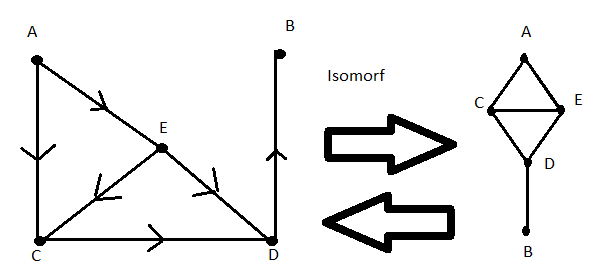
<https://www.youtube.com/watch?v=HmQR8Xy9DeM> – Introduction to Graph Theory (video).

<http://math.tut.fi/~ruohonen/GT_English.pdf> - Introduction to Graph Theory (slides).

<https://www.youtube.com/watch?v=zPgsNsOfxQ8> – NNA graf teori (video).

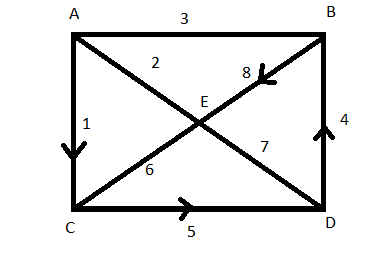
Graf teori er et afsnit i denne rapport, som omhandler en generel forklaring på graf teori, hvorefter teorien bag ”Nærmeste Nabo Algoritme” vil blive beskrevet, herefter forklares ”udregningstid” i rute-algoritmer, og til sidst Traveling Salesman Problem. Alt dette beskrives, for at give et udgangspunkt for implementering af en hensigtsmæssig algoritme i programmet for denne rapport.

Matematikken bag graf teori er ét aspekt af emnet, hvor visualisering og tegning er en anden. Den matematiske del behandler kombinationerne af *knuder* og *kanter*. En knude er et punkt, i vores tilfælde en attraktion som skal besøges, hvor en kant er vejen derhen. En kant er derfor længden fra ét punkt til det næste.  
I graf teori er begrebet ”graf” mere fleksibelt, da punkterne ikke nødvendigvis har x, y eller z værdier, alt efter antallet af dimensioner man behandler det i. Grafen i denne form er en afbildning af punkter i den form, hvor det virker hensigtsmæssigt. Heraf opstår isomorfiske modeller, som er forskellige afbildninger, af selv samme graf.



Figur 1, afbildning af isomorfi.

Denne matematik-type er stadig under udforskning, da der endnu ikke er en fuldstændig løsning på problemer i teorien, for blandt andet ”Traveling Salesman Problem”. Heriblandt findes mange typer af problemer, hvor forskellige teoretiske løsninger kan bruges.  
En løsning er den Nærmeste Nabo Algoritme (NNA, Nearest Neighbor Algorithm), som behandler et problem der opstår, når en række knuder skal indgå, og kun skal indgå én gang, hvilket betyder, at der ikke må være en løkke (loop). Der er to måder at gennemgå en rute på, som kaldes hhv. Hamiltonian Paths og Hamiltonian Circuits. Paths er ruter, der ikke ender tilbage ved start-knuden, hvor en Hamiltonian Circuit er en løkke fra start-knuden, gennem alle knuder, og tilbage til start-knuden. Hamiltonian Paths og Circuits vælger begge knude efter knude at følge kanten der fra den enkelte knude har den mindste værdi.  
NNA kan opnåes gennem brug af Hamiltonian Paths, hvilket er en ”rute” gennem knuder på grafen. I NNA vil en kant have en værdi, og disse værdier er bestemmende for, hvilken kant der skal følges. Fra den knude der behandles, skal kanten med den laveste værdi følges. Dette kan dog optimeres, ved at lave en matrice over kanterne fra alle knuder. Her tages højde for hvilke kanter der samlet set giver den korteste rute, uden brug af løkker.



Figur 2, Hamiltonian Path. Følger kanter med laveste værdier.

I dette tilfælde (figur 2), er den endelige rutes længde: 1+5+4+8 = 18. Spørgsmålet er så, er dette den koreste rute? Herefter opstilles en matrice, der beskriver alle kanter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | - | 3 | 1 | - | 2 |
| B | 3 | - | - | 4 | 8 |
| C | 1 | - | - | 5 | 6 |
| D | - | 4 | 5 | - | 7 |
| E | 2 | 8 | 6 | 7 | - |

De mulige ruter er:

ABDCE = 3+4+5+6 = 18.  
ACDBE = 1+5+4+8 = 18.  
AEBDC = 2+8+4+5 = 19.  
ABECD = 3+8+6+5 = 22.  
ABEDC = 3+8+7+5 = 23.  
ACEBD = 1+6+8+4 = 19.  
ACEDB = 1+6+7+4 = 18.  
AECDB = 2+6+5+4 = 17.

Den korteste rute er altså AECDB, hvilket er 1 kortere end den antagede rute. Den optimale Hamiltonian Path er derfor denne rute. Dette tager NNA ikke højde for, da den starter i en valgt start-knude, og derefter følger kanten, med den derfra laveste værdi. Det smarte ved NNA er, at den ikke kræver meget kraft for en computer at udføre, hvorimod at finde den optimale Hamiltonian rute, vil være langt mere compliceret. NNA tager dog ikke højde for, hvad konsekvenser de skridt den tager, har for det endelige resultat.

Udover NNA, findes også Dijkstra’s algoritme, hvor første step er, at bestemme ende-knuden, og sætte dens distance til nul. Denne knude sættes til at være den første knude, som behandles. I det en knude er checket færdig, vil denne knude markeres som ”besøgt”, og kanten med den mindste værdi følges, og næste knude markeres som ”nuværende” knude. En kant bliver kun fulgt, hvis det er den korteste rute, tilregnet tidligere kanter.  
Problematikken med Dijkstra’s algoritme i forhold til dette projekt er, at den checker den korteste rute fra start-knude til slut-knude, men den indkluderer ikke nødvendigvis alle knuder som oplyses. I det denne rapport er afgrænset til fugleflugtslinjer, vil Dijkstra’s ikke være den optimale. Hvis en rute igennem en by, hvor der er tilregnet veje, stier og andre knuder, vil Dijkstra’s være det bedste valg. Denne algoritme vil også være i brug ved den optimale løsning.

Ved brug af NAA vil en udregningstid for algoritmen beskrives som O(Nd), hvor N er kardinaliteten (antallet af knuder), d er antallet af dimensioner, og O er udregningstiden. Dette vil sige, at for hver knude vi inddrager i udregningen, vil udregningstiden bliver antallet af dimensioner større. Denne udregningstid er baseret på, at udregningen bliver gjort gennem en ”naiv” metode, hvor udregningen konstant bliver testet for, hvorvidt den ”nuværende” beregnede rute, er kortere eller længere end den hurtigste hidtil.