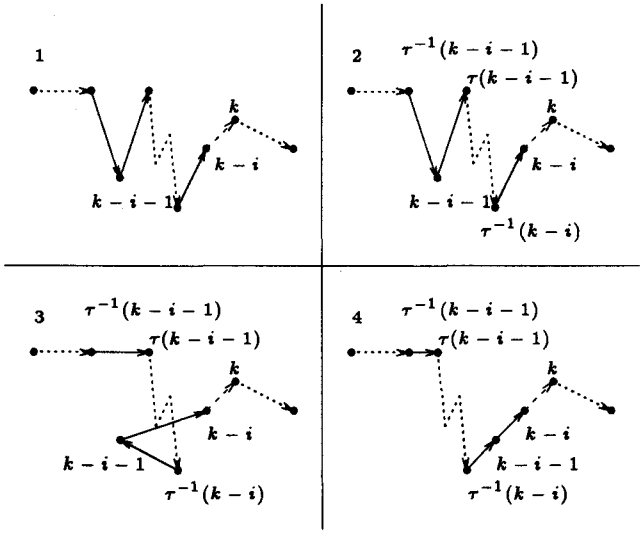
For at tydeliggøre dette, se da figur C.2, hvor hver situation nævnt i figur C.1 er tegnet, i den version beviset bruger. En pil er en kant. En stiplet pil er et antal kanter, måske 0, der forbinder de angivne punkter. En prikket kant er det samme, men typisk er vi enten ikke interesseret i disse kanter, eller ved ikke ret meget om hvordan de forløber. Hvis det vides, kan det ses af punkternes højde på tegningen, hvilke punkter der har højest nummer, fx. vil $k - i$ kunne ses liggende over $k - i - 1$. På tegningerne i figur C.2 er der angivet mange flere punkter, end der er navngivet. Dette er fordi, de senere vil blive navngivet, for at beviset kan gennemføres, men deres navne er ikke nødvendige for at overskue situationen.



Figur C.2: Tegninger til enkelttilfælde i beviset.

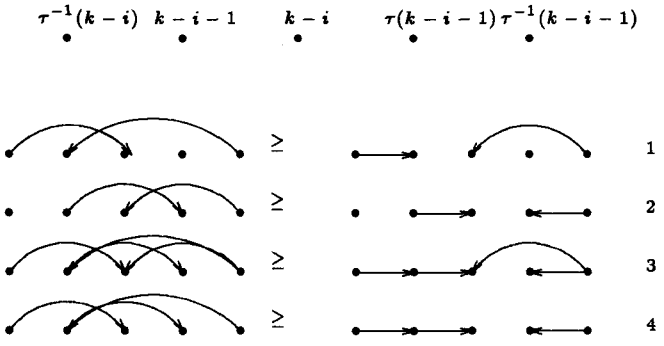
Måden hvert bevis gennemføres på er ved hjælp af en tegning at vise, hvordan transformationen virker ("før og efter"), og så bevise at denne transformation opfylder de fastsatte krav, og at man kan bygge en sammensat transformation vha. den.

Således vises på figur C.3 transformation Ia, som de andre navngivet efter den situation, den forsøger at reparere.



Figur C.3: Skitse af transformation Ia.

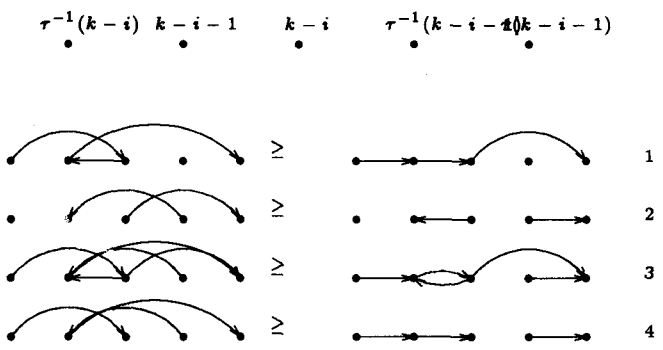
Transformation Ia fjerner altså det dårlige par, der fremgår af første billede i figur C.3. Denne situation ses mere detaljeret i andet billede. Transformationen er da at udskifte nogle kanter med nogle andre, så situationen i tredje billede opstår. For at kunne vurdere denne situation bedre, ses den "rettet ud" i fjerde billede. Beviset må deles i to, fordi det ikke vides om 1: $\tau^{-1}(k-i-1) < \tau(k-i-1)$ eller 2: $\tau^{-1}(k-i-1) > \tau(k-i-1)$. 1 bevises ved hjælp af figur C.4.



Figur C.4: Bevis til transformation $Ia1$.

I linje 1 ses et eksempel på A1, i linje 2 ses et eksempel på B4, i linje 3 ses summen af de ovenstående linjer, og i linje 4 er kanter, der er på begge sider af ulighedstegnet, fjernet. Resultatet er de kanter, der er forskellen på situation 2 og 3 i figur C.3.

Hvis vi er i situation 2 bruges figur C.5.



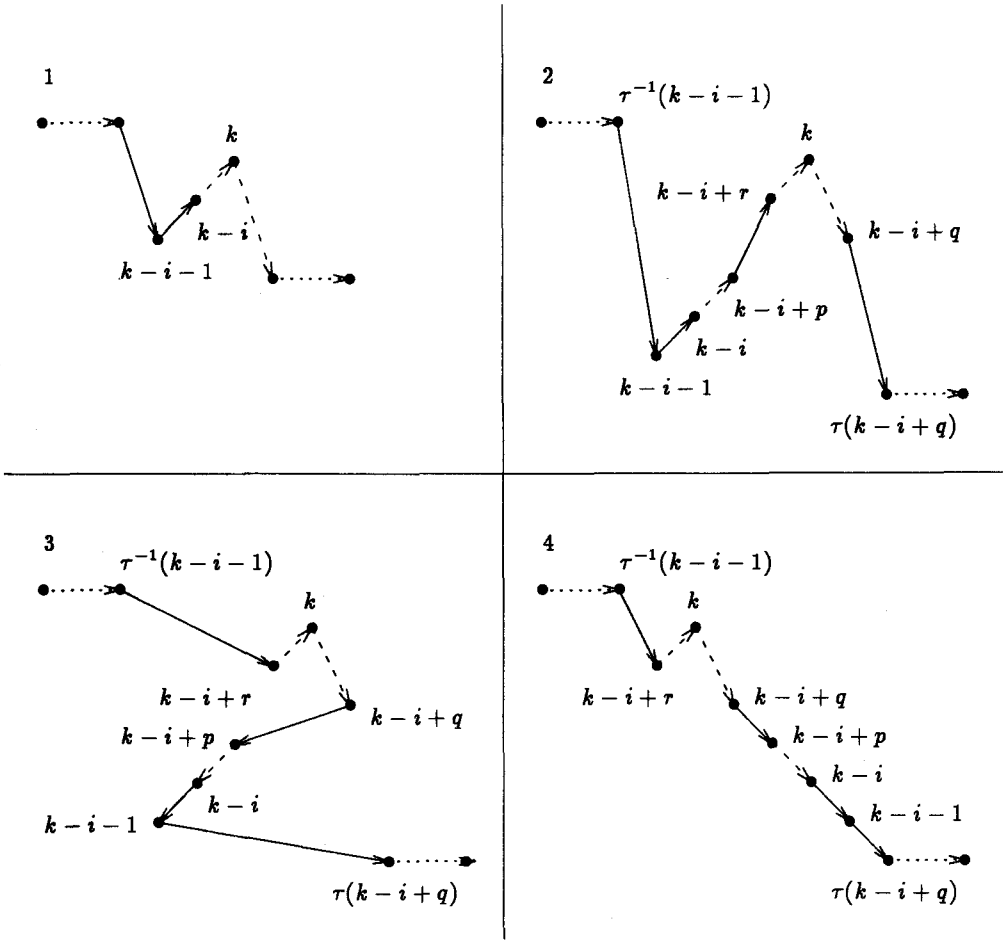
Figur C.5: Bevis til transformation $Ia2$.

