Forord

Denne rapport er udarbejdet som adgangsgivende projektrapport til uddannelsen bachelor of science software engineering på Aalborg Universitet Esbjerg. Målgruppen for denne rapport er vejleder og censor på uddannelsen. Afleveringsfristen er fredag den 25. januar 2002. Forløbet er i alt 5 uger incl. helligdage. Rapporten er udarbejdet af Claus Hallas Nielsen, Mikael Heinze, Jens Iwan Jørgensen og Jeppe Moritz Hansen.

Afleveret den 25. januar 2002 af:

Claus Hallas Nielsen	Mikael Heinze
Jens Iwan Jørgensen	Jeppe Moritz Hansen

Indholdsfortegnelse

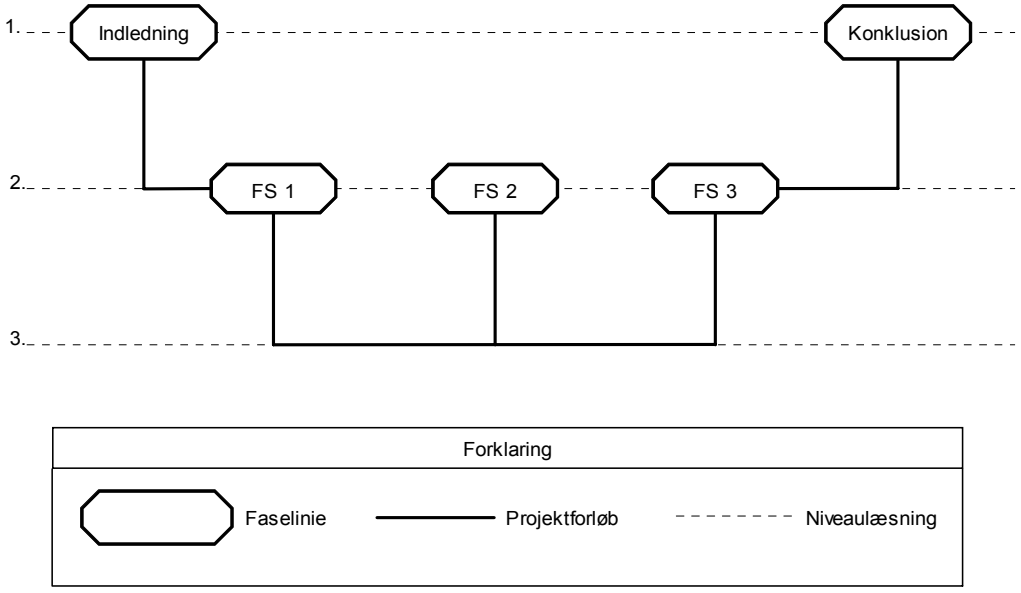
1. Indledning

Et af computerverdenens mere komplekse problemer er kendt som travelling salesman problem (TSP). TSP er en problemstilling der kort beskrevet består af at finde den korteste rute for en handelsrejsende, der har en række byer der ønskes besøgt, og hvor start og slut by er den samme. TSP er dog blot blevet et navn og en forklaringsform, på et mere generelt problem der findes i flere varianter. Mere generelt står man her med en problemstilling hvor man har en lang række punkter der ønskes besøgt, og den samlede rejse mellem punkterne ønskes at være så lille som mulig, og til sidst står man hvor man startede.

Vi vil i vores rapport tage udgangspunkt i en lidt anden problemstilling end en handelsrejsende. Vi vil i rapporten tage udgangspunkt i en maskine der laver huller i printplader. I en samlebånds produktion kan det være af yderste vigtighed at en sådan maskine kører optimalt, og ikke blot tager punkterne der skal bores i en umiddelbar orden. I sidste ende kan det have ganske stor betydning om en maskine bruger 10 sek. på at lave hullerne i en printplade, i stedet for 12 sek. Med andre ord, produceres der 5 eller 6 printplader pr. minut.

I en maskine der borer disse huller, må det antages at boret har en udgangsposition. Denne udgangsposition skal boret også ende med at stå i, inden næste printplade påbegyndes. Denne problemstilling er ikke anderledes end TSP der eksemplificeres ved hjælp af en handelsrejsende. Borets udgangsposition svare til start og slut by, og hullerne der skal bores, svarer til byerne der skal besøges netop en gang.

Denne rapport er skrevet således at det er muligt at læse den uafhængigt i tre niveauer (afbilledet i nedenstående figur). Første niveau består af problemformulering samt konklusion. Ved læsning opnås indblik i væsentlige valg i projektperioden. Ved læsning på andet niveau, som er de enkelte faseliniers beslutninger, samt definition af den nye fase, opnås der indblik i de enkelte fasers problemstillinger, samt konklusionen herpå. Tredje niveau er fuld læsning af de enkelte faser. Dette giver selvsagt indsigt i al argumentation i fasen.



Figur 1: Læseniveauer

Læsning af rapporten kan kombineres. Ved at læse andet niveau opnås en bred indsigt i hele projektforløbet. Ønskes yderligere uddybning af en fases indhold kan denne i detaljer læses.

2. Problemformulering

Ved boring af huller i printkort i elektronikindustrien er det vigtigt at boringen sker hurtigt, for at få flest mulige produkter ud inden for et tidsrum. Problemet består i at få boremaskinen til at bruge mindst mulig tid på at bevæge sig mellem de enkelte huller, for til sidst at returnere til sit udgangspunkt.

Ved nye printkort skal der ske en beregning af den rute som tager kortest tid og slutter ved udgangspunktet. Denne tid skal være så kort som mulig, men da selve beregnings tiden kan blive længere end tiden for blot at bore hullerne i pladen tilfældigt, skal problemet løses sådan at beregningstiden er optimal.

Vi skal analysere problemet og på baggrund af analysen opstille modeller til løsning af problemet. Der skal træffes et valg af model ud fra kriterier som er vigtige for løsning af problemet (kriterierne bestemmes senere i rapporten). Ved implementering er der er krav til simplicitet og gennemskuelighed i koden for at opnå genanvendelighed.

Vi skal demonstrere forståelse af matematiske begreber benyttet i forbindelse med problemløsningen.

Løsningen skal implementeres i Java og her skal den praktiske anvendelighed demonstreres.

3. Projektgrundlag

I projektgrundlaget, som er indeholdt projektetableringsfasen, vil formål og mål blive klarlagt. Herudover skal der tages stilling til en lang række faktorer som omgiver projektet. Det være sig rammerne, de fysiske og materielle, intern og ekstern kommunikation og interessenter. Der vil blive redegjort for projektets strategi, indeholdende en strategianalyse. På baggrund af denne træffes der et begrundet valg af projektmodel. Projektgrundlaget afsluttes med udarbejdelse af en projektplan, som indeholder estimater og beskrivelser af de enkelte faselinier.

3.1. Generelt

Som indledning til projektgrundlaget er formål og mål stillet op. Dette er gjort for at skabe klarhed og fælles forståelse omkring hvor projektet skal bære hen og hvordan vi når dertil.

Formål

Udvikling af en matematisk model til løsning af vores valgte datalogiske problemstilling og derpå konstruere en model til løsning af en væsentlig del af problemet.

Mål

En projektrapport indeholdende en anbefaling af algoritme til løsning af problemet. Løsningen skal dække en væsentlig del af problemet, samt implementeres i Java. Projektrapporten skal samtidig vise forståelse af matematiske begreber brugt i

forbindelse med opgaveløsning og diskret matematik og dokumentere vores viden om disse i form af fyldestgørende beskrivelser.

3.2. Rammer

Projektet er fastsat til 5 uger, hvilket ikke kan ændres. Indeholdt i de 5 uger er helligdage i forbindelse med jul og nytår. Projektgruppen består af 4 deltagere der møder kl. 8:30 og går hjem kl. 15:00, vi har alle de samme forudsætninger. Det skal dog nævnes, selvom der er blevet uddelegeret roller er projektets forløb og slutresultat et fælles ansvar.

Materiel/Værktøj
Der er mulighed for adgang til pc'ere for både rapportskrivning og programmering på AUE hvor der er mulighed for udskrivning til printer. Ved rapportskrivning anvendes der Microsoft® Office2000®. Da projektgruppen har den største programmerings-erfaring i Java, er Java valgt som implementeringssprog.

Kommunikation
Kommunikationen i projektgruppen foregår hovedsagelig uformelt. Den eksterne kommunikation foregår primært via mødeindkaldelse, hvilket er projektgruppens ansvar at arrangere.

Interessenter
Projektgruppen har ingen egentlig styregruppe. Projektets vejleder, er projektgruppens sikring mod ikke at gå i den forkerte retning. Vejlederen kan kun rådgive, ikke tvinge projektgruppen i en bestemt retning. Uddannelse og erfaringsgrundlag er hovedsagelig det samme for alle projektets deltagere. Projektgruppen har ingen interessenter/brugere at interagere med.

Roller
Følgende roller er blevet uddelegeret. Rollen som arkivar er blevet uddelegeret til Jens Iwan Jørgensen. Det er hans ansvar at strukturere og vedligeholde dokumentationen og konfigurationsstyring. Rollen som projektleder har vi valgt ikke at uddele til en bestemt person, men i stedet lade rolle tilfalde projektgruppen som helhed.

3.3. Strategi

Ved projektstart forsøges det at skabe et overblik over de umiddelbare forhindringer som projektet udsættes for. Dette med henblik på at få defineret områder som udgør potentielle risici for projektets forløb og endelige resultat. I virkelige og længerevarende projekter udformes den overordnede strategi for projektet med udgangspunkt i de vilkår projektet skal leve under. Samt vægtes synlige risici og eventuelle konfrontationer der kan opstå med brugere og interessenter.

Da denne rapport skal søge at afdække problemstillinger i en tænkt situation og da projektet ikke har nogle brugere eller interessenter, har vi valgt at undlade disse analyser.

Sygdom er hver mands herre, der er ikke noget man kan gøre for at gardere sig. Sandsynligheden for aftalt fravær er sat til høj, det er ikke en risiko – men en forudsætning. Vi kan forudse at der vil være dage hvor gruppemedlemmer er væk. Hvis det bliver et stort problem må gruppen diskutere om det bliver nødvendigt at inddrage tid udover den aftalte arbejdstid.

Ikke aftalt fravær er en modsætning til gruppens høje ambitionsniveau. Bliver det et problem, må det diskuteres på møde hvorvidt alle har samme ambitionsniveau, eller om der eventuelt skal findes et kompromis. I yderste konsekvens kan der blive tale om ekskludering af et medlem fra gruppen, og dermed projektet. Denne beslutning må tages af den resterende gruppe i enstemmighed, i konsultering med skolen.

Den sidste risiko som projektet indeholder, er at emnet der arbejdes med er ukendt for alle projektgruppens medlemmer. Det vil derfor være vigtigt at få gennemgået den kendte teori der skal benyttes for at kunne løse projekts problemstilling, med bedst mulig tid til rådighed. Under løsning af problemstillingen bør de sværeste opgaver løses først, hvis det er muligt, sådan at de svære opgaver ikke skubbes til slutningen af projektet. Da dette vil give en stor arbejdsbyrde i slutningen af projektet. Såfremt der vil være opgaver som ikke står til at løse inden for den givne tidsramme, vil det være nødvendigt at afslutte denne opgave og beskrive hvad det var der var utilstrækkeligt for at opgaven kunne løses. Styrken hos projektgruppen er at der forventes et resultat af høj kvalitet. Dette sætter projektgruppens motivation høj fra starten.
Ingen af opgaverne i implementeringen betragtes som en risikofaktor, da alle medlemmerne af projektgruppen tidligere har arbejdet med Java som programmeringssprog. Dog gælder det også her, at svære opgaver løses først.

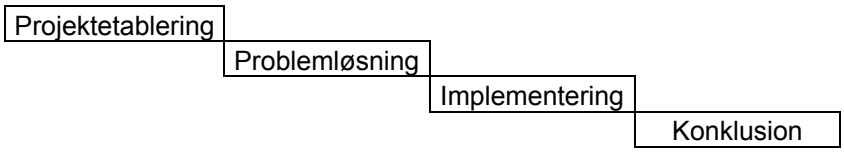
Valg af strategi
Det fastsættes to strategier. Først en for projektmodellen og derefter en for projektforløbet.

Det er vigtigt at vælge en projektmodel med mulighed for at få kortsigtede mål, sådan at projektgruppen kan føre en stram styring med projektforløbet. Desuden bør projektmodellen være af den type der ikke tillader iteration tilbage til en tidligere fase. Dette er for at kunne nå en slutning på en fase, sådan at det ikke bliver ved med at komme ændringer til fase, hvilket kan sætte fremdriften i projektet i stå. Dog bør man kunne ændre i dokumentation for en fase, såfremt der afdækkes fatale fejl. Mindre ændringer kan tilføjes i et separat afsnit i slutningen af dokumentation for en fase.
Strategien for projektmodellen er følgende: ”En faseopdelt projektmodel uden iteration for at kunne opnå delresultater, sådan at en fremdrift i projektet kan spores, med henblik på at opretholde motivationen hos projektgruppen.”

Strategien for projektforløbet skal tage hensyn til de svagheder der er i projektet, gerne ved brug af de styrker der ligger. Sygdom kan der ikke gøres noget ved, hvilket der kan med ikke aftalt fravær. Ved at tilstræbe at holde motivationen høj, tilsikres det i en vis grad at det enkelte projektgruppemedlem møder som aftalt. En måde at holde motivationen høj på er ved at der fremkommer resultater.
Strategien for projektforløbet er følgende: ”Løs svære opgaver først, for at fremkomme med resultater, som sikrer en høj motivation.”

3.4. Projektmodel

Med udgangspunkt i projektets strategi mener vi at vandfaldsmodellen er mest hensigtsmæssig.



Figur 2: Vandfalds projektmodellen

Figuren viser vandfaldsmodellen som grundmodel. Startfasen er projektetableringen som andre typiske projekter, med forholdsvis lav usikkerhed. Herefter den matematiske problemløsning. Det er ikke klarlagt hvad præcist denne fase vil komme til at indeholde, dette defineres først senere i forløbet. Problemløsningsfasen efterfølges af implementeringsfasen. Her konstrueres en prototype på baggrund af problemløsningsfasens løsningsmodeller. Endeligt konkluderes på projektets forløb og resultat. Stram styring opnås ved korte og præcist defineret faser, hvor projektets løbende resultat holdes op i mod aktiviteter indeholdt i fasen.

3.5. Arbejdsform, organisation og møder

Projektgruppen er sammensat af personer der ikke har arbejdet sammen om projekter tidligere. Dette kan give gnidninger, da der kan opstå uenighed om hvordan de enkelte opgaver skal gribes an. Modsat kan dette vendes til noget konstruktivt, da det giver projektgruppen mulighed for at lære af hinanden. Vi har valgt at styre projektet ved uddeling af opgaver fra en pulje, til en eller flere personer der påtager sig ansvaret for løsningen af en af opgaverne. Puljen revurderes på morgen- og projektmøde. Fordelingen foregår løbende.

Beslutningsprocesser
Projektet kører i en flad organisation, hvor alle har en stemme. Enstemmighed i vigtige beslutninger er en hjørnesten i projektet. Beslutningsprocessen er beskrevet for den enkelte fase i projektplanen.

Vejledermøder
Vejlederen har ingen reel magt over projektet. Vejlederen kan besvare tvivlsspørgsmål og bruges som sparringspartner. Vejlederen skal have materialet to døgn før mødet. Der afholdes som udgangspunkt vejledermøder med ca. en til to ugers mellemrum. Møderne foregår på AUE.

Faseliniemøder
Der afholdes faseliniemøder når det fremgår af tidsplanen. Ved faseliniemødet skal planen revurderes for evt. ændringer. Mangler et punkt i fasen at blive færdiggjort, vil faseliniemødet blive afholdt med bl.a. fastlæggelse af dato for et endeligt faseliniemøde, og færdige punkter blive behandlet som planlagt.

Morgenmøder
Det er aftalt at afholde morgenmøde hver dag.

Projektmøder
Der kan indkaldes til projektmøder af et enkelt gruppemedlem, der foregår svarende til morgenmøde.

3.6. Konfigurationsstyring og projektarkiv

Alle dokumenter bliver vedligeholdt/opdateret af arkivaren. Arkivaren har det overordnede ansvar for at klippe de færdige dokumenter ind i hoveddokumenterne. Der findes en mappe til hver projektdeltager. Denne mappe er projektdeltagerens eget ansvar og det er derfor kun projektdeltageren som foretager ændringer i sin egen mappe. I denne mappe ligger kladder, som hver projektdeltager laver og når projektdeltageren mener han er færdig, skrives dokumentet ud til fælles gennemgang. Når dokumentet er endeligt godkendt, af samtlige projektdeltagere, kopieres det til en mappe til færdige dokumenter.

Til kildekode er en mappe, hvori versioner af koden lægges. Hver version gemmes i en separat mappe, og når der påbegyndes en ny version, så arbejdes der altid på en kopi, som efter endt behandling får et nyt versionsnummer. Der er følgende krav til udviklerne af programmet:

- Alle metoder skal indeholde initialer, pre og post betingelser og dato for ændring/oprettelse.
- Hver klasse skal indeholde en klassebeskrivelse, så koden bliver lettere at læse både under og efter endt udvikling.
- Alle variabler, metoder og klasser skal have sigende navne og alle skal være på engelsk. Æ, ø og å, må derfor heller ikke benyttes i forbindelse med samme.
- Det er op til den individuelle udvikler, selv at vurdere hvornår en kildekode er godkendt/testet til evt. at sætte sammen med en anden kildekode. Det tager for lang tid, hvis alle udviklere skal sætte sig ind i detaljer omkring hver stump kildekode.

