Udfaldshåndtering i lavspændingsnet

af Aske Butze-Ruhnenstierne og Svend Knarhøj Johannsen

Aske Butze-Ruhnenstierne

Svend Knarhøj Johannsen

Abstract

I dag tager vi det alle for givet, at der er strøm i stikkontakterne når vi har brug for det og at strømafbrydelser hører til de absolutte sjældenheder. For at opretholde den høje forsyningssikkerhed der forventes, anvender energiselskaberne en lang række systemer til at overvåge elforsyningsnettet, så fejl kan forudses og forebygges.

Når fejl alligevel opstår på elforysningsnettet er det vigtigt, at de hurtigt kan lokaliseres og afhjælpes. I denne rapporten præsenteres forskellige metoder til lokalisering af fejlene og til automatisk allokering af resourcer, så fejlene kan udbedres. Metoderne til fejlfinding omhandler blandt andet algoritmerne Breadth First Search og Edmunds-Karp, samt teknologierne Constraint Based Reasoning og specielt Array Teknologi. Til gruppering af de fundne fejl er blandt andet Markov Clustering processen benyttet.

I projektet er der udviklet en prototype, der implementerer tre forskellige metoder til fejlfinding og to forskellige metoder til gruppering. Prototypen er et værktøj til måling af ydeevnen af de forskellige metoder.

De designmæssige overvejelser der er gjort under udvikling af ovenstående metoder er beskrevet, og der er lavet både teoretiske og praktiske analyser af køretid og begrænsninger. I den sammenhæng er der fokuseret på, hvilke overvejelser man bør gøre sig med henblik på skalerbarhed.

Vi vil gerne takke Per Klitgaard for generelle råd og vejledning omkring elforsyningen i Danmark samt Philip Heede og Informi GIS for værdifuld information om DONG Energys nuværende system, kravene til systemet og strukturen af DONG Energys elforsyningsnet. Desuden tak til Array Technology for bistand i forbindelse med udvikling af projektets arrayteknologiske aspekter og vores vejleder Peter Falster for støtte gennem hele projektforløbet.

Summary

One takes it for granted that the power outlets work as intended and that power outages rarely occurs. To provide the high quality of service that is expected, power distribution companies deploy an array of systems to detect and counter errors.

This paper presents a number of different approaches to detect and cluster errors in a power grid. The deployed methods make use of algorithms such as Breadth First Search and Edmunds-Karp, as well as the technologies Constraint Based Reasoning and in particular Array Technology. In terms of clustering the Markov Clustering process is of particular interest.

The system is intended as both an error detection system and a resource planning system. This is why both error detection algorithms and clustering are described - once a number of errors have been detected, clustering is used to issue an appropriate number of work orders.

The paper provides a mathematical definition of how errors are detected in a power grid. Based on this definition, three error detection algorithms are designed:

- ATOMS - based on Array Technology
- BOMS - based on graph theory and in particular Breadth First Search
- EOMS - based on flow analysis and the Edmunds-Karp algorithm

The three error detection algorithms are further enhanced with the ability to cluster nodes in the grid, and the ability to adapt to the current grid by processing reports from the technicians working in the field.

The thoughts and considerations that went into the development of the said methods have been described. With regard to the implementation significant effort has gone into describing the aspects of performance as well as scalability.

The project has produced a sample application capable of detecting errors and issuing work orders in a power grid with 540,000 nodes in just 11 seconds. In an emergency situation, where major parts of the grid are unpowered, the application is able to issue work orders in approximately 10 minutes.

The final chapters of the paper describe a thorough testing of the described algorithms, and compares theoretical and real-life running times. The main conclusion from the testing is that given the actual structure of the power grid, the EOMS algorithm is by far the fastest.

A number of enhancements to the topological approach are proposed at the end of the paper. Especially drawing in information about construction work and zoning from GIS systems could strengthen the model a great deal.

Indholdsfortegnelse

1 Indledning

Denne rapport omhandler fejlfinding i elforsyningsnet.

På de større elforsyningsnet, højspændingsnettet, er hver komponent tilsluttet et overvågningssystem, der automatisk alarmerer en tekniker, hvis der er mistanke om fejl. På lavspændingsnettet, der anvendes til at distribuere strøm det sidste stykke vej fra en transformatorstation ud til den enkelte forbruger, er hver komponent endnu ikke udstyret med automatisk overvågning. I denne rapport vil vi analysere muligheden for at bruge elforsyningsnettets topologi, sammenholdt med fejlrapporter fra forbrugerne til at lokalisere fejl i lavspændingsnettet.

Udgangspunktet for projektet er, at Informi GIS har været involveret i udvikling af et system til lokalisering af fejl i DONG Energys lavspændingsnet. På baggrund af Informi GIS' erfaringer, vil denne rapport belyse mulighederne for at anvende Array Teknologi og klassisk grafteori i forbindelse med denne problemstilling.

Rapporten indledes med en beskrivelse af elforsyningsnettes opbygning og en præcicering af, hvilket problem denne rapport omhandler. Der opstilles en matematisk formuleting af problemet, så man med udgangspunkt i denne kan konstruere algortimer til at løse problemet på en computer.

Derefter designes tre algoritmer til lokalisering af fejl i elforsyningsnettet. Disse tre algoritmer er:

- Array Technology Outage Management System (ATOMS) – der ved hjælp af array teknologi løser fejlfindingsproblemet.
- Breadth First Search Outage Management System (BOMS) – der ved hjælp af traditionel grafteori løser fejlfindingsproblemet.
- Edmunds-Karp Outage Management System (EOMS) – der modellerer fejlfindingsproblemet som et maksimum flow problem.

For at oversætte de fundne fejl til arbejdsordrer, så der kan allokeres mandskabsresourcer til at løse problemet, ønsker man at gruppere fejlene. Derfor suppleres de tre ovenævnte algortimer med to forskellige metoder til gruppering af knuderne i en graf. De to metoder der anvendes er:

- Markov Clustering – der finder naturlige grupper i en graf ved hjælp af lineær algebra.
- Sektionering – der benytter sig af en simpel heuristik til gruppering.

Med alle de primære algoritmer på plads, drejer rapporten derefter fokus over på indlæring, dvs. hvordan tilbagemeldinger fra teknikerne i marken kan anvendes til løbende at gøre systemet bedre. Der bliver opstillet heuristikker for, hvordan de ovenstående algoritmer kan fintunes på baggrund af empiriske data.

Dette projekt har, udover denne rapport, produceret et stykke software kaldet "Outage Management System". Softwaren giver mulighed for at afprøve alle de ovenstående algoritmer på forskellige elforsyningsnet og med forskellige fejlscenarier. Efter afsnitet om algoritmerne indeholder rapporten et afsnit om, hvorledes "Outage Management System" er implementeret samt en kort brugervejledning.

Ved hjælp af "Outage Management System" er det lavet en række testkørsler af de forskellige algoritmer, så de teoritiske køretider kan sammenholdes med de faktiske køretider.

For at kunne generere testeksempler af varierende størrelse, er der også implementeret en lille hjælpapplikation til konstruktion af elforsyningsnet. Med udgangspunkt i kendte net fra DONG Energy kan der opbygges nye elforsyningnet af vilkårlig størrelse, med samme struktur som det virkelige net.

Nettets topologi er kun én ud af flere paramtere der kan anvendes til at lokalisere fejl. Rapporten afsluttes derfor med en række anbefalinger til, på hvilke områder beregningsmodellen kan udvides i fremtiden.

2 Terminologi

I dette afsnit gennemgås den terminologi, der bruges gennem rapporten. Afsnittet indeholder desuden en beskrivelse af den grafnotation der benyttes, samt en oversigt over de hyppigst anvendte matematiske symboler og forkortelser. Afsnittet er ment som en reference.

Generelt set anvender vi termer fra grafteori, og ikke fra elforsyningsindustrien.

Alle literaturhenvisninger er angivet i skarpe parenteser, eksempelvis: [LITT]

2.1 Det fysiske net

Delnet beskriver et lavspændingsnet der, når man ser bort fra højspændingsnettet, kan behandles som et simpelt sammenhængende el-net.

Elforsyningsnet beskriver det fysiske el-net, eksempelvis DONG Energys el-net i København og Nordsjælland.

Fejlfindingsproblemet er at finde frem til hvilke forbindelser der med stor sandsynlighed er defekte ifølge en given fejlrapport.

Fejlmelding bruges når en forbruger har informeret elforsyningsselskabet om at vedkommende er uden strøm

Fejlrapport beskriver en række fejlmeldinger indsamlet over et tidsrum.

Forbindelse er det fysiske kabel mellem fordelingsskabe.

En forbindelse kan være **defekt** / **ikke defekt**.

Forbruger benyttes om en enkelt husstand der tilsluttet lavspændingsnettet.

En forbruger kan være **afbrudt** / **ikke afbrudt**.

Fordelingsskab er et knudepunkt i lavspændingsnettet.

Højspændingsnet beskriver de dele af elforsyningsnettet hvor spændingen er over 400 V.

Lavspændingsnet er de dele af elforsyningsnettet hvor spændingen er 400 V.

Maskenet er et cyklisk elforsyningsnet med én strømkilde.

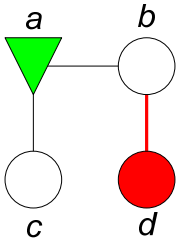
Radialnet er et ikke-cyklisk elforsyningsnet med én strømkilde.

Stikledning er det fysiske kabel mellem forbrugere og fordelingsskabe.

Strømkilde er et knudepunkt i et lavspændingsnet der er forbundet til et højspændingsnet.

Ø-net er et cyklisk elforsyningsnet med to eller flere strømkilder.

2.2 Det abstrakte net



Figur 1: Grafrepræsentation af et elforsyningsnet.

Graf anvendes om en matematisk graf, som abstraktion fra elforsyningsnettet eller et delnet.

Kant benyttes om en forbindelse mellem to knuder i en graf.

En kant kan være **defekt** (rød) / **ikke defekt**. (sort)

Knude beskriver et knudepunkt i en graf og angives som en cirkel.

En knude kan være **afbrudt** (rød) / **ikke afbrudt**. (hvid)

Strømkilde er angivet med en grøn trekant.

2.3 Matematiske symboler

G	En graf.
V	Alle knuder i en graf. (eng. *vertex*)
E	Alle kanter i en graf. (eng. *edge*)
$N_n(V)$	Alle knuder der ligger *n* eller færre kanter fra en knude i V.
$p(v_1, v_2)$	Indikatorfunktion for stier mellem knuder.
$P(V_1, V_2)$	Indikatorfunktion for stier mellem mængder af knuder.
t	En enkelt knude der er angivet som strømkilde.
T	En mængde af knuder der er angivet som strømkilder.
F	En fejlrapport - en mængde af knuder der er fejlmeldt.
$R(G, T, F)$	Tilstanden af et elforsyningsnet.
S	En forklaring - en mængde af kanter der, givet en fejlrapport, bringer grafen tilbage i en lovlig tilstand.
Q_∞	Alle forklaringer.
Q_n	Mængden af forklaringer der udelukkende benytter sig af knuder, der ligger *n* eller færre kanter fra en knude i fejlrapporten.
u	Indikatorfunktion for afbrudte knuder.
$K(S)$	Konsekvensen af en forklaring - en mængde af knuder der nødvendigvis må være afbrudt.

2.4 Forkortelser

AMB	Filformat for kompilerede (binære) arraymodeller.
AML	Array Markup Language - filformat til beskrivelse af en arraymodel i Array Studio
AMS	Array Model Source - filformat til beskrivelse af en arraymodel i Array Studio
ATOMS	Algoritme til udfaldshåndtering baseret på Array Teknologi
BFS	Breadth First Search algoritmen
BOMS	Algoritme til udfaldshåndteing baseret på Breadth First Search
CM	Combined Model - arraymodel der anvendes af ATOMS til at finde alternative forklaringer. Modellen er en kombination af ECM og PPM.
CSOP	Constraint Satisfaction Optimization Problem
CSP	Constraint Satisfaction Problem.
ECM	Edge Cutting Model - arraymodel der anvendes af ATOMS til at finde defekte kanter.
EOMS	Algoritme til udfaldshåndtering baseret på Edmunds-Karp.
GIS	Geografisk Informations System
MCL	Markov Clustering - metode til gruppering af en graf
OMS	Outage Management System (udfaldshåndteringssystem)
PPM	Power Propagation Model - arraymodel der anvendes af ATOMS til at finde afbrudte knuder

3 Problemformulering

Fokus for dette projekt er fejlfinding i elforsyningsnet på baggrund af fejlmeldinger fra forbrugerne. Kort sagt kan det problem, der analyseres i denne rapport beskrives således:

> *Givet en eller flere fejlmeldinger fra forbrugerne, bestem da hvilke komponenter i elforsyningsnettet, der med størst sandsynlighed er skyld i disse fejl, således at de teknikere der skal udbedre fejlene, kan påbegynde deres fejlsøgning på det mest hensigtsmæssige sted.*

Der vil blive taget udgangspunkt i data fra DONG Energy, således at strukturen af de elforsyningsnet der anvendes ligger så tæt på virkeligheden som muligt.

I den typiske situation meddeler en eller flere forbrugere telefonisk, at de ikke har strøm. DONG Energy akkumulerer fejlmeldinger over et tidsrum, og sammenholder alle modtagne fejlmeldinger for, på den baggrund, at vurdere hvor fejlen/fejlene med størst sandsynlighed er opstået. Det er denne vurdering vi ønsker at automatisere i dette projekt. Rapporten vil fokusere på fejlfindingsproblemet med en algoritmisk indgangsvinkel, da det hurtigt viser sig, at man ikke kan gennemføre en "brute force"[1] beregning på hele elforsyningsnettet. For at finde de mest sandsynlige forklaringer på de fejlmeddelelser der er modtaget, er man nødt til at udvikle algoritmer, der på en effektiv måde finder de steder i netværket, der med stor sandsynlighed indeholder fejl.

For at kunne udvikle algoritmer til at løse fejlfindingsproblemet, er det nødvendigt med en præcis beskrivelse af problemet. En del af dette projekt er derfor, at analysere hvad der identificerer en sandsynlig forklaring på en fejl i et elforsyningsnet. Endvidere skal der opstilles en formel matematisk definition af problemet, som kan danne grundlag for argumentation for korrekthed af algoritmerne.

Målet med projektet er at implementere flere forskellige algoritmer, der løser fejlfindingsproblemet og derefter analysere styrker og svagheder, i form af kvaliteten af de fundne forklaringer og med hensyn til køretid. Specielt vil der blive lagt vægt på algoritmer, der udnytter styrkerne ved arraybaseret logik og der vil blive fokuseret på, hvilke udvidelser til den arraybaserede model man kunne ønske sig, for bedre at kunne løse denne type problem.

[1] En brute force algoritme er en algoritme, der slavisk gennemsøger alle løsninger for at finde den bedste.

Udgangspunktet for at udvikle disse algoritmer vil være en række kendte algoritmer og principper indenfor følgende områder:

- Constraint Based Reasoning
- Array Based Logic
- Grafteori
- Flowanalyse
- Graph Clustering

De algoritmer der beskrives i dette projekt benytter udelukkende topologien i elforsyningsnettet som grundlag for beregningen, dvs. afstanden mellem de enkelte komponenter samt hvordan de er koblet. Det vil blive tilstræbt at designe algoritmerne således, at flere parametre senere kan medtages i beregningen. Desuden vil rapporten indeholde en perspektivering, der beskriver et udvalg af andre parametre, som formentlig vil kunne forbedre beregningen. Hvorledes det fysiske elforsyningsnet omsættes til en abstrakt grafrepræsentation, der kun indeholder topologi, vil også blive behandlet i rapporten.

4 Afgrænsning

Her beskrives de forudsætninger der er gældende for vores indgangsvinkel til fejlfindingsproblemet, desuden vil vi afgrænse problemstillingen for at bevare fokus i rapporten. Det er vigtigt at fastslå, at dette projekt kun omhandler håndterning af fejl i lavspændingsnettet. Vi vil derfor ikke se på elforsyningsnet med højere spændinger. Endvidere vil vi betragte de strømkilder, hvor lavspændingsnettet får strøm fra højspændingsnettet, som ufejlbarlige. Hvis der opstår en fejl på en af disse strømkilder, findes der allerede systemer til at detektere dette.

For at kunne håndtere problemet indenfor en rimelig tidsramme, må kompleksiteten reduceres. Vi vælger at gøre dette, ved at transformere det faktiske elforsyningsnet til et abstrakt netværk repræsenteret ved en graf. Der ligger en væsentlig afgrænsning i abstraktionen mellem det fysiske elforsyningsnet og grafen. Grafrepræsentationen skjuler mange detaljer omkring netværket, eksempelvis størrelse og placering af sikringer. Sådanne oplysninger vil i nogle tilfælde kunne bruges til at lave beregninger på primære fejl og følgefejl, hvilket altså ikke er fokus for dette projekt.

Formålet med projektet er ej heller at skrive software, der kan konstruere graf-repræsentationen ud fra eksempelvis GIS[2] data. I projektet vil alle inputdata være grafer, der tænkes genereret af at andet system.

Ved hjælp af elektrotekniske love er det muligt, at forudsige en del om, hvor og hvordan fejl opstår og forplanter sig i et elforsyningsnet. I dette projekt betragtes alle forbindelser (kanter i grafrepræsentationen) som havende samme sandsynlighed for fejl. Algoritmen vil altså udelukkende basere sig på grafanalyse og ikke på elektrotekniske beregninger.

Udover de elektrotekniske detaljer findes der andre parametre, der kan påvirke sandsynligheden for, at en bestemt forbindelse er defekt, f.eks. hvorvidt der er tale om en jord- eller luftledning. Da datagrundlaget ikke er tilstrækkeligt detaljeret til, at disse sandsynligheder kan beregnes, vil de ikke blive brugt i projektet.

Endvidere ligger det udenfor projektets rammer at vurdere, hvor mange teknikere der skal allokeres til at reparere de defekte forbindelser.

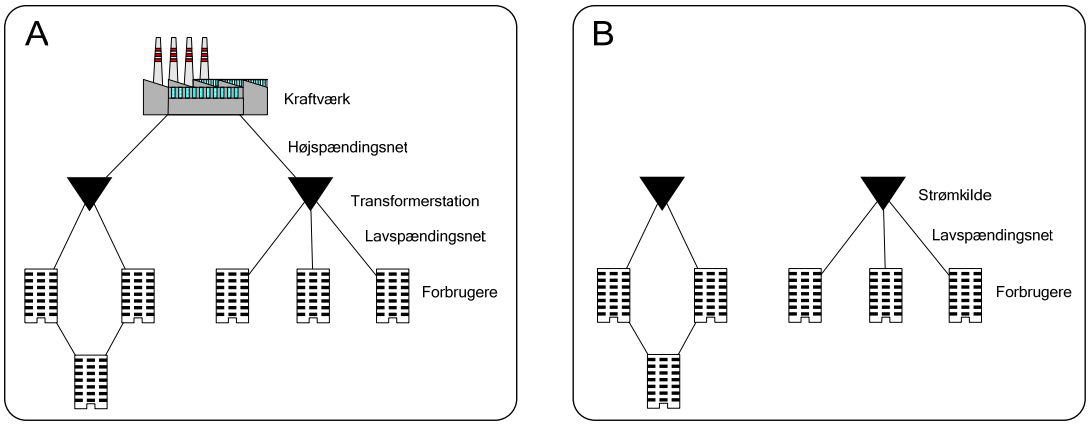
Det software der udvikles i projektet er tænkt som en prototype, der belyser de algoritmiske aspekter af forskellige metoder til at finde fejl i elforsyningsnet. Der er altså ikke tale om software, der er egnet til at indgå i et produktionsmiljø, men nærmere et analyseværktøj DONG Energy og andre kan bruge til at vurdere, hvilke algoritmer de vil basere deres produktionsmiljø på.