I linje 1 og 2 ses eksempler på B3, i linje 3 ses summen af de ovenstående linjer, og i linje 4 er kanter, der er på begge sider af ulighedstegnet, fjernet. Resultatet er som før det ønskede.

(Dette udgør en tilføjelse i forhold til *Gilm 85* og en rettelse i forhold til *Demi 79*, der siger B4 i stedet for B3, begge gange.)

Hermed er transformationen for tilfælde Ia fundet, og det er bevist at turen ikke bliver længere. Det andet krav var, at det dårlige par skulle forsvinde, og ikke alene sker dette, faktisk kan denne transformation opfylde den sammensatte transformations krav: der bliver ikke nye dårlige par med højere $k' - i'$. Derfor er dette også den sammensatte transformation, og efter denne er man et skridt nærmere en pyramide-tur.

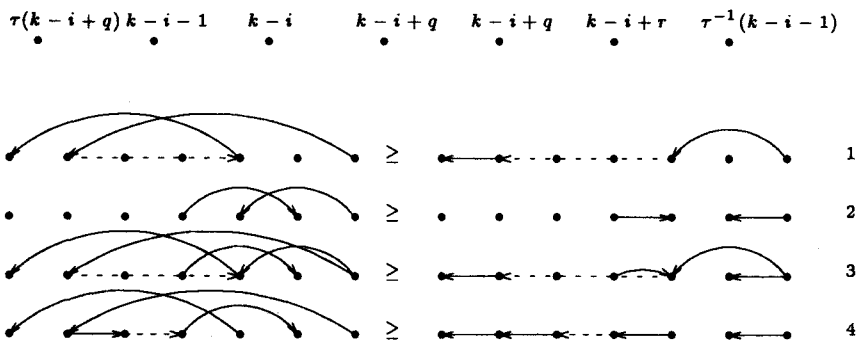
Herefter behandles tilfælde Ib, se figur C.6.



Figur C.6: Skitse af transformation Ib.

Her skal det nævnes, at punktet $k-i+q$ er det mindste punkt på højre side af $B(\tau,k)$, så $k-i+q$ ligger højere end $k-i$ på bjergets venstre side. $k-i+p$ og $k-i+r$ er punkterne på venstre side af bjerget, umiddelbart under og over $k-i+q$. Bemærk, at der kan ligge punkter mellem $k-i+q$ og $k-i+r$, så $r-p \geq 2$. $p<q<r$. $p+1=q$.

Beviset for, at denne transformation kan bruges, ses primært af figur C.7.

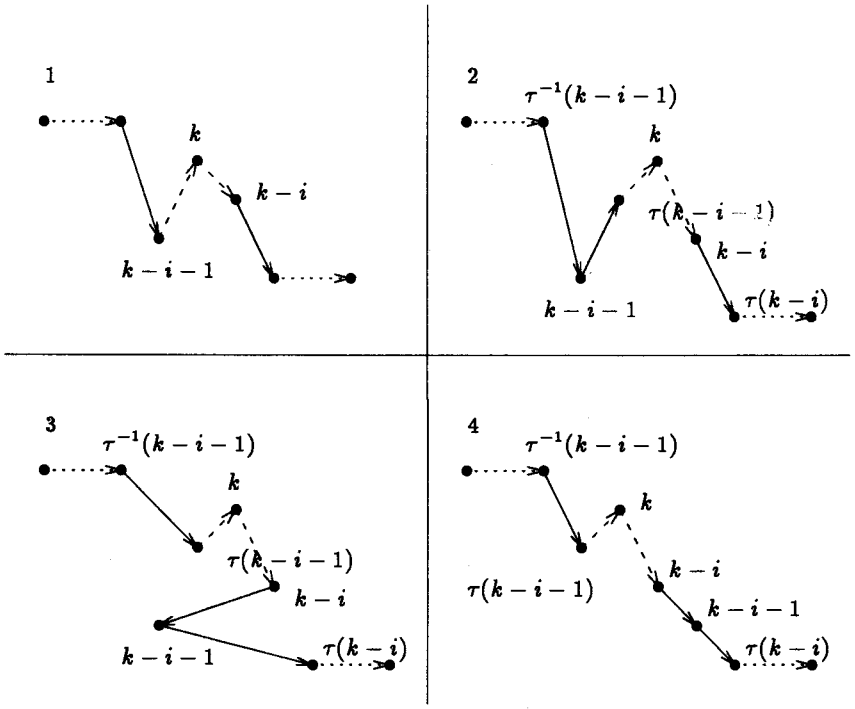


Figur C.7: Bevis til transformation *Ib*.

I linje 1 (af figur C.7) ses et eksempel på A4, hvor den stiplede kant som før betyder, at dette er en række kanter, der gennemløber alle punkter mellem endepunkterne. I linje 2 er et eksempel på B3. I linje 3 ses summen, og linje 4 er ens kanter fjernet. Resultatet er det ønskede, dvs. de kanter der er forskellen på før og efter transformationen.