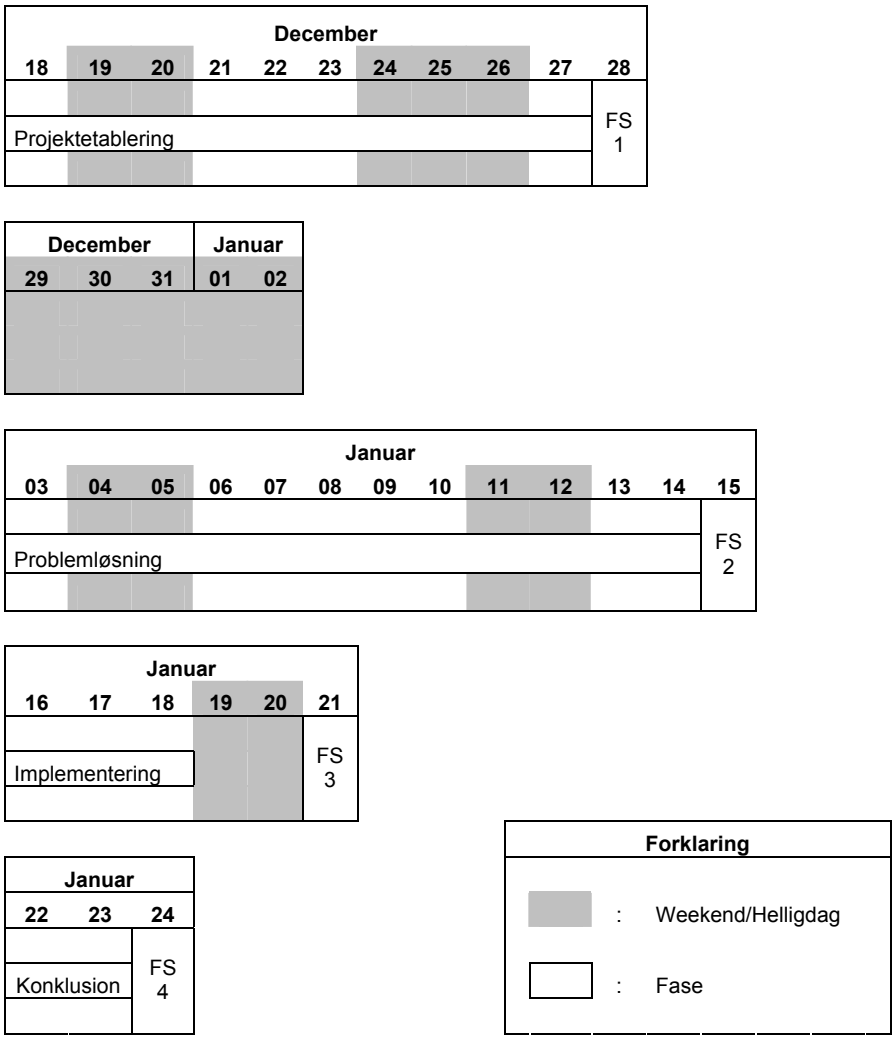
Der er følgende formkrav til rapporten. Typografien der anvendes er Arial, størrelse 12 og linieafstand 1. Margin-indstillingerne er 3 cm for top og bund og i siderne 2 cm. Side-hoved og fod er 1,25 cm. Tabeller er centreret, med billedtekst centreret under figur eller tabel.

4. Projektplan

Projektplanen beskriver forløbet af et projekt. Projektplanen vil omhandle de enkelte faser i projektet og deres ressourceforbrug. Desuden vil faselinier blive fastsat og vurderings- kriterier og procedurer sat op. En tidsplan udarbejdes og skal give projektdeltagerne noget at styre efter. Herudover vil der ske opfølgninger på faselinierne i form af et faseliniemøde, revurdering af tidsplanen mv.

4.1. Estimering og tidsplan

For de faselinier, som indgår i den indledende del af projektmodellen, er der klart definerede aktiviteter og produkter. Nedenstående er første udkast til tidsplanen. Denne tidsplan er dog retningsgivende og kan naturligvis ændres undervejs i projektforløbet. Første og endelige version vil være at finde i rapporten, eventuelt øvrige versioner er at finde i bilag.



Figur 3: Tidsplan

Ved vurdering af varigheden af projektet vil vi anvende ”analogi”, hvor der benyttes erfaringer fra tidligere projekter til at estimere det indeværende projekt.
I forhold til tidligere projekter med kortere varighed, er estimeringen af det igangværende projekt mere vanskeligt. Traditionelt er estimering af et projektforløb en proces af gradvis tilretning. Projektetableringen er identisk med erfaringer fra tidligere projekter og vi har derfor allerede et godt indblik i hvor lang tid, det varer. Problemløsningsfasen ligner derimod ikke det vi har haft erfaringer med tidligere og det er derfor sværere at estimere.

4.2. Faselinier

Da indholdet i en fase reelt først kan defineres ved afslutning af foregående fase, på grund af den opgaveusikkerhed der ligger i projektforløbet, er det besluttet først at definere fasen der. Ved hver faselinie vurderes de produkter fasen indeholder iht. deres vurderingskriterie(r). Minimumskrav til definition af en fase, er de aktiviteter som fasen indeholder, ressourceforbruget, samt de produkter og aktiviteter der skal forberedes til næste fase og for selve faselinien; hvilke produkter som afsluttes i fasen og punkternes vurderingskriterie(r).

Fase 1	Projektetablering	
Aktiviteter	• Problemformulering • Projektgrundlag • Projektplan	
Ressourcer	• Projektgruppen • Løbende projektmøder	
Faselinie	Godkendelse af projektetablering	
Betydning	Fælles forståelse for projektets forløb	
Vurderings-Procedure	Hver aktivitet/produkt godkendes ved enstemmighed, med mindre at anden vurderingsprocedure specifikt er angivet	
	Produkt	**Vurderingskriterie(r)**
	Projektetablering indeholdende: projektgrundlag og -plan	• Formål og mål for projektet er korrekt defineret • Overordnet strategi for projektet er defineret og denne varetager projektets svage sider, risici og problem-formuleringens fokus. • Argumenteret valg af projektmodel iht. projektets strategi • Estimater og tidsplan er udarbejdet og disse er, under hensyntagen til usikkerheden i projektets enkelte faser, retvisende
Klargøring	• Produkter og aktiviteter for næste fase diskuteres, defineres og uddelegeres • Revurdere og tilrette estimater og tidsplan	

Tabel 1: Fase 1 – Projektetablering

Ovenfor ses fase 1. Fasen indeholder de produkter, som skal strukturere projektets forløb værende projektetableringen (dvs. udarbejdelse af projektgrundlag og –plan). Selve projektplanen er kun beskrevet for projektets indledende faser.

5. Faselinie 1 – Projektetablering

Produkterne i fasen blev gennemlæst. Problemformulering blev godkendt uden bemærkninger, det samme gjorde estimat og tidsplan samt indeholdet af projektets indledende faser. Der var bemærkninger til overordnet strategi og projektmodel, men begge blev godkendt. Ændringer vurderes til det næste projektmøde.

I fasen er rammer og retningslinier for projektforløbet fastlagt. Vandfaldsmodellen skal føre projektet gennem de kendte faser i udviklingen. Ligeledes giver modellen mulighed for at stoppe op, orientere sig og finde nye indsatsområder.

5.1. Definition af fase 2 - Problemløsning

Nedenstående er definition af anden fase. Fasen indeholder en gennemgang af den teori vi vil anvende senere i fasen til forklaring af de enkelte løsningsmodeller, samt til evt. konstruktion af nye løsningsmodeller.

Fase 2	Problemløsning	
Aktiviteter	• Gennemgang af anvendt teori • Løsningsforslag til alternative algoritmer	
Ressourcer	• Projektgruppen • Løbende projektmøder	
Faselinie	Defineret problemområde	
Betydning	Projektgruppen har en fælles forståelse for den anvendte teori og det fremkomne produkt.	
Vurderings-Procedure	Hver aktivitet/produkt godkendes ved enstemmighed, med mindre at anden vurderingsprocedure specifikt er angivet	
	Produkt	**Vurderingskriterium**
	Forklaring af teori	• Skal indeholde; grafer, træer, Eulerkreds og -tur, Hamiltonkreds og -tur, tidskompleksitet
	TSP	• Beskrivelse af TSP, samt problemstillingerne omkring den tunge kompleksitet
	Alternative modeller	• Gennemgang af alternative løsningsmodeller, samt en vurdering af disse
Klargøring	• Produkter og aktiviteter for næste fase diskuteres, defineres og uddelegeres • Revurdere og tilrette estimater og tidsplan	

Tabel 2: Fase 2 – Problemløsning

6. Anvendt teori

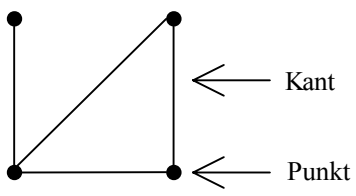
Der vil i dette afsnit blive gennemgået relevant teori for løsning af TSP problematikken. Den gennemgåede teori er ikke komplet for de enkelte emner, men indeholder kun teori relevant for TSP.

6.1. Grafer

I følgende afsnit vil vi så vidt muligt beskrive hvad grafer er. Dette skylles at TSP problemstilling og vores løsningsforslag vil benytte grafer og en mængde grafteori. Vi vil så vidt mulig forsøge at dække det mest elementære samt nogle af de begreber vi senere vil benytte.

Hvad er en graf

Grafer er strukturer der består af punkter og kanter. Kanter er det som forbinder de forskellige punkter i en graf, med hinanden. For bedre at forstå grafer kan det være nødvendig at bruge et eksempel i den virkelige verden. Lad os forestille os vi har et computernetværk. Netværket består af en række pc'er som indbyrdes er forbundet via netværkskabler. Hver pc i netværket svarer til et punkt i en graf, og hver netværkskabel er en kant. En kant kan være, men behøves ikke, at være retningsbestemt. Der findes en række af forskellige typer af grafer, der primært adskiller sig ved, hvordan det er tilladt at forbinde punkterne. I det følgende afsnit vil vi forklare forskellige typer af grafer, og hvilke egenskaber der identificerer dem.



Figur 4: Et eksempel på en graf

Valens

Valensen (engelsk degree) af et punkt er defineret som antallet af kanter der går ud fra punktet. Notationsformen for valens er deg(p) hvor p er punktet.

Specielt for en pseudograf kan der findes loops og en af disse tæller altid 2 til valensen af punktet.

For orienterede grafer gælder det at for kanter som går fra et punkt skrives valensen som $deg^-(p)$ og fra kanter som går til et punkt skrives valensen $deg^+(p)$.

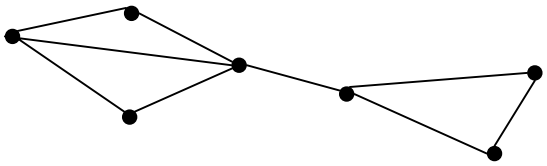
Ture og kredse

En helt grundlæggende terminologi i grafteorien omhandler ture og kredse. En tur består i at følge en række kanter rundt i en graf. Hvis det er muligt at komme fra punkt A til punkt B ved at følge kanterne rundt i grafen, taler man om der er en tur fra punkt A til B. En tur bliver til en kreds i det øjeblik udgangspunkt og slutpunkt er det samme. Derudover kan man have en simpel tur eller en simpel kreds. En tur eller kreds er simpel i det øjeblik alle kanter i grafen kun er brugt en gang.

Simpel graf

En simpel graf er, som navnet antyder, den simpleste form for graf. Denne type graf danner bla. grundlag for multigrafen og pseudografen. Med det menes der at enhver simpel graf også er en multi- eller pseudograf, hvor disse grafers unikke kendetegn ikke er benyttet.

For en simpel graf gælder: Grafen består af en række punkter. Der er maksimalt én kant der forbinder to forskellige punkter, og kanterne er ikke retningsbestemt.

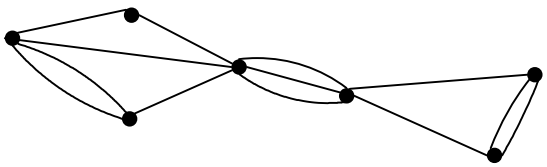


Figur 5: Et eksempel på en simpel graf

Multigraf

I nogle tilfælde er det nødvendigt at have grafer hvor det er tilladt at have flere kanter mellem de samme to punkter. En sådan graf kaldes for en multigraf.

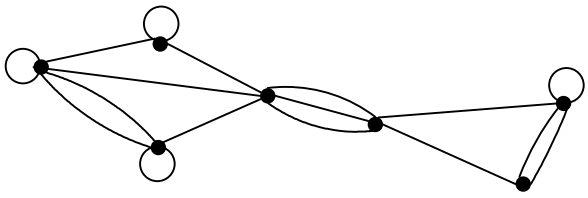
For en Multigraf gælder: Grafen består af en række punkter. Der er tilladt at have flere kanter mellem to forskellige punkter, og kanterne er ikke retningsbestemte.



Figur 6: Et eksempel på en multigraf

Pseudograf
I pseudografer er det som i multigrafen tilladt at have flere kanter mellem de samme to punkter. Det er desuden tilladt at have en kant hvor start og slutpunkt er den samme. Eller sagt på en anden måde, have en kant der kun er forbundet til et punkt.

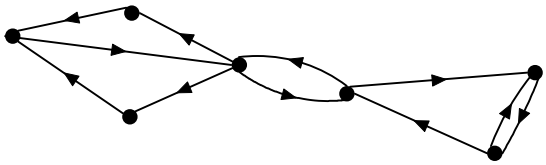
For en Pseudograf gælder: Grafen består af en række punkter. Der er tilladt at have flere kanter mellem to punkter. Det er tilladt at have en kant associeret med kun et punkt, og kanterne er ikke retningsbestemte.



Figur 7: Et eksempel på en pseudograf

Retningsbestemt graf
I nogle tilfælde er det nødvendigt at bruge grafer hvor hver kant er retningsbestemt. Er en kant retningsbestemt er det muligt at komme fra det ene punkt til det andet, men det er ikke muligt at komme tilbage til udgangspunktet via samme kant.

For en retningsbestemt graf gælder: Grafen består af en række punkter, og enhver kant der forbinder grafens punkter, er retningsbestemt.

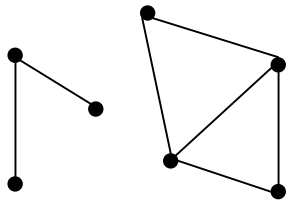


Figur 8:Et eksempel på en retningsbestemt graf

Foruden de forskellige typer af grafer der findes, kan en graf have nogle andre specielle egenskaber. I det følgende afsnit vil vi komme ind på nogle af de specielle egenskaber en graf kan have, hvad en graf med en sådan egenskab hedder og hvad egenskaben er.

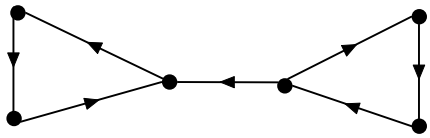
Sammenhængende grafer
Ikke alle punkter i en graf er nødvendigvis forbundet via en tur. I nogle tilfælde kan man risikere at have en graf hvori der ligger et punkt, eller en gruppe punkter, som ikke er forbundet med en anden gruppe punkter i grafen. Men i de tilfælde hvor samtlige punkter i grafen på den ene eller anden måde er forbundet via kanter, taler man om en sammenhængende graf. I retningsbestemte grafer taler man om at den både kan være stærk forbundet og svag forbundet.

For en sammenhængende graf gælder: En ikke retningsbestemt graf er sammenhængende hvis det er muligt via kanterne i en graf at komme fra et vilkårligt punkt til et hvilket som helst andet punkt.



Figur 9: Et eksempel på en graf der IKKE er sammenhængende

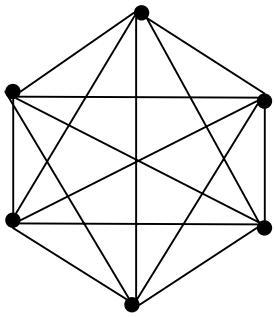
For en stærk retningsbestemt graf gælder:
En retningsbestemt graf er stærk forbundet hvis det er muligt via kanterne i grafen at komme fra et vilkårligt punkt til et hvilket som helst andet punkt. Det er kun muligt at følge en kant den vej dens retningen angiver.
For en svag retningsbestemt graf gælder:
En retningsbestemt graf er svag forbundet hvis det er muligt, når man ignorere retningsangivelsen på kanterne, at komme fra et vilkårligt punkt til et hvilket som helst andet punkt.



Figur 10: Et eksempel på en graf der er svag forbundet, men ikke stærk forbundet

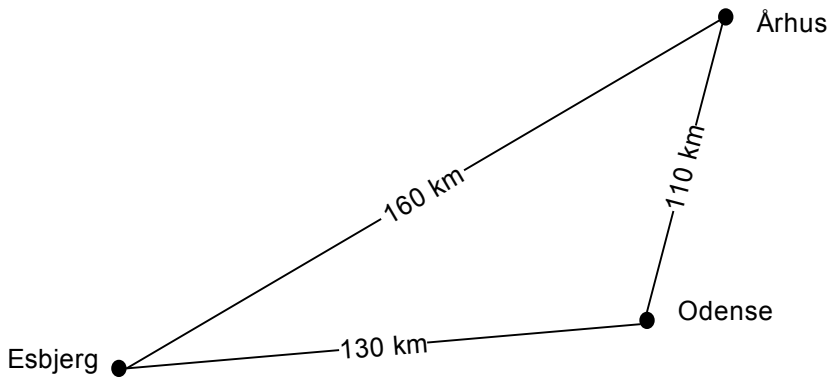
Komplette grafer
En komplet graf er en graf hvor et vilkårligt punkt, med en kant, er forbundet til alle andre punkter i en graf. Det er med andre ord muligt at gå fra et punkt direkte til et andet punkt, uden at skulle igennem et tredje punkt på vejen.

For en komplet graf gælder:
En komplet graf er en graf hvor der er en kant mellem hvert eneste par af punkter.



Figur 11: Et eksempel på en komplet graf

Vægtede grafer
En vigtig egenskab grafer ofte har, er at kanterne har en værdi. Alt efter hvilket problem grafen repræsenterer kan disse værdier have en forskellig mening. Nogle eksempler kunne være distance, pris eller tider. En graf hvor kanterne har en værdi kaldes for en vægtet graf. I en problemstilling hvor man har en række byer der ønskes besøgt, kunne man repræsentere problemet med en graf hvor hvert punkt er en by. Kanternes værdi kunne så f.eks. være afstanden mellem byerne, eller den tid det ville tage at rejse fra den ene til den anden by.



Figur 12: Et eksempel på en vægtet graf

For en vægtet graf gælder: En graf hvor kanterne der forbinder de forskellige punkter, har en værdi, kaldes for en vægtet graf.

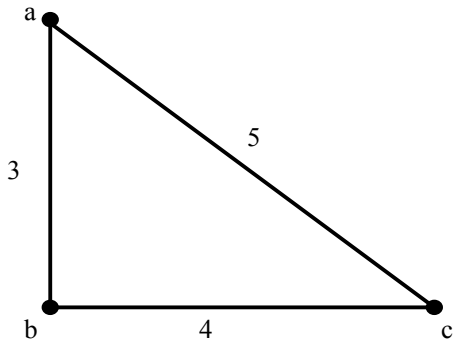
Grafer kan have yderligere egenskaber, men de er ikke beskrevet her da vi ikke mener de har nogen relevans i forbindelse med TSP problematikken.

6.2. Trekantsuligheden

For en trekant gælder:

I en trekant T, med punkterne a, b og c, vil ingen af kanterne {a, b}, {b, c} og {a, c} være længere end summen af de to andre.

I forhold til Figur 13 ses at summen af de to kanter {a, b} og {b, c} er større end kanten {a, c}. 3 + 4 > 5. Uanset hvordan man kombinerer kanterne vil den kant der står alene altid være mindre end summen af de to andre.
Summen af kanterne {a, b} og {b, c} er den mindste kombination der findes i trekanten illustreret i Figur 13.



Figur 13: Trekant

I forhold til vores problemstilling er trekantsuligheden altid opfyldt, da kanternes vægt svarer til den faktiske længde mellem punkterne.

Uligheden i en kreds

I en komplet graf som overholder trekantsuligheden, gælder det også at en ulighed er opfyldt for kredse med n punkter.