[2] Geografisk Informations System se [LONGLEY]

5 Elforsyningsnet

Da fokus for dette projekt er håndtering af fejl i DONG Energys lavspændingsnet, er datagrundlaget en abstrakt repræsentation af netop lavspændingsnettet. Når man kun betragter lavspændingsdelen af elforsyningsnettet, bliver resultatet ikke ét sammenhængende net, men en række mindre delnet, se Figur 2. Hvorledes disse delnet er forbundet af højspændingsnettet, ligger udenfor rammerne af dette projekt.

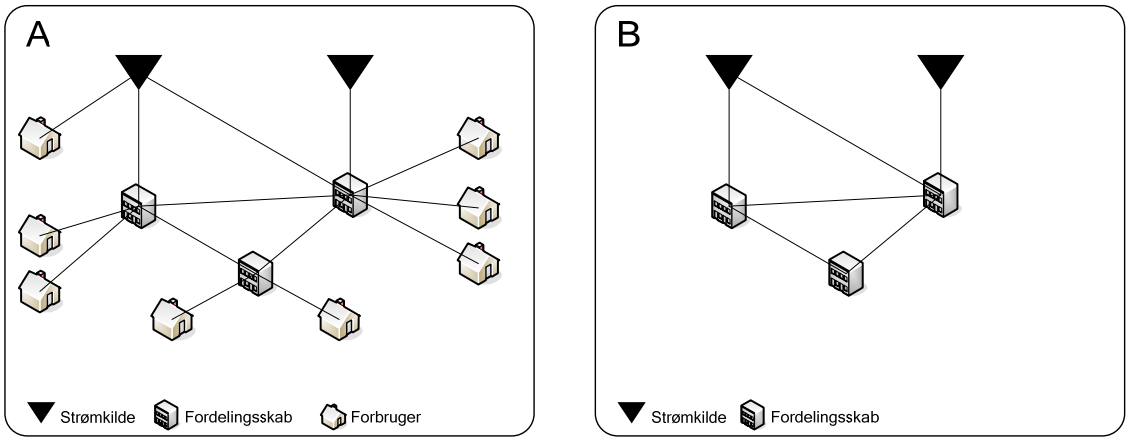
DONG Energys samlede elforsyningsnet består af ca. 400.000 fordelingsskabe spredt over et stort antal delnet. De største delnet indeholder ca. 3000 fordelingsskabe.



Figur 2: A: Skematisk repræsentation af et elforsyningsnet. Hele nettet er sammenhængende B: Repræsentation hvor det kun er lavspændingsnettet der betragtes. Nettet splittes nu op i flere delnet.

I det fysiske elforsyningsnet er der flere spændingsniveauer fra kraftværk til forbruger, men for overskuelighedens skyld, er disse slået sammen til ét niveau i figuren. Når højspændingsdelen skæres væk, har hvert lavspændingsnet en eller flere strømkilder, hvorfra delnettet forsynes med strøm, Figur 2 B. Da fejl i højspændingsnettet ikke betragtes i dette projekt, vil en strømkilde antages altid at virke - det er altså ikke muligt at forklare en fejl i lavspændingsnettet med, at en strømkilde er afbrudt. Denne begrænsning er acceptabel, da der på højspændingsdelen af elforsyningsnettet, findes automatiske systemer til detektering af sådanne fejl.

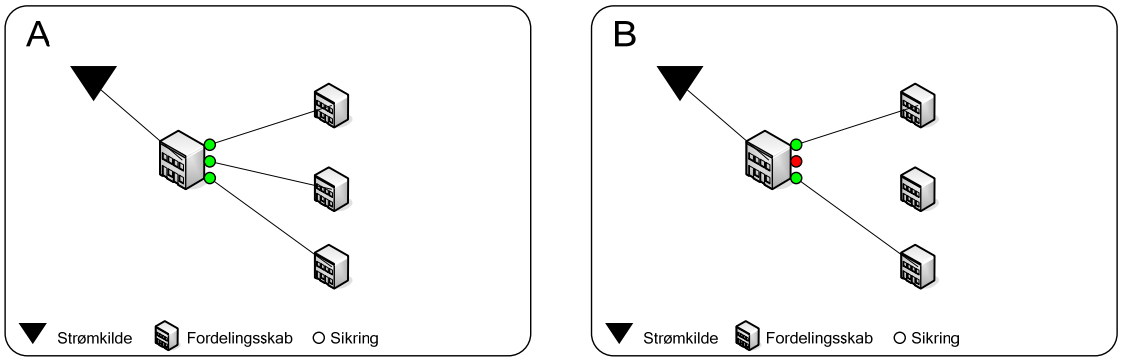
Udover kun at betragte lavspændingsnettet, indlægges der yderligere den begrænsning, at stikledninger fra fordelingsskabene ud til de enkelte forbrugere ikke medtages i beregningen, se Figur 3. Den repræsentation af elforsyningsnettet der danner grundlag for beregningen, medtager kun fordelingsskabe, transformatorer ol.



Figur 3: A: Lavspændingsnet med forbrugere. B: Den del af nettet fra A, der anvendes som datagrundlag i dette projekt.

Hvorledes fejl på stikledninger ud til de enkelte forbrugere lokaliseres, behandles i afsnittet Problemdefinition.

Nogle af de komponenter, der anvendes i lavspændingsnettet, har separate sikringer på hver indgang eller udgang. I grafrepræsentationen af elforsyningsnettet modelleres dette ved, at en forbindelse fjernes fra nettet, hvis en sikring er sprunget. På Figur 4 A ses et fordelingsskab med tre udgange, der hver har en separat sikring. På Figur 4 A er ingen sikringer sprunget, hvorfor der er forbindelse videre til de tre fordelingsskabe længere nede i nettet. På Figur 4 B er sikring nummer to sprunget, hvilket modelleres ved, at forbindelsen fra denne udgang fjernes fra nettet. Det er altså ikke nødvendigt at behandle sikringer som en selvstændig objekttype i nettet, da deres funktion udelukkende modellers ved, at indsætte eller fjerne forbindelser.



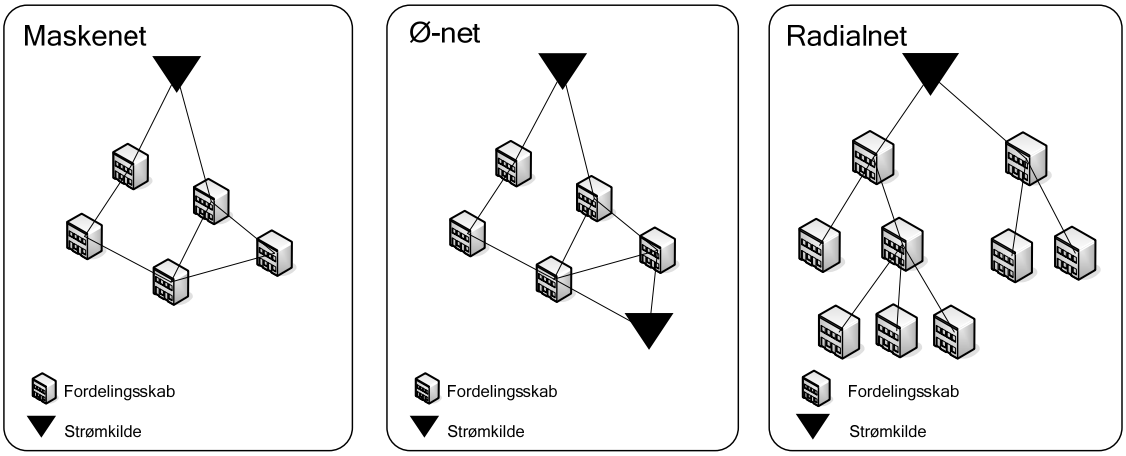
Figur 4: Hvis en sikring springer, modelleres det ved at fjerne forbindelsen fra netværket.

5.1 Kategorisering af delnet

De enkelte delnet i netværket kategoriseres i tre typer:

- Maskenet - ringforbundne net med én strømkilde
- Ø-net - ringforbundne net med to eller flere strømkilder
- Radialnet - ikke ringforbundne net med én strømkilde

Det elforsyningsnet DONG Energy driver i hovedstadsområdet er en sammenkobling af elforsyningsnettene fra det tidligere Københavns Energi og NESA. Københavns Energi har fortrinsvis benyttet maske- og ø-net mens NESA i højre grad har anvendt radialnet. Man er derfor nødt til at tage højde for, at alle tre typer net kan forekomme. På Figur 5 er de tre typer delnet illustreret.



Figur 5: De tre typer delnet der anvendes i dette projekt.

For en gennemgang af de forskellige typer af elforsyningsnet se [NÆRFØR]

6 Problemdefinition

Fejl i elforsyningsnettet håndteres i tre trin:

1. Lokalisering af fejl (fejlfindingsproblemet)
2. Generering af arbejdsordrer
3. Afrapportering

Forud for denne proces ligger naturligvis et arbejde med at modtage fejlmeldinger fra forbrugerne og akkumulere dem over tid. I dette afsnit gives en gennemgang af de tre trin i processen. Det primære fokus for dette projekt er udvikling af algoritmer til løsning af fejlfindingsproblemet, hvorfor trin 1 vil få den mest detaljerede gennemgang i det følgende.

6.1 Fejlfindingsproblemet

Med udgangspunkt i problemformuleringen kan fejlfindingsproblemet defineres som:

> *Givet en række fejlmeldinger fra forbrugere samt information om elforsyningsnettets topologi, find da den eller de forbindelser, der med størst sandsynlighed er årsag til fejlen.*

Som det fremgår af ovenstående er der ikke tale om et problem, der altid har en entydig løsning. Der kan være forskellige vurderinger af, hvilke parametre og vægte der skal anvendes, for at beregne sandsynligheden for at en bestemt fejl skyldes netop en bestemt tilstand i nettet. I dette afsnit vil de parametre, der påvirker beregningen blive beskrevet og en generel metode til løsning af problemet vil blive skitseret.

Udgangspunktet for at løse fejlfindingsproblemet er en repræsentation af elforsyningsnettet, som den der blev beskrevet i afsnittet Elforsyningsnet, samt en række fejlmeldinger fra forbrugerne.

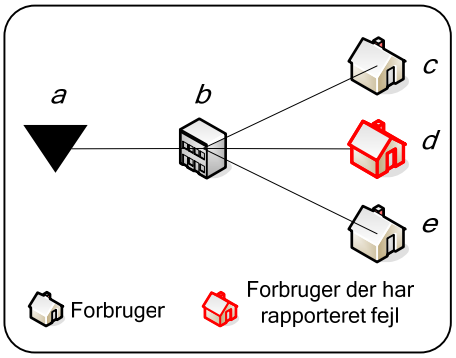
6.1.1 Akkumulering af fejl

For at få et realistisk billede af omfanget af en fejl, er det nødvendigt at akkumulere fejlmeldinger over et tidsrum, for at danne en samlet fejlrapport. Når en fejl i elforsyningsnettet opstår, vil der gå en kortere eller længere periode, før alle berørte forbrugere har rapporteret fejlen. Ved normal drift akkumuleres fejlmeldinger i op til 10 minutter, mens der i stormtilstand[3] kan bruges akkumuleringstider på helt op til 30 minutter. Problemet er at beregne, hvor mange af de modtagne fejlmeldinger, der skyldes en og samme fejl samt hvor denne fejl er placeret. Det videre forløb afhænger af, hvor mange fejlmeldinger der er modtaget.

[3] En speciel drifttilstand forsyningsselskaberne overgår til efter storm ol.

6.1.2 Fejl på stikledninger

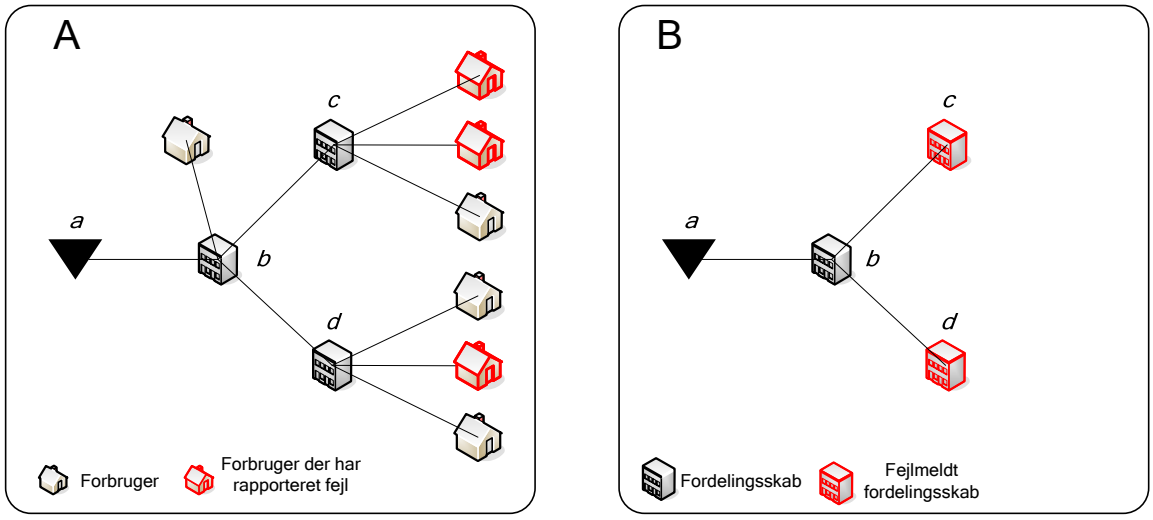
Hvis der i et område af elforsyningsnettet kun modtages en enkelt fejlrapport, placeres fejlen på stikledningen fra forsyningsskabet ind til forbrugeren. I dette tilfælde er der ikke brug for yderligere beregning, da der direkte kan genereres en arbejdsordre om at efterse den pågældende stikledning. I eksemplet på Figur 6 herunder, illustreres netop denne situation. I dette tilfælde vil der genereres en arbejdsordre om at efterse forbindelsen *bd*.



Figur 6: Enkeltstående fejl. Der genereres en arbejdsordre om at efterse stikledningen *bd*.

6.1.3 Større fejl

Hvis der modtages flere fejlmeldinger i samme område, er der formentlig tale om en større fejl på et eller flere fordelingsskabe. I dette tilfælde vil det være nødvendigt at foretage en beregning, for at lokalisere den mest sandsynlige kilde til fejlen.



Figur 7: A: Eksempel på fejlmeldinger fra forbrugere. B: Oversættelse til fejlrapport for elforsyningsnettet.

I Figur 7 vises hvorledes fejlmeldinger fra forbrugerne oversættes til en fejlrapport i nettet. Der ses bort fra stikledninger ud til den enkelte forbruger og fejlene placeres på de

fordelingsskabe forbrugerne er tilknyttet. Når forklaringen på fejltilstanden i figuren skal findes, er der to muligheder.

- Forbindelserne *bc* og *bd* er defekte
- Forbindelsen *ab* er defekt.

Hvilken af disse to forklaringer der betragtes som mest sandsynlig, afhænger af hvordan de forskellige elementer i forklaringen vægtes. Man kan sige, at der til hver forklaring er tilknyttet en sandsynlighed, der sammensættes af to parametre:

- $\boldsymbol{\alpha}$: Sandsynligheden for at et fordelingsskab virker
- $\boldsymbol{\beta}$: Sandsynligheden for at en forbindelse virker

De to mulige forklaringer vil opnå forskellig sandsynlighed, afhængigt af de to parametre. I den forklaring, hvor forbindelserne *bc* og *bd* er defekte, skal to forbindelser fejlrapporteres. I den alternative forklaring, hvor kun *ab* er defekt, reduceres antallet af fejlrapporterede forbindelser til en, til gengæld er en konsekvens af denne forklaring, at fordelingsskabet *b* også må være afbrudt.

Det følger intuitivt, at det er usandsynligt, at et stort antal forbindelser går i stykker på samme tid, og omvendt er det usandsynligt, at et stort antal forbrugere uden strøm ikke rapporterer fejlen. Derfor findes den mest sandsynlige forklaring som en vægtning mellem antallet af afbrudte forbrugere og antallet af defekte forbindelser. Denne vægtning udtrykkes som forholdet mellem α og β.

I afsnittet Formalisering af problemet vil en formel matematisk formulering af algoritmen til håndtering af større fejl blive givet. Endvidere vil det i afsnittet Indlæring blive analyseret, hvorledes parametrene α og β kan fastlægge empirisk.

6.2 Generering af arbejdsordrer

Resultatet af at behandle en fejl med ovenstående algoritme er, at systemet finder en eller flere forbindelser i elforsyningsnettet, der med stor sandsynlighed er defekte. Når der skal genereres arbejdsordrer, genereres der én ordre for hvert område af nettet, hvor der er rapporteret fejl. I eksemplet fra Figur 7, hvor tre fejlmeldinger slås sammen til en fejlrapport, og fejlrapporten forklares med en eller to defekte forbindelser, resulterer det i én arbejdsordre om at efterse den/de pågældende forbindelser. Hvis man har separate klynger af fejl i forskellige dele af nettet, og i særdeleshed hvis fejlene spreder sig over flere forskellige delnet, genereres en arbejdsordre for hver klynge af fejl. En enkeltstående fejl på en stikledning resulterer altid i én arbejdsordre.

For at undgå ulykker og udnytte ressourcerne optimalt, sørger man så vidt muligt for kun at generere én arbejdsordre for hvert delnet. På denne måde vil en tekniker altid vide hvorvidt der er spænding på en forbindelse, da vedkommende er den eneste der arbejder på det pågældende delnet. På store delnet kan det dog være hensigtsmæssigt at generere flere arbejdsordrer, hvis fejlene ligger i separate klynger.

6.3 Afrapportering

Når en tekniker har afsluttet arbejdet på en af de arbejdsordre systemet udsender, skal den faktiske årsag til fejlen afrapporteres. Det primære formål med afrapporteringen er, at forsyningsselskabet skal dokumentere deres forsyningssikkerhed, således at man præcis kan sige, hvad risikoen for en afbrydelse er. Disse tal skal indrapporteres til myndighederne. De tre oplysninger i teknikerens rapport, der er relevante i forbindelse med beregning af forsyningssikkerhed er:

- Hvilken arbejdsordre knytter afrapporteringen sig til.
- Hvilke forbindelser var defekte.
- Hvornår blev problemet løst

Rapporten vil i de fleste tilfælde indeholde en lang række andre oplysninger, f.eks. tidsforbrug, materialeforbrug etc.

Første trin i behandling af teknikerens rapport er konsekvensberegning. Med udgangspunkt i de forbindelser, teknikeren har rapporteret som defekte, beregnes hvilke forbrugere der har været uden strøm. Eksemplet i Figur 7 A kunne resultere i en arbejdsordre om at efterse forbindelserne *bc* og *bd*, hvis denne forklaring blev vurderet som værende mest sandsynlig. Da teknikeren ankommer til stedet, finder han imidlertid ud af, at det faktisk er forbindelsen *ab* der er defekt, hvilket han rapporterer. Resultatet af konsekvensberegningen bliver, at alle tre fordelingsskabe på Figur 7 B har været afbrudt, i modsætning til kun to som først antaget.