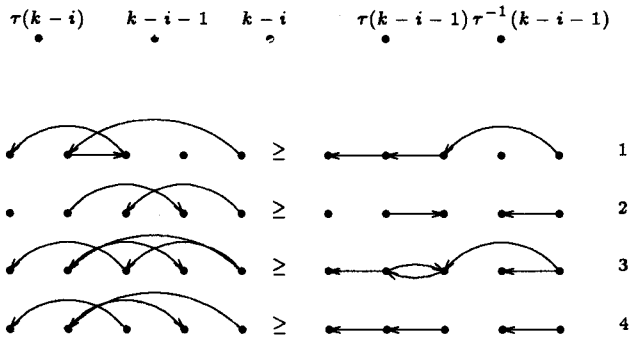
Som det ses, er dette muligvis ikke en god nok transformation, fordi $k-i+r$ hvis forskellig fra k er en dal, og dermed giver anledning til et nyt dårligt par, nemlig $(k, i-r-1)$ (rettet i forhold til *Gilm 85*) fordi $k-(i-r-1)-1 = k-i+r$ er det nye problem. Dvs. $k-i+r+1 \geq k-i$, og dermed er der dannet et nyt "højere" problem. Imidlertid er situationen stadig den samme, dvs. efter transformationen er vi igen i *Ib* eller *Ic*. Yderligere er venstre side af bjerget blevet kortere.

Her følger en tegning af transformation *Ic*, figur C.8.



Figur C.8: Skitse af transformation Ic.

At denne transformation virker, bevises bl.a. vha. figur C.9.



Figur C.9: Bevis til transformation Ic.

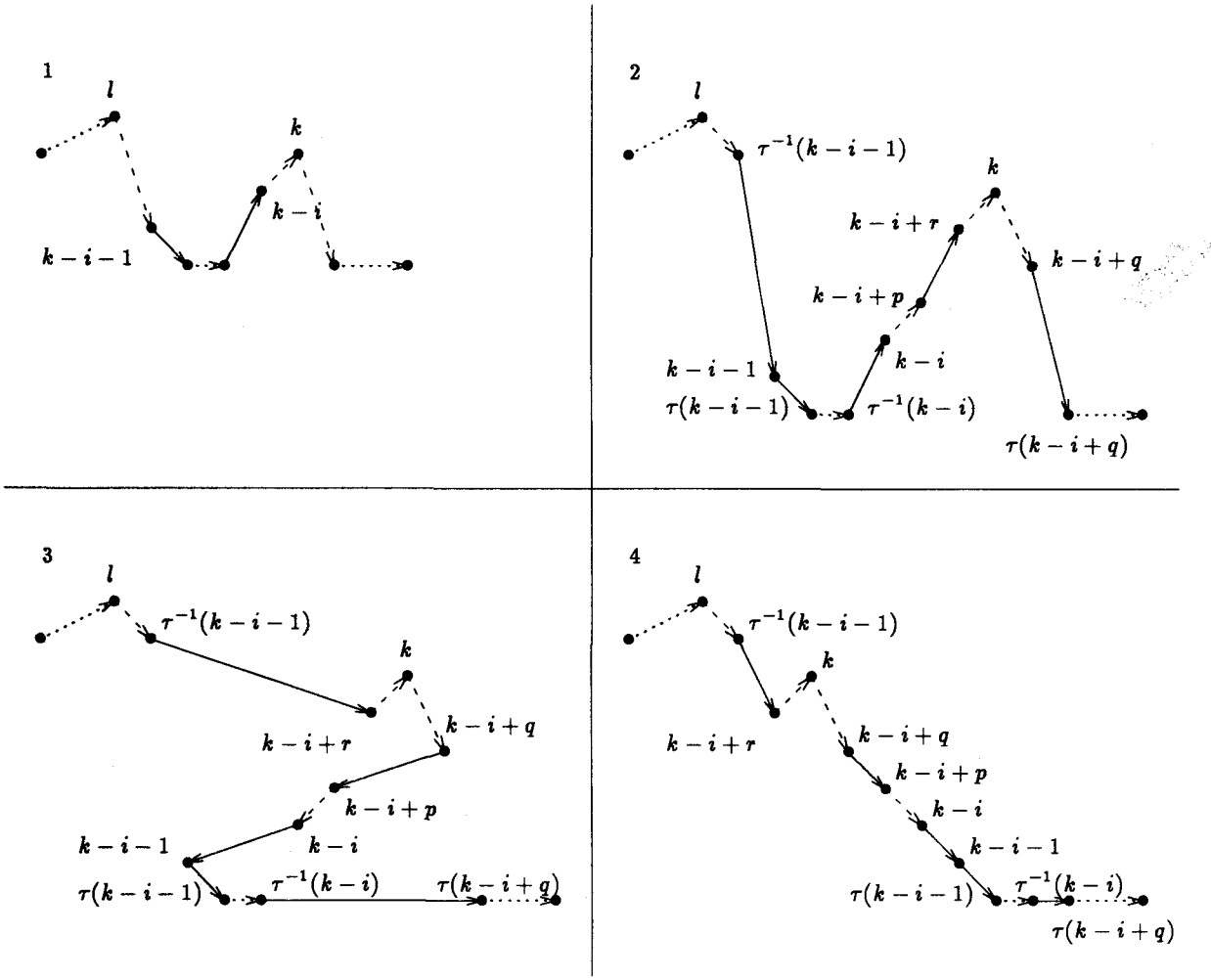
I linje 1 ses et eksempel på A4, i linje 2 B3, i linje 3 summen, i linje 4 er dublet-kanter fjernet. Resultatet er det ønskede.

(Dette er en tilføjelse i forhold til begge kilder.)

Faktisk er dette et spejlbillede af transformation $Ia1$.

Da transformationen for Ib gør noget lignende, dvs. muligvis skaber et nyt dårligt par, men gør venstre side af bjerget kortere, vil disse 2 transformationer tilsammen kunne skaffe dette dårlige par helt af vejen, så k til sidst ikke er en tinde mere, men en del af bjerget til venstre for $B(\tau, k)$.

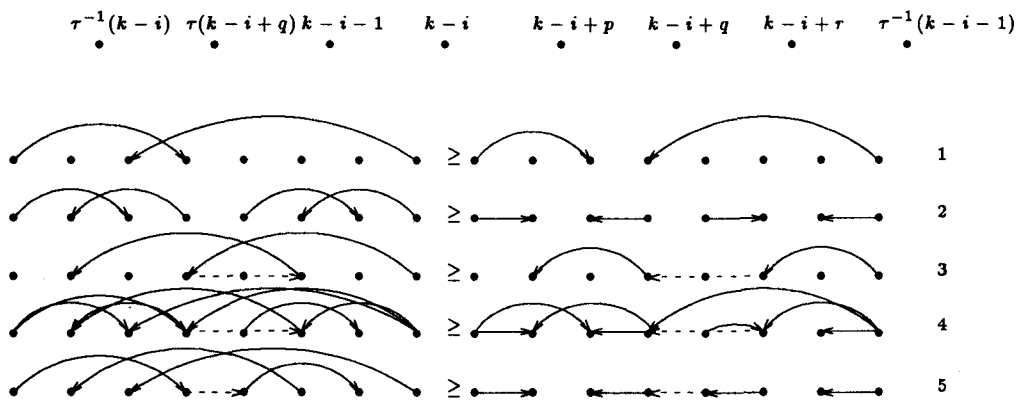
Herefter behandles transformation IIa, som ses på figur C.10.