Givet punkterne $a_1, \ldots, a_n$, hvor $n \geq 3$, i en komplet graf som opfylder trekantsuligheden gælder:

$$Vægt\{a_1, a_n\} \leq Vægt\{a_1, a_2\} + Vægt\{a_2, a_3\} + \ldots + Vægt\{a_{n-1}, a_n\}$$

Dette vil blive bevist med et induktionsbevis.

Basistrin
I en trekant gælder:

$$Vægt\{a_1, a_3\} \leq Vægt\{a_1, a_2\} + Vægt\{a_2, a_3\}$$

Dette er gældende jf. trekantsuligheden og antages derfor at være korrekt.

Induktionstrin
Givet n + 1 punkter $a_1, a_2, \ldots, a_n, a_{n+1}$ i en komplet graf som opfylder trekantsuligheden. Der antages at for n af disse punkter gælder:

$$(1)\ Vægt\{a_1, a_n\} \leq Vægt\{a_1, a_2\} + Vægt\{a_2, a_3\} + \ldots + Vægt\{a_{n-1}, a_n\}$$

Det skal vises at så gælder også:

$$(2)\ Vægt\{a_1, a_{n+1}\} \leq Vægt\{a_1, a_2\} + Vægt\{a_2, a_3\} + \ldots + Vægt\{a_{n-1}, a_n\} + Vægt\{a_n, a_{n+1}\}$$

Ifølge trekantsuligheden gælder:

$$(3)\ Vægt\{a_1, a_{n+1}\} \leq Vægt\{a_1, a_n\} + Vægt\{a_n, a_{n+1}\}$$

Ved at indsætte højre side af (1) i stedet for første led i højre side af (3) fås (2):

$$(4)\ Vægt\{a_1, a_{n+1}\} \leq Vægt\{a_1, a_2\} + Vægt\{a_2, a_3\} + \ldots + Vægt\{a_{n-1}, a_n\} + Vægt\{a_n, a_{n+1}\}$$

Det er antaget at (1) gælder. (3) er gældende jf. trekantsuligheden. Da højre side af (1) indsat i (3) giver (4) = (2), må (2) være gældende. Det er hermed bevist at vægten af en kant i en kreds altid er mindre end eller lig med summen af resterende kanter i kredsen.

□

6.3. Træer

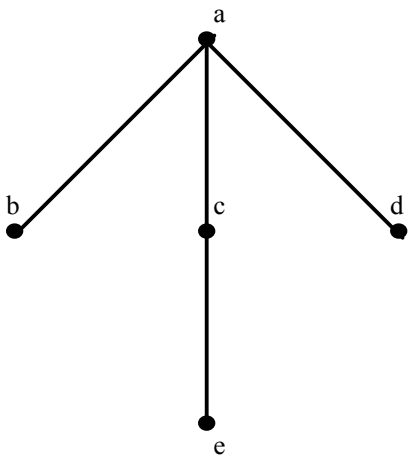
Først gives en definition på hvad et træ er. Efterfølgende defineres hvad et udspændende træ af, med henblik på at definere et minimum udspændende træ. Dette afsnit er udarbejdet på baggrund af afsnit 8.1, 8.5 og 8.6 i DMIA.

Definition på træer

For et træ gælder:
En sammenhængende ikke-retningsbestemt graf uden en simpel kreds. Desuden kan siges at en ikke-retningsbestemt graf er et træ, hviss der er en unik simpel tur mellem alle par af grafens punkter.

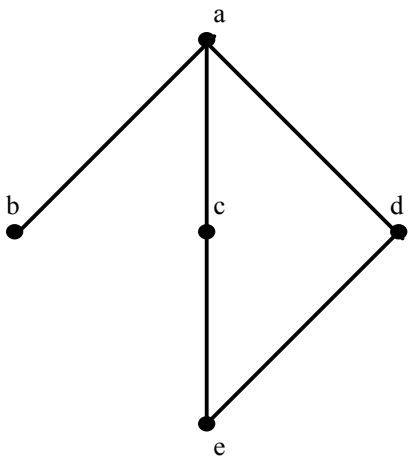
Figur 14 viser en graf som også er et træ. Grafen er sammenhængende, da alle punkter har forbindelse til de andre punkter. Kanterne er ikke retningsbestemte, samt

der ikke er nogen en simpel kreds. Der er dog en unik simpel tur mellem alle par af grafens punkter. Et eksempel på en sådan unik simpel tur er fra punkt a til punkt e. Man kan kun komme fra punkt a til e ved at benytte kanterne {a, c} og {c, e}.



Figur 14: Graf som er et træ

Det der gør at Figur 15 ikke er et træ er at der findes en simpel kreds i grafen. Kredsen er som følger med udgangspunkt i punkt a: {a, c}, {c, e} ,{e, d}, {d, a}. Desuden findes ikke en unik simpel tur mellem alle par af grafens punkter. Man kan komme fra punkt a til e af to veje. Enten af kanterne {a, c}, {c, e} eller {a, d}, {d, e}.



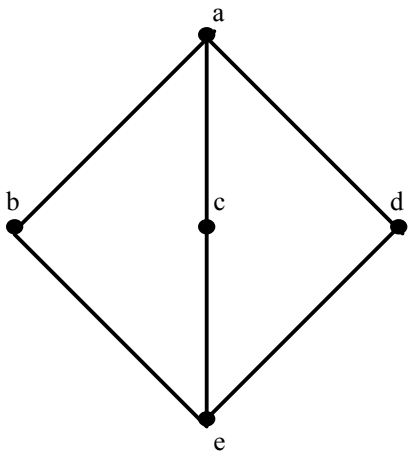
Figur 15: Graf som ikke er et træ

For et træ gælder: Et træ med n punkter har n-1 kanter.

I forhold til Figur 14 ses at der er 5 punkter, men kun 4 kanter.

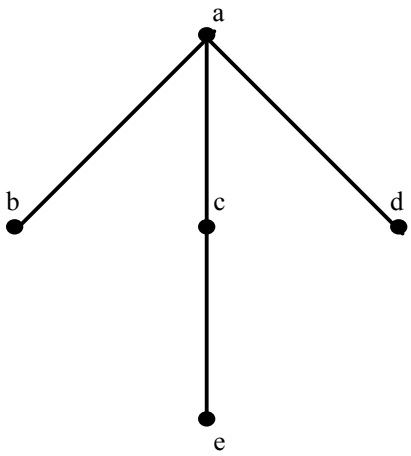
Udspændende træer

Et udspændende træ er konstrueret med udgangspunkt i en simpel graf. Givet en simpel sammensat graf G = (V, E), hvor V er mængden af punkter og E er mængden af kanter. Et udspændende træ er en graf konstrueret på baggrund af G indeholdende alle punkter fra G. Figur 16 viser en simpel sammensat graf G.



Figur 16: Graf der kan indeholde et udspændende træ

Figur 17 viser et udspændende træ af grafen G, vist i Figur 16. Alle punkter fra den oprindelige graf G er medtaget i det udspændende træ. Den viste løsning er ikke den eneste løsning.



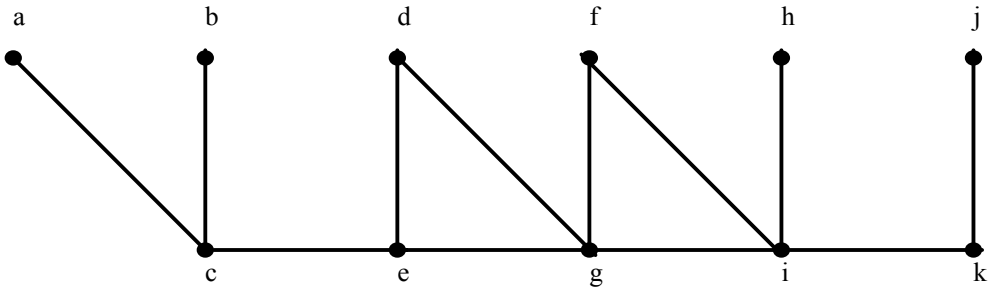
Figur 17: Udspændende træ på baggrund af graf i Figur 16

For en simpel graf gælder: En simpel graf er sammensat, hviss den har et udspændende træ.

Vi viser to algoritmer som kan finde udspændende træer i en graf. Den første er dybde-først og den anden er bredde-først. Dybde-først gennemgås først og derefter bredde-først.

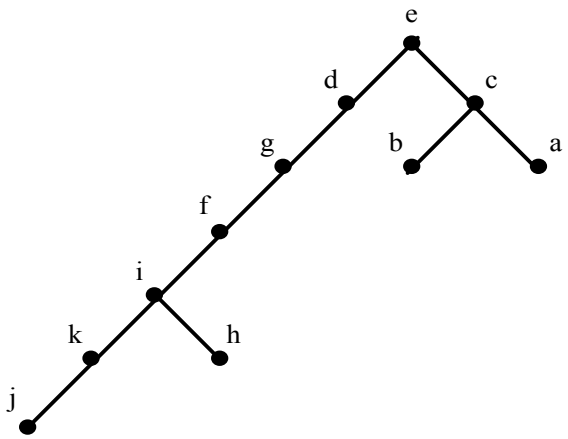
Dybde-først

Dybde-først algoritmen tager udgangspunkt i et hvilket som helst punkt i en simpel sammensat graf. Herfra tilføjes kanter, som ikke danner kredse. Hver ny kant er forbundet med det punkt der sidst er arbejdet med, som ikke allerede er tilføjet til træet. Der fortsættes sådan så længe det er muligt. Herefter følges kanterne tilbage til udgangspunktet. Hvis ikke alle punkter er forbundet tilføjes kanterne til disse kanter efterhånden som man bevæger sig tilbage af de kanter man allerede har medtaget. Hele forløbet gentages indtil alle punkter er medtaget i træet. Dybde-først er også kendt som backtracking, netop fordi man bevæger sig baglæns for at få alle punkter med. Figur 18 viser en sammensat simpel graf G.



Figur 18: Graf til dybde-først

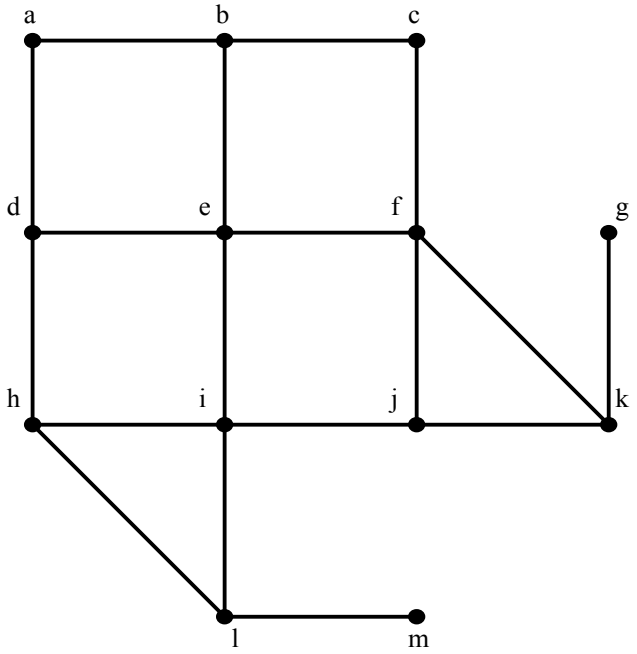
En udspændende træ konstrueret ud fra G med dybde-først algoritmen er vist i Figur 19. Der tages udgangspunkt i punkt e. Kanterne tilføjes i følgende rækkefølge: {e, d}, {d, g}, {g, f}, {f, i}, {i, k}, {k, j}, {h, i}. Derefter tilføjes {e, c}, {c, a}, {b, c}. Den viste løsning er ikke den eneste løsning.



Figur 19: Dybde-først træ på baggrund af Figur 18

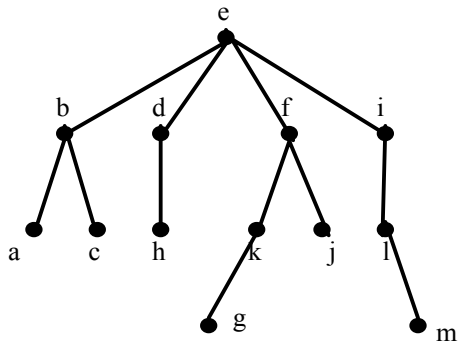
Dybde-først, er som navnet antyder, en algoritme som opbygger træet i så mange niveauer som muligt og derefter gør det bredere. I modsætning hertil findes bredde-først.

Bredde-først
Ved bredde-først vælges et hvilket som helst punkt i en simpel sammensat graf som udgangspunkt. Alle kanter som er forbundet til udgangspunktet vælges. Punkterne som er forbundet til roden udgør niveau 1. Herefter tages alle punkterne i niveau 1 som udgangspunkt og deres kanter vælges, hvor der ikke dannes kredse. Punkterne forbundet til niveau 1 udgør niveau 2. Igen vælges alle punkterne som udgangspunkt. Sådan forsættes indtil alle punkter er med i træet. Figur 20 viser en sammensat simpel graf G.



Figur 20: Graf til bredde-først

Figur 21 viser et udspændende træ på baggrund af grafen G vist i Figur 20. Der tages udgangspunkt i punkt e. Kanterne tilføjes i følgende rækkefølge: {e, b}, {e, d}, {e, f}, {e, i}, {b, a}, {b, c}, {d, h}, {f, k}, {f, j}, {i, l}, {k, g}, {l, m}. Den viste løsning er ikke den eneste.



Figur 21: Bredde-først på baggrund af Figur 20

Minimum udspændende træer

For et minimum udspændende træ gælder: Et udspændende træ af en sammensat vægtet graf der har den mindste mulige sum af kanternes vægt.

Vi viser to forskellige algoritmer som kan benyttes til at finde et minimum udspændende træ i en givet simpel sammensat graf.

Første algoritme er Prim's algoritme, her udtrykt i pseudokode:

procedure P(G: sammensat vægtet ikke-retningsbestemt graf med n punkter)
T := en minimum vægtet kant
for i := 1 **to** n-2
begin
e := en kant med minimum vægt, forbundet til et punkt i T. Må ikke danne en simpel kreds i T hvis den tilføjes T
T := T + e
end
Output er et minimum udspændende træ T af grafen G.

[DMIA, side 582]

Prim's algoritme tager som udgangspunkt en kant med den mindst mulige vægt fra grafen G. Derefter gentages følgende. Der tages en kant med mindst mulig vægt, som er forbundet med et punkt i T. Denne kant tilføjes T, såfremt den ikke vil danne et simpelt kredsløb i T når den tilføjes. Dette gentages lige så mange gange som der punkter i G fratrukket 2. Grunden til at det er 2 og ikke 1, er at T fra starten lægger beslag på én kant. For at antallet af kanter for T skal passe, skal der være antallet punkter i G fratrukket 1.

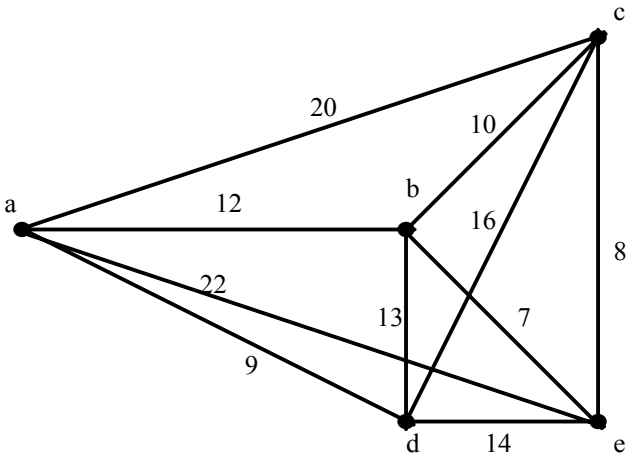
Anden algoritme er Kruskal's algoritme, her udtrykt i pseudokode:

procedure K(G: sammensat vægtet ikke-retningsbestemt graf med n punkter)
T := tomt træ
for i := 1 **to** n-1
begin
e := hvilken som helst kant i G med den mindste vægt, der ikke danner en simpel kreds, hvis den tilføjes til T
T := T + e
end
Output er et minimum udspændende træ T af grafen G.

[DMIA, side 584]

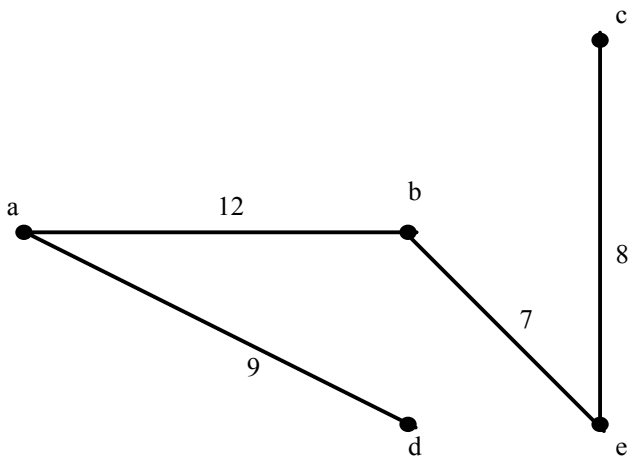
Kruskal's algoritme gentager følgende. Den/eller en af kanterne med mindst vægt tilføjes T, hvis den ikke vil danne en simpel kreds når den tilføjes. Dette gentages ligeså mange gange som der er punkter fratrukket 1 i G.

Figur 22 viser en sammensat vægtet graf G.



Figur 22: Sammensat vægtet graf

Med udgangspunkt i Figur 22 forklares hvordan et minimum udspændende træ dannes ved hjælp af henholdsvis Prim's og Kruskal's algoritme. Figur 23 viser det minimum udspændende træ. For Prim's algoritme gælder at kanterne tilføjes i følgende rækkefølge, med udgangspunkt i punkt b: {b, e}, {e, c}, {b, a}, {a, d}. For Kruskal's algoritme tilføjes kanterne i følgende rækkefølge: {b, e}, {e, c}, {a, d}, {b, a}. Her tages ikke udgangspunkt i noget punkt. Den samlede sum af kanternes vægt er 36.



Figur 23: Minimum udspændende træ på baggrund af Figur 22

6.4. Eulerkredse

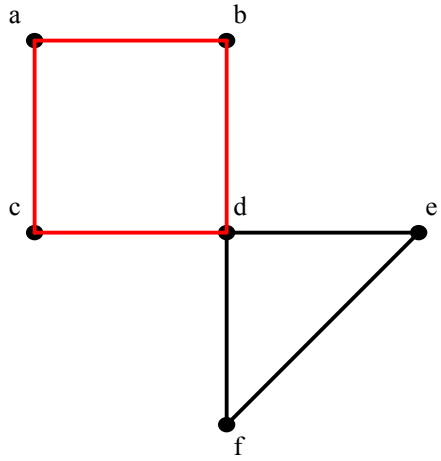
Det påstås at der er en Eulerkreds i en sammenhængende multigraf når alle punkter har lige valens. Vi vil nu bevise denne påstand.