Varigheden af afbrydelsen kan beregnes på grundlag af, hvornår den første af de fejlrapporter der resulterede i den pågældende ordre er modtaget samt tidspunktet hvor teknikeren rapporterede problemet løst. Med oplysninger om afbrydelsens omfang og varighed, har man tilstrækkelige informationer til at beregne forsyningssikkerheden i elforsyningsnettet.

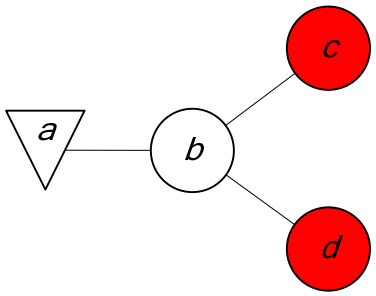
Udover konsekvensberegningen har afrapporteringen også speciel interesse for netop dette projekt, da hver rapport kan bruges til at estimere hensigtsmæssige værdier af parametrene α og β og på den måde løbende forbedre systemet. Dette vil blive beskrevet nærmere i afsnittet Indlæring.

7 Formalisering af problemet

Her gives en formel matematisk definition af fejlfindingsproblemet, for at kunne beskrive kravene til en algoritme, der løser dette problem.

7.1 Grafrepræsentation af elforsyningsnettet

Selve elforsyningsnettet beskrives som en ikke-orienteret graf $G = (V, E)$, hvor V er mængden af knuder i grafen og E er kanterne mellem de enkelte knuder. Eksemplet fra Figur 7 B er oversat til grafrepræsentation som vist på Figur 8.



Figur 8: Grafrepræsentation af nettet fra Figur 7 B, med knuderne *c* og *d* fejlmeldt.

En graf kan lagres på flere forskellige måder. En figur som den på Figur 8 er svær at behandle i en computer, hvorfor man typisk vælger at lagre grafen på listeform eller matrixform. På listeform ser grafen fra Figur 8 således ud:

$$G = (V, E) = (\{a, b, c, d\}, \{(a, b), (b, c), (b, d)\}) \quad (1)$$

På listeform er grafen en liste af knuder efterfulgt af en liste af par, der repræsenterer de knuder der er forbundet af kanter. På matrixform bliver grafen på Figur 8 til:

$$G = \begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 1 & 0 & 0 \\ b & 1 & 0 & 1 & 1 \\ c & 0 & 1 & 0 & 0 \\ d & 0 & 1 & 0 & 0 \end{array} \quad (2)$$

På matrixform angiver et 1-tal at to knuder er forbundet.

Da grafen er ikke-orienteret, gælder:

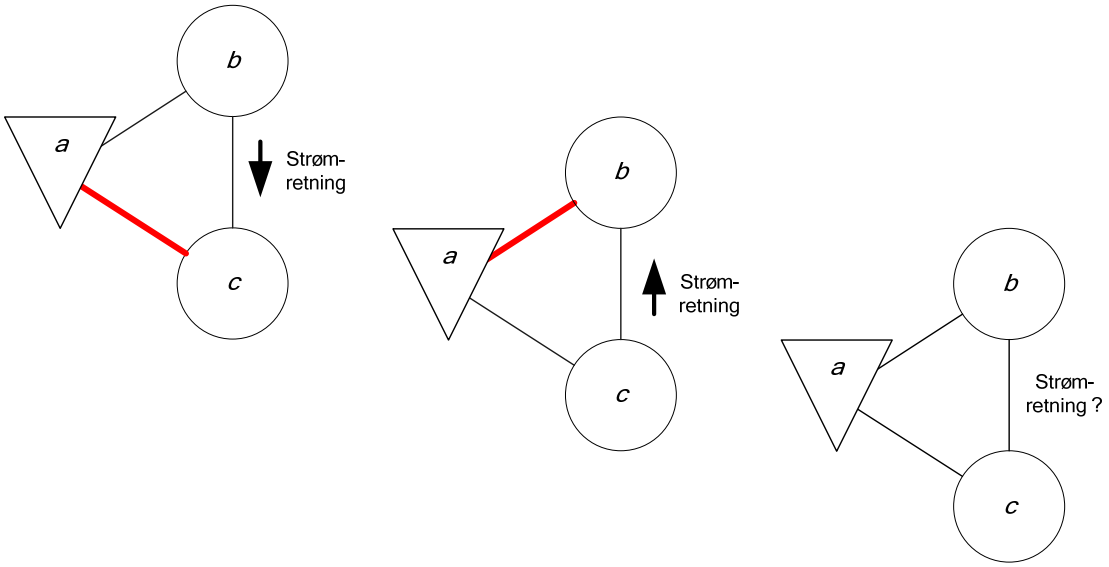
$$(a, b) \in E \Leftrightarrow (b, a) \in E \quad (3)$$

altså hvis kanten *(a, b)* findes, findes kanten *(b, a)* også. I listerepræsentationen af grafen (1) skrives kun en af kanterne, f.eks. *(a, b)*, mens eksistensen *(b, a)* er givet implicit via

(3). I matrixrepræsentationen (2) observeres egenskaben fra (3) ved, at matricen er symmetrisk omkring diagonalen. En mere kompakt repræsentation af grafen kunne derfor være, kun at lagre alle tal der ligger over diagonalen.

Fordelen ved listerepræsentationen er primært at den er kompakt, dvs. at hukommelsesforbruget er lavt. Fordelen ved en matrixrepræsentation er at visse operationer, som eksempelvis at afgøre om to knuder er forbundet af en kant, beregningsmæssigt er lettere. Se i øvrigt [CORMEN] kapitel 22 for en detaljeret gennemgang af de forskellige metoder til lagring af en graf.

Elforsyningsnettet modelleres som en ikke-orienteret graf, da det i maske- og ø-net ikke altid er muligt at forudsige, hvilken vej strømmen løber. I nogle tilfælde vil strømretningen også afhænge af, om der findes defekte forbindelser andre steder i nettet. Derfor er den mest fleksible repræsentation af problemet en ikke-orienteret graf, hvor den faktiske strømretning ikke har indflydelse på løsningen af problemet. De skiftende strømretninger i et maskenet er illustreret på Figur 9.



Figur 9: Strømretning i en forbindelse kan ikke altid fastlægges. Knuden *a* er en strømkilde og røde kanter er defekte.

Det maksimale antal kanter ud fra hver knude i grafen er $|V|-1$, da en knude i grafen maksimalt kan være forbundet én gang til alle andre knuder, og en knude ikke må have en kant der fører tilbage til den selv. Denne egenskab lægger en øvre grænse på antallet af kanter i en graf, $|E|$:

$$|E| < |V|^2 \qquad (4)$$

Når man repræsenterer et elforsyningsnet med eksempelvis 100 knuder, vil hver knude langt fra have 99 forbindelser til andre knuder. For en graf der repræsenterer et

elforsyningsnet gælder der altså, at $|E| << |V|^2$, hvilket også kan udtrykkes som at graden af grafen er lav. Denne egenskab er vigtig, da den kan bruges til at forbedre køretiden for de algoritmer, der vil blive designet gennem projektet.

Definition 1: Naboer
Mængden af knuder der kan nås ved at følge maksimalt *n* kanter startende fra *v* betegnes $N_n(v)$. Mængden $N_1(v)$ kaldes naboer til v. $N_n(V')$ defineres som mængden af knuder, der kan nås fra mindst en knude, $v \in V'$, ved at følge maksimalt *n* kanter. Naboer til en mængde af knuder, betegnes $N_1(V')$.

Definition 2: Stier
En mængde af kanter, $\{(v_1, v_2), (v_2, v_3), \cdots, (v_{n-2}, v_{n-1}), (v_{n-1}, v_n)\} \subseteq E$ kaldes en sti fra v_1 til v_n.
En sti i grafen er en mængde af kanter, der tilsammen forbinder knuden v_1 med en anden knude v_n. Eksempelvis er stien fra *c* til *d* i Figur 8 $\{(c,b),(b,d)\}$. Bemærk at der i maske- og ø-net ofte findes flere forskellige stier mellem to knuder.

Definition 3: Indikatorfunktion for stier
Funktionen $p : V \times V \to \{0,1\}$ defineres som:

$$p(v_1, v_2) = \begin{cases} 0 & \textit{hvis der ikke findes en sti fra } v_1 \textit{ til } v_2 \\ 1 & \textit{hvis der findes en sti fra } v_1 \textit{ til } v_2 \end{cases} \qquad (5)$$

I eksemplet i Figur 9 er alle knuder forbundet til alle andre knuder, hvorfor $p(v_1, v_2) = 1$ for alle værdier af $(v_1, v_2) \in V^2$.

Definition 4: Indikatorfunktion for stier mellem mængder
Funktionen $P : V^n \times V^m \to \{0,1\}$ defineres som:

$$P(V_1, V_2) = \max(p(v_1, v_2)) \qquad v_1 \in V_1; v_2 \in V_2 \qquad (6)$$

hvor $V_1, V_2 \subseteq V$

dvs. $P(V_1, V_2) = 1$ hvis og kun hvis der eksisterer mindst en sti fra en knude v_1 i V_1 til en knude v_2 i V_2.

7.2 Fejlrapporter

For at kunne håndtere fejlrapporter i nettet, indføres to specielle delmængder af knuderne V i grafen.

Definition 5: Strømkilder

Mængden af alle knuder, der fungerer som strømkilder, jf. afsnittet Elforsyningsnet, betegnes *T*. Det gælder at $T \subseteq V$.

Definition 6: Fejlmeldte knuder
Mængden af alle fejlmeldte knuder i grafen betegnes *F*. Det gælder at $F \subseteq V$. Hele mængden *F* kaldes en fejlrapport.

Endvidere gælder relationen,

$$T \cap F = \varnothing, \tag{7}$$

da en strømkilde jf. afsnittet Elforsyningsnet aldrig kan fejlmeldes. En fejlrapport som den i Figur 8 viste repræsenteres ved mængden $F = \{c, d\}$.

Definition 7: Indikatorfunktion for tilstanden af et netværk
Tilstanden af et netværk betegnes $R(G,T,F)$ og defineres som:

$$R(G,T,F) = \begin{cases} 1 & T = \varnothing \vee F = \varnothing \\ inv(P(T,F)) & T \neq \varnothing \vee F \neq \varnothing \end{cases}, \tag{8}$$

dvs. netværket er i en lovlig tilstand, $R(G,T,F) = 1$, hvis og kun hvis der ikke findes en sti gennem grafen fra en fejlmeldt knude til en strømkilde.

7.3 Forklaring af fejlrapport

Det antages, at der for grafrepræsentationen af nettet $G = (V,E)$ hvor $F = \varnothing$ gælder, at $P(\{v\},T) = 1$ for alle knuder $v \in V \setminus T$. Dvs. hvis der ikke er modtaget fejlrapporter fra forbrugere, antages alle knuder i netværket at være forbundet til mindst en strømkilde. Det er dog ikke noget krav at det forholder sig sådan. Hvis der findes knuder der ikke er forbundet til en strømkilde, er nettet stadig i en lovlig tilstand, jf. (8).

Ifølge ovenstående antagelse vil enhver fejlrapport bringe nettet i en ulovlig tilstand, da fejlmelding af en vilkårlig knude $v \in V \setminus T$ vil medføre, at der findes en sti fra *v* til en strømkilde $t \in T$

Definition 8: Forklaring
Givet grafen $G = (V,E)$ og fejlrapporten $F \subseteq V$, da kaldes en mængde af kanter $S \subseteq E$ en forklaring af *F,* hvis og kun hvis grafen $G' = (V, E \setminus S)$ er i en lovlig tilstand.

Betragtes eksemplet fra Figur 8, med fejlrapporten $F = \{c, d\}$ ses to mulige forklaringer, S_a og S_b hvor $S_a = \{(b,c),(c,d)\}$ og $S_b = \{(a,b)\}$

Den samlede mængde af alle forklaringer, der forklarer en given fejlrapport *F* kaldes Q_∞. I eksemplet ovenfor er der kun to mulige forklaringer på *F*, nemlig S_a og S_b, hvorfor $Q_\infty = \{S_a, S_b\} = \{\{(b,c),(c,d)\},\{(a,b)\}\}$

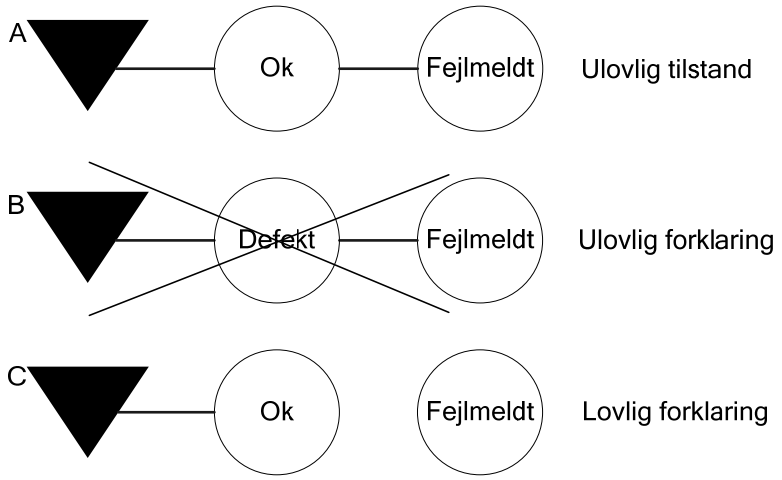
Størrelsen af en forklaring, $|S|$, betegner antallet af kanter indeholdt i S. I ovenstående eksempel er $|S_a| = 2$ og $|S_b| = 1$.

Det samlede antal forklaringer på en fejlrapport betegnes $|Q_\infty|$. I eksemplet er $|Q_\infty| = 2$ men i det generelle tilfælde gælder det:

$$|Q_\infty| < 2^{|E|}, \qquad (9)$$

da enhver kombination af at fjerne en eller flere kanter fra *G* er en potentiel forklaring på *F*.

For at finde en forklaring på en ulovlig tilstand, eliminerer man stier fra en fejlrapporteret knude tilbage til strømkilderne. I den model af elforsyningsnettet der anvendes i dette projekt, elimineres en sti altid ved at fjerne kanter fra grafen; **en sti kan aldrig elimineres ved at markere knuder i nettet som defekte**, se Figur 10.



Figur 10: En fejlrapport forklares altid ved at fjerne kanter fra grafen. A: Der modtages en fejlrapport som bringer nettet i en ulovlig tilstand. B: Fejlen kan ikke forklares ved en defekt knude. C: Fejlen fra A forklares ved at fjerne en kant, hvorved grafen kommer tilbage i en lovlig tilstand.

7.4 Konsekvens af en forklaring

Til enhver forklaring *S* hører en konsekvens $K(S)$, hvorved der forstås den mængde af knuder i *V*, der afbrydes når kanterne i *S* fjernes fra *E*.

Definition 9: Indikatorfunktion for afbrudte knuder
Funktionen $u : V \to \{0,1\}$ angiver om en knude er afbrudt eller ej. $u(v) = 0$ hvis og kun hvis knuden er afbrudt.

$$u(v) = \begin{Bmatrix} 1 & v \in T \\ P(\{v\}, T) & v \notin T \end{Bmatrix}, \quad (10)$$

Per definition kan en strømkilde, $v \in T$, ikke være afbrudt, jf. afsnittet Elforsyningsnet. Alle andre knuder i grafen er afbrudt, hvis og kun hvis der ikke findes mindst en sti til en strømkilde.

Konsekvensen $K(S)$ er defineret som mængden af knuder $V' \subseteq V$ der er afbrudt, givet en fejlrapport på grafen *G* med forklaringen *S*.

$$V' = K(S) = \{v \in V \mid u(v) = 0\} \quad (11)$$

I eksemplet ovenfor bliver konsekvenserne af de to forklaringer $K(S_a) = \{c, d\}$ og $K(S_b) = \{b, c, d\}$

Antallet af afbrudte knuder i en konsekvens betegnes $|K(S)|$. I ovenstående eksempel bliver $|K(S_a)| = 2$ og $|K(S_b)| = 3$

7.5 Objektfunktionen

I langt de fleste tilfælde vil der findes mere end en mulig forklaring, *S*, på en fejlrapport *F*, jf. (9). Det er derfor nødvendigt at tilknytte en sandsynlighed til hver forklaring, således at den mest sandsynlige kan findes. Til dette formål anvendes objektfunktionen:

$$f(S) = \alpha \cdot |K(S)| + \beta \cdot |S| \quad (12)$$

hvor α og β er vægte med følgende tolkninger:

α = Den vægt der knyttes til at afbryde en knude i forklaringen.
β = Den vægt der knyttes til at markere en kant som defekt i forklaringen.

Som beskrevet i afsnittet Fejlfindingsproblemet er store værdier af både $|S|$ og $|K(S)|$ usandsynlige, hvilket fører frem til, at problemet matematisk set er at finde den forklaring, $S_{\min}$, der minimere værdien $f(S)$ således at

$$f(S_{\min}) \leq f(S) \; \forall \; S \in Q_\infty \quad (13)$$

I Tabel 1 herunder illustreres, at forskellige værdier af α og β kan gøre udslaget for, om forklaringen S_a eller S_b fra eksemplet betragtes som mest sandsynlig:

Vægte, α, β	Forklaring, S	Værdi af objektfunktion, $f(S)$
$\alpha=1$ $\beta=2$	S_a	$f(S_a)=1\cdot 2+2\cdot 2=6$
	S_b	$f(S_b)=1\cdot 3+2\cdot 1=5$ ✓
$\alpha=2$ $\beta=1$	S_a	$f(S_a)=2\cdot 2+1\cdot 2=5$ ✓
	S_b	$f(S_b)=2\cdot 3+1\cdot 1=7$

Tabel 1: Forskellige værdier for α og β og deres indflydelse på objektfunktionen.

En relativ høj α værdi betyder at der er en høj vægt forbundet med at have afbrudte knuder i en forklaring, man stræber altså her efter forklaringer der primært forklarer afbrydelsen ved hjælp af defekte kanter. Omvendt betyder en relativ høj β værdi at der er en høj vægt forbundet med at markere en kant som defekt, man stræber altså her efter forklaringer der forklarer afbrydelsen ved afbrudte knuder.

7.6 Reduktion af løsningsrummet

I praksis viser det sig, at beregning af Q_∞ hurtigt bliver meget omfattende. Hvis man eksempelvis har et elforsyningsnet med 34 forbrugere og to af dem rapporterer fejl, vil en fuldstændig beregning af udtrykket Q_∞ kræve, at alle kombinationer af at afbryde en eller flere af de resterende 32 forbrugere evalueres. Således vil man allerede for dette lille net skulle evaluere $2^{32}\approx 4{,}3$ mia. kombinationer. Derfor indføres et mere begrænset løsningsrum, f.eks. Q_2, hvilket er mængden af alle forklaringer på fejlrapporten F, der kun inkluderer knuder der ligger maksimalt to kanter fra en fejlrapporteret knude i F.

Definition 10: Afgrænsede løsningsrum

Q_n er en delmængde af den samlede mængde forklaringer, $Q_n \subseteq Q_\infty$. Q_n er defineret ved, at en vilkårlig forklaring, S, tilhører Q_n hvis og kun hvis $K(S)\subseteq N_n(F)$. Altså, hvis alle knuder der afbrydes som konsekvens af forklaringen ligger maksimalt n kanter fra en fejlrapporteret knude, tilhører forklaringen Q_n.

Hvis man betragter fejlrapporten $F=\{c,d\}$, se Figur 8, får man følgende mulige forklaringer.

