\chapter{Grafteori}

Grafteori er et afsnit i denne rapport, som omhandler en generel forklaring på grafteori, hvorefter teorien bag Nearest neighbor algoritmen, Dijkstra’s algoritmen, Double Minimum Spanning Tree og Traveling salesman problem vil blive beskrevet. Alt dette beskrives, for at give et udgangspunkt for implementering af en hensigtsmæssig algoritme i programmet for denne rapport.

\section{Begrebsbeskrivelse}

En knude er et punkt på grafen. Knuder vil i denen rapports sammenhæng være attraktioner. \newline

En kant forbinder to knuder. Kanter kan vægtes, i denne rapport vil vægtningen af kanter være afstanden mellem de to knuder som kanten forbinder. \newline

En path er en rute gennem et antal knuder, hvor startknuden ikke er endeknuden. \newline

Et circuit er en rute gennem et antal knuder, hvor startknuden også er endeknuden. \newline

Et Hamiltonian circuit er et circuit, hvori alle knuder er besøgt én gang, med undtagelse af slutpunktet, som skal være det samme som startpunktet. \newline

I denne rapport søges et circuit, som går igennem alle knuder én gang, hvor en samlet vægtning af kanterne (længden af ruten) bliver så kort som mulig. Dette er i grafteori også kaldet Traveling Saleman Problem. En umiddelbar løsning til TSP ville ville være at undersøge alle mulige Hamiltonian circuits, og finde den korteste. Problemet er, at når en graf med *n* knuder skal undersøges, skal der bruges (*n* – 1)!/2 udregninger. Dette skyldes, at efter en startknude er valgt, vil der være *n* – 1 mulige knuder som næste punkt, og derefter *n* – 2 osv. Da der ikke er en bestemt retning på Hamiltonian circuits, vil der altid være to veje, hvor den ene beskriver den anden bare omvendt. Derfor fås (*n* – 1)!/2. \newline  
Med 25 knuder vil der være (24)!/2, som svarer til 3.1 x 10^23 forskellige Hamiltonian circuits. Hvis det antages, at det tager et nanosekund at udregne ét Hamiltonian circuit, vil dette tage cirka 10 millioner år, at udregne alle circuits, og finde den optimale. Derfor bruges algoritmer, som udregner en løsning som er tæt på den optimale. Nedenfor vil tre sådanne algoritmer blive beskrevet. (side 715, Discrete Mathematics) \newline

\subsection{Nearest neighbour algoritmen}

Fremgangsmetode: \newline  
Vælg en startknude. \newline  
Følg kanten med den laveste vægtning. \newline  
Check om der er flere ubesøgte knuder tilbage, hvis ja, gå tilbage til trin 2. \newline  
Gå tilbage til startknuden. \newline

Fra den knude der behandles, skal kanten med den laveste værdi følges. Hvis alle knuder er besøgt, skal kanten der går tilbage til startknuden vælges. \newline

Eksempelvis i dette tilfælde (figur 4.2), er den endelige rutes længde, udregnet med NNA: 1+5+4+8+2 = 20. \newline

NNA udregner ikke med sikkerhed en optimal rute, hvilket skyldes at den ikke tager højde for konsekvenserne af de skridt den tager. Selvom det første skridt er det korteste blandt de mulige, kan det i det lange løb godt ende med at blive en meget længere rute der bliver sammensat. En måde dette kan optimiseres, er ved at køre NNA flere gange med forskellige startpunkter. Her kaldet en udvidet NNA. Nedenfor er en udvidet NNA beskrevet. \newline

Fremgangsmetode: \newline  
Udfør NNA på en ny ikke testet startknude. \newline  
Er der knuder der ikke er testet som startknude, hvis ja, gå tilbage til trin 1. \newline  
Vælg det korteste circuit af de testede.

\begin{tabular}{| l | l | l | l | l | l |}

\hline

& A & B & C & D \\ \hline

A & - & 8 & 1 & 2 \\ \hline

B & 8 & - & 6 & 3 \\ \hline

C & 1 & 6 & - & 5 \\ \hline

D & 2 & 4 & 5 & - \\ \hline

\hline

\end{tabular}\newline

De mulige ruter er: \newline

ACDBA = 1 + 5 + 4 + 8 = 18\newline

BDACB = 4 + 2 + 1 + 6 = 13\newline

CADBC = 1 + 2 + 4 + 6 = 13\newline

DACBD = 2 + 1 + 6 + 4 = 13\newline

Den korteste rute er altså BDACB, hvilket er 5 kortere end den antagede rute. Den optimale circuit fundet med udvidet NNA er derfor denne rute. Dette tager standard NNA ikke højde for, da den starter i en valgt start-knude, og derefter følger kanten, med den derfra laveste værdi. NNA kræver ikke lige så mange beregninger som en brute force udregning kræver, og kan derfor bruges i praksis, dog er den ikke sikker på at finde den optimale rute.

/subsection{Double Minimum Spanning Tree}

Double Minimum Spanning Tree er algoritmen der ligger til grund for Christofides algoritme, hvor DMST har tre trin: \newline  
Først oprettes et minimum spanning tree, som indkluderer alle knuder. \newline  
Alle kanter duplikeres. \newline  
Den korteste rute findes mellem disse kanter, hvor en knude kun besøges én gang. \newline  
Hvis der ikke er kanter til ubesøgte knuder, oprettes en ”genvej” fra den nuværende knude til en ubesøgt knude \citep{DMST}. \newline

Et minimum spanning tree findes ved at tage så få kanter som muligt, med den laveste vægtning, så alle knuder er besøgt blot én gang.

Da dette projekt forholder sig til en forholdvis kort rute, men hovedsagligt tager udgangspunkt i en rute hvortil der kan tilføjes flere punkter, for at lave en interessant rute, er NNA valgt som den bedste kandidat. Dette skyldes, at Dijkstra’s ikke nødvendigvis indkluderer alle knuder, da den finder en kort rute fra startpunkt til slutpunkt. DMST finder en mere optimal rute end NNA, men samtidig kræver den længere tid at udregne, og derfor blev NNA valgt som den optimale algoritme til dette projekt.