En Eulerkreds begynder i punktet a. Kredsen fortsættes gennem punkt b. For at komme fra punkt a til punkt b benyttes kanten {a, b}. Kanten {a, b} tilføjer 1 til valensen for punkt a. Hver gang kredsen passerer et punkt må valensen tilføjes 2. For at komme til et punkt skal der være 1 kant og for at komme fra punktet må der være 1 kant. Når kredsen afsluttes tilføjes punkt a 1 i valens, da kredsen skal slutte i det punkt hvor den startes. Dette giver punkt a en valens som er lige. 1 til valensen ved start af kredsen og 1 til valensen ved slutning af kredsen. Eventuelt 2 til valensen hver gang startpunktet passeres under konstruktion af kredsen. Alle andre punkter end a vil have en lige valens, da der tilføjes 2 til valensen hver gang punktet passeres.

Det kan konkluderes at hvis en sammenhængende graf har en Eulerkreds, så har hver punkt lige valens.

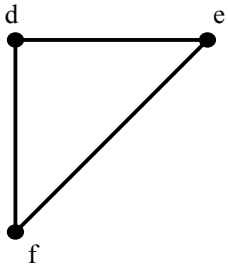
Vi vil nu bevise at det gælder for alle sammenhængende multigrafer at de har en Eulerkreds, hvis alle punkter har lige valens.

Antag at grafen G er en sammenhængende multigraf, hvor alle punkterne har lige valens. Der konstrueres en simpel kreds begyndende i et tilfældigt punkt a i grafen G. Lad $punkt_0$ = a. Der vælges en tilfældig kant {$punkt_0$, $punkt_1$} som er tilknyttet punkt a. Konstruktionen af den simple kreds {$punkt_0$, $punkt_1$}, {$punkt_1$, $punkt_2$}, … , {$punkt_{n-1}$, $punkt_n$} forsættes så længe det er muligt. Dette er illustreret i Figur 24. Kredsen startes i punkt a. Kanterne tilføjes i rækkefølgen: {a, c}, {c, d}, {d, b} og {b, a}.



Figur 24: Sammenhængende multigraf

Grafen har et begrænset antal kanter og derfor vil kredsen slutte. Den simple kreds starter med kanten {a, x} og slutter med {y, a}. Dette er fordi hver gang kredsen passerer et punkt med lige valens, så benyttes én kant til at komme til punktet og én kant til at komme fra punktet. Med startpunktet a benyttes én kant {a, c} til at komme fra a og kanten {b, a} til at komme tilbage til a.
Turen kan have benyttet alle kanterne eller også har den ikke. Er alle kanterne benyttet er der konstrueret en Eulerkreds.
I Figur 24 fjernes de kanter der allerede er benyttet til den simple tur, samt de punkter som ikke er forbundet til de resterende kanter. Følgende punkter fjernes: a, b og c. Kanter der fjernes {a, b}, {a, c}, {c, d} og {d, b}. Dette giver en delgraf H af G illustreret i Figur 25.



Figur 25: Delgraf H

Da grafen G, Figur 24, er sammenhængende må delgrafen H, Figur 25, have mindst et punkt fælles med den simple kreds der er fjernet. I dette tilfælde er det punktet d. I H har alle punkter lige valens. G har lige valens. Da den simple kreds der er konstrueret har benyttet et lige antal kanter fra punktet d, må valensen af d stadig være lige. I delgrafen H konstrueres en simpel kreds begyndende i d, ved at tilføje kanter så længe det er muligt. Kredsen slutter i punkt d. En tur kan være følgende: d, e, f, d.

Der konstrueres en kreds i grafen G, Figur 24, ved at tilføje kredsen fra delgrafen H, til den eksisterende simple kreds i G. Dette gøres ved punktet d, som er fælles for de to grafer. Den konstruerede kreds er som følger: a, c, d, e, f, d, b, a.
Delgrafen H skal ikke nødvendigvis være sammenhængende. Er den ikke dette forsættes indtil alle ikke-sammenhængende dele af H er en del af den nye kreds.

Ved at sammenlægge de to kredse fås en Eulerkreds.

For en sammenhængende multigraf gælder:
Der findes en Eulerkreds hviss hver af punkterne har lige valens.

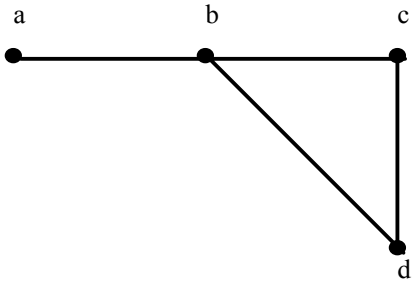
6.5. Hamilton

Først forklares hvad en Hamiltontur og -kreds er. Afslutningsvis vises hvordan man kan fastslå om der findes en Hamiltonkreds eller ej i en given graf.
Dette afsnit er udarbejdet på baggrund af 7.5 i DMIA.

Hamiltontur

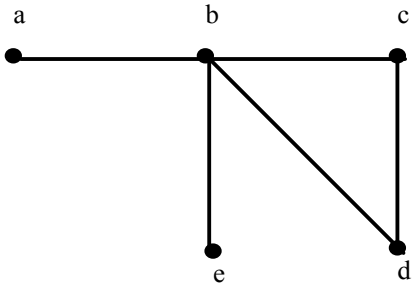
For Hamiltontur gælder:
Givet en graf G = (V, E). Med udgangspunkt i et af punkterne i mængden V, skal alle de andre punkter passeres én gang, og kun en gang. Det er et krav at slutpunktet ikke er det samme som udgangspunktet.

I Figur 26 er vist en graf med en med en Hamiltontur. Turen er givet ved følgende kanter {a, b}, {b, c}, {c, d}. Den foreslåede Hamiltontur er ikke den eneste løsning.



Figur 26: Graf med Hamiltontur

Til forskel viser Figur 27 en graf der ikke har en Hamiltontur. Grunden til at Hamiltonturen ikke findes er at det ikke er muligt at lave en tur i grafen, hvor alle punkter besøges uden at punkt b besøges to gange.



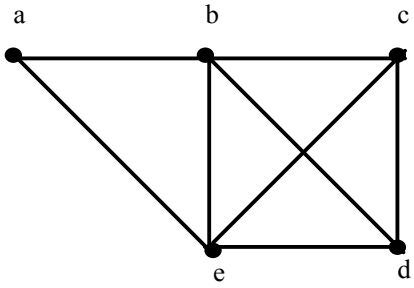
Figur 27: Graf uden Hamiltontur

Hamiltonkreds

I forlængelse af en Hamiltontur findes en Hamiltonkreds.

For en Hamiltomkreds gælder:
Givet en graf G = (V, E). Kravene til en Hamiltonkreds er, ligesom ved en Hamiltontur, at alle punkter i mængden V skal passeres én gang, og kun en gang, dog med undtagelse af udgangspunktet. Udgangspunktet skal også være det punkt der afsluttes i.

I Figur 28 er vist en graf med en Hamiltonkreds. Kredsen er givet ved følgende kanter {a, b}, {b, c}, {c, d}, {d, e}, {a, e}. Den foreslåede Hamiltonkreds er ikke den eneste løsning.



Figur 28: Graf med Hamiltonkreds

Til forskel viser Figur 26 en graf der ikke har en Hamiltonkreds. Grunden til at der ikke findes en Hamiltonkreds er at man ikke kan komme tilbage til udgangspunktet a uden at skulle passere b to gange. Grafen indeholder en Hamiltontur.

For en Hamitonkreds gælder:
Hvis G er en sammensat simpel graf med 3 eller flere punkter, så har G en Hamiltonkreds, såfremt valensen af hvert punkt er mindst n/2."
[DMIA, side 483]

Dette benyttes for at kunne fastslå om en Hamiltonkreds findes i en graf eller ej.
For at illustrere brugen af sætningen, vil den blive benyttet til at fastslå om der findes en Hamiltomkreds i Figur 28, hvilket der gør. Der findes 5 punkter i grafen, hvilket er mere end 3. Valensen af punkterne er følgende: deg(a) = 2, deg(b) = 4, deg(c) = 3, deg(d) = 3, deg(e) = 4. Da alle valenserne er mindst 2 (5/2 = $\lfloor 2{,}5 \rfloor \Rightarrow 2$) og der er 5 punkter findes der en Hamiltonkreds.
I Figur 27 kan vi ikke fastslå om der findes en Hamiltonkreds. Der findes 4 punkter i grafen, hvilket er mere en 3. Dette er tilfredsstillende. Valensen af punkterne er følgende: deg(a) = 1, deg(b) = 4, deg(c) = 2, deg(d) = 2, deg(e) = 1. Da valensen af a og e ikke er mindst 2 (4/2 = 2), kan vi ikke fastslå om der findes en Hamiltonkreds.

Sætningen kan være brugbar ved større grafer, som ikke umiddelbart er til at overskue. Sætningen kan ikke fastslå om der findes en Hamiltonkreds. Der findes ikke, på nuværende tidspunkt, en algoritme der kan løse opgaven. Der findes dog nogle enkelte udledninger.

For en Hamiltonkreds gælder:
En graf indeholdende et punkt med valensen 1, kan ikke have en Hamiltonkreds. Det er fordi man ikke kan komme videre fra punktet uden at skulle benytte en allerede benyttet kant. Hvis det er udgangspunktet kan man ikke komme tilbage til punktet og afslutte sin kreds.

For en Hamiltonkreds gælder:
Givet en graf indeholdende et punkt med valensen 2. Her skal begge kanter indgå i Hamiltonkredsen. Dette er for at punktet skal passeres og man skal kunne komme videre. Hvis det er udgangspunktet skal man kunne komme tilbage til punktet og afslutte sin kreds.

For en Hamiltonkreds gælder:
Når et punkt er passeret, kan man under den videre konstruktionen af en Hamiltonkreds, undlade at tage punktets resterende kanter i betragtning.

6.6. Tidskompleksitet

Når man taler om algoritmer er der en meget vigtig egenskab der gør sig gældende. Det er hvor lang tid en algoritme tager at eksekvere på en computer. Når man taler om eksekveringstid, menes ikke den nøjagtige tid i timer, minutter eller sekunder denne algoritme kører. Det man er interesseret i er hvor hurtigt algoritmens tidsforbrug vokser, når problemet vokser. Vil et dobbelt antal af elementer forøge tidsforbruget med en tiendedel, til det dobbelte eller med en faktor 1000. Til at beskrive en algoritmes tidskompleksitet bruger man noget kaldet store O notation, eller O().

O() notationen er en matematisk måde, hvorpå man kan udtrykke hvor meget tidsforbruget for en rutine vokser. Skriver vi at en rutine har en tidskompleksitet på $O(n^2)$, menes der at den højst tager $k*n^2$ tid at behandle n elementer. Siger vi en rutine er $O(n^2)$, er det worst case betragtning, afviklingstiden kan vokse langsommere. Det er den tungeste betingelse der er dominerende. Man ignorerer alle konstanter og multiplerende faktorer. $O(0,5 * n^2 + 3n)$ er det samme som $O(0,5 * n^2)$ som igen er det samme som $O(n^2)$.

Ens tidskompleksiteter for algoritmer er ikke direkte sammenlignelige. To algoritmer med tidskompleksiteten $O(n^2)$ kan have forskellige afviklingstider, da konstanterne kan være forskellige.

Et eksempel kunne være to algoritmer med henholdsvis kompleksiteten $O(0,5 * n^2 + 3n)$ og $O(100 * n^2 + 3n)$. Begge algoritmer kategoriseres som $O(n^2)$ men afviklingstiderne er vidt forskellige pga. konstanternes indvirken på afviklingstiden.

O() notationen bruges ikke kun til tidsestimater. Den kan også bruges til at repræsentere andre ressourcer en rutine bruger, f.eks. hukommelse.

6.7. Klassificering af problemer

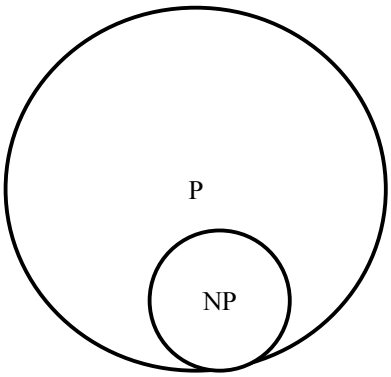
Problemer er placeret i forskellige kategorier. Kategorierne der beskrives er Polynomielle (P), Non-determiniske polynomielle (NP), NP-fuldstændige.

P og NP problemer

For at et problem kan placeres i P, skal problemet kunne løses af en algoritme i polynomiel tid. At et problem kan løses i polynomiel tid vil sige at tidskompleksiteten af problemet er $O(n^k)$ forhold til input n.
Et problem, der kan besvares med ja eller nej, er i NP, hvis der hver gang svaret er ja, findes en algoritme som kan kontrollere dette svar. Denne algoritme skal have en tidskompleksitet der er polynomiel, altså $O(n^k)$.
Problemer der er P er også NP. Dette er fordi at problemet kan besvares i polynomiel tid. Samme algoritme som besvarede problemet kan ligeledes kontrollere dette svar i polynomiel tid.

Et NP-fuldstændigt problem er et problem der er i NP og hvor alle andre NP-problemer kan omformuleres til et specialtilfælde af dette problem i polynomiel tid.

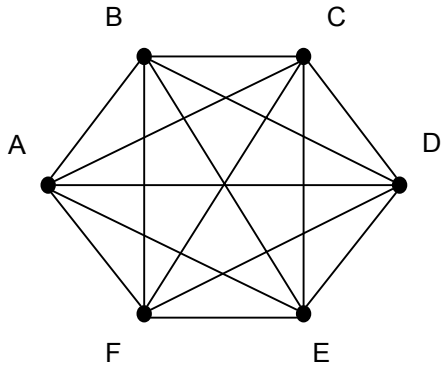
TSP problemet er NP-fuldstændigt. Der kendes ikke nogle algoritme, der kan bevise at en løsning af TSP er optimal, i polynomiel tid ($O(n^k)$).



Figur 29: Mængden af problemer i NP indeholdt i mængden af problemer P

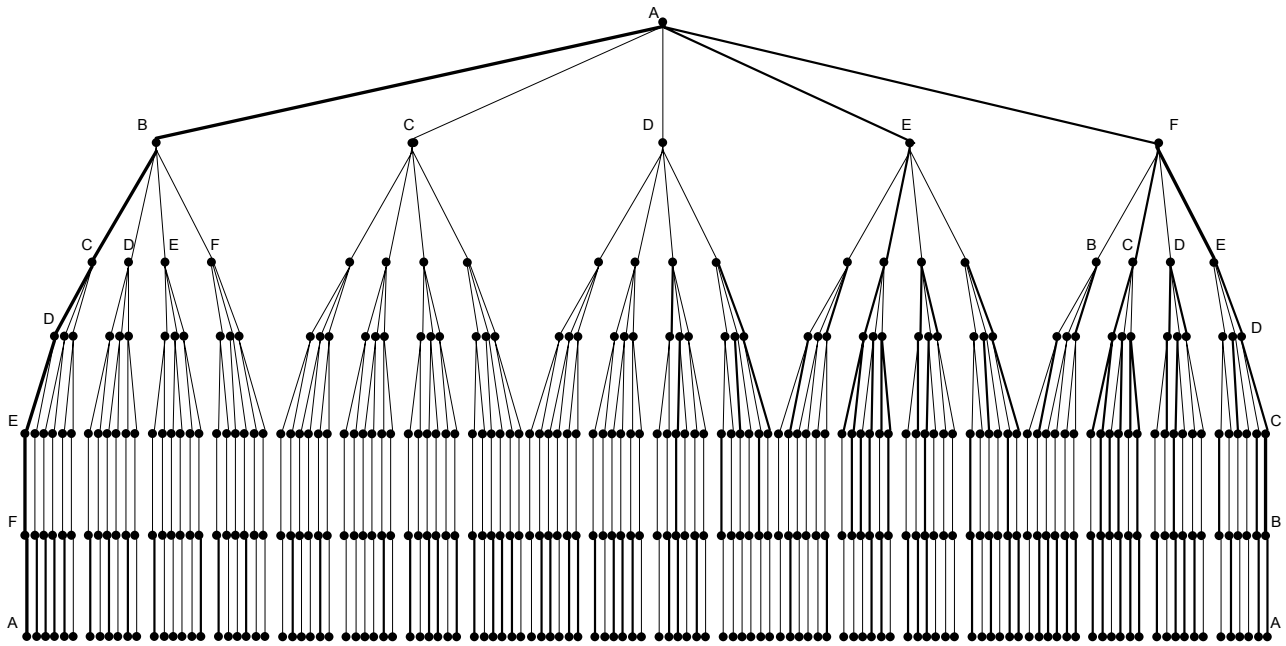
7. Brute Force

Først til vi se lidt på baggrunden af TSP problematikken, dernæst give et overblik af kompleksiteten af Brute Force.



Figur 30: Komplet graf med 6 punkter

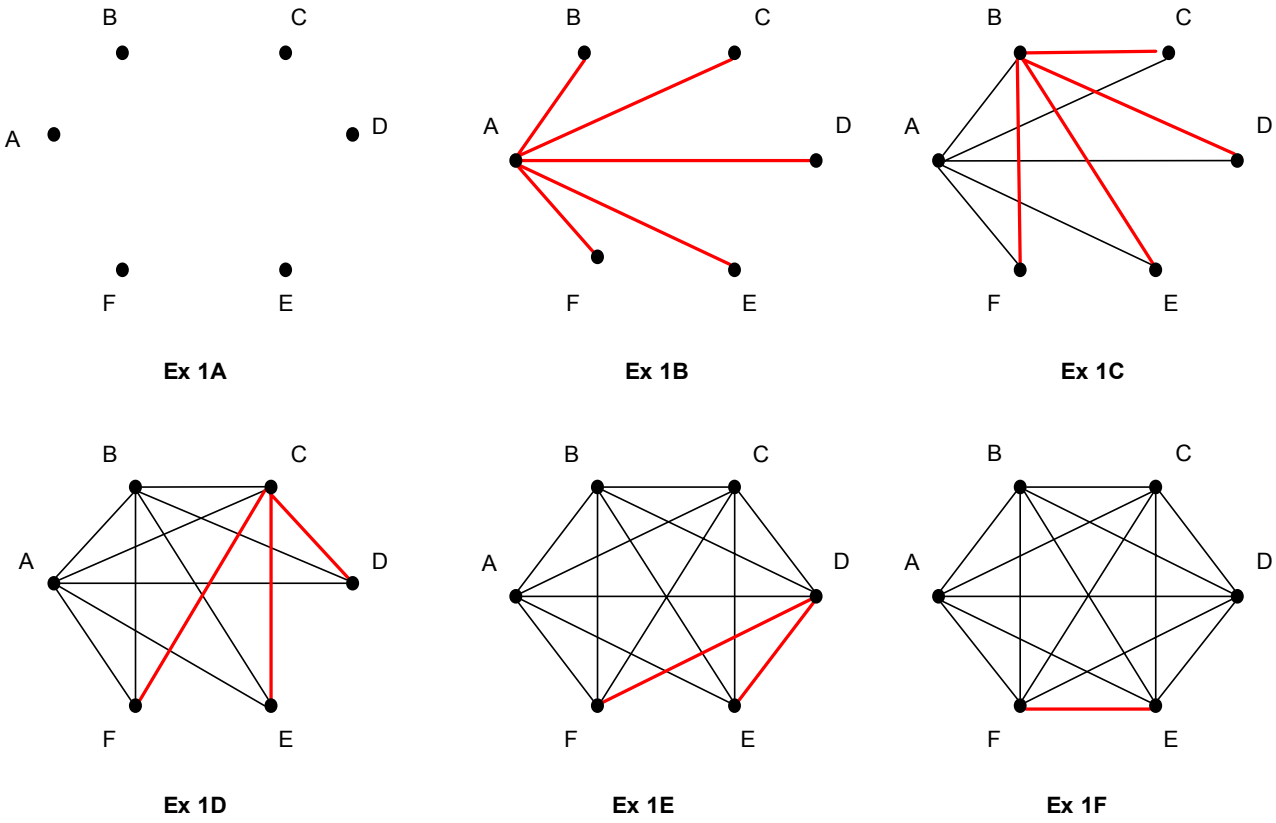
Ovenstående viser en komplet graf med 6 punkter. For at kunne bestemme den hurtigste Hamiltonkreds (fremefter benævnt som HK) med samme start og slutpunkt, er det nødvendigt med Brute Force, at beregne alle HK. Fra startpunktet er der 5 mulige veje at gå. Ved næste punkt er der for hvert punkt 4 mulige punkter (da HK først til sidst returnerer til udgangspunktet). Herefter er der 3 mulige punkter for hvert punkt (da kredsen ej heller må gå via tidligere besøgte punkter), dernæst 2 og endelig 1 punkt. Da HK har været via alle punkter, skal den returnere til udgangspunktet. Nedenstående figur viser træet for denne HK og de muligheder der er af HK for en graf med 6 punkter.