- $S_a=\{(b,c),(b,d)\}\quad K(S)=\{c,d\}$
- $S_b=\{(a,b)\}\quad K(S)=\{b,c,d\}$
- $S_c=\{(a,b),(b,c)\}\quad K(S)=\{b,c,d\}$
- $S_d=\{(a,b),(b,d)\}\quad K(S)=\{b,c,d\}$
- $S_e=\{(a,b),(b,c),(b,d)\}\quad K(S)=\{b,c,d\}$

hvoraf det følger at $Q_0 = \{S_a\}$ og $Q_1 = \{S_a, S_b, S_c, S_d, S_e\}$. Da knuden *a* i Figur 8 er strømkilde og derfor ikke kan afbrydes, ses det endvidere at $Q_\infty = Q_1$.

Det skal bemærkes at, i det samlede løsningsrum findes en lang række forklaringer, der aldrig bliver relevante i forhold til at bestemme $S_{\min}$. Betragt f.eks. S_c, S_d og S_e i ovenstående liste; det er tydeligt, at alle disse løsninger vil være dårligere end S_b, da den eneste forskel på disse løsninger og S_b er, at de inkluderer ekstra kanter, der ikke bidrager til at forklare *F*. Når der skal designes algoritmer til at løse problemet, skal det derfor tilstræbes, at udelukke disse overflødige forklaringer så hurtigt som muligt. Dette vil blive nærmere beskrevet i afsnittet Algoritmer.

Om antallet af forklaringer i de afgrænsede løsningsrum, $|Q_n|$ gælder der:

$$|Q_0| \le |Q_1| \le \cdots \le |Q_\infty| \quad \text{da} \qquad Q_0 \subseteq Q_1 \subseteq \cdots \subseteq Q_\infty . \tag{14}$$

Denne egenskab, sammenholdt med objektfunktionen, kan også bruges til at optimere de algoritmer der designes. Da objektfunktionen minimerer et forhold mellem antallet af afbrudte knuder og fejlramte kanter er det sandsynligt, at $S_{\min}$ skal findes i et af de mindre løsningsrum, hvorfor det er tilstrækkeligt at betragte disse. Dette giver en voldsom reduktion af det regnearbejde, der skal udføres for at løse problemet.

Definition 11: Umiddelbar forklaring
Den umiddelbare forklaring, S_U, betegner den løsning, $S \in S_0$ for hvilken det gælder,

$$|S_U| \le |S| \quad \textit{for alle } S \in S_0 \tag{15}$$

Den umiddelbare forklaring er altså den forklaring, der kun medtager forbindelser direkte ind mod en afbrudt knude, og kun de forbindelser der fører til en strømkilde.

7.7 Delkonklusion

I dette afsnit er fejlfindingsproblemet blevet formuleret ud fra grafteoretiske termer og matematisk mængdelære. De vigtigste af de matematisk objekter der er indført er:

1. En graf, *G*, der består af en mængde knuder, *V*, og en mængde kanter *E*.
2. En fejlrapport som er en delmængde af knuderne i *V*, nemlig alle de knuder forbrugerne har fejlmeldt, *F*.
3. En forklaring, *S*, som er en mængde kanter der skal fjernes fra grafen, for at forklare hvorfor en eller flere knuder er afbrudte.
4. En mængde af forklaringer, *Q*.
5. Konsekvensen af en forklaring, $K(S)$. Konsekvensen er en mængde af knuder, nemlig alle de knuder der er afbrudt, hvis kanterne i *S* fjernes fra grafen.

Endvidere er der i afsnittet introduceret en objektfunktion, for at knytte en sandsynlighed til hver at de fundne forklaringer, så den mest sandsynlige kan vælges.

Med udgangspunkt i den matematiske formulering vil der i næste afsnit blive udviklet algoritmer, så fejlfindingsproblemet kan løses effektivt på en computer.

8 Konfigurationsproblemer

I dette afsnit vil vi kort beskrive, hvad vi i denne rapport forstår ved et konfigurationsproblem. (eng. Constraint Satisfaction Problem)

Et konfigurationsproblem består af en række variable, som hver kan antage værdier indenfor et domæne. Desuden består problemet af en række relationer mellem disse variable. Som eksempel betragtes variablen a med domænet $\{1,2,3,4,5\}$, dvs. a kan antage heltallige værdier mellem 1 og 5. Hvis der indføres endnu en variabel, b, med samme domæne som a, kan relationen $a+b \leq 8$, bruges til at beskrive, at ikke alle kombinationer af a og b er lovlige.

Bemærk, at alle de konfigurationsproblemer vi behandler i denne rapport, defineres med diskrete domæner på alle variable.

For en præcis matematisk definition af konfigurationsproblemer se [TURNER].

I nogle tilfælde er man ikke kun interesseret i at finde en lovlig løsning[4] til et konfigurationsproblem, men i at finde den bedste af de lovlige løsninger. For at kunne definere den bedste løsning, indføres en objektfunktion i problemet, der til hver lovlig løsning knytter en godhed. Man ønsker således at finde den løsning, der maksimerer godheden. Konfigurationsproblemer hvor man leder efter den bedste løsning kaldes på engelsk Constraint Satisfaction Optimization Problems.

I denne rapport vil forkortelsen CSP blive brugt om almindelige konfigurationsproblemer og CSOP betegner konfigurationsproblemer, hvor der tilknyttes en godhed til de lovlige løsninger.

Det skal bemærkes, at både CSP og CSOP er velkendte NP-complete problemer [CORMEN], hvilket vil sige, at der sandsynligvis ikke findes algoritmer, der kan løse det generelle konfigurationsproblem i polynomiel tid.

8.1 Arrayteknologi

Arrayteknologi betegner en konkret metode til løsning af konfigurationsproblemer. Metoden bygger på arrays og matematiske operationer på disse, hvilket også giver metoden dens navn.

I denne rapport vil vi anvende den implementering af arrayteknologien, der stilles til rådighed af firmaet Array Technology i deres Array Studio produkt. Array Studio kan konstruere og behandle almindelige konfigurationsproblemer, CSP, men ikke direkte CSOP.

[4] En løsning til et konfigurationsproblem er en værditildeling til alle variable, der overholder alle relationer i problemet.

Da konfigurationsproblemet som nævnt er NP-complete, vil arrayteknologien i nogle tilfælde kræve lang tid for at løse et problem. En stor del af arbejdet kan dog udføres i kompileringsfasen, så problemet kan præprocesseres på et tidpunkt, hvor systemet ikke er hårdt belastet.

Hvorledes konfigurationsproblemer kan beskrives ved hjælp af arrays, samt det matematiske grundlag for array teknologien er beskrevet i [TURNER].

9 Algoritmer

I dette afsnit beskrives designet af de algoritmer, der er implementeret i projektet. Da målet med projektet er at undersøge, hvilke muligheder der er for at løse fejlfindingsproblemet, har vi implementeret tre forskellige algoritmer. Først og fremmest er der implementeret en algoritme, der bruger array teknologi i videst muligt omfang. Derudover er der implementeret en grafteoretisk algoritme, der opererer efter samme grundlæggende metode som arrayalgoritmen, samt en algoritme baseret på maksimum flow i en grafer. De tre algoritmers ydelse vil blive sammenlignet i afsnittet Benchmarking.

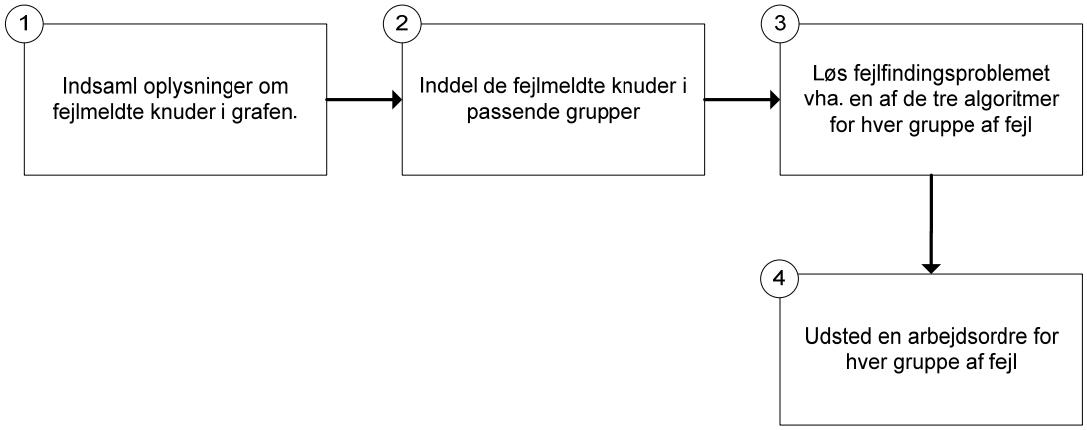
Algoritme	**Teknologi**
ATOMS	Benytter array teknologi i videst muligt omfang. For at løse fejlfindingsproblemet benyttes 3 forskellige arraymodeller.
BOMS	Denne algoritme anvender ikke array teknologi, men løser fejlfindingsproblemet udelukkende ved brug af klassisk grafteori. Den grundlæggende fremgangsmåde er den samme som for array algoritmen.
EOMS	Løser fejlfindingsproblemet ved at betragte det som et maksimum flow problem. Der anvendes en modificeret version af Edmunds-Karp algoritmen.

Tabel 2: Oversigt over OMS-algoritmer.

De tre algoritmer til løsning af fejlfindingsproblemet betegnes overordnet som Outage Management System (OMS) algoritmer, og navngives derfor ATOMS, BOMS og EOMS.

Udover de primære OMS algoritmer er det nødvendigt med endnu en algoritme, for at kunne håndtere udfald, nemlig en algoritme til at gruppere grafen.

Figur 11 viser den samlede struktur af udfaldshåndteringen i et elforsyningsnet. Først inddeles de fejlmeldte knuder i passende grupper, så antallet af arbejdsordrer bliver så lavt som muligt. Disse grupper vil ofte svare til et helt delnet, men for de største delnet kan man godt udstede to arbejdsordrer indenfor samme delnet. Herefter anvendes en af OMS algoritmerne på hver gruppe af fejl, for at finde forklaringen på fejlene. Slutteligt udstedes en arbejdsordre for hver gruppe af fejl på baggrund af den forklaring OMS algoritmen har beregnet.



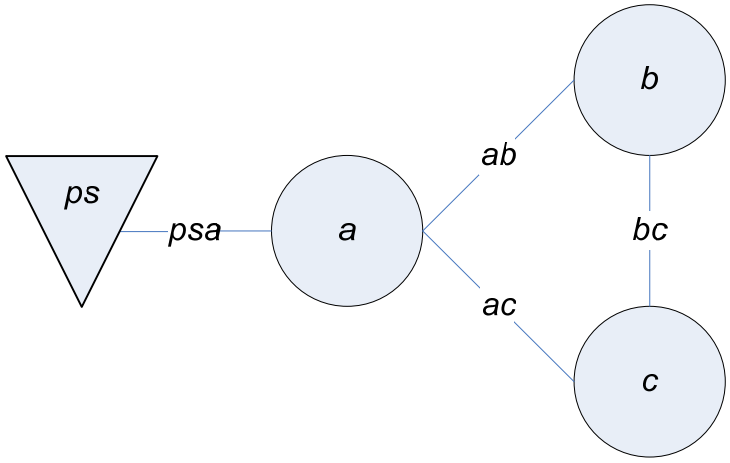
Figur 11: Den overordnede struktur af udfaldshåndteringen i elforsyningsnet.

I afsnittene herunder beskrives først de tre OMS algoritmer og derefter to metoder til gruppering af grafen. Den sidste del af algoritme afsnittet vil omhandle metoder til at fastlægge objektfunktionsparametrene α og β ud fra empiriske observationer.

9.1 ATOMS

Her beskrives en algoritme, der løser fejlfindingsproblemet ved brug af flere forskellige arraymodeller. Først gives en oversigt over de tre arraymodeller der anvendes, samt en beskrivelse af hvordan de opbygges ud fra grafens topologi. Derefter beskrives algoritmen i detaljer og samspillet mellem de tre modeller bliver gennemgået.

Til illustration af, hvorledes arraymodellerne konstrueres, vil grafen på Figur 12 blive anvendt. Knuden *ps* er strømkilde, mens de tre knuder *a*, *b* og *c* er almindelige knuder, der kan være afbrudt eller ikke afbrudt.



Figur 12: Eksempel på en graf til illustration af arraymodellerne.

9.1.1 Edge Cutting Model

Den første arraymodel vi konstruerer, har vi valgt at kalde "Edge Cutting Model" forkortet til ECM. ECM modellen bruges til at finde alle defekte kanter i grafen, når det er givet, hvilke knuder der er afbrudt.

I ECM modellen repræsenteres grafen ved en boolean variabel for hver knude og en boolean variabel for hver kant. Værdien af knude-variablerne fortolkes som afbrudt/ikke afbrudt, mens kant-variablerne fortolkes som defekt/ikke defekt.

I Boks 1 herunder vises AMS[5] koden for den ECM model, der konstrueres ud fra grafen på Figur 12.

```
Variables
     Node :
     {
          Ps : Boolean;
          A : Boolean;
          B : Boolean;
          C : Boolean;
     };

     Edge :
     {
          A_Ps : Boolean;
          A_C : Boolean;
          A_B : Boolean;
          B_C : Boolean;
     };

Relations
     Edge.A_Ps -> (Node.A=Node.Ps);
     Edge.A_C -> (Node.A=Node.C);
     Edge.A_B -> (Node.A=Node.B);
     Edge.B_C -> (Node.B=Node.C);
```

Boks 1: AMS kode for ECM modellen.

AMS koden i ovenstående eksempel kan genereres automatisk ud fra grafens struktur, vha. denne algoritme:

```
Generer en boolean variabel for hver knude i grafen.
Generer en boolean variabel for hver kant i grafen.

For (e = Alle kanter i grafen)
     v1 = Første knude forbundet til e
     v2 = Anden knude forbundet til e
     Indsæt relation: e -> (v1 = v2)
```

Boks 2: Pseudokode for generering af ECM modellen ud fra grafens topologi.

Som det fremgår af ovenstående AMS-kode, er antallet af variable og relationer i modellen lineært i forhold til antallet kanter, $|E|$, og knuder, $|V|$, i grafen. Af pseudokoden i Boks 2 ses det, at køretiden[6] for at opbygge AMS filen vil være $O(V+E)$, bemærk at V

[5] Array Model Source sproget er beskrevet i hjælpen til Array Studio.
[6] Den asymptotiske notation, der anvendes til angivelse af køretider i denne rapport, er beskrevet i [CORMEN]

og E angiver antallet af knuder og kanter i forbindelse med køretiden. Det kan ikke forventes, at køretiden for at kompilere arraymodellen og efterfølgende lave opslag i denne også opfører sig lineært, da størrelsen af det samlede løsningsrum udvikler sig ekspotentielt.

For at anvende ECM modellen påtrykkes fejlrapporten modellen. Dette gøres ved at oversætte fejlrapporten til en statevektor, der direkte kan bruges til et opslag i ECM modellen. Som beskrevet i afsnittet Formalisering af problemet, er en fejlrapport defineret som en delmængde af alle grafens knuder, nemlig de knuder hvorpå der er rapporteret fejl. I eksemplet på Figur 13 på næste side, anvendes fejlrapporten $F = (b, c)$. Oversættelsen til statevektor ser således ud:

$$F = \{b, c\} \quad \Rightarrow \quad \begin{pmatrix} ps \\ a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{16}$$

Bemærk, at kun de variable, der repræsenterer en knude i grafen, påtrykkes en værdi, da man netop ønsker, at ECM modellen skal fastlægge værdierne af alle de variable, der repræsenterer kanter i grafen. Når værdien af alle kanter slås op i ECM fås følgende:

$$\begin{pmatrix} psa \\ ab \\ ac \\ bc \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \vee \quad \begin{pmatrix} psa \\ ab \\ ac \\ bc \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{17}$$

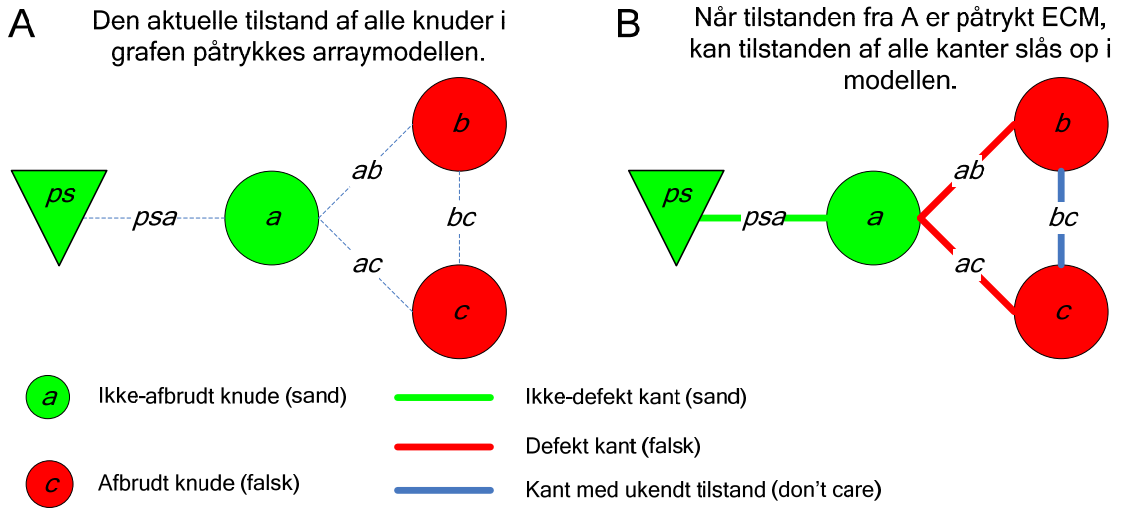
Da ECM modellen i dette eksempel er 8-dimensionel, fire knuder + fire kanter, og opslaget laves med en statevektor der fikserer fire af de otte dimensioner, er der altså fire frie dimensioner tilbage. Svaret i (17) er altså i realiteten et array i fire dimensioner, men for overskuelighedens skyld, vælger vi at repræsentere det som to 4-dimensionale vektorer, nemlig de to lovlige tilstande der eksisterer.

Som det fremgår af AMS koden i Boks 1, afhænger værdien af de variable der repræsenterer kanterne i grafen kun af de variable der repræsenterer knuderne. Der er altså ingen indbyrdes relationer mellem de enkelte kanter. Konsekvensen af denne egenskab er, at når alle knude-variable i grafen fikseres gennem (17), vil alle kant-variable antage en af værdierne {0,1,don't care} uafhængigt af værdien af alle andre kanter. Derved kan de to lovlige kombinationer af kant-variablernes værdi reduceres fra to vektorer (17) til én vektor (18):

$$\begin{pmatrix} psa \\ ab \\ ac \\ bc \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ don't\ care \end{pmatrix} \tag{18}$$

Reduktionen fra (17) til (18) er mulig for alle ECM modeller. Generelt gælder, at ECM modellens størrelse er defineret ved antallet af knude-variable, $|V|$, og antallet af kant-variable, $|E|$. Når alle knudevariable fikseres gennem en statevektor med længden $|V|$, vil alle lovlige kombinationer af kanterne kunne repræsenteres med en vektor med længden $|E|$, hvor hver kant tillægges en værdi i udfaldsrummet {0,1,don't care}.