Figur C.10: Skitse af transformation IIa.

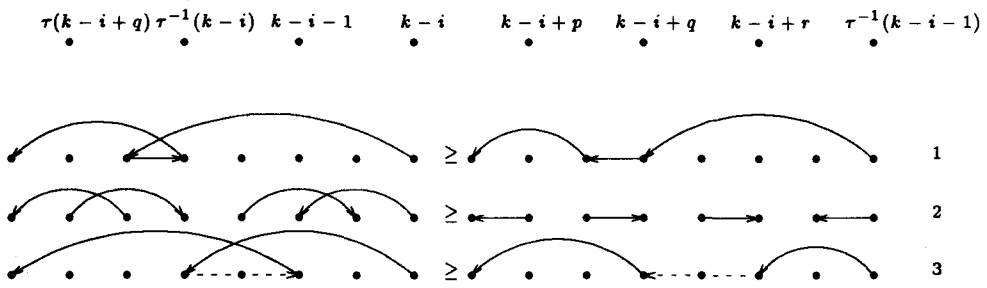
Som det måske fremgår af figur C.10 er der mulighed for, at en ny tinde opstår, enten i $\tau^{-1}(k-i)$ eller i $\tau(k-i+q)$.

Beviset for transformationen hvis $\tau^{-1}(k-i) < \tau(k-i+q)$ fremgår af figur C.11.

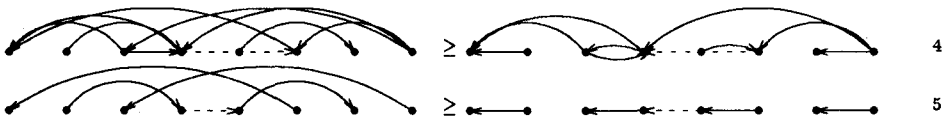


Figur C.11: Bevis til transformation $IIa1$.

I linje 1 og 2 ses tre eksempler på B3, og i linje 3 et eksempel på A4. I linje 4 ses summen, og i linje 5 er ens kanter elimineret. Resultatet er det ønskede. Herefter ses beviset for $\tau^{-1}(k-i) > \tau(k-i+q)$, på figur C.12 og figur C.13.



Figur C.12: Bevis til transformation $IIa2$.

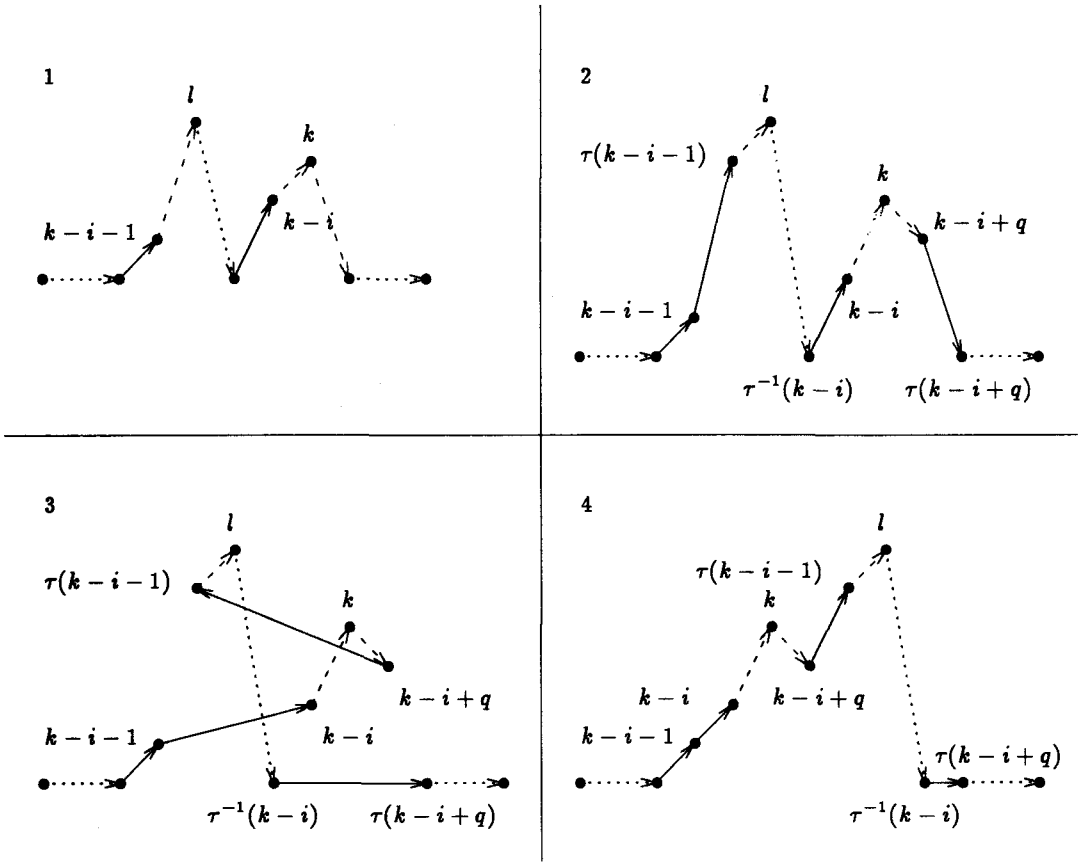


Figur C.13: Bevis til transformation $IIa2$.

I linje 1 ses A2 i brug, i linje 2 B3 og B4, og i linje 3 A4. I linje 4 ses summen, og i linje 5 er ens kanter elimineret. Resultatet er igen det ønskede.

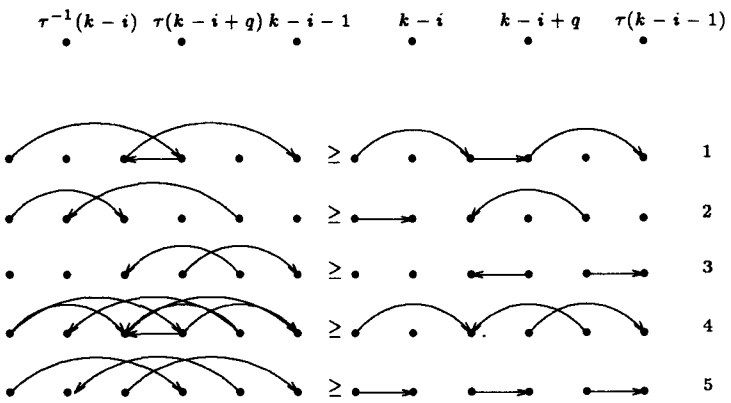
Denne transformation bruges når $k - i - 1$ er skråning i et forkert bjerg, og resultatet bliver måske et nyt dårligt par, af typen Ib eller Ic. Derefter vil en række af disse transformationer klare problemet. Desuden er venstre side af bjerget allerede ved denne første transformation blevet kortere, og muligvis forsvundet.