Figur 31: Træ for komplet graf i Figur 30

Ovenstående træ er antallet af mulige HK med udgangspunkt i punktet A. I alt er der fremkommet 120 mulige ruter. Her skal det bemærkes at ved beregning af samtlige HK, fremkommer hver mulig HK to gange. I ovenstående eksempel er to HK hhv. a, b, c, d, e, f, a og a, f, e, d, c, b, a angivet med fed streg. Disse HK er identiske, den ene er blot i modsat rækkefølge. Dette giver totalt 60 forskellige mulige HK i grafen. Havde grafen haft bare et punkt mere (i alt 7 punkter) havde ovenstående træ været en gren af et større træ, men 5 grene som denne og dermed havde grafen haft 360 forskellige mulige HK.

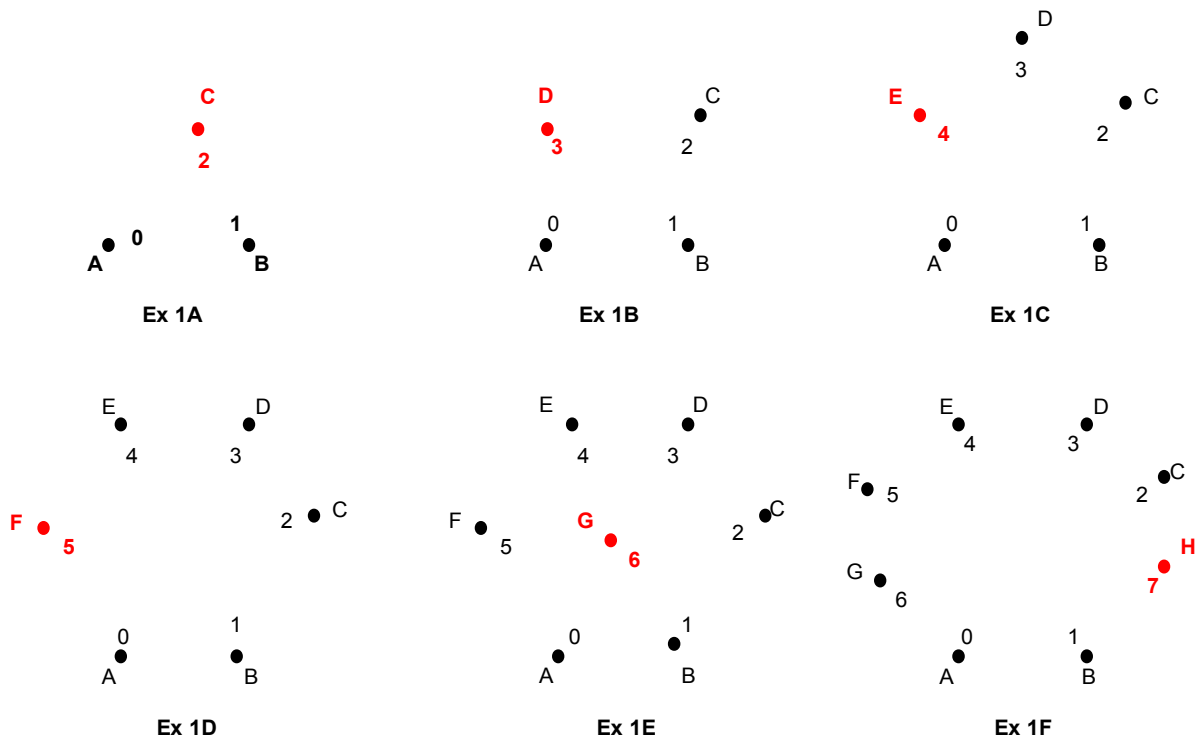
Nedenstående figur viser hvorledes en komplet ikke retningsbestemt graf opbygges og hvorledes antallet af kanter afhænger af antallet af punkter.



Figur 32: Tilgangen af kanter i en graf

Eksemplet i ovenstående figur har 6 punkter (ses i Ex 1A). Punkt A forbindes til de 5 øvrige punkter (Ex 1B), dette giver 5 kanter til grafen. Efterfølgende forbindes (Ex 1C) B til de punkter den ikke har kontakt med, da B allerede er i kontakt med A, kommer der kun 4 nye kanter til grafen. Dette gøres ligeledes med C, D og E, her kommer der hhv. 3, 2 og 1 kant til grafen. F er som sidste punkt allerede forbundet til A, B, C, D og E via de tidligere tegnede kanter.

Antallet af kanter stiger i forhold til antallet af punkter. Nedenstående ses eksempler med 3, 4, 5, 6, 7 og 8 punkter.



Figur 33: Tilgangen af kanter i en graf

Ovenstående figur viser at hver gang at der kommer et nyt punkt til grafen, så kommer der antallet af punkter -1 kanter til. Nedenstående ses formlen til at beregne antallet af kanter for hvert punkt i en komplet graf.

$$P(x) = \sum_{i=1}^{n} \frac{n(n-i)}{2}$$

Nedenstående ses formlen til at beregne antallet af kanter i en komplet graf, hvor n er antallet af punkter.

$$P(x) = \frac{n^2 - n}{2}$$

Antallet af kombinationer er som vist i dette afsnit, for en graf med 6 punkter er:(5 x 4 x 3 x 2 x 1) / 2 = 60, ved et fast udgangspunkt. Nedenstående er formlen for antallet af kombinationer af HK i en graf, hvor n er antallet af punkter.

$$P(x) = \frac{(n-1)!}{2}$$

For bedre at forstå kompleksiteten af Brute Force er der i nedenstående tabel angivet antallet af kombinationer for intervallet: 0 < n < 26.

n	Antal forskellige mulige kombinationer
1	0
2	1
3	1
4	3
5	12
6	60
7	360
8	2.520
9	20.160
10	181.440
11	1.814.400
12	19.958.400
13	239.500.800
14	3.113.510.400
15	43.589.145.600
16	653.837.184.000
17	10.461.394.944.000
18	177.843.714.048.000
19	3.201.186.852.864.000
20	60.822.550.204.416.000
21	1.216.451.004.088.320.000
22	25.545.471.085.854.700.000
23	562.000.363.888.804.000.000
24	12.926.008.369.442.500.000.000
25	310.224.200.866.620.000.000.000

Tabel 3: Sammenhæng mellem antallet af punkter (n) og mulige forskellige ruter

Det ses tydeligt at antallet af kombinationer vokser voldsomt. Tidskompleksiteten for Brute Force er O(n!). Eksempelvis kan nævnes at, hvis en rute med 25 punkter ønskes beregnet (24! /2) og hvis hver HK tager 1 nanosekund (10^{-9}) at beregne, vil det tage 10 millioner år at finde den korteste HK.

Det er med udgangspunkt i denne problemstilling, at vi ønsker at belyse hvorledes man kan nedbringe tidskompleksiteten og samtidige opnå en acceptabel HK.

8. Approksimerede løsningsforslag

Selvom det ikke er bevist at der ikke findes en algoritme, der i værste tilfælde af polynomiel tid kan løse et NP-problem, bør man prøve at finde en approksimations-algoritme, der kan løse ens problem, hvis det tilhører kategorien NP-fuldstændig.

En approksimationsalgoritme er ikke garanteret at finde den optimale løsning, men i stedet en tilfredsstillende løsning i forhold til tidskompleksiteten og en løsning der højst er k gange højere end den optimale løsning.

8.1. 2-gange minimum algoritmen

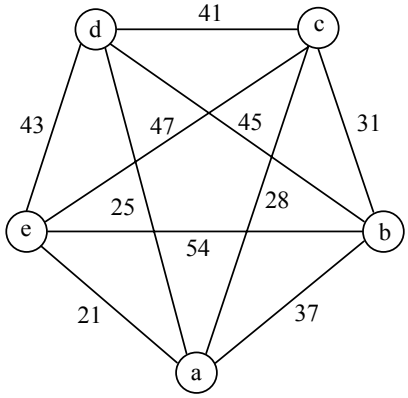
2-gange minimum er en tilnærmet algoritme der kan løse TSP. Algoritmen vil give et resultat der i værste tilfælde, vil give en kreds der er dobbelt så lang som den

optimale hamilton kreds. Det skal dog nævnes at algoritmen i langt de fleste tilfælde vil give en væsentlig bedre løsning. Algoritmen forudsætter en ikke orienteret, komplet graf med et antal knuder k, hvor k > 2, og grafen skal overholde trekantsuligheden. Algoritmen har følgende fremgangsmåde:

1. Et minimum udspændende træ findes i grafen.
2. Kanterne i træet fordobles. Herved skabes en multigraf.
3. I multigrafen skabes en eulerkreds.
4. Eulerkredsen omdannes ved hjælp af genveje til en hamiltonkreds.

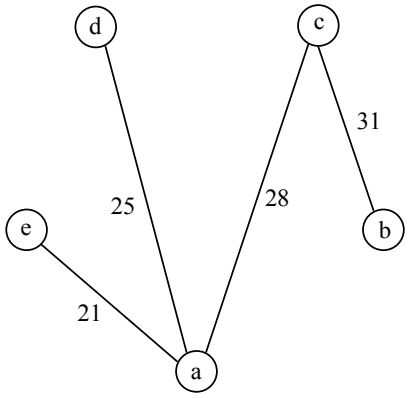
Vær indledningsvis opmærksom på at fjernes der en kant i en hvilken som helst Hamiltonkreds i en graf, vil det producere et udspændende træ af grafen. Dette træ vil aldrig kunne være mindre end et minimum udspændende træ. Af denne grund må den mindste Hamiltonkreds altid have større vægt end et minimum udspændende træ.

Efterfølgende vil der gives et eksempel på hvordan algoritmen virker. Givet er en komplet graf G, Figur 34.



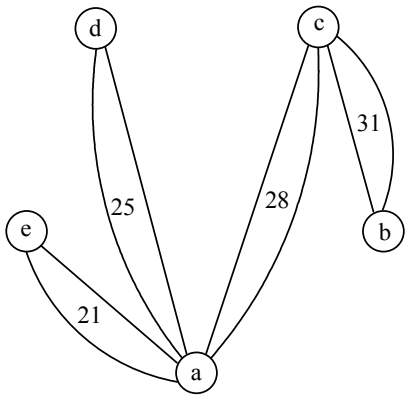
Figur 34: Den komplette graf G. hvori vi ønsker at finde en Hamiltonkreds

Ved hjælp af f.eks. Prims eller Kruskals algoritme kan et minimum udspændende træ findes. Dette giver os følgende graf T, Figur 35.



Figur 35: Det minimum udspændte træ T fundet i grafen G

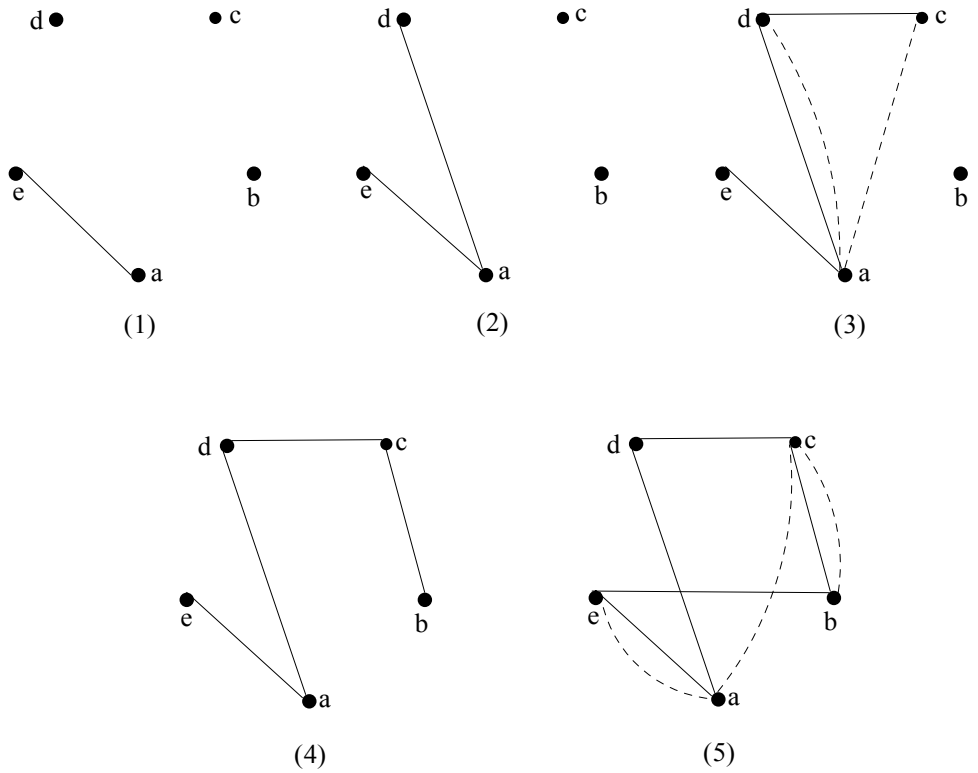
I grafen T fordobles samtlige de tilbageværende kanter for på denne måde at skabe en multigraf med mulighed for en Eulerkreds. Dette ville give os følgende graf T_2, Figur 36.



Figur 36: Grafen T_2 indholdene en Eulerkreds bygget omkring minimum udspændte træ

Da valensen af alle punkterne nu er lige og grafen er sammenhængende kan der i grafen nu skabes en Eulerkreds (se afsnit 6.4). Eulerkredsen kunne være {e,a,d,a,c,b,c,a,e}. Dette vil nu gøre os i stand til at besøge samtlige punkter i grafen G, ved at benytte hver kant én gang. Den samlede tur vil være den dobbelte længde af et minimum udspændende træ. Da længden af et minimum udspændende træ, som nævnt tidligere, altid vil være kortere end den optimale Hamiltonkreds, kan vi fastslå at kredsen maksimalt vil have den dobbelte længde af den optimale Hamiltonkreds.

Endnu er dette ikke en løsning på TSP, da det kræver at resultatet er en Hamiltonkreds. Dette løses ved at benytte genveje. N-kantsuligheden sørger for at en genvej aldrig vil blive længere en ruten tilbage over allerede besøgte punkter.

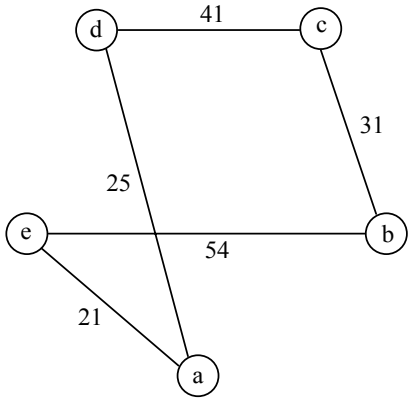


Figur 37: Hamiltonkredsen H findes ud fra Eulerkredsen T_2

Figur 37 viser hvordan Eulerkredsen i T_2 omdannes til Hamiltonkredsen H.

1. Først vælges et udgangspunkt. I dette tilfælde e. Eulerturen følges til det næste ubesøgte punkt a.
2. Fra a følges Eulerkredsen videre til næste ubesøgte punkt d.
3. Fra d er der ikke direkte kontakt til flere ubesøgte punkter. Eulerkredsen følges videre til det næste ubesøgte punkt. I dette tilfælde c. Punkterne d og c forbindes med en genvej. I dette tilfælde garanterer trekants uligheden at {d, c} ikke vil have højere vægt end {d, a} + {a, c}.
4. Turen følges til punktet b.
5. Alle punkter har nu været besøgt og start og slutpunkt forbindes. N-kantsuligheden garanterer at {b, e} ikke vil have højere vægt end {b, c} + { c, a} + { a, e}.

Vi ender derefter med følgende Hamiltonkreds som er 2-gange minimum algoritmens approksimerede løsning på TSP. Se Figur 38.



Figur 38: Hamiltonkredsen H fundet ved hjælp af 2-gange minimum algoritmen

Løsningen til TSP i grafen G, vil ifølge 2-gange minimum ende med vægten:

$$21 + 25 + 41 + 31 + 54 = 172.$$

2-gange minimum er en relativ hurtig algoritme til løsning af TSP. Algoritmen består af følgende operationer:

1. Finde minimum udspændte træ. – $O(n^2)$ [TTSP s. 155].
2. Fordoble samtlige kanter for at skabe en Eulerkreds. – $O(n)$.
3. Løbe Eulerkredsen igennem og skabe genveje, for til sidst at ende med en Hamiltonkreds. – $O(n)$ [TTSP s. 155].

Dette vil give en samlet tids kompleksitet på $O(n^2)$.