På Figur 13 A illustreres, hvordan fejlrapporten og statevektoren fra (16) ser ud på grafen. Figur 13 B viser resultatet af opslaget i arraymodellen (18), hvor hver kant tillægges en værdi.



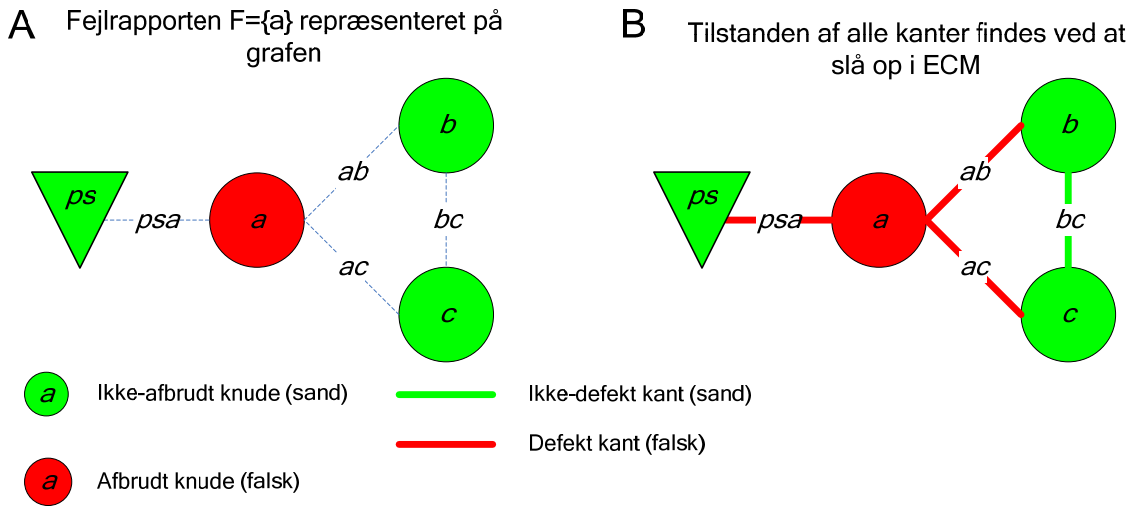
Figur 13: Eksempel på anvendelse af ECM modellen

Helt i overensstemmelse med hvad man kunne forvente, bliver svaret i ovenstående eksempel, at kanterne *ab* og *ac* er defekte, hvilket også virker som en meget sandsynlig forklaring på, hvorfor *b* og *c* er afbrudte.

Med notationen fra afsnittet Formalisering af problemet kan man udtrykke funktionen af ECM modellen således: Når ECM modellen for en simpel graf *G* påtrykkes en fejlrapport *F* findes den umiddelbare løsning S_U.

9.1.2 ECM modellens begrænsninger

I ovenstående eksempel gav ECM modellen det korrekte svar på, hvilke defekte kanter der bedst forklarede den observerede situation. Det viser sig dog, at der er et grundlæggende problem med ECM modellen: Den kan ikke beregne alle konsekvenserne af en defekt forbindelse, og kan derfor foreslå forklaringer, der ikke er lovlige. Herunder, på Figur 14, illustreres et eksempel, hvor ECM modellen giver en ulovlig forklaring som svar.



Figur 14: Eksempel på fejlagtigt svar fra ECM modellen

På Figur 14 B laves et opslag i ECM modellen med statevektoren (*ps, a, b, c*) = (1, 0, 1, 1), hvorved alle kanter i grafen tildeles en værdi. Opslaget i modellen giver svaret:

$$\begin{pmatrix} psa \\ ab \\ ac \\ bc \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{19}$$

Igen er svaret i (19) teknisk set et 4-dimensionelt array, men grundet ECM modellens specielle struktur kan de lovlige kombinationer repræsenteres i en vektor med fire elementer.

Problemet med resultatet i dette eksempel er, at ECM modellen ikke beregner, at konsekvensen af forklaringen $S = \{(ps,a),(a,b),(a,c)\}$ er, at b og c også er afbrudt. Derfor foreslås en forklaring, hvor b og c ikke er afbrudt, hvilket er umuligt.

9.1.3 Power Propagation Model

Som ovenstående eksempel illustrerer, er det grundlæggende problem med ECM modellen, at den ikke kan beregne alle konsekvenserne af en forklaring, $K(S)$. For at beregne konsekvensen af, at en eller flere kanter i grafen er defekte, konstrueres endnu en arraymodel, nemlig den model vi herefter vil betegne som Power Propagation Model (PPM). Denne model simulerer strømmens udbredelse i elforsyningsnettet, således at man ud fra information om defekte forbindelser, kan finde alle afbrudte knuder.

Det viser sig, at en arraymodel der kan finde afbrudte knuder bliver meget omfattende at generere. Hvorvidt en knude, *v*, er afbrudt eller ej, afhænger af alle stier fra *v* tilbage til en strømkilde, jf. Definition 9 i afsnittet Formalisering af problemet. Hvis der blot findes én sti, der ikke går via en defekt forbindelse, er knuden ikke afbrudt. Antallet af stier mellem to tilfældige knuder i en vilkårlig graf vokser eksponentielt i forhold til antallet af kanter, hvorfor antallet af relationer i arraymodellen også vil blive eksponentielt.

Man kan derfor med fordel betragte det inverse problem, nemlig at finde alle knuder som **ikke** er afbrudt. Om en knude ikke er afbrudt afhænger kun af knudens umiddelbare naboer, $N_1(v)$: Hvis der findes mindst en nabo, som ikke er afbrudt, og kanten ud til denne nabo ikke er defekt, er den aktuelle knude ikke afbrudt. Der findes mange eksempler på umiddelbart nærliggende problemer, der i beregningsmæssig kompleksitet er vidt forskellige. Eksempelvis er det meget svært at finde længste vej mellem to knuder i en graf, mens det er simpelt at finde den korteste vej.

Med dette ræsonnement som grundlag, kan der opbygges en model, der kan finde alle ikke afbrudte knuder, og dermed implicit finde alle afbrudte knuder. AMS-koden til PPM modellen for eksempelgrafen fra Figur 12 er vist i Boks 3 herunder:

```
Variables

     Node :
     {
          Ps : Boolean;
          A : Boolean;
          B : Boolean;
          C : Boolean;
     };

     Edge :
     {
          A_Ps : Boolean;
          A_C : Boolean;
          A_B : Boolean;
          B_C : Boolean;
     };

Relations
     Node.Ps = 1;
     Node.A = (Node.C and Edge.A_C) or
              (Node.B and Edge.A_B) or
              (Node.Ps and Edge.A_Ps);
     Node.B = (Node.C and Edge.B_C) or
              (Node.A and Edge.A_B);
     Node.C = (Node.B and Edge.B_C) or
              (Node.A and Edge.A_C);
```

Boks 3: AMS kode for PPM modellen.

Ligesom i ECM modellen repræsenteres grafen som en boolean variabel for hver knude og kant. Fortolkningen af variablene er også den samme som i ECM modellen, dog indføres Definition 5 og formel (7) fra Formalisering af problemet (en strømkilde kan ikke afbrydes) som en specifik relation for alle strømkilder.

PPM modellen kan også generes automatisk ud fra grafens topologi, som det er illustreret med pseudokoden i Boks 4.

```
Generer en boolean variabel for hver knude i grafen.
Generer en boolean variabel for hver kant i grafen.

For (v = Alle knuder i grafen)
     Hvis v er strømkilde
          Indsæt relation: v = 1   // Strømkilder virker
                                   // altid.
     Ellers
          Indsæt relation: v = 1 hvis
               mindst en nabo virker og
               forbindelsen til denne nabo virker.
```

Boks 4: Pseudokode til at generere PPM modellen for en graf.

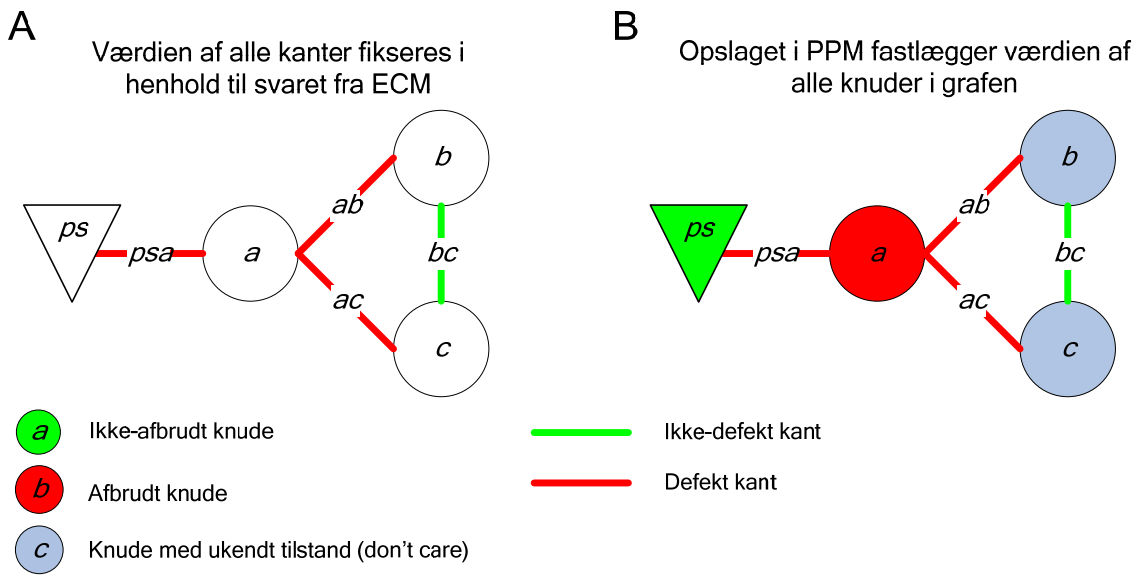
Antallet af variable og relationer i modellen bliver, ligesom for ECM, lineært i forhold til antallet af kanter og knuder i grafen. Køretiden for generering af PPM modellen er $O(V+E)$. Køretiden for opbygning af modellen kan ikke tages som udtryk for køretiden af kompilering og opslag i modellen. I afsnittet Benchmarking vil tidsforbruget ved kompilering og opslag i modellerne blive undersøgt nærmere.

9.1.4 Eksempel på anvendelse af PPM modellen

Anvendelsen af PPM modellen er i en hvis forstand modsat af anvendelsen af ECM. I PPM konstrueres en statevektor, der fikserer værdien af alle kanter, hvorefter et opslag i modellen med denne statevektor giver værdien af alle knuder. Faktisk kan resultatet af opslaget i ECM modellen, (19), direkte bruges som statevektor til at slå op i PPM. Dog skal det lige bemærkes, at hvis svaret fra ECM indeholder kanter med værdien "don't care", se f.eks. vektoren (18), skal disse kanter rettes til værdien 1 (sand), inden vektoren bruges til opslag i PPM. Når vektoren (19) anvendes til opslag i PPM fås følgende resultat:

$$\begin{pmatrix} ps \\ a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ don't\ care \\ don't\ care \end{pmatrix} \qquad (20)$$

Figur 15 A herunder illustrer, hvordan statevektoren fra (19) ser ud på grafen, og Figur 15 B viser PPM svaret (20) indtegnet på grafen.



Figur 15: Eksempel på anvendelse af PPM modellen. Resultatet fra eksemplet i Figur 14 anvendes.

Tolkningen af svaret i (20) er, at det kun er knuden *ps* der har strøm, når kanterne *psa*, *ab* og *ac* er afbrudt. Dermed vides det implicit, at knuderne *a*, *b* og *c* alle er afbrudt, selvom modellen kun markerer *a* som direkte afbrudt. De to don't care knuder, *b* og *c* skal også betragtes som afbrudte.

Det bør pointeres, at mens don't care variable i ECM modellen fortolkes som ikke defekte kanter, fortolkes don't care variable i PPM modellen som afbrudte knuder.

Efter at PPM har været anvendt, bringes ECM endnu engang i spil. Svaret fra PPM modificeres, så alle don't care knuder betragtes som afbrudte, dvs. vektor (20) bliver til:

$$\begin{pmatrix} ps \\ a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad (21)$$

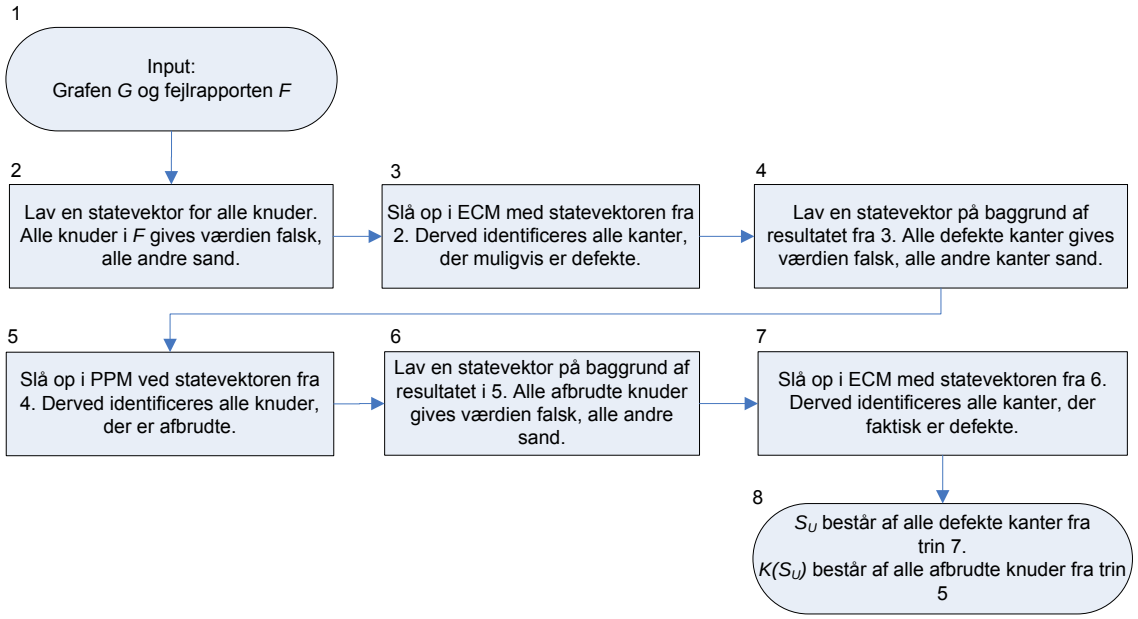
hvilket svarer til fejlrapporten $F=\{a, b, c\}$. Nu anvendes (21) til opslag i ECM modellen, hvilket giver svaret:

$$\begin{pmatrix} psa \\ ab \\ ac \\ bc \end{pmatrix} = \begin{pmatrix} 0 \\ don't\ care \\ don't\ care \\ don't\ care \end{pmatrix} \qquad (22)$$

Forklaringen af den oprindelige fejlrapport, $F = \{a\}$, kan nu udledes fra (21) og (22):

- $S_U = \{(ps, a)\}$ følger af (22)
- $K(S_U) = \{a, b, c\}$ følger af (21)

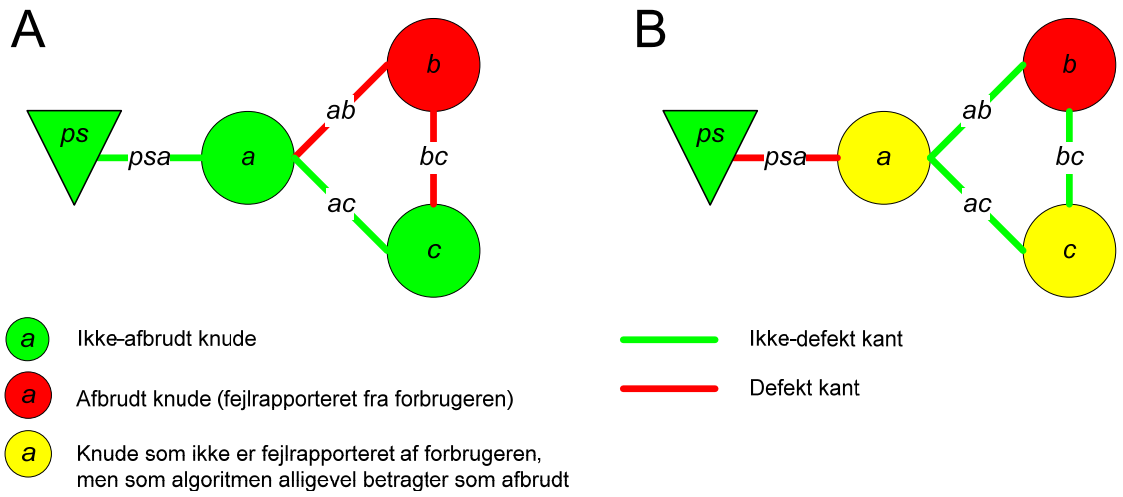
Ved brug af både ECM og PPM er det altså muligt at beregne både S_U og $K(S_U)$ for en given fejlrapport *F*. Flowdiagrammet i Figur 16 viser, hvordan ECM og PPM modellerne spiller sammen, for at finde frem til S_U.



Figur 16: Flowdiagram over, hvordan de to arraymodeller bruges til at finde frem til S_U og $K(S_U)$.

9.1.5 Alternative forklaringer

Ifølge problemdefinitionen er S_U ikke altid den mest sandsynlige forklaring på en fejlrapport. Derfor skal algoritmen være i stand til at finde frem til alternative forklaringer, hvor et antal ikke fejlmeldte knuder medtages, for til gengæld at forsøge at nedbringe antallet af defekte kanter. Figuren herunder illustrerer, hvordan den samme fejlrapport, $F = \{b\}$, kan forklares på to forskellige måder.



Figur 17: To mulige forklaringer på samme fejlrapport.

På Figur 17 gives to forskellige forklaringer på fejlrapporten F={b}.

- $S_A = \{(a,b),(b,c)\}$ $K(S_A) = \{b\}$
- $S_B = \{(ps,a)\}$ $K(S_B) = \{a,b,c\}$

Hvilken af disse forklaringer der betragtes som mest sandsynlig afhænger af objektfunktionen, se afsnittet Formalisering af problemet.

9.1.6 Derived Relation

For at ATOMS kan udlede alternative forklaringer på en effektiv måde, anvendes funktionen *derived relation* i array runtime miljøet. I dette afsnit gives en generel introduktion til derived relation og i det efterfølgende afsnit beskrives hvordan funktionen konkret benyttes til at udlede alternative forklaringer.

For at beregne en derived relation kræves tre inputs:

1. En arraymodel med et antal variable og relationer.
2. En statevektor, der fikserer værdien af et vilkårligt antal variable i modellen. Bemærk at denne statevektor kan udelades, hvis ingen variable i modellen skal fikseres på forhånd.
3. En liste over en eller flere af modellens variable, for hvilke man ønsker at beregne derived relation.

Givet de tre ovennævnte inputs, vil array runtime miljøet generere en tabel indeholdende alle lovlige kombinationer af de variable der er valgt i input 3, under hensyntagen til både modellens begrænsninger og den påtrykte statevektor.

Eksempel 1
En arraymodel med tre boolean variable betragtes, Boks 5:

```
Variables
  A : boolean;
  B : boolean;
  C : boolean;

Relations
  A->B;
  B->C;
```

Boks 5: Arraymodel til illustration af derived relation.

Hvis der i ovenstående model beregnes derived relation af B og C, uden der først påtrykkes en statevektor, fås resultatet:

B	C
0	0
1	1
0	1

Resultatet af derived relation beregningen indeholder kun de variable, der indgår i beregningen (B, C) og kun de lovlige kombinationer af værditildeling til disse. I dette eksempel udelukkes værdisættet B=1 og C=0 da dette ikke opfylder relationen B->C.