Derefter følger transformation IIb, se figur C.14.



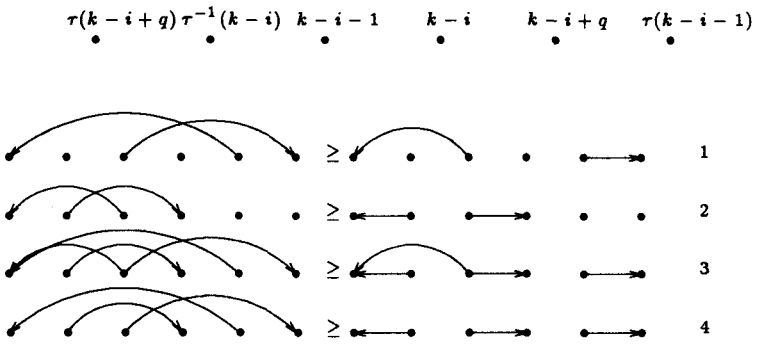
Figur C.14: Transformation IIb.

At dette er en transformation for $\tau^{-1}(k-i) < \tau(k-i+q)$ bevises i figur C.15.



Figur C.15: Bevis til transformation $IIb1$.

I linje 1 bruges A3, i linje 2 B3 og i linje 3 B4.
(Tilføjelse i forhold til begge kilder.)
For $\tau^{-1}(k-i) > \tau(k-i+q)$ følger beviset af figur C.16.



Figur C.16: Bevis til transformation $IIb2$.

I linje 1 og 2 ses eksempler på B4.
(Tilføjelse i forhold til begge kilder.)
Denne transformation giver også kun endnu et dårligt par, af type som et spejlbillede til de betragtede Ib eller Ic. Derfor kan disses transformation klare resten af problemet, og igen, er den (nu) højre side af bjerget blevet kortere.

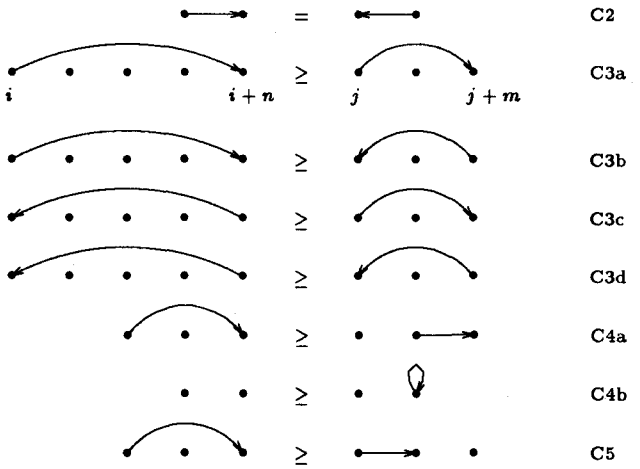
I det ovenstående fremgår det ikke, hvad der sker, hvis visse punkter overlapper, således at situationen er simplere end vist. I de tilfælde jeg har undersøgt vil det være nok at simplificere transformationen, og dermed dens bevis, simpelthen ved at smide kanter væk, der går mellem sammenfaldende punkter.
QED

På det tidspunkt, denne sætning kom frem, var der kendt andre klasser af problemer, for hvilke pyramideture var nok at gennemsøge. Artiklen indeholder derfor et bevis for, at disse andre klasser er specialtilfælde af Demidenko-matricer.

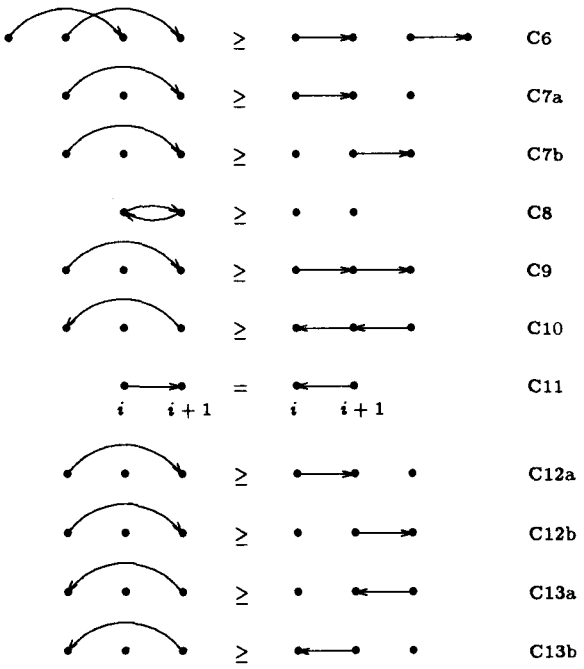
Definition 8 *Lad der være 8 klasser af matricer, nummereret fra 0 til 7.*

M_0: *Demidenko-matricer.*
M_1: $a_{ij} = a_i b_j, a_i \leq a_{i+1}, b_j \geq b_{j+1}$ *(C1)*
M_2: $a_{ij} = a_{ji}$ *(C2)*
$|i-j| < |k-l| \Rightarrow a_{ij} \leq a_{kl}$ *(C3)*
M_3: *C2*
$i < j \leq k \Rightarrow a_{ik} \geq a_{jk}$ *(C4)*
$i < j \leq k \Rightarrow a_{ik} \geq a_{ij}$ *(C5)*
M_4: *C2*
$i < j < k < l \Rightarrow a_{ik} + a_{jl} \geq a_{ij} + a_{kl}$ *(C6)*
M_5: *C2*
$i < j < k \Rightarrow a_{ik} \geq max\{a_{ij}, a_{jk}\}$ *(C7)*
M_6: $a_{ij} + a_{ji} \geq 0$ *(C8)*
$i < j < k \Rightarrow a_{ik} \geq a_{ij} + a_{jk}$ *(C9)*
$i < j < k \Rightarrow a_{ki} \geq a_{ji} + a_{kj}$ *(C10)*
M_7: $a_{i,i+1} = a_{i+1,i}$ *(C11)*
$i < j < k \Rightarrow a_{ik} \geq max\{a_{ij}, a_{jk}\}$ *(C12)*
$i < j < k \Rightarrow a_{ki} \geq max\{a_{ji}, a_{kj}\}$ *(C13)*

Som før er det lige så nemt (eller nemmere) at se hvad der foregår på nogle tegninger, se figur C.17 og figur C.18. Nogle af reglerne spalter op i flere tilfælde, og til disse er stort set kun at bemærke, at for C3 er $n > m$.



Figur C.17: Matriceklassernes egenskaber.



Figur C.18: Matriceklassernes egenskaber.

Sætning 34 $\bigcup_{i=1}^{7} M_i \subseteq M_0$.