Bevis

Hvis H_{min} er den optimale Hamiltonkreds løsning til TSP, kan vi trække en kant fra H_{min}. Vi får så i stedet et udspændt træ. Lad os kalde det $H_{træ}$. Det minumum udspændende træ T i grafen, vil højst have samme vægt som $H_{træ}$. Vi kan derfor konkludere at:

$$Vægt(T) < Vægt(H_{min})$$

Lad T_2 være den multigraf der fremkommer ved at duplikere alle kanter i T. T_2 vil så indeholder en sammenhængende Eulerkreds af lige valens. Denne Eulerkreds vil have den dobbelte længde af det minimum udspændte træ.

$$Vægt(T_2) = 2 \cdot Vægt(T)$$

Sidst skabes Hamiltonkredsen $H_{løsning}$ ved at forbinde to punkter direkte i stedet for at gå tilbage over et allerede besøgt punkt. Denne genvej kan ifølge N-kantsuligheden ikke have højere vægt en den tidligere tur.

$$Vægt(H_{løsning}) \leq Vægt(T_2)$$

Deraf følger:

$$\boxed{Vægt(H_{løsning}) \leq 2 \cdot Vægt(T) < 2 \cdot Vægt(H_{min})}$$

Efterfølgende er 2 gange minimum udtrykt ved hjælp af pseudokode.

```
Procedure ToGangeMinimum(G: Graf med n punkter)
Begin
  {Følgende 6 linier er Kruskals algoritme til minimum udspændte træ}
  T := Graf med punkterne fra grafen G;
  For i := 1 to n-1 do
  Begin
    e := kanten med mindst vægt i G der ikke danner en simpel kreds;
    T := T + e;  {Kanten e tilføjes T}
  End;  {T er minimum udspændte træ i G}

  {Kanterne i T fordobles for at skabe en Eulertur}
  While (T har punkter med ulige valens) do
  Begin
    e := kant mellem to punkter i T hvor der kun er én kant;
    e:= 2 · e;  {Kanten fordobles}
  End;  {T indeholder en Eulertur skabt af minimum udspændte træ}

  {Eulerturen omdannes til en Hamiltonkreds}
  H := Graf med punkterne fra grafen T;
  P1 := vilkårligt punkt i grafen T;
  P2 := P1;  {P2 husker sidste punkt kun besøgt en gang}
  While (H ikke indeholder en Hamiltonkreds) do
  Begin
    If (Eulerkredsen kan følges fra P1 til endnu ikke besøgt punkt) then
    Begin
      P1 := næste ubesøgte punkt i Eulerkredsen i T;
      e := kant mellem P1 & P2;
      H := H + e;
      P2 := P1;  {P2 huskes som sidste punkt i Hamiltonkredsen}
    End else  {Der gås tilbage til et allerede besøgt punkt}
    Begin
      P1 := Eulerkredsen i T følges til tilbage til tidligere besøgt punkt;
      If (P1 = startpunkt) then  {Hamilton kredsen lukkes}
      Begin
        e := kant mellem P1 og startpunktet;
        H := H + e;
      End;
    End;
  End;

End; {H indeholder en Hamilton kreds med aproksimeret løsning til TSP}
```

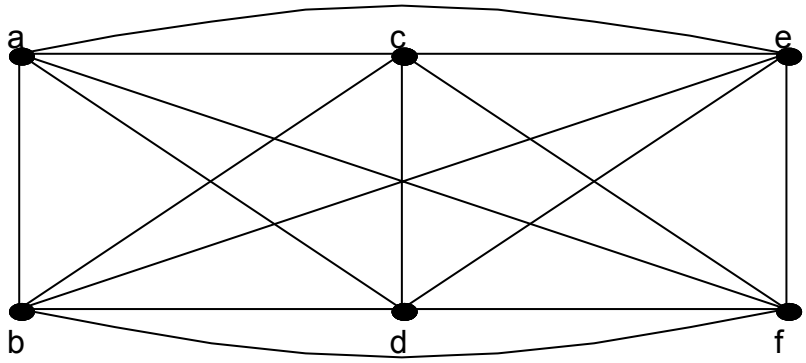
8.2. Christofides algoritme

Christofides algoritme giver et resultat der maksimalt er 3/2 gange så stort som den mindste Hamilton kreds. Vi kan vise dette ved at gennemgå de enkelte trin i denne algoritme.

1. Der konstrueres et minimum vægt udspændende træ
2. Alle punkter i træet med ulige valens findes og der konstrueres en delgraf indeholdende punkterne og kanterne som støder op til begge og der udføres en minimum vægt parring af delgrafen

3. Der skabes en forening af træet og minimum vægt parringen
4. Der konstrueres en Eulerkreds i denne nye forenede graf
5. Der konstrueres en Hamiltonkreds på baggrund af Eulerkredsen

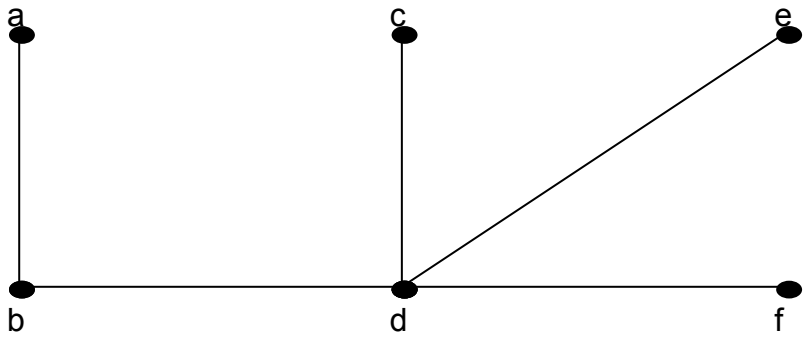
Der tages udgangspunkt i et eksempel, den komplette graf G. Algoritmen kan kun bruges i en komplet ikke orienteret graf hvor trekantsuligheden gælder (det gør den i G). Vægtene er ikke taget med for overskuelighedens skyld.



Figur 39: Grafen G

Ligesom i 2 gange minimum algoritmen tager denne algoritme udgangspunkt i at der konstrueres et minimum udspændende træ.

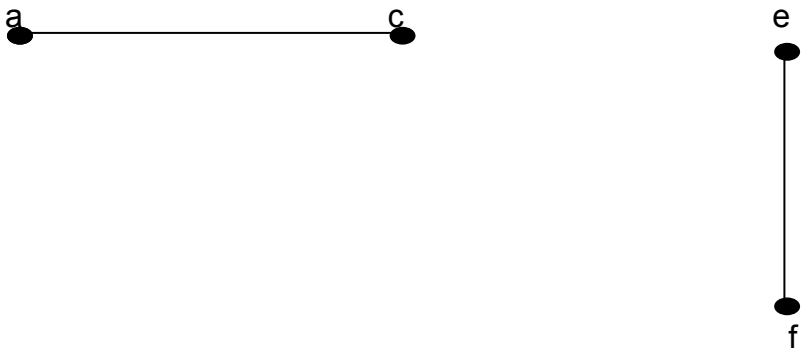
I dette eksempel kalder vi det minimum udspændende træ for T_{min}. Fordelen ved dette træ, er at det altid vil være den tur mellem alle punkter med mindst vægt. Og at et sådant træ kan laves med Prims algoritme og en tidskompleksitet på $O(n^2)$.



Figur 40: Minimum udspændende træ T_{min}

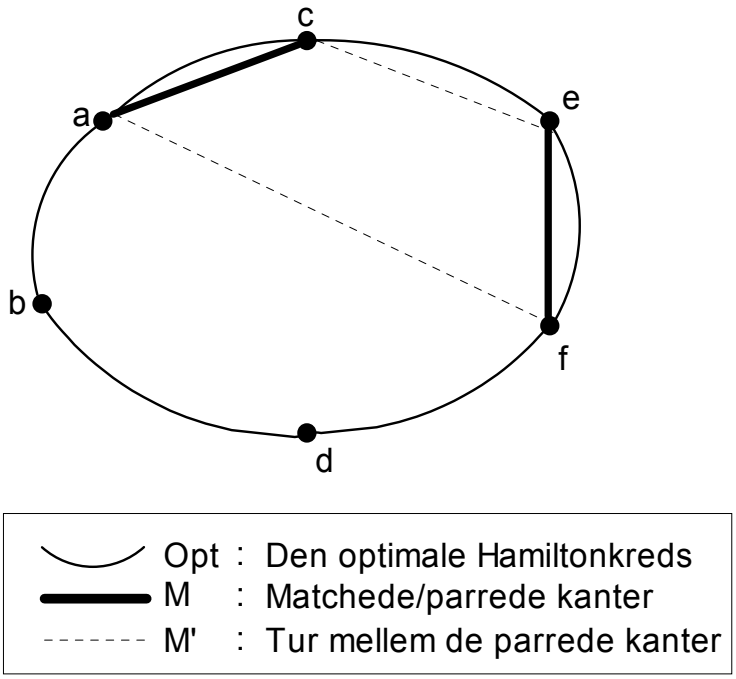
Her adskilles chiristofides og 2 gange minimum algoritmen, ved at i Christofides findes alle punkter i træet med ulige valens med henblik på en senere minimum vægt parring af disse.

Træet T_{min} : Punkter med ulige valens i T_{min}: deg(a)=1, deg(c)=1, deg(e)=1, deg(f)=1. Minimum vægt parring af disse punkter giver at summen af vægten af kanterne {a,c} og {e,f} giver den mindste vægt for punkterne med ulige valens.



Figur 41: Graf M: Minimum vægt parring af punkter med ulige valens i T_{min}

Denne minimum vægt parring har en tidskompleksitet på $O(n^3)$. Minimum parringen sikrer at alle punkter i grafen har lige valens, så snart alle punkter har lige valens er det muligt at konstruere en Eulerkreds. Samtidig gælder det pga. trekantsuligheden at minimum vægt parringen er mindre end eller lig med ½ gange den optimale rute mellem punkterne med lige valens. Vi vil vise at dette er sandt.



Figur 42: Matching af punkter

På grund af trekantsuligheden gælder det at:

$$(1)\ \text{Vægt(M)} + \text{Vægt (M')} \leq \text{Vægt (Opt)}$$

Da parringen sker med de mindste kanter må det også gælde at:

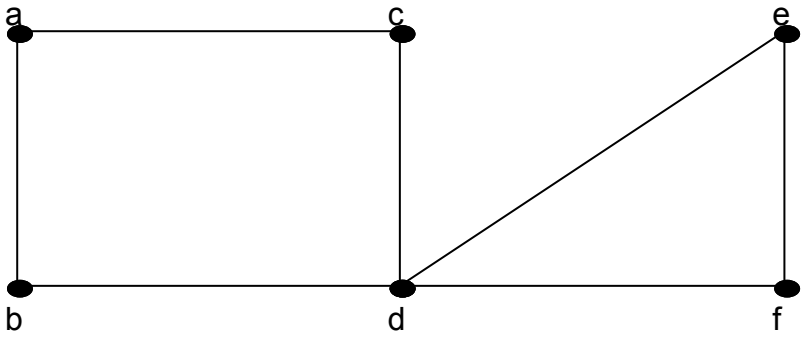
$$(2)\ \text{Vægt (M)} \leq \text{Vægt (M')}$$

Ud af det kan vi udlede følgende:

(fra 1)	Vægt (M) ≤ Vægt (Opt) - Vægt (M')
(Vægt(M) fra 2)	Vægt (M) ≤ Vægt (Opt) - Vægt (M)
(Vægt(M) flytter side)	2Vægt (M) ≤ Vægt (Opt)
(Dividerer med 2)	Vægt (M) ≤ ½ Vægt (Opt)

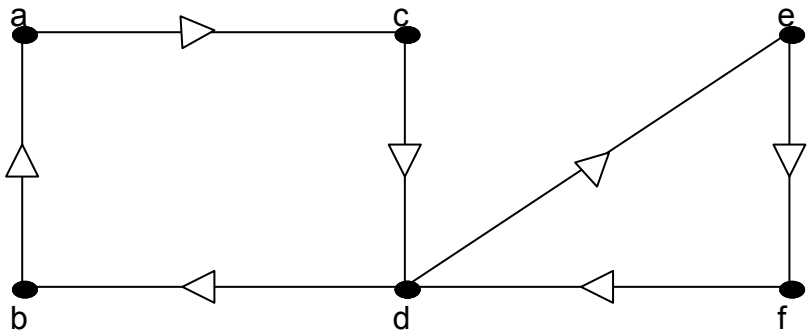
Heraf kan det ses at Vægt(M) altid er mindre end eller lig med en halv gange den optimale Hamiltontur.

For at kunne løse vores problem som er TSP, er det nu nødvendigt at skabe en Hamiltonkreds. For at kunne skabe en Hamiltonkreds er vi dog nødt til først at lave en Eulerkreds. For at lave denne skal vi bruge en graf TM som er en forening af grafen M og grafen T_{min} for at udnytte at Eulerkredsen maksimalt bliver 1½ gange den mindste Hamiltonkreds ved at sammenlægge parringen som tillægger maks. ½ og træet som tillægger maks. 1.



Figur 43: Graf TM: Forening af T og M

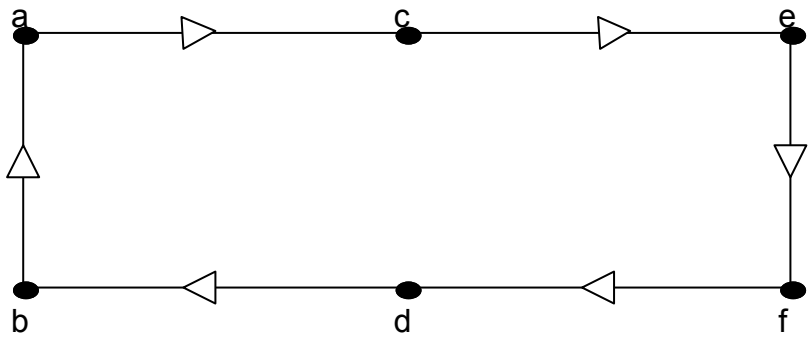
Denne graf vil altid indeholde en Eulerkreds da alle ulige valenser har fået tilføjet 1 kant. Vi bruger derefter algoritmen til konstruktion af Eulerkredsen, afsnit 6.4.



Figur 44: Eulerkreds i grafen TM

Når Eulerkredsen er lavet kan vi lave den om til en Hamiltonkreds vha. algoritmen indeholdt i 2 gange minimum, afsnit 8.1. Man skal naturligvis benytte sig af at TM er en del af den komplette graf G og det er derfor det er muligt at konstruere en Hamiltonkreds ud fra graferne TM og G.

Hamiltonkredsen H kommer til at se således ud. Algoritmen udnytter ligesom parringen, trekantsuligheden, derfor kan vægten af den fundne hamiltonkreds aldrig blive større end Eulerkredsen. Ser man på eksemplet herunder kan man se at vejen fra c til e, går gennem kanten {c,e}. Det ses hermed tydeligt at vejen er kortere hvis der ses på Figur 44 hvor Eulerkredsen må gennem kanten {c,d} og {d,e} for at nå fra c til e.



Figur 45: Løsning 3/2 metoden: Hamilton tur i grafen TM

Denne løsning, er den approksimerede løsning til grafen G. For at beregne tidskompleksiteten, for hele Christofides algoritme, er det nødvendigt at se på de enkelte delelementer.
Vi ser igen på hele algoritmen samlet:

1. Der konstureres et minimum vægt udspændende træ
   Tidskompleksiteten for Prims algoritme er $O(n^2)$

2. Alle punkter i træet med ulige valens findes og der konstrueres en delgraf indeholdende punkterne og kanterne som støder op til begge og der udføres en minimum vægt parring af delgrafen.
   Tidskompleksiteten for at finde alle punkter i træet med ulige valens er O(n) og parringen er $O(n^3)$ [TTSP, side 162]

3. Der skabes en forening af træet og minimum vægt parringen
   Igen tidskompleksiteten O(n)

4. Der konstrueres en Eulerkreds i denne nye forenede graf.
   Konstruktion af Eulerkreds er også O(n)

5. Der konstrueres en Hamiltonkreds på baggrund af Eulerkredsen via shortcuts
   Denne del har også tidskompleksiteten O(n)

Sammenholder man disse kompleksiteter er det klart at tidskompleksiteterne for matching af ulige punkter er dominerende. Matching af ulige punkter har kompleksiteten $O(n^3)$ og den samlede tidskompleksitet for algoritmen er derfor $O(n^3 + n^2 + 4n)$, hvilket svarer til $O(n^3)$. Og dette er tiden for at opnå en approksimeret løsning på under 3/2 gange den mindste Hamiltonkreds.

8.3. Heuristiske algoritmer

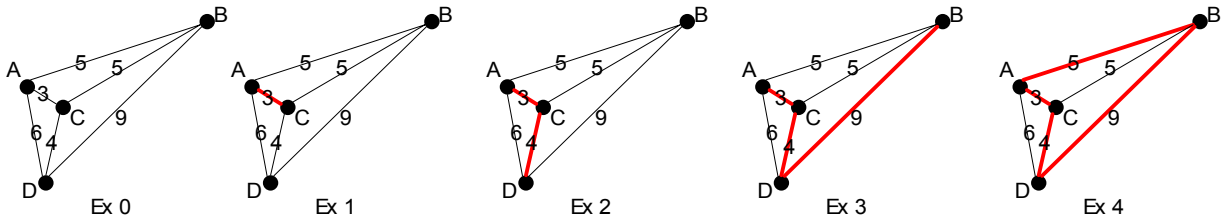
Betegnes som algoritmer der ikke med garanti giver optimale løsninger og som man ej kan bevise at virker, men som kan give et til fredsstillende resultat og som er en tidsbillig approksimation i forhold til Brute Force algoritmen. Vi tager udgangspunkt i to typer heuristiske algoritmer; en type genererer en rute ud fra et udgangspunkt, den anden type tager udgangspunkt i en tur og tilføjer punkter til turen.

Vi vil først beskrive Nearest Neighbor, som er af den første type. Dernæst, vil vi beskrive Nearest insertion der er af den anden type. Denne er kategoriseret med to andre algoritmer; farthest- og cheapest insertion, som ikke gennemgås her. Fælles

for indsættelses algoritmerne er at de tager udgangspunkt i en tur med to punkter. Metoden de arbejder på er identiske, blot er kriteriet for at indlemme yderligere punkter i den eksisterende tur forskellig for algoritmerne. Da de alle er varianter af hinanden, vil vi kun se på hvorledes nearest insertion virker.

Nearest Neighbor
Ud fra et tilfældigt valgt punkt vælger Nearest Neighbor algoritmen (fremefter benævnt som NN) det punkt som ligger nærmest, dette gentages for alle punkter indtil alle punkter er indeholdt i turen, bemærk at algoritmen ikke må returnere til allerede besøgte punkter.

Øjensynligt lyder algoritmen fornuftig, men algoritmen har den svaghed at den let kan isolere sig i et hjørne af grafen og når de nærmeste punkter er taget, så er der kun yderpunkter tilbage at besøge.



Figur *46: NN-algoritmen*

Ex 0 viser grafen NN-algoritmen skal finde ruten i. Først vælges kant {A,C}, da denne er kortest, herefter i Ex 2 vælges {C,D}, endelig vælges {D,B} og ruten afsluttes i Ex 4 med {B, A}. Den fundne rute er i alt 21. Havde denne graf været løst via Brute Force ville den funde tur være A, D, C, B, A. Dette havde givet en rute på i alt 20. I dette eksempel finder algoritmen ikke den optimale Hamiltonkreds.

Kompleksiteten af NN-algoritmen er $O(n^2)$, hvor n er antallet af punkter. Argumentationen er at NN-algoritmen for hvert punkt den møder, skal sammenligne alle punktets naboer og da grafen er komplet vil dette være n-1 sammenligninger for hvert punkt.

NN-algoritmen er nem at implementere og er blandt de hurtigste. Hvor kravet om en hurtig løsning vejere tungere end kravet til den fremkomne løsning og hvor antallet af punkter er højt, er algoritmen en umiddelbar mulighed.