Eksempel 2
Hvis man før beregning af derived relation fikserer variablen A til værdien 1 fås følgende resultat:

B	C
1	1

Når statevektoren fikserer A=1 resterer der kun én lovlig tildeling af værdier til variablerne B og C, nemlig B=1 og C=1, som følge af modellens to relationer.

Beregningen af en derived relation foregår i praksis, ved at beregne det kartesiske produkt af domænerne for alle de variable, der indgår beregningen. I de to ovenstående eksempler drejer det sig om variablerne B og C, der begge har domænet {0,1}, hvorfor det kartesiske produkt bliver:

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\} \qquad (23)$$

Når dette produkt er beregnet, vil array runtime miljøet for hvert element evaluere, om det overholder modellens relationer eller ej. Mængden af alle de lovlige elementer i det kartesiske produkt er resultatet af derived relation beregningen. Antallet af elementer i et

kartesisk produkt er produktet af størrelserne af de mængder produktet beregnes af, hvoraf det ses, at køretiden for at beregne en derived relation vokser eksponentielt i forhold til antallet af variable der indgår i relationen.

9.1.7 Bestemmelse af alternative forklaringer

For at bestemme alternative forklaringer, inddrages en tredje arraymodel – "Combined Model" (CM). CM modellerer både kanter og knuder i grafen, og kan derfor betragtes som en kombination af ECM og PPM. Herunder, Boks 6, ses AMS koden til CM modellen:

```
Variables

     Node :
     {
          Ps : Boolean;
          A : Boolean;
          B : Boolean;
          C : Boolean;
     };

     Edge :
     {
          A_Ps : Boolean;
          A_C : Boolean;
          A_B : Boolean;
          B_C : Boolean;
     };

Relations
     Edge.A_Ps = (Node.A=Node.Ps);
     Edge.A_C = (Node.A=Node.C);
     Edge.A_B = (Node.A=Node.B);
     Edge.B_C = (Node.B=Node.C);
     Node.Ps;
     Node.A = (Node.C and Edge.A_C) or
              (Node.B and Edge.A_B) or
              (Node.Ps and Edge.A_Ps);
     Node.B = (Node.C and Edge.B_C) or
              (Node.A and Edge.A_B);
     Node.C = (Node.B and Edge.B_C) or
              (Node.A and Edge.A_C);
```

Boks 6: AMS kode for CM modellen.

Pseudokoden til generering af CM modellen vil ikke blive gentaget her, da den fremkommer direkte, ved at kombinere koden for ECM og PPM modellerne.

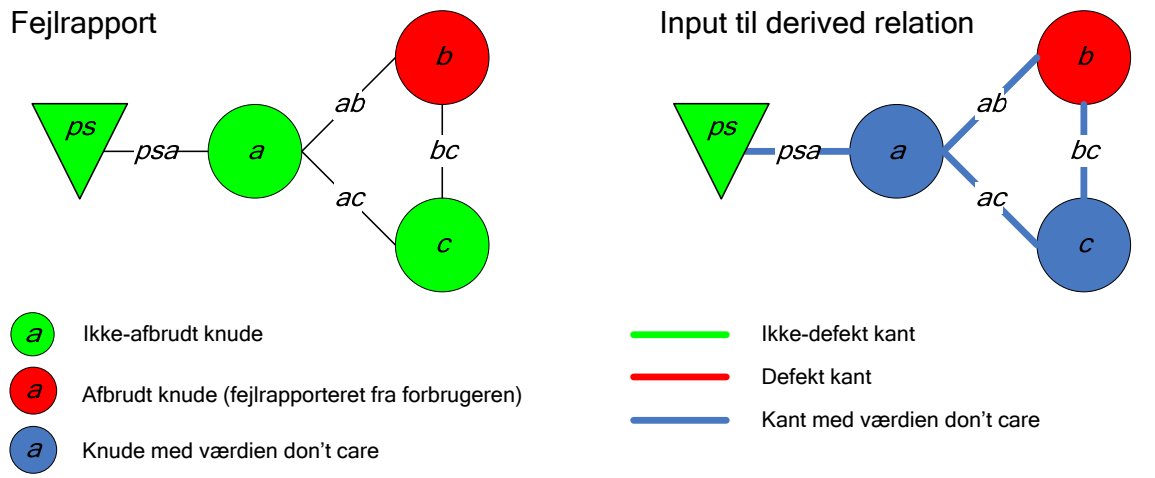
Ved beregning af en derived relation, se foregående afsnit Derived Relation, på CM, kan man direkte beregne Q_n ud fra grafen G og fejlrapporten F ved hjælp af algoritmen i Boks 7.

Grunden til at CM modellen ikke erstatter ECM og PPM modellerne er, at man ved brug af CM modellen ikke er i stand til at placere den fundne forklaring tæt ind til de fejlrapporterede knuder, hvorfor S_U ikke kan bestemmes.

```
Find alle knuder, der ligger højest n kanter fra en knude
i F. Betegn disse knuder som X.

Find alle kanter, der er forbundet til en knude i X.
Betegn disse kanter Y.

Generer en statevektor, hvor:
     Alle knuder i F sættes til falsk.
     Alle øvrige knuder, der ikke ligger i X, sættes til
     sand.
     Alle kanter der ikke ligger i Y sættes til sand.

Beregn en derived relation, der medtager netop de
variable der repræsenterer knuderne i X og kanterne i Y
og statevektoren påtrykkes inden beregningen.

Derived relation tabellen indeholder nu alle forklaringer
i Qn.
```

Boks 7: Pseudokode for input til derived relation.

Her gives et eksempel på beregning af derived relation. Igen betragtes Figur 12, med fejlrapporten $F = \{b\}$. Fejlrapporten, samt den statevektor der påtrykkes arraymodellen, når derived relation beregnes, er illustreret på Figur 18:



Figur 18: Input til derived relation for beregning af Q_1.

De variable, der medtages i beregningen af derived relation er knuderne *a* og *c* samt alle kanter i grafen, der er forbundet til enten *a* eller *c* - i dette minieksempel vil det sige samtlige kanter i grafen. Resultatet af derived relation beregningen er vist i Tabel 3.

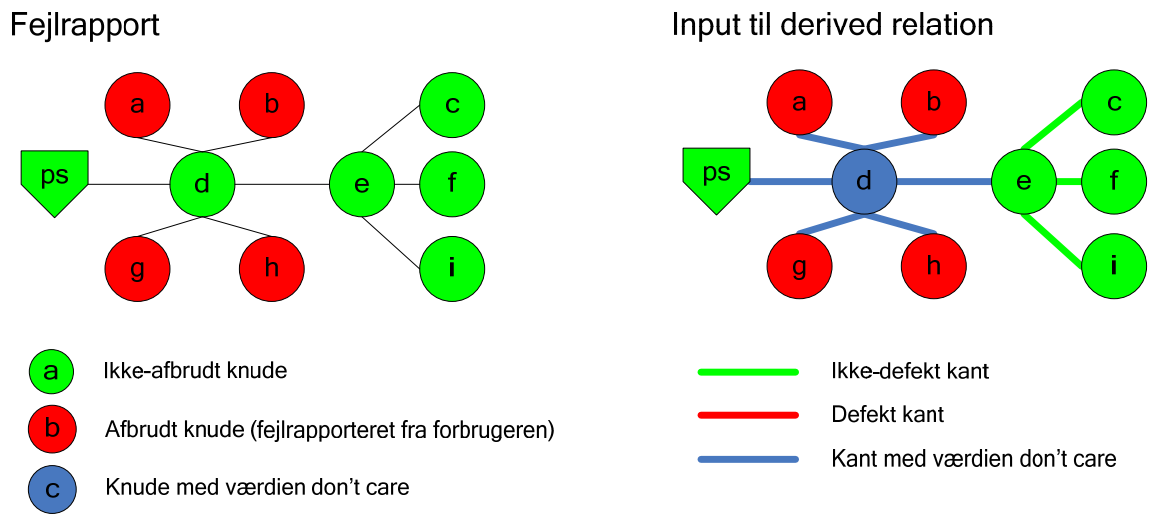
a	***c***	***psa***	***ab***	**ac**	**bc**
Ikke afbrudt	Ikke afbrudt	Ikke defekt	Defekt	Ikke defekt	Defekt
Afbrudt	Afbrudt	Defekt	Ikke defekt	Ikke defekt	Ikke defekt
Ikke afbrudt	Afbrudt	Ikke defekt	Defekt	Defekt	Ikke defekt

Tabel 3: Resultat af derived relation beregning.

Som det ses stemmer denne løsning overens med definitionen af Q_1 fra Definition 10.

CM modellen har det samme problem som ECM modellen, nemlig at den ikke kan beregne alle konsekvenser af en forklaring *S*. I eksemplet fra Figur 18 giver derived relation beregningen præcis de forklaringer der udgør hele Q_1. Hvis grafen bliver mere kompliceret, kan derived relation beregningen i nogle tilfælde medtage løsninger, der ikke er gyldige. **Dog er man sikker på, at resultatet af beregningen af derived relation i *n* niveauer, altid vil indeholde alle de forklaringer der udgør Q_n.** For at komme fra resultatet af derived relation beregningen til Q_n, er det altså kun et spørgsmål om at fjerne eventuelle ikke-gyldige forklaringer.

På Figur 19 ses et eksempel, hvor derived relation beregningen giver både gyldige og ikke-gyldige forslag til forklaring af en fejlrapport.



Figur 19: Illustration af fejlrapporten *F={a, b, g, h}* og den derived relation beregning der bruges til at beregne Q_1.

Resultatet af den derived relation der er illustreret i Figur 19 bliver:

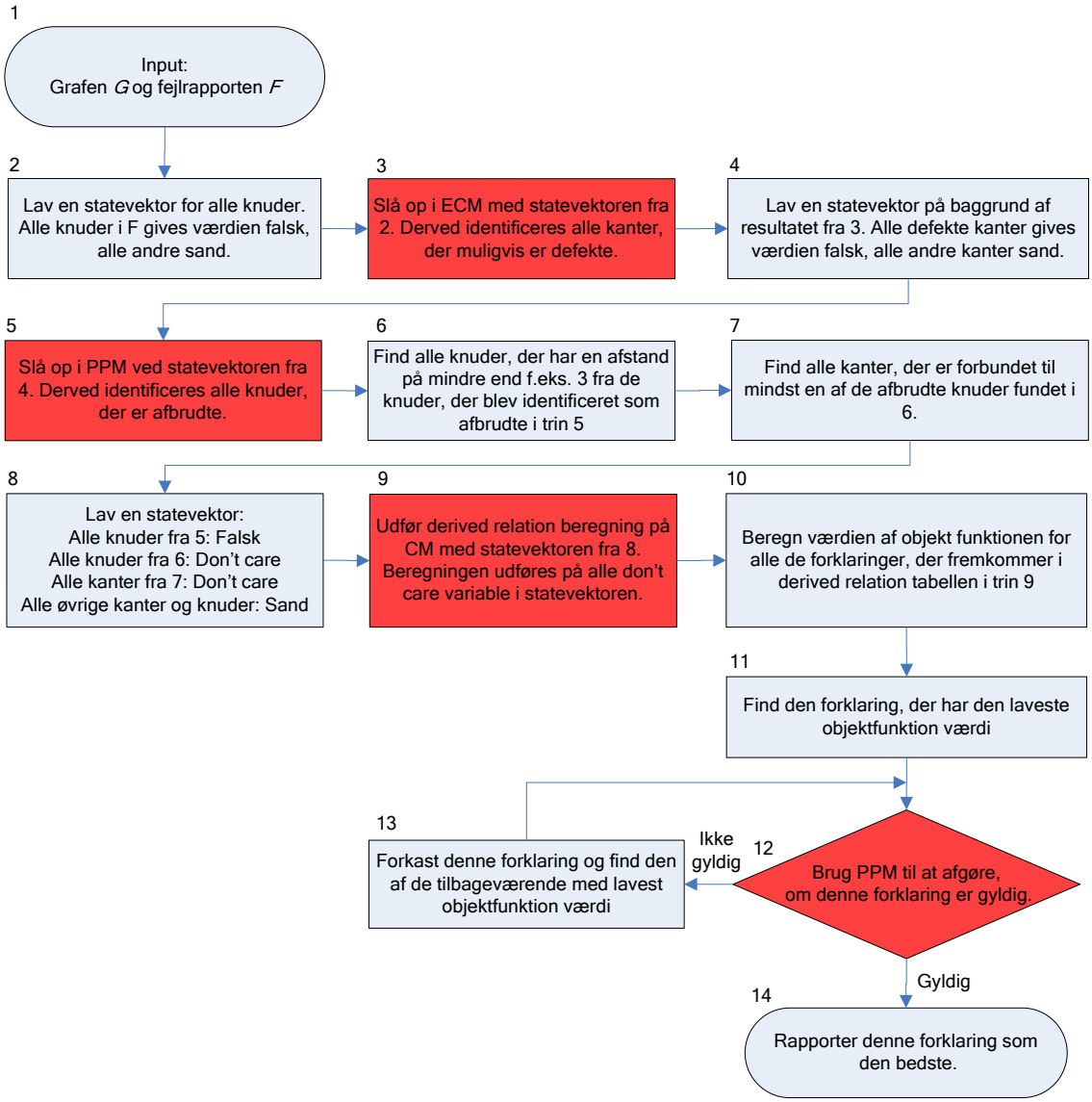
d	**psd**	**ad**	**bd**	**dg**	**dh**	**de**
Afbrudt	Defekt	Ikke defekt	Ikke defekt	Ikke defekt	Ikke defekt	Defekt
Ikke afbrudt	Ikke defekt	Defekt	Defekt	Defekt	Defekt	Ikke defekt

Tabel 4: Resultat af derived relation.

Af resultaterne i Tabel 4 herover ses det, at arraymodellen ikke beregner alle konsekvenserne af at afbryde kunden *d*. Derived relation beregningen kommer frem til, at den bedste forklaring på fejlrapporten er $S = \{(ps, d), (d, e)\}$ hvilket skyldes, at der i beregningen ikke tages højde for, at man ved denne forklaring også afbryder knuderne *e, c, f* og *i*. Når denne konsekvens tages med ind i billedet viser det sig, at den umiddelbare løsning, nederst i Tabel 4, er den mest hensigtsmæssige i dette tilfælde. Det er derfor nødvendigt at kontrollere hver forklaring i derived relation resultatet for, om der skal medtages yderligere konsekvenser af beregningen end dem, der findes direkte. Konsekvensberegning foretages vha. PPM modellen på præcis samme måde, som når konsekvenserne af resultatet fra ECM modellen skal beregnes, se afsnittet Power Propagation Model.

9.1.8 Flowdiagram af hele ATOMS algoritmen

Flowdiagrammet på Figur 20 illustrerer hele forløbet af den arraybaserede algoritme, ATOMS. Forløbet er opdelt i 14 trin, hvor de røde trin indebærer opslag i en arraymodel. Bemærk specielt trin 9, hvor der udføres en derived relation beregning. I dette trin vil der blive foretaget mange opslag i CM modellen.



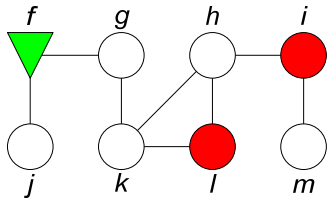
Figur 20: Flowdiagram over hele ATOMS algoritmen.

Ved brug af den fremgangsmåde der er blevet beskrevet her, er det muligt at løse fejlfindingsproblemet ved hjælp af tre arraymodeller. Ydeevnen af denne algoritme vil blive analyseret i afsnittet Benchmarking, hvor algoritmens egenskaber også vil blive sammenlignet med andre måder at løse fejlfindingsproblemet på.

9.2 BOMS

I dette afsnit beskrives en grafalgoritme, der løser fejlfindingsproblemet. Grafalgoritmen benytter sig af en let modificeret udgave af den velkendte Breadth First Search (BFS) algoritme.

Selve grafalgoritmen kan inddeles i tre trin. I dette afsnit vil vi gennemgå algoritmen trin for trin med udgangspunkt i et konkret eksempel. Udgangssituationen kan ses på Figur 21. Knude *f* er angivet som strømkilde mens knuderne *i* og *l* er fejlmeldt af forbrugerne, $F = \{i, l\}$.

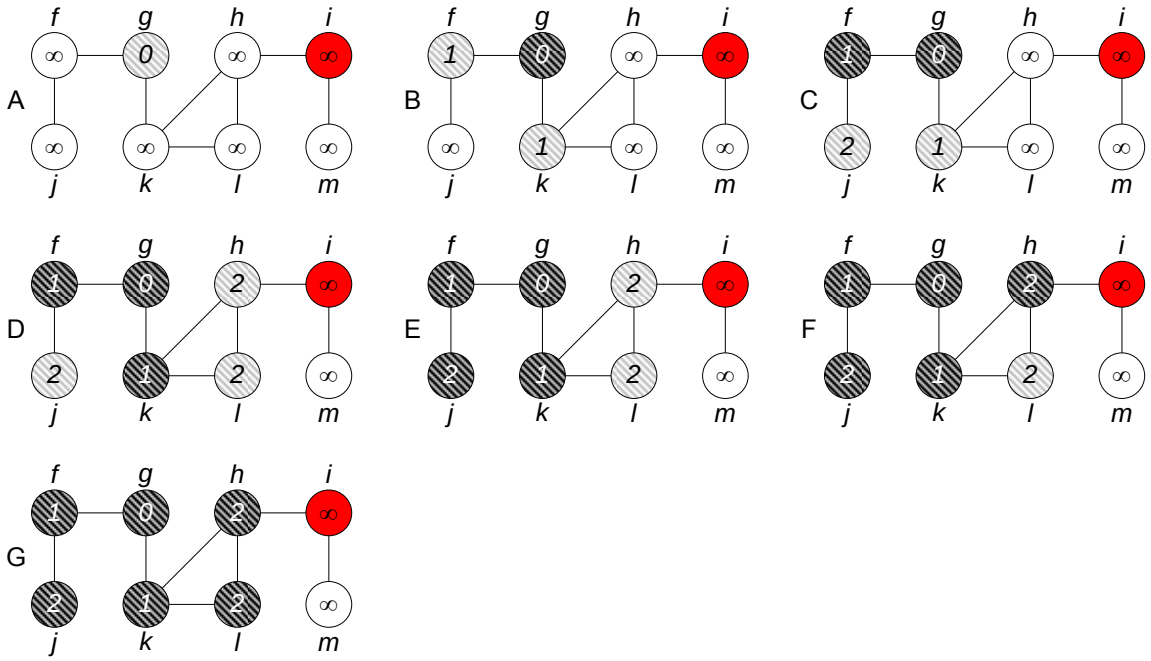


Figur 21: Udgangssituationen for grafalgoritmen.

9.2.1 Breadth First Search

I BOMS benyttes BFS op til flere gange, derfor beskrives denne algoritme kort i dette afsnit. BFS er en algoritme til at finde den korteste afstand til samtlige knuder i en graf. I forhold til den i [CORMEN] beskrevne BFS algoritme, adskiller denne implementering sig på to punkter.

- Antallet af kilder er udvidet fra 1 til et vilkårligt antal.
- Det er muligt at angive et antal fejlmeldte knuder. Søgningen kan ikke forsætte forbi disse knuder.