Bevis: Hver for sig bevises $M_1 \subseteq M_0, M_4 \subseteq M_0, M_2 \subseteq M_4, M_3 \subseteq M_5, M_5 \subseteq M_4, M_6 \subseteq M_0$ og $M_7 \subseteq M_0$. Det checkes for $M_i \subseteq M_0$ om A1 og B1 gælder, mens A2 og B2 springes over, beviset for disse er symmetrisk.

$M_1 \subseteq M_0$: Det checkes om C1 $\Rightarrow$ A1, dvs. kan A1 vises at være sand. Start med at skrive A1 op.

$a_{i,j+1} + a_{j+1,j} + a_{jk} \geq a_{ij} + a_{j,j+1} + a_{j+1,k}$
$\Leftrightarrow a_{i,j+1} + a_{j+1,j} + a_{jk} - a_{ij} - a_{j,j+1} - a_{j+1,k} \geq 0$
$\Leftrightarrow a_i b_{j+1} + a_{j+1} b_j + a_j b_k - a_i b_j - a_j b_{j+1} - a_{j+1} b_k \geq 0$
$\Leftrightarrow a_{j+1}(b_j - b_k) + a_j(b_k - b_{j+1}) + a_i(b_{j+1} - b_j) \geq 0$
$\Leftrightarrow a_{j+1}(b_j - b_{j+1} + b_{j+1} - b_k) + a_j(b_k - b_{j+1}) + a_i(b_{j+1} - b_j) \geq 0$
$\Leftrightarrow a_{j+1}(b_j - b_{j+1}) + a_{j+1}(b_{j+1} - b_k) + a_j(b_k - b_{j+1}) + a_i(b_{j+1} - b_j) \geq 0$
$\Leftrightarrow (a_{j+1} - a_j)(b_{j+1} - b_k) + (a_{j+1} - a_i)(b_j - b_{j+1}) \geq 0$

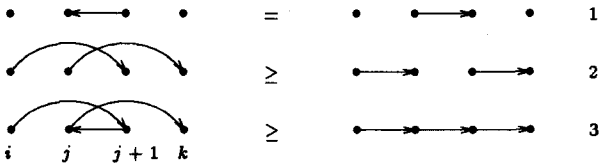
Pga. C1 er det opfyldt, at $a_i \leq a_j \leq a_{j+1} \leq a_k$ og $b_i \geq b_j \geq b_{j+1} \geq b_k$, og deraf følger, at alle parenteserne er ikke-negative, og deraf følger igen, at uligheden er opfyldt. Dvs. C1 $\Rightarrow$ A1.

Det checkes om C1 $\Rightarrow$ B1. Skriv B1 op.

$a_{i,j+1} + a_{kj} \geq a_{ij} + a_{k,j+1}$
$\Leftrightarrow a_{i,j+1} + a_{kj} - a_{ij} - a_{k,j+1}$
$\Leftrightarrow a_i b_{j+1} + a_k b_j - a_i b_j - a_k b_{j+1}$
$\Leftrightarrow a_i(b_{j+1} - b_j) + a_k(b_j - b_{j+1})$
$\Leftrightarrow (a_k - a_i)(b_j - b_{j+1})$

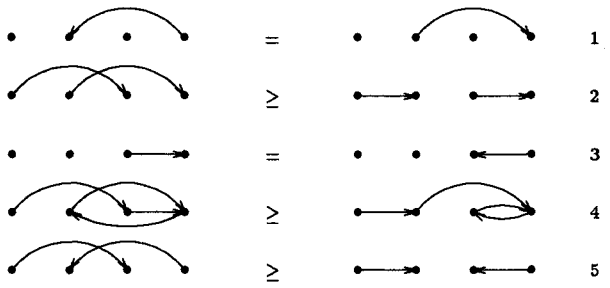
Pga. C1 er $a_i \leq a_k$ og $b_j \geq b_{j+1}$, dermed er begge parenteser ikke-negative, og uligheden er opfyldt. Dvs. C1 $\Rightarrow$ B1.

$M_4 \subseteq M_0$: (Tilføjelse i forhold til kilden.) At A1 gælder fremgår af figur C.19.



Figur C.19: Bevis for $M_4 \subseteq M_0$.

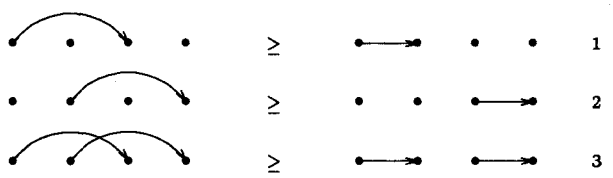
I første linje ses C2, i anden linje C6, og i tredje linje summen, der netop er A1.
At B1 gælder fremgår af figur C.20.



Figur C.20: Bevis for $M_4 \subseteq M_0$.

I første og tredje linje bruges C2, i anden linje bruges C6. Den reducerede sum er netop B1.

$M_2 \subseteq M_4$: (Tilføjelse i forhold til kilden.) C2 medfører C2. C6 følger af figur C.21.

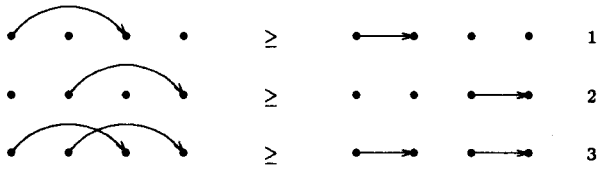


Figur C.21: Bevis for $M_2 \subseteq M_4$.

I første og anden linje bruges C3a. Summen i tredje linje er C6. Dermed følger at $M_2 \subseteq M_4$.

$M_3 \subseteq M_5$: (Tilføjelse i forhold til kilden.) C2 medfører C2. C4a medfører C7b. C5 medfører C7a. Dermed følger alle krav i M_5 af M_3's.

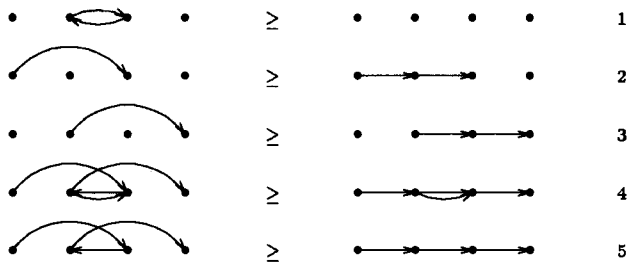
$M_5 \subseteq M_4$: (Tilføjelse i forhold til kilden.) C2 medfører C2. C6 følger af figur C.22.



Figur C.22: Bevis for $M_5 \subseteq M_4$.

I første linje bruges C7a, i anden linje C7b. Summen er C6. Deraf følger $M_5 \subseteq M_4$.