Efterfølgende er Nearest Neighbor udtrykt ved hjælp af pseudokode.

```
NearestNeighbor(G: Komplet ikke orienteret graf med n punkter)
begin
    R = Grafen G uden kanter
    min = punkt i grafen
    start = Et punkt i grafen fra kant med mindste vægt

    for i=0 to n-1 do

    begin
            min = punktet med mindste kant til punktet start til hvilket der ikke
            i forvejen er en sti i R.
       R := R samt kanten mellem min og start
       If der er en sti til min then
          start := min        // startpunktet sættes til det nye punkt
    end;

R := R med en kant mellem start og første punkt i grafen

end; {R har en hamiltonkreds}
```

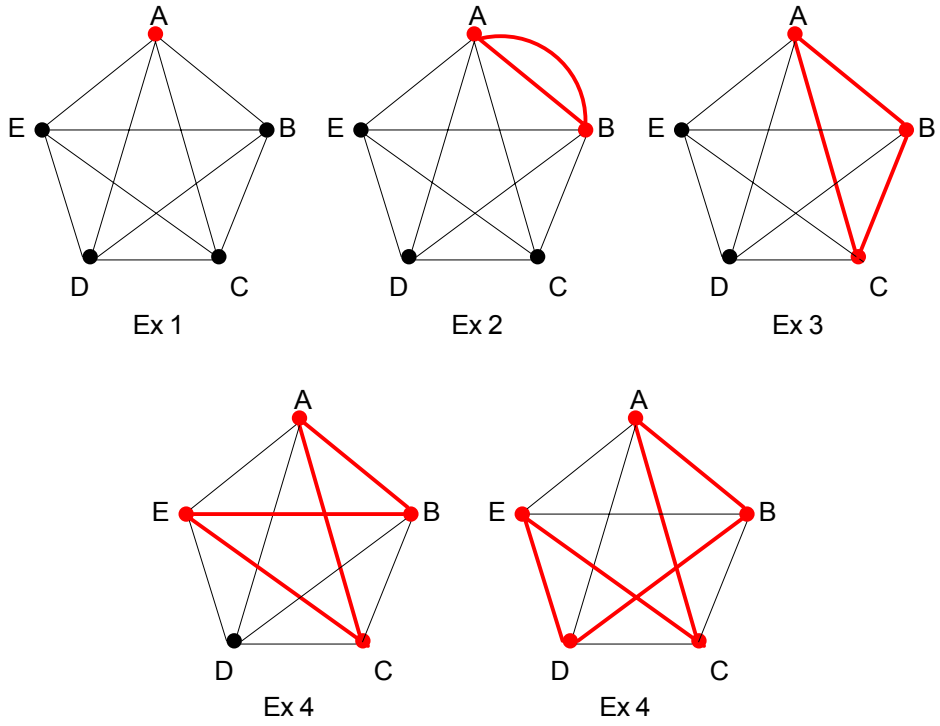
Nearest insertion
I nearest insertion (fremefter benævnt som NI) vælges to punkter som har den korteste afstand til hinanden. Igen findes for hvert punkt ikke medtaget i turen, den minimale afstand til et af punkterne indeholdt i turen. Dette gøres indtil at alle punkter er indeholdt i turen.

Eksempel
Nedenstående matrix viser afstanden mellem alle punkter i grafen G.

	A	B	C	D	E
A	-	1	2	5	4
B		-	3	4	3
C			-	3	2
D				-	3
E					-

Tabel 4: Matrix for afstandene i Tabel 51: Nearest insertion

Afstandene mellem punkterne i Figur 47 er ikke medtaget af hensyn til overskueligheden.



Figur 47: Nearest insertion

Ex 1: Først vælges A som udgangspunkt for turen.

Ex 2: Herefter vælges B, da B er det punkt er tættest på A, Kredsen K vil så indeholde A og B. Kanten {A, B} dobles for at danne en kreds således at K indeholder kanterne {A, B} og {B, A}.

Ex 3: Igen vælges det punkt er tættest på K, i dette tilfælde C. Da C er nærmest A indskydes C mellem A og B og kanterne i K vil således være {A, B}, {B, C} og {C, A}.

Ex 4: Der er nu to punkter tilbage, der ikke er medtaget i K, hhv. D og E. Den korteste afstand fra D til et punkterne i K er 3 og for E er den korteste afstand 2 (til C). Der beregnes for K, hvor E skal indsættes for at skabe den mindste tilgang til summen af kanter i K.

Sum({A, E} + {E, B} – {A, B}) = 6
Sum({B, E} + {E, C} – {B, C}) = 2
Sum({A, E} + {E, C} – {A, C}) = 4.

E indsættes mellem B og C da dette giver den mindste tilgang til K.

Ex 5: Endeligt skal D tilføjes K. Dette sker ved samme beregninger:

Sum({A, D} + {B, D} – {A, B}) = 8
Sum({B, D} + {E, D} – {E, B}) = 4
Sum({E, D} + {C, D} – {E, C}) = 4
Sum({C, D} + {A, D} – {A, C}) = 6

D indsættes mellem B og E da dette giver den mindste tilgang til K. Den endelige K er så A, B, D, E, C, A, med vægt på 12.

Tidskompleksitet
Først findes det punkt der er nærmest startpunktet, dette giver n-1 sammenligninger af afstande, hvor n er antallet af punkter. Der tilføjes to kanter mellem startpunktet og det fundne punkt for at danne en kreds.

Herefter foretages der ved første punkt som indsættes i kredsen n – 2 sammenligninger (i alt 3) og n – 3 beregninger (i alt 2).

Herefter foretages n – 3 sammenligninger (i alt 2) og for det optimale hvert fundet punkt, foretages n – 2 beregninger (i alt 3) af tilgangen til den eksisterende kreds.

Næste gennem løb giver n – 4 sammenligninger (i alt 1) og n – 1 beregninger (i alt 4).

Heraf kan nedenstående formel til beregning af antal beregninger i algoritmen udledes.

$$P(x) = \sum_{i=2}^{n}((n-i)\cdot i)+i$$

Figur 48: Formel til beregning af antallet af beregninger

For hvert punkt ikke indeholdt i grafen, gennemløbes afstanden til et af punkterne i grafen, hvilket giver en tidskompleksitet på $O(n^2)$. Indeholdt i dette gennemløb er beregningen af hvor det fundne punkt skal indsættes i kredsen, hvilket har en tidskompleksitet på O(n). Da denne funktion er indeholdt i det foregående gennemløb giver det en samlet tidskompleksitet på $O(n^3)$.

Fælles for algoritmerne er at tidskompleksiteten er $O(n^3)$. Hver gang der indsættes et punkt i ruten, gennemløbes der for hvert punkt ikke indeholdt i grafen afstanden til samtlige punkter indeholdt i turen. Nedenstående udledes tidskompleksiteten af ovenstående formel til beregning af antallet af beregninger for algoritmen.

$$\sum_{i=2}^{n}((n-i)\cdot i)+i \le \sum_{i=2}^{n}(n-i)\cdot i \le \sum_{i=2}^{n} n\cdot i \le \sum_{i=2}^{n} n\cdot n$$

$$= \overbrace{n\cdot n + n\cdot n + n\cdot n + n\cdot n + ... + n\cdot n}^{n-2}$$

$$= n\cdot n\cdot n$$

$$= n^3$$

Figur 49: Tidskompleksiteten for algoritmen

Tidskompleksiteten er et udtryk for hvordan algoritmens afviklingstid vokser ved ændring af n. I dette tilfælde har algoritmen en tidskompleksitet på $O(n^3)$. Tidskompleksiteten er en worst-case betragtning. Ved n=100, er n^3=1.000.000, men rent faktisk er det eksakte antal beregninger for algoritmen 171.402.

Efterfølgende er Nearest Insertion udtrykt ved hjælp af pseudokode.

```
Procedure NearestInsertion (Graf G)
// Pre:  G=(V, E), hvor G er en ikke retningsorienteret multigraf
// Post: en Hamiltonkreds for grafen G
Begin
    i := tilfældig valgt punkt i mængden V  {startpunktet}
    j := det punkt der har den mindste afstand til i
    C:= G(V,Ø) {Grafen C sættes lig med G, med samme punkter, men uden
kanter}
    Kanten { i, j} og { j, i} indsættes i X
    While V_C > E_C
    Begin
        Lad r være det punkt der har den mindste afstand til punkterne i kredsen C
        For i=0 to Antal(E_C) do
        Begin
            Lad e_i være den i'ende kant i C
            Lad u og k være de punkter i V_C der er forbundet via e_i
            Lad v være Sum({u, r}+{k, r}-{u,k})
            Lad v_opt være den laveste af alle v
        End;
        For v_opt, tilføj (u, r) og (k, r) til C og slet (u, k) fra C
    End;
End;
```

9. Teoretisk anbefaling

Nedenfor laves en sammenligning af de gennemgåede algoritmer. Algoritmerne sammenlignes på baggrund af deres tidskompleksitet, samt hvor langt fra den optimale løsning algoritmens svar garanteres at ligge.

Algoritme	**Tidskompleksitet**	**Kvalitet**
Brute Force	O(n!)	Optimal
2 Gange Minimum	$O(n^2)$	2 · Optimal
Christofides	$O(n^3)$	3/2 · Optimal
Nearest Neighbor	$O(n^2)$	∞ · Optimal
Nearest Insertion	$O(n^3)$	∞ · Optimal

Tabel 5: Sammenligning af algoritmer

Kvaliteten af løsningerne kan kategoriseres i tre katogorier. Den eksakte løsning, løsninger der kan bevises at ligge inde for et bestemt interval, samt algoritmer der ikke garanterer noget, men i de fleste tilfælde vil give et fornuftigt resultat.

Skal der vælges en algoritme ud af ovenstående, anbefales det at der vælges en algoritme der garanterer at den fremkomne løsning har en øvre grænse i forhold til det optimale løsning. I forhold til vores problemstilling anbefales det at der tages udgangspunkt i 2 Gange Minimum algoritmen. Denne har et garanteret udfaldsområde og samtidig den laveste tidskompleksitet. Løsningen kan bruges som udgangspunkt og løbende optimeres/erstattes af de øvrige algoritmers løsninger.

10. Faselinie 2 – Problemløsning

Gennemgang af den anvendte teori, viste sig at indeholde teori ikke senere brugt i fasen. Problemet var at projektgruppen ikke på forhånd vidste hvilket teori var relevant at medtage i rapporten, da problemområdet på forhånd var usikkert. Vi har valgt ikke at fjerne de passager det drejer sig om. Problematikken omkring TSP er blevet synliggjort.

Af gennemgåede alternative løsningsmodeller er; 2 Gange Minimum, Christofides, Nearest Neighbor og Nearest Insertion. Algoritmerne er valgt udfra gennemgåede teori i undervisningen, samt fra udleverede materiale. Der er algoritmer som projektgruppen kunne have tænkt at anvende, men da dokumentation for disse ikke kunne fremskaffes, er disse undladt. Dette er også gældende for matching algoritmen, som er en del af Christofides algoritme.

10.1. Definition af fase 3 – Implementering

Nedenstående er definition af tredje fase. Fasen indeholder implementering af udvalgte algoritmer fra fase 2. Vores anbefaling til valg af algoritme vil ske på baggrund af gennemførte test.

Fase 3	Implementering	
Aktiviteter	• Implementering • Anbefaling	
Ressourcer	• Projektgruppen • Løbende projektmøder	
Faselinie	Implementering af GUI, grafmodel og algortimer	
Betydning	Funktionsdygtig program med algoritmer implementeret.	
Vurderings-Procedure	Hver aktivitet/produkt godkendes ved enstemmighed, med mindre at anden vurderingsprocedure specifikt er angivet	
	Produkt	**Vurderingskriterium**
	Implementering	• GUI. Skal kunne tegne punkter og kanter, samt tilbyde mulighed for at vælge en algoritme at teste. • Algoritmer. 2 Gange Minimum, Nearest Neighbor og Nearest Insertion • Graf klasse. Skal repræsenterer en ikke orienteret vægtet multigraf.
	Anbefaling	• Anbefalinger skal ske på baggrund af tests foretaget med implementerede algoritmer.
Klargøring	• Produkter og aktiviteter for næste fase diskuteres, defineres og uddelegeres • Revurdere og tilrette estimater og tidsplan	

Tabel 6: Fase 3 – Implementering

11. Implementering

Alt beskrivelse vedrørende brugerfladeprogrammering er undladt i denne rapport, da vi ikke mener at det er relevant for det matematiske i denne rapport.

Til at modellere en graf er der konstrueret en graf- og en punktklasse. Punktklassen giver mulighed for at modellere et punkt med følgende oplysninger: en identifikation samt et koordinat. Grafklassen implementerer en graf ved at oprette og kontrollere det antal instanser af punktklassen som der bedes om. Oplysninger om kanter holdes i adjecentmatrix. Grafen der skal modelleres må ikke være retningsbestemt eller være en multi- eller pseudograf

Følgende funktionalitet findes i graf klassen:

- Generering af tilfældig graf med ønskede antal punkter, samt beregning af kantvægt (længden) i forhold til punkternes koordinater.
- Sletning og indsættelse af kant.
- Forespørgsel på hvorvidt to punkter er naboer.
- Forespørgsel på hvorvidt der findes en tur mellem to punkter.
- Konstruktion af minimum udspændende træ ved Kruskals algoritme.

I henhold til den teoretiske gennemgang af to gange minimum fremgår det bl.a. at algoritmen har en tidskompleksitet på $O(n^2)$. Det er i visse kilder fremgået at to gange minimum ville kunne løses med en tidskompleksitet på $O(n \log(n))$. Vi har dog ikke været i stand til at finde frem til en kilde, da på en troværdig måde kunne dokumentere denne påstand. Vi har desuden heller ikke i vores implementation været i stand til at få den under $O(n^2)$. Dette skyldes at vores implementerede løsning benytter en nestet løkke. Dvs. en løkke kørende til n, indholdene en anden løkke kørende til n.

Med hensyn til implementationen, adskiller to gange minimum sig også fra den måde algoritmen teoretisk arbejder på. To gange minimum består bl.a. af at lave et minimum udspændt træ til en Eulerkreds ved at ligge alle kanter dobbelt. Vi var dog begrænset af at vores graf implementation ikke kunne håndtere multigrafer. Det var med andre ord ikke muligt at have mere end én kant mellem to punkter. Løsningen blev at benytte en stak datastruktur hvor et punkt blev ”pushet” hver gang et punkt blev passeret første gang. Den fulgte rute kunne så følges tilbage (backtraces) ved at ”poppe” stakken.

Christofides algoritme involverer brugen af en minimum parring med tidskompleksiteten $O(n^3)$. Det har ikke været muligt for os at implementere Christofides algoritme, netop af denne grund. Vi kunne dog se en mulighed for minimum vægt parring i med kompleksiteten $O(n!)$. Vi koncentrerede os derfor om de andre algoritmer, da et praktisk sammenligningsgrundlag ikke var reelt med denne tidskompleksitet.

Nearest Neighbor algoritmen kunne vi implementere uden videre problemer, da denne algoritme var den mindst komplekse at implementere.

Implementeringen af Nearest Insertion er fortaget iht. afsnittet der behandler denne algoritme. Den opnåede tidskompleksitet er $O(n^3)$. Ifølge [HDCM, side 694], kan implementeringen af algoritmen gøres med en tidskompleksitet på $O(n^2)$. I vores implematation beregnes afstanden fra hvert punkt ikke indeholdt i kredsen, til hvert punkt i kredsen, dette sker ved hvert gennemløb. Denne handling er redundant. Ved at gemme beregningerne kunne der opnås en lavere tidskompleksitet, og dermed sænke afviklingstiden af algoritmen. Modellen der kunne illustrere denne metode, er nedenstående angivet.

Ved første gennemløb oprettes en prioritetskø, med alle beregnede afstande til punkterne i kredsen. For hvert punkt der indsættes i kredsen, fjernes alle dens afstande i køen og der beregnes afstande for alle punkter i køen til dette punkt. Hver gang at der skal indsættes et punkt i kredsen, kunne det første punkt i køen vælges. Tidskompleksiteten af denne variant ville være $O(n^2)$, da der ved hvert gennemløb blot skulle beregnes afstanden fra et punkt til alle punkter ikke indeholdt i kredsen, i alt $O(n) \times O(n) = O(n^2)$).

12. Praktisk anbefaling

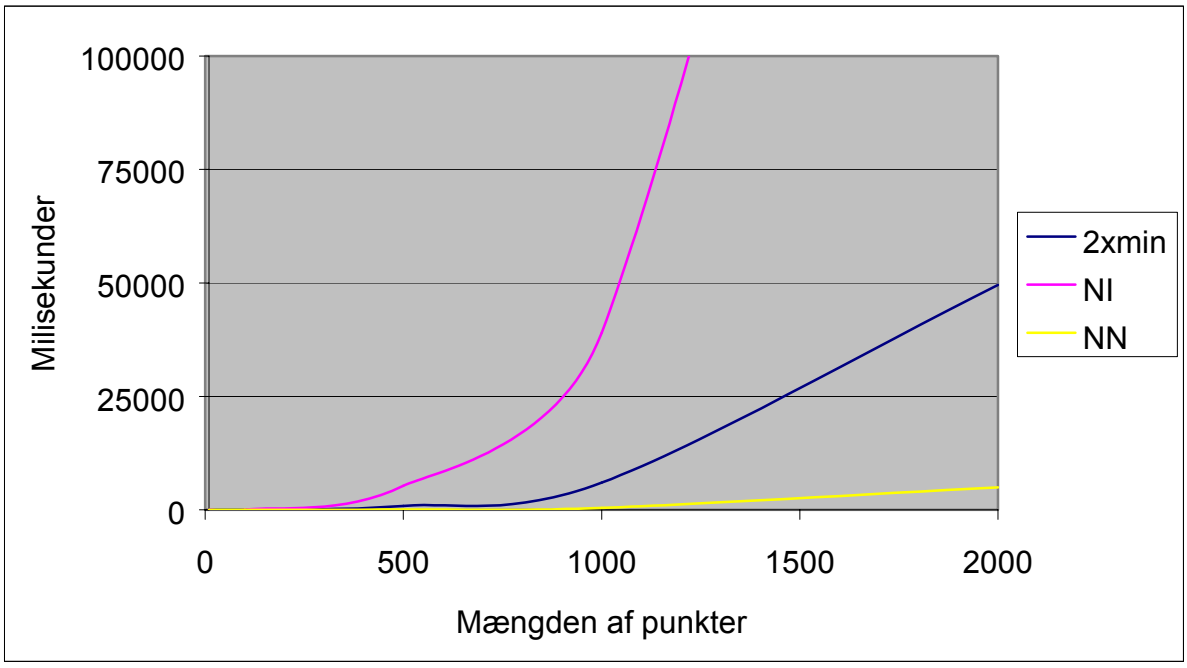
For at vurdere den praktiske brug af de fundne algoritmer er der udført en række tests. Disse tests skal vise om den teoretiske anbefaling også holder i praksis. Testen tager udgangspunkt i 10 tilfældige grafer med et varierende antal punkter. Alle testresultater kan ses i bilag (afsnit 17.3). De relevante vil blive gennemgået her.

Der blev udarbejdet en serie af 10 tests for hvert interval af punkter, som gav de gennemsnitlige resultater i Tabel 7.