Figur 22: Virkemåden af BFS. Uudforskede knuder er farvet hvide. Udforskede knuder er farvet grå, mens knuder for hvilke alle umiddelbare naboer er udforsket er farvet sorte. Fejlmeldte knuder er farvet røde.

Figur 22 beskriver virkemåden af BFS. I udgangstilstanden farves alle knuder hvide. Hvid betyder at knuden endnu ikke er blevet udforsket. Herefter farves alle knuder, der er angivet som kilder for søgningen grå og deres afstand sættes til nul. Grå betyder at knuden er udforsket. Desuden farves alle knuder der er fejlmeldte røde. Søgningen er nu initialiseret som vist på Figur 22 A. Knude *g* er angivet som kilden og på knude *i* er der rapporteret fejl. I udgangssituationen er afstanden til kilden *g* sat til nul, mens afstanden til de resterende knuder er uendelig.

Her begynder en iterativ proces, der først vælger en af de udforskede knuder; på billede B er knude *g* blevet valgt. Dernæst findes afstanden til samtlige naboer og de farves grå. Naboerne til knude *g* er knuderne *f* og *k*. Når alle naboer er fundet farves den aktive knude sort. Resultatet af første gennemløb kan ses på Figur 22 B. Afstanden til knuden *g*'s naboer findes som afstanden til *g* selv plus én.

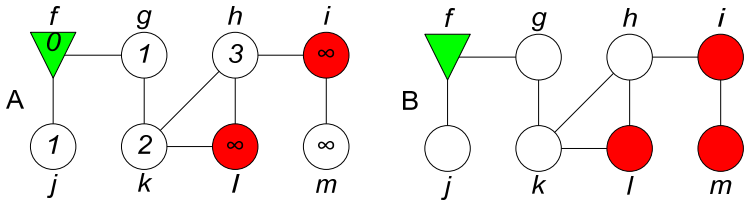
De efterfølgende gennemløb af den iterative proces kan ses på Figur 22 C til G. Læg dog mærke til Figur 22 F, hvor knude *h* vælges, men naboen, knude *i,* bliver ikke farvet grå, da den er fejlmeldt. Afstanden til knude *i* bliver heller ikke fundet, da fejlmeldte knuder per definition ikke kan nås. Læg mærke til, at søgningen bliver blokeret af knude *i* og derfor aldrig når knude *m*.

Algoritmen finder alle førstenaboer før den finder andennaboer, ligeledes finder den alle andennaboer før den finder tredjenaboer. Det er derfor den hedder Breadth First Search. Denne egenskab kan benyttes til at forbedre køretiden i nogle tilfælde - hvis man kun er interesseret i første- og andennaboer, $N_1(V)$ og $N_2(V)$, kan søgningen afbrydes, når disse er fundet.

9.2.2 Bestemmelse af den umiddelbare forklaring

Med udgangspunkt i BFS algoritmen, som beskrevet ovenfor, vil der nu blive skitseret en metode til løsning af fejlfindingsproblemet, der bygger på denne algoritme.

Det første trin er at finde ud af, hvilke knuder der nødvendigvis må være afbrudt, givet de af forbrugerne fejlmeldte knuder, $F = \{i, l\}$. Dette gøres ved at benytte sig af BFS, hvor strømkilderne bruges som udgangspunkt for søgningen. Som nævnt ovenfor markeres de fejlmeldte knuder med en bestemt farve, der blokerer søgningen. Resultatet af søgningen er angivet på Figur 23 A. Alle knuder hvortil afstanden er uendelig, må nu regnes for at være afbrudte og de markeres derfor således, se Figur 23 B hvor knuden *m* nødvendigvis må være afbrudt.

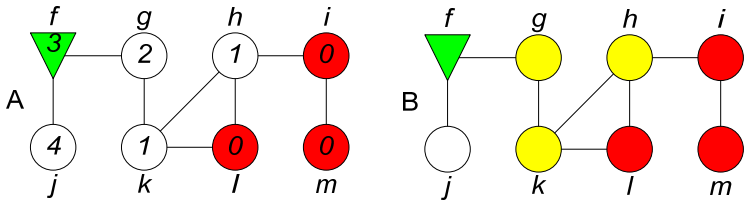


Figur 23: Første trin i BOMS. Alle knuder der nødvendigvis må være afbrudt findes.

Den umiddelbare forklaring, S_U, kan nu bestemmes, ved at finde alle kanter, der forbinder en afbrudt knude med en ikke afbrudt knude. I dette tilfælde bliver $S_U = \{(k,l),(h,l),(h,i)\}$.

9.2.3 Bestemmelse af alternative forklaringer

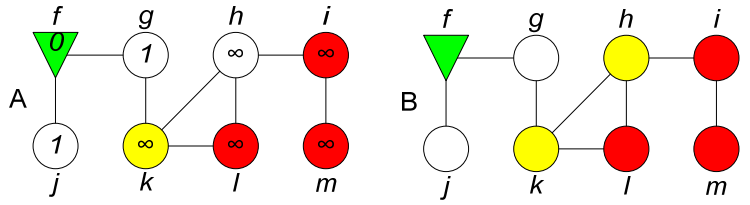
For at bestemme et antal alternative forklaringer betragtes naboerne til de afbrudte knuder. For at finde disse naboer benyttes BFS igen, denne gang med alle afbrudte knuder som udgangspunkt for søgningen. Resultatet af BFS kan ses på Figur 24 A. Der udtages nu et antal niveauer af naboer, i vores eksempel udtages både første- og andennaboer, men det er muligt at ændre, hvor mange niveauer af naboer man vil udtage, alt efter hvor langt væk fra de fejlrapporterede knuder man ønsker at søge efter alternative forklaringer. På Figur 24 B er første- og andennaboer farvet gule.



Figur 24: Andet trin i BOMS. Naboer til fejlmeldte knuder findes.

Det tredje trin i BOMS er at evaluere alle kombinationer af gule knuder med hensyn til objektfunktionen (12), som er beskrevet i afsnittet Formalisering af problemet. Det viser sig at en del af kombinationerne af gule knuder er ugyldige; hvis vi for eksempel vælger at knuderne *g* og *k* er afbrudte, må *h* ifølge grafens struktur nødvendigvis også være afbrudt. For at undgå at evaluere ugyldige forklaringer, benyttes BFS igen efter hvert

valg af gule knuder. På Figur 25 A er den forklaring valgt, hvor knuden *k* og de røde knuder er afbrudt. Efter BFS er evalueret med strømkilden som udgangspunkt ses det, at knuden *h* nødvendigvis også må være afbrudt. Derfor er det den forklaring, hvor både knude *h* og *k* er afbrudt, der bliver evalueret i dette tilfælde, som vist på Figur 25 B.



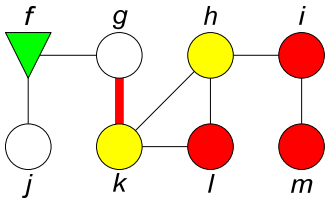
Figur 25: Tredje trin i grafalgoritmen. Alle kombinationer af gule knuder evalueres.

Da dette eksempel er forholdsvis lille, kan vi opskrive samtlige kombinationer af gule knuder, og den dertil hørende værdi af objektfunktionen. Alle de lovlige kombinationer af gule knuder udgør Q_2, da de gule knuder blev fundet som første- og andennaboer til de afbrudte knuder. Objektfunktionen er $f(S) = \alpha \cdot |K(S)| + \beta \cdot |S|$ hvor α og β er variable der kan ændres af brugeren. Dette er gjort i Tabel 5, hvor gule knuder der nødvendigvis også må være afbrudt er angivet i parentes. Den bedste forklaring er den, der minimere værdien af objektfunktionen.

Kombination af gule knuder	**Resultatet af objektfunktionen** $\alpha = 0{,}5$ og $\beta = 0{,}6$	
-	1,8	
g, (h), (k)	2,1	
h	1,7	
k, (h)	1,6	✓
g, h, (k)	2,1	
g, k, (h)	2,1	
h, k	1,6	✓
g, h, k	2,1	

Tabel 5: Vægtning af alternative forklaringer.

Som det fremgår af Tabel 5 afhænger den bedste løsning i høj grad af valget af α og β. Med vores valg af α og β giver objektfunktionen det bedste resultat, 1,6, hvis knuderne *h* og *k* også afbrydes, hvorved det kun er kanten mellem *g* og *k* der er defekt. Man bør desuden bide mærke i, at nogle af forklaringerne evalueres flere gange. Dette skyldes konsekvensberegningerne, for eksempel bliver forklaringen hvor knuderne *g*, *h* og *k* er afbrudt evalueret hele fire gange. Den valgte forklaring er illustreret på Figur 26.

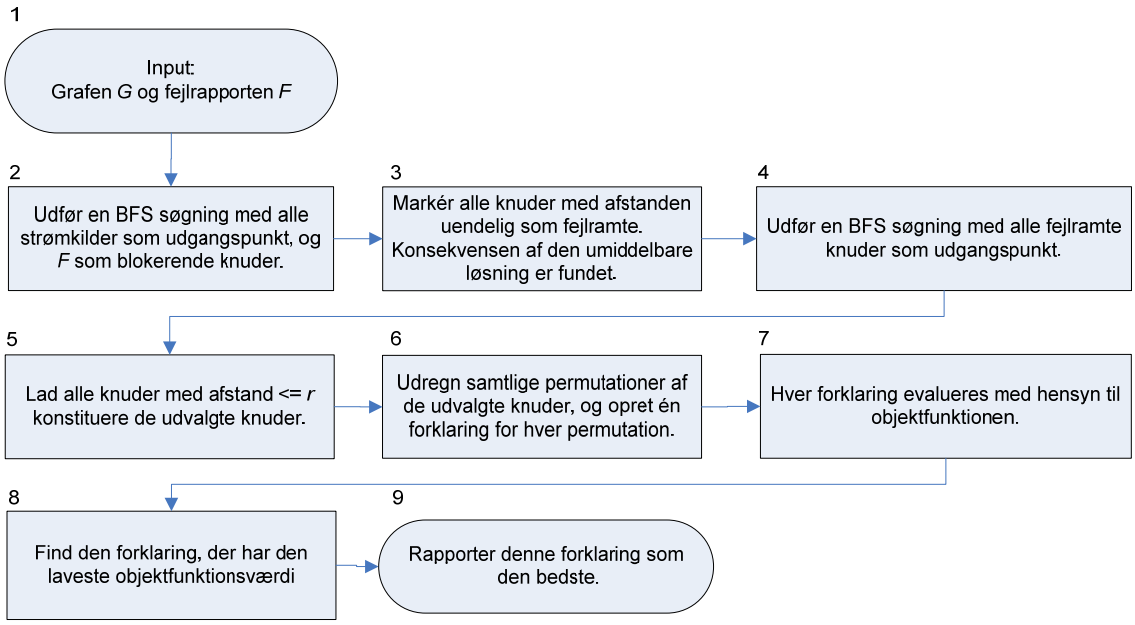


Figur 26: Den valgte forklaring hvor knuderne *h* og *k* er afbrudt og kanten mellem *g* og *k* er defekt.

Antallet af valgte forklaringer der skal evalueres, for at bestemme de alternative forklaringer, Q_n , stiger eksponentielt i forhold til antallet af naboer der udtages. I eksemplet herover, hvor Q_2 bestemmes, udgør første- og anden-naboer til de afbrudte knuder i alt tre knuder. Derfor bliver antallet af alternative forklaringer, hvoraf ikke alle er gyldige, $2^3 = 8$ forklaringer. For større grafer med flere afbrudte knuder kan man forudse, at ydeevnen af denne algoritme hurtigt bliver dårlig - f.eks. vil blot 30 naboer resultere i, at $2^{30} = 1.073.741.824$ forklaringer skal evalueres. Algoritmens ydeevne vil blive analyseret nærmere i afsnittet Benchmarking.

9.2.4 Flowdiagram af hele BOMS algoritmen

Flowdiagrammet på Figur 27 viser hele forløbet af grafalgoritmen, forløbet er inddelt i 9 trin. Parametren *r* kan modificeres af brugeren.



Figur 27: Flowdiagram over hele BOMS algoritmen.

9.3 EOMS

I dette afsnit beskrives endnu en algoritme til løsning af fejlfindingsproblemet. Her vil blive taget udgangspunkt i klassiske grafalgoritmer til bestemmelse af maksimum flow og mindste snit i grafer, nærmere bestemt Ford-Fulkersons metode og den konkrete implementering Edmunds-Karp. Fejlfindingsalgoritmen baseret på Edmunds-Karp gives navnet EOMS.

9.3.1 Ford-Fulkerson

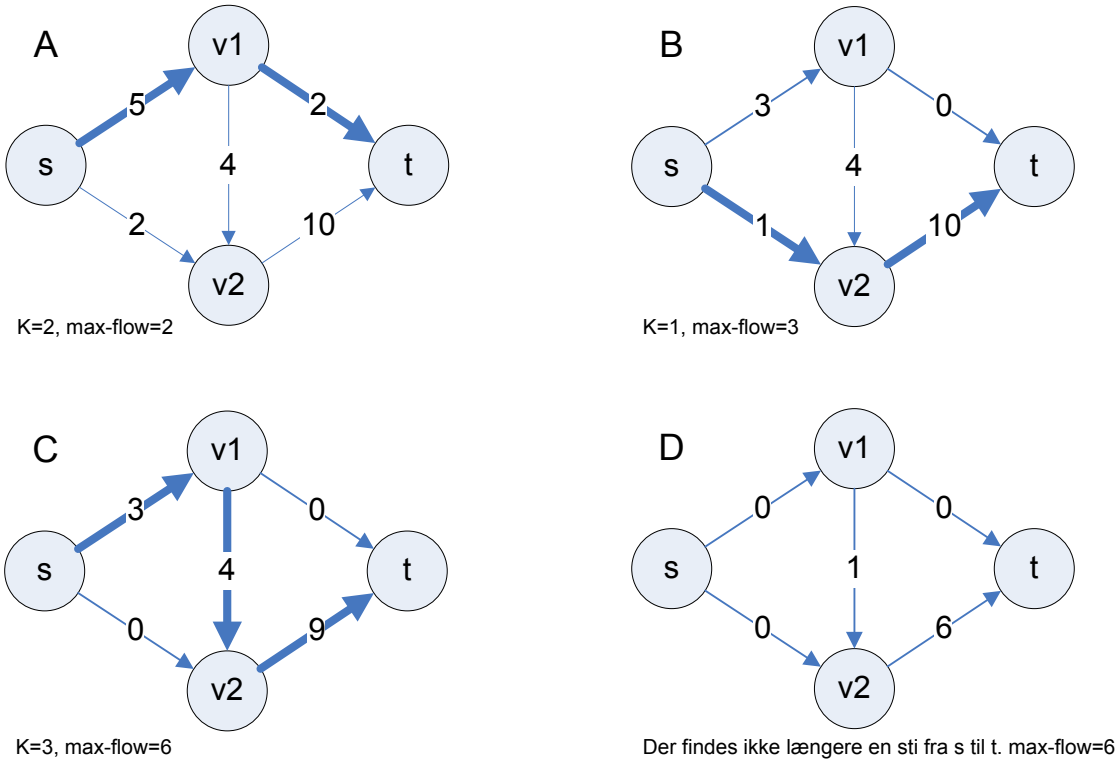
Ford-Fulkerson [VATTER] opererer på en orienteret graf, hvor hver kant har en kapacitet tilknyttet. I grafen vælges én knude som kilde, herefter betegnet *s*, og én knude som dræn, herefter betegnet *t*. Maksimum flow problemet er at bestemme, hvor meget der kan "pumpes" gennem grafens kanter fra kilde til dræn, under hensyntagen til den begrænsede kapacitet i de enkelte kanter. Pseudokoden for Ford-Fulkerson er vist i Boks 8.

```
Ford-Fulkerson(G,s,t)

     flow = 0

     Loop indtil der ikke længere findes en sti fra s til t
          P = en sti fra s til t
          K = den laveste kapacitet af en kant i P
          Træk K fra kapaciteten af alle kanter i P
          flow = flow + K

     returner flow
```

Boks 8: Pseudokode for Ford-Fulkerson. Finder maksimalt flow fra *s* til *t*

I ord kan Ford-Fulkerson algoritmen beskrives således: Find en sti fra *s* til *t* og find "flaskehalsen" på stien, ved at finde den kant, der har den laveste kapacitet. Eliminer denne sti, ved at trække flaskehalsens kapacitet fra hele stien. Derved kommer den/de kanter der udgør flaskehalsen ned på en overskydende kapacitet på 0, hvorved den aktuelle sti reelt er fjernet fra grafen. Denne proces gentages, indtil der ikke længere findes en sti fra *s* til *t*. Summen af alle de kapaciteter, der er trukket ud af grafen, udgør det maksimale flow fra *s* til *t*.

På figuren herunder, Figur 28, er Ford-Fulkerson illustreret på en simpel graf.



Figur 28: Eksempel på Ford-Fulkerson. Det maksimale flow fra *s* til *t* er 6.

9.3.2 Edmunds-Karp

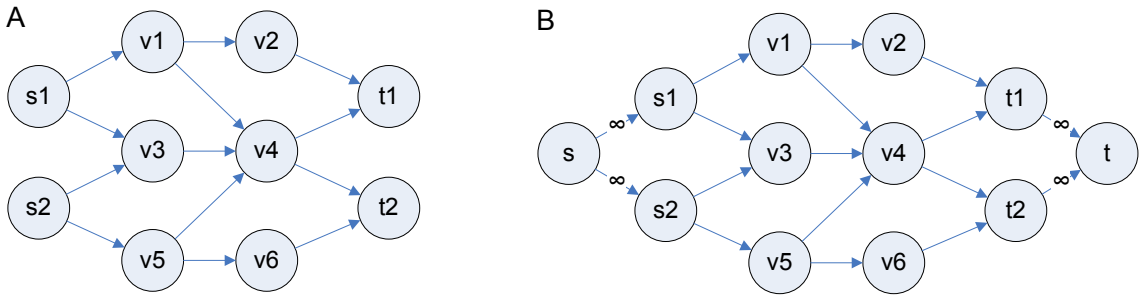
Edmunds-Karp algoritmen [CORMEN] er meget nært beslægtet med Ford-Fulkerson metoden - faktisk er den eneste forskel, at Edmunds-Karp foreskriver præcis, hvordan en sti fra *s* til *t* skal bestemmes, mens Ford-Fulkerson blot betragter vilkårlige stier fra *s* til *t*. Edmunds-Karp kan således betragtes som en speciel implementering af Ford-Fulkerson.

I Edmunds-Karp algoritmen skal stien fra *s* til *t* bestemmes ved brug af Breadth First Search (BFS), se afsnit Breadth First Search, hvorved det garanteres, at den sti fra *s* til *t* der betragtes, er den korteste mellem de to knuder. Det skal bemærkes, at BFS søgningen ikke bruger kanternes kapacitet som afstande, men betragter alle kanter som havende længden 1. Med "den korteste sti" menes altså den sti, der består af færrest kanter.

9.3.3 Flere kilder og dræn

Indtil nu er det kun beskrevet, hvordan Ford-Fulkerson kan bruges til at bestemme det maksimale flow fra én kilde til ét dræn. Det er dog simpelt at modificere algoritmen, så den kan anvendes på problemer med flere kilder og dræn. I stedet for at modificere selve metoden, indføres et trin til pre-processering af grafen, hvorved et problem med flere kilder og dræn kan transformeres til et problem med en enkelt kilde og et enkelt dræn. Ved at tilføje to nye knuder til grafen, en superkilde og et superdræn, og derefter tilføje kanter med ubegrænset kapacitet fra superkilden til alle kilder og fra alle dræn til

superdrænet, kan det maksimale flow findes med en helt almindelig Ford-Fulkerson beregning fra superkilde til superdræn. Metoden er illustreret på Figur 29 herunder.