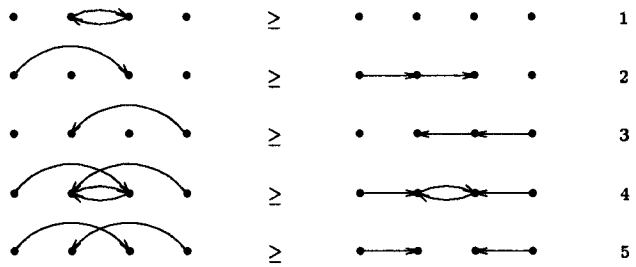
$M_6 \subseteq M_0$: Det skal bevises at A1 gælder i M_6. Se figur C.23.



Figur C.23: Bevis for $M_6 \subseteq M_0$.

I første linje bruges C8, i anden og tredje linje C9. I fjerde linje er summen og i femte linje den reducerede sum, netop A1.

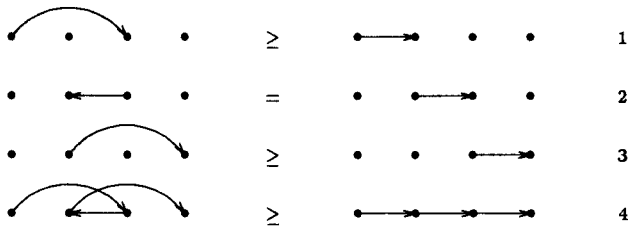
Det skal bevises at B1 gælder i M_6. Se figur C.24.



Figur C.24: Bevis for $M_6 \subseteq M_0$.

I første linje bruges C8, i anden linje C9, i tredje linje C10. I fjerde linje er summen, og i femte linje den reducerede sum, netop B1.

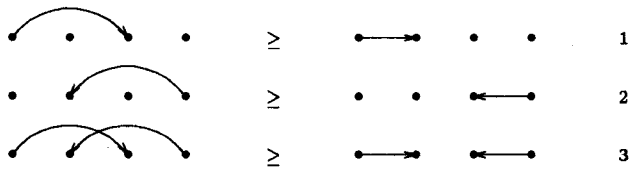
$M_7 \subseteq M_0$: (Tilføjelse i forhold til kilden.) Det skal bevises at A1 gælder i M_7. Se figur C.25.



Figur C.25: Bevis for $M_7 \subseteq M_0$.

I første linje bruges C12a, i anden linje bruges C11, og i tredje linje C12b. I fjerde linje er summen, netop A1.

Det skal bevises at B1 gælder i M_7. Se figur C.26.



Figur C.26: Bevis for $M_7 \subseteq M_0$.

I første linje bruges C12a, i anden linje C13a. I tredje linje er summen, netop B1.

QED

Bibliografi

Alle mine kilder skal være hos dig!
(DDS 396)

Bibliografi sorteret primært efter årstal, sekundært efter første forfatters efternavn.
Min kilde til de benchmark-problemer, jeg har anvendt, er en ftp til softlib.cs.rice.edu.

[Eu 1759] L.Euler. *Solution d'une question curieuse qui ne paroit soumise à aucune analyse.*
s.310-337, Mem.Acad.Sci.Berlin 15, 1759.

[Va 1771] A.T.Vandermonde. *Remarques sur les problèmes de situation.*
s.566-574, Histoire de l'Acadèmie des Sciences (Paris), 1771.

[Vo 1831] B.F.Voigt. *Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein. Von einem alten Commis-Voyageur.*
Ilmenau, genoptrykt 1981, Verlag Bernd Schramm, Kiel; 1831.

[Ha 1858] W.R.Hamilton. *On a new system of roots of unity.*
s.415-416, Proc.Royal Irish Acad.6, 1858.

[Meng 30] K.Menger. *Das Botenproblem.*
Menger 1932, 9.Kolloquium (5.II.1930), 12; 1930.

[Metr 53] N.Metropolis, A.Rosenbluth, M.Rosenbluth, A.Teller & E.Teller.
Equation of state calculations by fast computing machines.
s.1087-1092, J.Chem.Phys., 21, 1953.

[Dant 54] G.B.Dantzig, D.R.Fulkerson & S.M.Johnson. *Solution of a large-scale traveling-salesman problem.*
s.393-410. Oper.Res.2, 1954.

[Floo 56] M.M.Flood. *The traveling-salesman problem.*
s.61-75, Oper.Res.4, 1956.

[Gilm 64] P.C.Gilmore & R.E.Gomory. *Sequencing a one state-variable machine: a solvable case of the traveling salesman problem.*
s.655-679, Oper.Res.12, 1964.

[Grah 66] R.L.Graham. *Bounds for certain multiprocessing anomalies.*
s.1563-1581, Bell System Tecj.J.45, 1966.

[Grah 69] R.L.Graham. *Bounds on multiprocessing timing anomalies.*
s.416-429, SIAM J.Appl.Math.17, 1969.

[Krol 71] Patrick Krolak & Wayne Felts. *A Man-Machine Approach Toward Solving the Traveling Salesman Problem.*
s.327-334, Communications of the ACM, vo.14, no.5, 1971.

[Lawl 71] E.L.Lawler. *A solvable case of the traveling salesman problem.*
s.267-269, Math.Programming, 1971.

[Lin 71] S.Lin & B.W.Kernighan. *An Effective Heuristic Algorithm for the Traveling-Salesman Problem.*
s.498-516, Oper.Res.21, 1971.

[Chri 72] Nicos Christofides & Samuel Eilon. *Algorithms for Large-scale Traveling Salesman Problems.*
s.511-518, Operational Research Quarterly vol.23, no.4, 1972.

[Holl 75] J.Holland. *Adaptation in Natural and Artificial Systems.*
University of Michigan Press, Ann Arbor, 1975.

[Chri 76] N.Christofides. *Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem.*
Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg, PA, 1976.

[Sahn 76] S.Sahni & T.Gonzalez. *P-complete approximation problems.*
s.555-565, J.Assoc.Comput.Mach.23, 1976.

[Garf 77] R.S.Garfinkel. *Minimizing wallpaper waste, part I: a class of traveling salesman problems.*
s.741-751, Oper.Res.25, 1977.

[Karp 77] R.M.Karp. *Probabilistic analysis of partioning algorithms for the traveling-salesman problem in the plane.*
s.209-224, Math.Oper.Res.2, 1977.

[Papa 77] C.H.Papadimitriou & K.Steiglitz. *On the complexity of local search for the traveling salesman problem.*
s.76-83, SIAM J.Comput.6, 1977.

[Rose 77] D.J.Rosenkrantz, R.E.Stearns & P.M.Lewis II. *An analysis of several heuristics for the traveling salesman problem.*
s.563-581, SIAM J.Comput.6, 1977.

[Corn 78] G.Cornuéjols & G.L.Nemhauser. *Tight bounds for Christofides' traveling salesman heuristic.*
s.116-121, Math.Programming 14, 1978.