	10	100	500	1000	2000
2xmin	0	23	834	5977	49554
NI	0	22	5295	39012	355961
NN	0	0	51	407	4920

Tabel 7: Gennemsnitlige afviklingstider for algoritmerne

Tider under 50 millisekunder var ikke mulige at måle og derfor er resultaterne for kun 10 punkter blevet 0. Det ses som ventet at NN og 2xmin er de hurtigste algoritmer med en tidskompleksitet på $O(n^2)$. NI har tidskompleksiteten $O(n^3)$ og er derfor naturligvis tidsmæssigt længere fra de andre, efterhånden som antallet af punkter stiger. For at illustrere dette vises algoritmernes voksende afviklingstid i forhold til antallet af punkter.



Figur 50:Sammenligning af gennemsnitlige afviklingstider

Figur 50 viser at NI vokser langt hurtigere end både NN og 2xmin. NN vokser meget langsommere end 2xmin selv om begge har en tidskompleksitet på $O(n^2)$. Dette skyldes at $O(n^2)$ kun siger noget om hvor hurtigt tidsforbruget stiger maksimalt uanset eventuelle konstanter som kan have indflydelse på tidsforbruget.

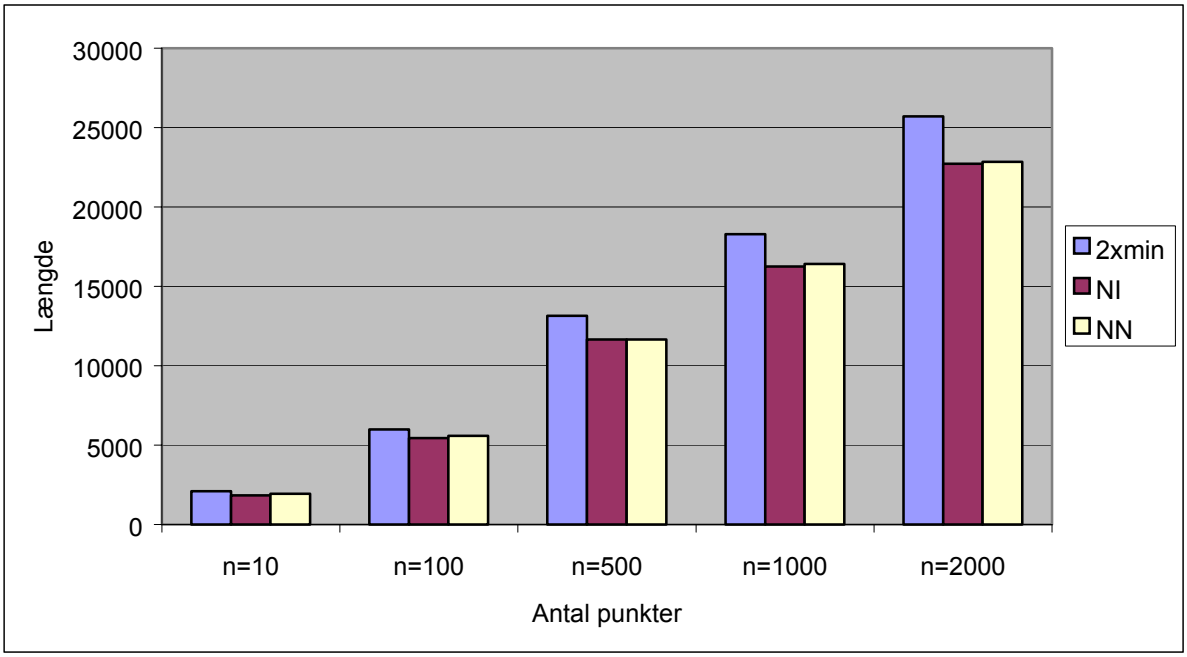
Udfra denne sammenligning kan der udledes at hvis man ønsker en hurtig algoritme til løsning af et problem så er NN algoritmen klart hurtigere, hvilket har betydning ved beregninger på mange punkter.

Det er imidlertid ikke nok blot at se på hvor hurtigt tidsforbruget stiger, men også hvor tæt resultatet kommer på den optimale Hamiltonkreds. Vi kender ikke den optimale Hamiltonkreds for graferne i de praktiske tests, da vi hverken har implementeret en Brute Force algoritme eller kunne finde en som kunne give den endelige løsning for blot 100 punkter, i den tid vi havde til rådighed. Vi kan til gengæld se hvilken algoritme som producerer den mindste Hamiltonkreds og derudfra vurdere hvilken algoritme som gennemsnitligt giver det bedste resultat i forhold til antallet af punkter. I Tabel 8 ses data for den gennemsnitlige længde af Hamiltonkredsene beregnet med de enkelte algoritmer.

	n=10	n=100	n=500	n=1000	n=2000
2xmin	2093	5978	13139	18290	25697
NI	1825	5438	11656	16259	22732
NN	1931	5582	11654	16419	22839

Tabel 8: Gennemsnitslængde af Hamiltonkreds

For at skabe et bedre billede af forskellene imellem algoritmerne er disse data illustreret i et diagram.



Figur 51: Sammenligning af gennemsnitslængde af Hamiltonkreds

Af Figur 51 kan man udlede at 2xmin algoritmens egenskaber for at finde den mindst mulige Hamiltonkreds forringes proportionelt med antallet af punkter i grafen i forhold til NN og NI algoritmerne. Gennemsnitligt er resultaterne for både NN og NI langt bedre end 2xmin, men det kan blot ikke garanteres at dette altid er gældende, da begge algoritmer er heuristiske.
NN og NI algoritmerne ligger generelt meget tæt og der er ikke nogen nævneværdig ændring efterhånden som antallet af punkter forøges.

2xmin algoritmen gav udfra de praktiske tests, ikke et godt gennemsnitligt resultat og var i forhold til NN algoritmen meget langsommere. NI algoritmen var langsommere

end 2xmin algoritmen, men producerede gennemsnitligt et langt bedre resultat. NN algoritmen var overlegen i tid og havde et resultat som kunne måle sig med den langsommere NI algoritme.

I relation til problemet omkring boring af huller i elektronikindustrien, er det ikke umiddelbart muligt at anbefale en af algoritmerne uden at kende til produktionen. Hvis produktionen skal skifte plader ud konstant vil en hurtig algoritme være den bedste, da en langsom beregningstid kan overgå den tid det tager at bore hullerne. Ønsker man at benytte sig af den samme plade hver gang er det bedre at tage en algoritme som giver det bedste resultat. I vores tilfælde vil NN algoritmen være at fortrække da den både er den hurtigste og producerer samtidig ligeså gode resultater som NI. Det er dog stadig et problem at NN algoritmen ikke garanterer en løsning som er bedre end 2xmin, så ønsker man sikkerhed for løsningen, så er 2xmin at foretrække.

13. Faselinie 3 – Problemløsning

Tre af de teoretisk gennemgåede algoritmer blev udvalgt til implementation. Det blev To gange minimum, Nearest Neighbor samt Nearest Insertion. Vi valgte at undlade Brute force implementation da den ikke kunne illustrere en løsning til problemet, snare hvad problemet reelt er. Christofides algoritme blev ligeledes udeladt da det ikke lykkedes at finde en algoritme til minimum vægtet parring med en tidskompleksitet mindre end O(n!). Efterfølgende blev de tre algoritmer koblet sammen med en simpel brugerflade, der grafisk kunne illustrere de 3 algoritmers løsning, til tilfældigt genererede komplette grafer.

14. Proceskonklusion

Proceskonklusionen er primært en erfaringsopsamling som skal evaluere om processen skal være anderledes i et nyt projekt.

Hele projektet igennem har vi haft formål og mål for øje og arbejdet ud fra dem. Vores viden er blevet dokumenteret som ønsket. Mødedisciplinen har været i top og vi har ikke haft nogen problemer med sygdom eller anden fravær.
Den uformelle kommunikation i projektgruppen har fungeret fint og kommunikationen til vejlederen har været uden problemer selvom vejlederen ikke var geografisk velplaceret.
Konfigurationsstyring og projektarkivet fungerede også udmærket og alle var enige om at denne måde, var den rigtige måde at styre dokumenter på. Koncentrationen var fra starten rettet mod dokumentation af vores matematiske forståelse af problemet. Faselinierne blev alle godkendt med få bemærkninger og ikke noget som rykkede tidsplanen

Tidsplanen holdt hele projektperioden igennem, hvilket kan tilskrives vores kortsigtede mål, som gjorde det muligt for alle i gruppen at vide præcis hvor langt vi var i projektet. Det skal nævnes at det korte projektforløb også gjorde det let overskueligt og at man i et længere forløb, kunne få brug for at tage andre styringsværktøjer i brug..

Strategien for projektet var bl.a. at løse svære opgaver først. Det var muligt at se resultater, og gav derfor som ventet en høj motivation, som også holdt hele projektperioden igennem. Projektet gav stor usikkerhed i starten, da emnerne kun var

overfladisk kendte. Dette resulterede i en stor del af projektarbejdet blev learning by doing.

Projektets estimering var fornuftig ved benyttelse af analogi, hvilket skyldes at vi alle har arbejdet i lignende projekter tidligere. Der var ikke mange problemer undervejs, det største problem var den, på nogen punkter, meget begrænsede litteratur og vores begrænsede kursusforløb. Selv om dette var tilfældet bevarede projektgruppen motivation hele projektet igennem.

15. Produktkonklusion

Rapportens indledende fase bestod i at opstille et matematisk problem der kunne modellere processen at beregne og bore hullerne i en printplade. Ved at anvende TSP som model for vores problem kunne vi derigennem udlede og forstå det konkrete matematiske problem der skulle søges løst i rapporten. Der findes et væld af algoritmer der giver en tilnærmelsesvis løsning af TSP problematikken, hvoraf fem er gennemgået teoretisk; Brute force, Christofides, To gange minimum, Nearest Neighbor og Nearest Insertion. Af disse er To gange minimum, Nearest Neighbor og Nearest Insertion implementeret.

Anbefalingen af algoritme afhænger af frekvensen af pladeændringer samt antallet af boringer. I forhold til den teoretisk og praktiske anbefaling er valget faldet på Nearest Neighbor, dog med det forbehold at der ikke kan garanteres en øvre grænsen for resultatet i forhold til den optimale Hamiltonkreds. Ved betragtning af de fremkomne resultater må det konkluderes at en lokal optimering kan give et bedre resultat, på grundlag af dette anbefales det at foretage lokale optimeringer ved længerevarende produktion af ens printplader.

16. Kildeangivelse

[TTSP] The Traveling Salesman Problem, John Wiley & Sons, E. L. Lawler m.f.

[CO] Combinatorial Optimization, John Wiley & Sons , Cook, Cunningham m.f.

[TSP] Traveling Salesman Problem, Aalborg Universitet, Busborg, Erichsen m.f.

[AP] Applies Combinatorics, John Wiley & Sons, Alan Tucker

[TPP] The Pragmatic Programmer, Addison-Wesley, Andrew Hunt, David Thomas

[HDCM] Handbook of Discrete and Combinatorial Mathematics, CRC Press , Kenneth H. Rosen

[DMiA] Discrete Mathematics and its Applications, McGraw-Hill, Kenneth H. Rosen

17. Bilag

17.1. Indhold af vedlagte cd

Den vedlagte cd indeholder følgende:

- Rapporten i Word format
- Rapporten i html format
- Class og java filer
- Java Runtime Environment v1.3.1

17.2. Kildekode

To gange minimum

```
class ToGangeMinimum
{
  public static Graph toGangeMinimum(Graph completeGraph)
  {
    Graph  minimumTree       = completeGraph.minTree();
    Graph  hamiltonCircuit   = new Graph(completeGraph.nofVertices());

    Stack backtracePath      = new Stack();
    boolean Hamilton         = false;
    int currentVertex        = 0;
    int lastHamiltonVertex   = 0;
    boolean[] visitedVertices = new boolean[completeGraph.nofVertices()];

    for (int i=0; i<completeGraph.nofVertices(); i++)
      visitedVertices[i] = false;

    visitedVertices[0] = true;

    while (!hamilton)
    {
      boolean keepSearching = true;
      for (int i=0; i<minimumTree.nofVertices() && keepSearching; i++)
      {
        if (minimumTree.neighbour(currentVertex, i) && !visitedVertices[i])
        {
          hamiltonCircuit.insEdge(lastHamiltonVertex, i, completeGraph.weight(lastHamiltonVertex, i));
          backtracePath.push(new Integer(currentVertex));
          lastHamiltonVertex = currentVertex = i;
          visitedVertices[i] = true;
          keepSearching = false;
        } //if
      } //for

      if (keepSearching)
      {
        if (!backtracePath.empty())
          currentVertex = ((Integer)backtracePath.pop()).intValue();
        else
        {
          hamilton = true;
          hamiltonCircuit.insEdge(lastHamiltonVertex, currentVertex,
                                  completeGraph.weight(lastHamiltonVertex, currentVertex));
        }
      } //if
    } //while
    return hamiltonCircuit;
  } //metode
} //class
```

Nearest Neighbor

```
class NearestNeighbor
{
    private Graph pgraf;         // den komplette graf hvori der skal findes en hamiltonkreds
    private Graph returgraf;  v // den graf der til slut returneres
    boolean []barray;            // boolsk array som fortæller om punkter er besøgt
```

```
 public NearestNeighbor()
 // Instantiering af objektet
 {
    // gør ingenting
 }

 public Graph beregn(Graph g)
 {
   pgraf=g;
   returgraf=new Graph(pgraf.nofVertices());
   barray = new boolean[pgraf.nofVertices()];

   for (int j=0;j<pgraf.nofVertices()-1;j++)  // Laver en liste over punkter som er besøgt
   {
        barray[j]=false;
   }

   int shortpoint=1655361;

    for (int m=0;m<pgraf.nofVertices()-1;m++) // Finder kant med minimum vægt(shortpoint)
    {
        for(int n=0;n<pgraf.nofVertices()-1;n++)
        {
                if (pgraf.weight(m,n)<shortpoint) shortpoint=m;
        }

    }

   int min=(-2);            // Instantiering af mindste punkt
   int start=shortpoint; // Startpunktet sat til punkt ved mindste kant
   barray[0]=true;       // Sætter første punkt til besøgt

   for (int i=0;i<pgraf.nofVertices()-1;i++)
   {
         if (min!=(-1))
         {
                min = findmin(start,pgraf.nofVertices());
                returgraf.insEdge(min,start,pgraf.weight(min,start));
                barray[min]=true;
              start=min;
          }
   }
   returgraf.insEdge(0,start,pgraf.weight(0,start)); // Indsætter kant mellem start og slut
   returgraf.vertex = pgraf.vertex; // Sætter koordinater til tegning
   return returgraf; // returnerer resultat (Hamilton kreds)

 }

private int findmin(int p, int nofVertices)
// finder mindste vægt mellem punkt p og alle andre punkter som ikke er brugt
{
     int min=1655361;
     int punkt = (-1);

     for (int i=0; i<nofVertices; i++)
     {
          if (!barray[i])
          {
                if (pgraf.weight(i,p)<min)
                {
                    punkt=i;
                    min=pgraf.weight(i,p);
```

```
                }
            }
        }
        return punkt;
}

} // End; Nearest Neighbor
```

Nearest Insertion

```
class NearestInsertion
{
    public void NearestInsertion()
    {
    }

    public Graph calculate(Graph gIn)
    {
        int mNumVertices       = gIn.nofVertices();
        Graph gOut             = new Graph(mNumVertices);    // Retur graf
        int InGOut             = 0;                  // Antal punkter i gOut
        boolean[] usedVertices  = new boolean[mNumVertices];  // Punkter i GOut

        for(int i=0; i<mNumVertices; i++) // Sætter alle besøgte til false
        usedVertices[i]= false;
        // Udgangspunkt tages det første punkt i grafen (0) og det punkt der ligger  nærmest,kanten
        indsættes i den nye graf

        int minWeight   = gIn.weight(0,1);
        int minVertices = 1;
        for (int i=2; i<mNumVertices; i++)
        {
                if (gIn.weight(0,i)<minWeight)
                {
                        minVertices = i;
                        minWeight   = gIn.weight(0,i);
                }
        }

                gOut.insEdge(0,minVertices,minWeight);
                InGOut +=2;
                usedVertices[0]          = true;
                usedVertices[minVertices] = true;
                // Punkter endnu ikke gOut indregnes en efter en
                while (InGOut !=mNumVertices)
                {
                    int minWeight3   = 999999;
                    int minVertices3 = -1;
                    for (int i=0; i<mNumVertices; i++)
                    {
                        if (usedVertices[i] == true)
                        {
                            for(int j=0; j<mNumVertices; j++)
                            {
                                if (usedVertices[j] == false)
                                {
                                    if (gIn.weight(j,i)<minWeight3)
                                    {
                                     minVertices3 = j;
                                        minWeight3   = gIn.weight(i,j);
                                    }
```

```
                        }
                    }
                }
            }
                int currentWeight=99999;
                int newWeight;
                int changePoint1=-1;
                int changePoint2=-1;
                for (int i=0; i<mNumVertices; i++)
                {
                    if (usedVertices[i] == true)
                    {
                        for(int j=i; j<mNumVertices; j++)
                        {
                            if (usedVertices[j] == true && j!=i)
                            {
    newWeight = (gIn.weight(minVertices3, i) + gIn.weight(minVertices3, j) - gOut.weight(i, j));


                                if (currentWeight > newWeight)
        {
        changePoint1  = i;
                                    changePoint2  = j;
                                    currentWeight = newWeight;
                                }
                                        }
                        }
                    }
                }
                usedVertices[minVertices3] = true;
                lnGOut++;
                If (lnGOut>3)  gOut.delEdge(changePoint1,changePoint2);
                gOut.insEdge(changePoint1,minVertices3, gIn.weight(changePoint1,minVertices3));
                gOut.insEdge(minVertices3,changePoint2, gIn.weight(changePoint2,minVertices3));
                }
                    gOut.vertex = gIn.vertex;
                    return gOut;
                }

}
```

17.3. Testresultater

n = 10

	Gns lgd	Gns Tid	Max Lgd	Min Lgd
2xmin	2092,8	0	2554	1421
NI	1824,9	0	2160	1407
NN	1931,3	0	2473	1443

n=100

	Gns lgd	Gns Tid	Max Lgd	Min Lgd
2xmin	5978,2	23	6466	5678
NI	5437,5	22	5669	5167
NN	5582,4	0	6099	5226

n=500

	Gns lgd	Gns Tid	Max Lgd	Min Lgd
2xmin	13138,6	834	13321	12801
NI	11656,4	5295	11933	11264
NN	11653,8	51	12072	11276

n=1000

	Gns lgd	Gns Tid	Max Lgd	Min Lgd
2xmin	18289,9	5977	18887	17942
NI	16258,6	39012	16683	15783
NN	16418,9	407	17079	15793

n=2000

	Gns lgd	Gns Tid	Max Lgd	Min Lgd
2xmin	25696,9	49554	26008	25281
NI	22731,6	355961	22995	22430
NN	22839,1	4920	23287	22473