[Demi 79] V.M.Demidenko. *TSP på kendt, asymmetrisk matrix.*
s.29-35, Vestsi Akad.Navuk BSSR Ser.Fiz.-Mat. Navuk 1, 1979.

[Bach 82] E.Bach, M.Luby & S.Goldwasser. *Privat kommunikation.*
1982.

[Kirk 83] S.Kirkpatrick, C.D.Gelatt Jr. & M.P.Vecchi. *Optimization by Simulated Annealing.*
s.671-680, Science, 220, 1983.

[Tuck 83] A.W.Tucker. *Brev til David Shmoys.*
17.februar 1983.

[Otte 84] R.H.J.M.Otten & L.P.P.P.van Ginneken. *Floorplan Design using Simulated Annealing.*
s.96-98, Proc.IEEE Int.Conference on Computer-Aided Design, Santa Clara, november 1984.

[Aart 85] E.H.L.Aarts & P.J.M.van Laarhoven. *Statistical Cooling: A General Approach to Combinatorial Optimization Problems.*
s.193-226, Philips J.of Research, 40, 1985.

[Corn 85] G.Cornuéjols, D.Naddef & W.R.Pulleyblank. *The traveling salesman problem in graphs with 3-edge cutsets.*
J.Assoc.Comput.Mach., 1985.

[Gilm 85] P.C.Gilmore, E.L.Lawler & D.B.Shmoys. *Well-solved special cases.*
s.87-144, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys; Wiley, New York, 1985.

[Gold 85a] D.E.Goldberg & R.Lingle. *Alleles, Loci and the TSP.*
s.154-159, Proceedings of the First International Conference on Genetic Algorithms, J.J.Grefenstette (editor); Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[Gold 85b] B.L.Golden & W.R.Stewart. *Empirical analysis of heuristics.*
s.207-250, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys; Wiley, New York, 1985.

[Hoff 85] A.J.Hoffman & P.Wolfe. *History.*
s.1-15, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys; Wiley, New York, 1985.

[John 85a] D.S.Johnson & C.H.Papadimitriou. *Computational complexity.*
s.37-85, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys; Wiley, New York, 1985.

[John 85b] D.S.Johnson & C.H.Papadimitriou. *Performance guarantees for heuristics.*
s.145-180, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization; E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys; Wiley, New York, 1985.

[Rome 85] F.Romeo & A.L.Sangiovanni-Vincantelli. *Probabilistc Hill Climbing Algorithms: Properties and Applications.*

s.393-417, Proc.1985 Chapel Hill Conference on VLSI, maj 1985.

[Huan 86] M.D.Huang, F.Romeo & A.L.Sangiovanni-Vincantelli. *An Efficient General Cooling Schedule for Simulated Annealing.*

s.381-384, Proc.IEEE Int.Conference on Computer-Aided Design, Santa Clara, november 1986.

[John 86] D.S.Johnson, C.R.Aragon, L.A.McGeoch & C.Schevon. *Optimization by Simulated Annealing: an Experimental Evaluation.*

List of Abstracts, Workshop on Statistical Physics in Engineering and Biology; Yorktown Heights, april 1984, revideret version 1986.

[Lund 86] M.Lundy & A.Mees. *Convergence of an Annealing Algorithm.*

s.111-124, Math.Prog., 34, 1986.

[Ackl 87] David H.Ackley. *An Empirical Study of Bit Vector Function Optimization.*

s.170-204, Genetic Algorithms and Simulated Annealing; Pitman, 1987.

[Gref 87] John J.Grefenstette. *Incorporating Problem Specific Knowledge into Genetic Algorithms.*

s.42-60, Genetic Algorithms and Simulated Annealing; Pitman, 1987.

[Laar 87] P.J.M. van Laarhoven & E.H.L.Aarts. *Simulated Annealing: Theory and Applications.*

Reidel, Dordrecht, 1987.

[Padb 87] M.Padberg & G.Rinaldi.

Oper.Res.Lett.6, 1, 1987.

[Sira 87] D.J.Sirag & P.T.Weiser. *Toward a unified thermodynamic genetic operator.*

s.116-22, Genetic Algorithms and Their Applications: proc. Second Int. Conf. on Genetic Algorithms, ed. Grefenstette; Lawrence Erlbaum Associates, 1987.

[Padb 88] M.Padberg & G.Rinaldi.

Report R.247, Instituto di Analisi Dei Sistini ed Informatica del CNR, Rom, 1988.

[Torr 88] Jan Torreele. *Optimization by Simulated Annealing.*

Speciale (?) ved Vrije Universiteit Brussel, 1988.

[Aart 89] Emile Aarts & Jan Korst. *Simulated Annealing and Boltzmann Machines.*

Wiley & Sons, 1989.

[Glov 89] Fred Glover & Harvey J.Greenberg. *New approaches for heuristic search: A bilateral linkage with artificial intelligence.*

s.119-130, European Journal of Operational Research 39, 1989.

[Liep 89] G.E.Liepins & M.R.Hilliard. *Genetic algorithms: foundations and applications.*

s.31-58, Annals of Operations Research, 21, 1989.

[Whit 89] D.Whitley, T.Starkweather & D'A.Fuquay. *Scheduling Problems and Travling Salesman: The Genetic Edge Recombination Operator.*

s.133-140, Proceedings of the Third International Conference on Genetic Algorithms, J.Schaffer; Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[Corm 90] Thomas H.Cormen, Charles E.Leiserson & Ronald L.Rivest. *Introduction to Algorithms.*

MIT Press, McGraw-Hill Book Company, USA/England, 1990.

[Dodd 90] Nigel Dodd. *Slow annealing versus multiple fast annealing runs - an empirical investigation.*

s.269-272, Parallel Computing 16, 1990.

[John 90] D.S.Johnson. *Local Optimization and the Traveling Salesman Problem.*

s.446-461, Proceedings of the 17th Colloquium on Automata, Languages and Programming, Springer, New York, 1990.

[Mill 91] Donald L.Miller & Joseph F.Pekny. *Exact Solution of Large Asymmetric Traveling Salesman Problems.*

s.754-761, Science, Vol.251, 1991.

[Yao 91] Xin Yao. *Simulated annealing with extended neighbourhood.*

s.169-189, Int.J.Comput.Math. 40, Nr.3/4, 1991.

[Ebse 92] Bjarne Ebsen & Morten Wulff Pedersen. *Parallellisering af Simulated Annealing og Genetiske Algoritmer anvendt på Travelling Salesman Problemet.*

Speciale ved Århus Universitet, 1992.

[Mich 92] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.*

Springer, 1992.

[Chan 94] Barun Chandra, Howard Karloff & Craig Tovey. *New Results for the old k-opt Algorithm for the TSP.*

s.150-159, Proceedings of the ACM-SIAM symposium on discrete algorithms, ACM, Philadelphia, 1994.