Figur 29: A: Maksimum flow problem med to kilder og to dræn. B: Modificeret version af problemet fra A, hvor der er tilføjet en superkilde og et superdræn

9.3.4 Mindste snit

Ved et snit i en graf forstås en opdeling af grafens knuder i to disjunkte mængder, *S*, *T* som tilsammen indeholder alle knuder i *V*. Opdelingen laves således, at kilden *s* ligger i *S* mens drænet *t* ligget i *T*.

Definition 12 - Snit i en graf
Et snit (S,T) i en graf $G=(V,E)$ er defineret som en opdeling af grafens knuder i to delmængder, *S*, *T*, hvor $S \subset V$ og $T = V \setminus \mathrm{S}$.

Definition 13 - Kapacitet af et snit
Kapaciteten af et snit, betegnet $f(S,T)$, beregnes som summen af kapaciteterne for de kanter $e \in E$, der forbinder en knude i *S* med en knude i *T*. Hvis der arbejdes med en orienteret graf, skal der vælges en positiv retning hen over snittet, og alle kapaciteter der indgår i $f(S,T)$ skal i dette tilfælde regnes med fortegn.

Ved det mindste snit forstås det snit mellem *s* og *t,* der har den lavest mulige kapacitet. Det følger intuitivt, at det mindste snit udgør "flaskehalsen" mellem *s* og *t*, hvorfor det mindste snit og det maksimale flow må være lige store. Det formelle matematiske bevis for, at dette faktisk gælder, kan ses i [CORMEN] side 657. Problemet med at bestemme det maksimale flow og problemet med at finde det minimale snit er altså meget nært beslægtede.

9.3.5 Køretid

Ford-Fulkerson metoden implementeret i Edmunds-Karp varianten har en køretid på $O(V \cdot E^2)$, hvor *V* er antallet af knuder i grafen og *E* er antallet af kanter. En anden måde at udtrykke køretiden på er $O(E \cdot f)$, hvor *f* er det maksimale flow der bestemmes af algoritmen. Dette udtryk gælder kun, så vidt BFS anvendes til at bestemme stier i grafen og alle kapaciteter er heltal. Argumentet for denne køretid er, at en sti fra *s* til *t* kan findes i tiden $O(E)$ med BFS, hvis initialiseringen af BFS "genbruges" i hver iteration. Da *f* i hver iteration inkrementeres med mindst en, vil antallet af stier der skal beregnes være

mindre end eller lig med f - derfor køretiden $O(E \cdot f)$. Begge måder at udtrykke køretiden på er bevist i [CORMEN].

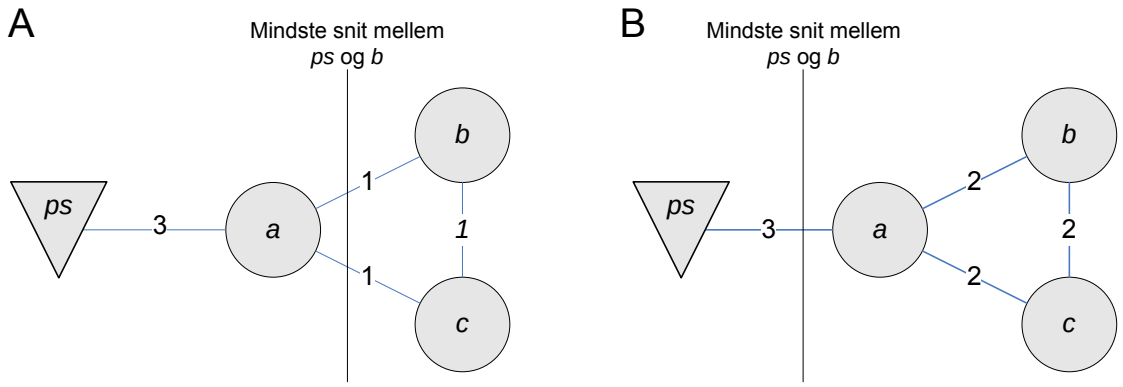
Det bemærkelsesværdige ved køretiden $O(E \cdot f)$ er, at hvis f er forholdsvis lille, vil Ford-Fulkerson køre ekstremt hurtigt.

En matematisk gennemgang af hele flow problemet og andre algoritmer til at løse dette problem kan ses i [CORMEN].

9.3.6 Repræsentation af fejlfindingsproblemet som et mindste snit problem

Det viser sig, at fejlfindingsproblemet kan omsættes til et mindste snit problem, ved at modificere grafrepræsentationen af nettet til et flow netværk og tilføje kapaciteter til alle kanter på en passende måde.

For at kunne løse fejlfindingsproblemet som et mindste snit problem, skal alle kanter i grafen tildeles en kapacitet. På Figur 30 illustreres, hvordan forskelle i kanternes kapacitet kan ændre placeringen af det mindste snit i grafen.



Figur 30: Eksempel på to forskellige tildelinger af kapaciteter til grafens kanter.

I mange af de tilfælde, hvor mindste snit problemer forekommer, følger kapaciteten naturligt af det domæne der modelleres af grafen. Eksempelvis kan mindste snit algoritmer anvendes til at finde "flaskehalse" i et vandforsyningsnet, hvor kapaciteten af de enkelte kanter er gennemstrømningskapaciteten af rørforbindelserne i nettet.

I fejlfindingsproblemet ledes der ikke direkte efter en fysisk flaskehals i nettet, men efter det snit der har den højeste sandsynlighed for fejl. Derfor vil de kapaciteter, der direkte kan bestemmes ud fra de elektrotekniske parametre, som eksempelvis spændingsfald og strømstyrke, ikke bidrage til at løse fejlfindingsproblemet. I stedet for må hver kant i grafen tildeles en kapacitet, der udtrykker sandsynligheden for, at netop denne kant er defekt. Da der som tidligere nævnt findes effektive algoritmer til at bestemme mindste snit, skal sandsynligheden endvidere udtrykkes således, at kanter med stor sandsynlighed for fejl har en lille kapacitet, mens kanter med lille sandsynlighed for fejl har større

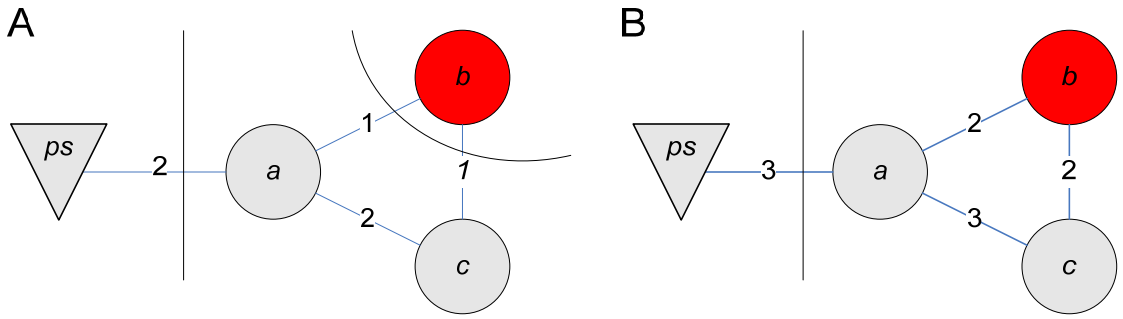
kapacitet. På den måde vil mindste snit algoritmerne foretrække kanter med stor sandsynlighed for fejl.

Ifølge afsnittet Formalisering af problemet er en forklaring på en fejlrapport defineret som en mængde af kanter, der skal fjernes fra grafen, så der ikke længere findes en sti fra en strømkilde til en fejlmeldt knude, Definition 8. En forklaring kan findes ved at bestemme det mindste snit, (S,T), mellem alle strømkilder og de fejlramte knuder, jf. ovenstående afsnit om flere kilder og dræn. De kanter der forbinder en knude i S med en knude i T udgør tilsammen en forklaring på fejlrapporten.

Som beskrevet i afsnittet Formalisering af problemet søges der efter en forklaring, som minimerer en vægtet sum af antallet af defekte kanter og afbrudte knuder. Da en god forklaring sjældent medtager mange ikke fejlmeldte knuder, er det vigtigt, at det mindste snit placeres tæt op af da fejlmeldte knuder. Derfor bliver afstanden til den nærmeste fejlmeldte knude meget afgørende for kanternes kapacitet. Kapaciteter til kanterne kan f.eks. beregnes som:

kapacitet = startkapacitet + afstand til nærmeste fejlrapporterede knude

På Figur 31 herunder er denne formel brugt, til at tildele kapaciteter i eksempelgrafen. Knuden *b* er fejlmeldt.



Figur 31: Afstandsafhængige kapaciteter. A: Startkapacitet = 0 B: Startkapacitet = 1.

Bemærk, at det mindste snit i en graf ikke nødvendigvis er entydigt. På Figur 31 A findes der to snit mellem *ps* og *b,* som begge har den mindst mulige kapacitet på 2. EOMS algoritmen er designet, så den i dette tilfælde vælger det snit, der ligger nærmest den fejlmeldte knude, altså i eksemplet det snit der ligger over kanterne *ab* og *ac*, se afsnittet Modifikation af Edmunds-Karp algoritmen.

På Figur 31 anvendes en konstant startkapacitet, uden hensyntagen til eksterne parametre. Man kan meget vel tænke sig, at startkapaciteten ikke skal være ens for alle kanter, eksempelvis kunne sandsynligheden for fejl på luftledninger hæves efter en storm. I afsnittet Perspektivering beskrives nogle af de parametre, der med fordel vil kunne inddrages i beregningen af kapaciteterne.

9.3.7 Modifikation af Edmunds-Karp algoritmen

Som beskrevet ovenfor er Edmunds-Karp designet til at finde kapaciteten af det mindste snit i grafen, $f(S,T)$ men altså ikke direkte de kanter der forbinder de to dele af grafen. Da man i EOMS algoritmen netop er interesseret i at finde de kanter, der forbinder de to delgrafer, mens den præcise kapacitet af snittet er underordnet, kan man med fordel modificere Edmunds-Karp algoritmen en smule.

Når den almindelige implementering af Edmunds-Karp anvendes på en sammenhængende graf, $G = (V, E)$, og alle kanter uden overskydende kapacitet[7] derefter fjernes fra grafen, bliver resultatet en skov af grafer. Oftest vil denne skov bestå af to grafer, nemlig en indeholdende alle kilder og en indeholdende alle dræn, men i specielle tilfælde vil grafen *G* blive splittet op i en skov bestående af tre eller flere grafer, se Figur 32 B. I EOMS algoritmen er man interesseret i at opsplitte grafen i netop to delgrafer, - en del med strøm og en uden, hvorfor en skov med mere end to grafer giver problemer. I stedet for at tilføje et ekstra trin i algoritmen, der kan "splejse" graferne i skoven sammen, så man altid kommer ned på netop to delgrafer, kan Edmunds-Karp modificeres, så det garanteres, at hvis alle kanter med en overskydende kapacitet på 0 fjernes fra grafen, er resultatet en skov bestående af præcis to grafer. For at gøre implementeringen af denne modifikation simplere, indføres den begrænsning, at alle kapaciteter skal være heltal. Hvis decimale kapaciteter forekommer, kan disse skaleres med en passende faktor, så de kan repræsenteres som heltal med en acceptabel afrunding.

Selve modifikationen består i at ændre den operator, der fratrækker den overskydende kapacitet langs en sti fra kilde til dræn. I den almindelige Edmunds-Karp trækkes den overskydende kapacitet i stien fra alle stiens kanter, hvorved en eller flere kanter bringes ned på en overskydende kapacitet på 0, Boks 8. I EOMS algoritmen modificeres denne operator som vist i Boks 9.

```
r = overskydende kapacitet i stien.

Gennemløb alle kanter i stien i topologisk rækkefølge
      a = den aktuelle kants kapacitet
      a = a-r
      førsteNulKant = sand

      Hvis a=0 og førsteNulKant = sand
            førsteNulKant = falsk
      Ellers
            a=1   //Sæt en kunstig kapacitet, for kun at
                  //"fjerne" én kant
```

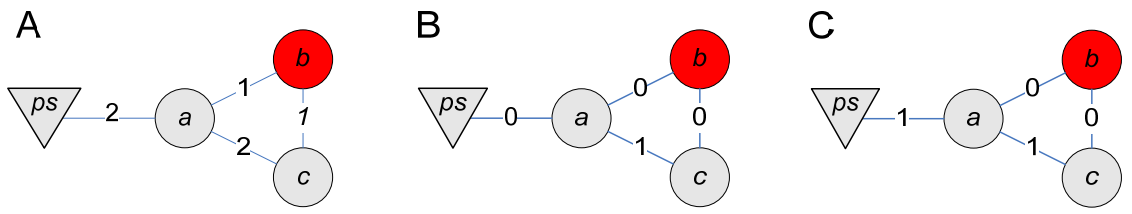
Boks 9: Modifikation af Edmunds-Karp algoritmen.

Det topologiske gennemløb af stiens kanter skal starte fra drænet, den fejlramte knude, og bevæge sig op mod strømkilden, også blot kaldet kilden. Da alle stier fra kilde til dræn i

[7] Kapaciteten efter kørsel af Edmunds-Karp, se Figur 28 D.

Edmunds-Karp er bestemt ved hjælp af BFS, kendes den topologiske rækkefølge af kanterne i stien allerede. Den modificerede operator er derfor ikke mere kompleks at evaluere end den oprindelige.

Ved ovenstående modifikation sikres det, at der for hver iteration af algoritmen kun er præcis én kant der bringes ned på en overskydende kapacitet på 0. Ved at sætte kapaciteten kunstigt til 1 på alle de øvrige kanter, der er kandidater til at få en kapacitet på 0, sikres det, at hvis en af disse kanter igen indgår i en sti fra kilde til dræn, og kanten ligger nærmest drænet, bringes denne kant ned på 0, jf. begrænsningen om heltallige kapaciteter. Ved at gennemløbe stien topologisk sikres det, at den kant der fjernes fra grafen er den af kandidaterne, der ligger nærmest de fejlmeldte knuder. Det er denne egenskab der sikrer, at det er snittet nærmest *b* der vælges i eksemplet på Figur 31 A.



Figur 32: Eksempel på forskellen mellem den almindelige Edmunds-Karp og den modificerede EOMS version. A: Starttilstanden. B: Overskydende kapacitet efter kørsel af standard Edmunds-Karp. C: Overskydende kapacitet efter kørsel af modificeret Edmunds-Karp.

Som eksemplet på Figur 32 viser, vil resultatet af at fjerne kanter uden overskydende kapacitet fra grafen på Figur 32 B resulterer i tre delgrafer, mens det på Figur 32 C kun resulterer i to delgrafer.

Sætning 1
Hvis den modificerede Edmunds-Karp algoritme køres på en sammenhængende, ikke orienteret graf $G = (V, E)$, med kilde $s \in V$ og dræn $t \in V$ hvorom det gælder at $s \neq t$, vil resultatet af at fjerne alle kanter uden overskydende kapacitet fra E være en skov bestående af præcis to delgrafer.

Bevis for Sætning 1
Betegn antallet af delgrafer i den skov der beskrives af sætningen som n.

Antag $n < 2$:
Da kilden *s* og drænet *t* jf. sætningen er to forskellige knuder i grafen, og den modificerede Edmunds-Karp først stopper, når der ikke længere findes en sti fra *s* til *t*, må grafen være splittet op i mindst to delgrafer. Dvs. $n<2$ kan ikke forekomme.

Antag $n > 2$:
Betragt en knude *v* tilhørende en sti fra *s* til *t* fundet med BFS. Da *v* ligger på en sti mellem *s* og *t* vides det, at *v* må have mindst en kant e_s der forbinder den til den delgraf der indeholder *s* og mindst en kant e_t der forbinder den til den delgraf der indeholder *t*.

Da stien er fundet med BFS, og derfor er den korsteste sti mellem *s* og *t* vides det endvidere, at $e_s \neq e_t$ og dermed at graden af *v* er større eller lig med 2. Da man, jf. ovenstående modifikation af Edmunds-Karp kun fjerner én kant per iteration, vil algoritmen maksimalt fjerne enten e_s eller e_t men aldrig dem begge, hvorved *v* aldrig kan udskilles som en selvstændig graf i skoven. Da der kun fjernes kanter fra knuder der ligger på en sti mellem *s* og *t*, vil enhver knude i grafen stadig være forbundet til enten *s* eller *t* efter kørsel af den modificerede Edmunds-Karp, hvorfor $n>2$ ikke er muligt.

■

Det skal bemærkes, at det ikke kan garanteres, at den modificerede Edmunds-Karp finder den korrekte værdi for kapaciteten af det mindste snit, $f(S,T)$, men da denne værdi ikke anvendes af EOMS betyder det ikke noget. Det kan ligeledes ikke garanteres, at det snit der bestemmes af den modificerede Edmunds-Karp faktisk er det mindste snit mellem strømkilder og fejlmeldte knuder, men derimod bliver snittet en vægtning af lav kapacitet og placering tæt på de fejlmeldte knuder, præcis som man ønsker det, når fejlfindingsproblemet skal løses. Ifølge Sætning 1 deler den modificerede Edmunds-Karp algoritme grafen over i præcis to delgrafer, hvorfor man er sikker på altid at finde en lovlig forklaring. I sammenligning med ATOMS og BOMS vil EOMS typisk betragte et mindre antal forklaringer, da de åbenlyst uhensigtsmæssige eller direkte ugyldige forklaringer aldrig beregnes.

9.3.8 Bestemmelse af alternative forklaringer

Ligesom det var tilfældet i ATOMS og BOMS algoritmerne, ønsker man også med EOMS at bestemme en række alternative forklaringer og slutteligt finde den bedste af disse forklaringer ved hjælp af objektfunktionen. For at bestemme alternative forklaringer i EOMS benyttes to forskellige metoder

- Formlen til beregning af kantkapaciteter justeres eller udskiftes, hvilket ofte vil fører til alternative placeringer af det mindste snit.
- En eller flere af de kanter, der indgår i det mindste snit, gives en kunstigt høj kapacitet, hvorefter det mindste snit med høj sandsynlighed vil flytte sig.

På Figur 31 er den første metode illustreret; forklaringerne fundet i Figur 31 A og Figur 31 B er forskellige udelukkende på grund af, at der anvendes forskellige initial kapaciteter for kanterne i de to grafer. I eksemplet på Figur 31 anvendes samme grundlæggende metode til at tildele kantkapaciteter, nemlig en lineær funktion af afstanden til nærmeste fejlrapporterede knude. Udover blot at ændre på konstanterne i formlen som i Figur 31, kan man forestille sig, at andre metoder som eksempelvis polynomiel eller eksponentiel afhængighed af afstanden til fejlmeldte knuder, vil kunne afsløre alternative forklaringer.

9.3.9 Køretidsanalyse

Køretiden for den modificerede Edmunds-Karp algoritme er ikke forskellig fra den almindelige Edmunds-Karp, nemlig $O(V+E^2)$, når man betragter worst case køretiden

på en orienteret graf. Når flowet beregnes på en ikke orienteret graf, som i dette tilfælde, kan køretiden dog bringes ned på $O(E^2)$ hvilket følger af beviset herunder.

Sætning 2 - Køretid af Edmunds-Karp på ikke orienterede grafer
Køretiden af Edmunds-Karp for at finde det maksimale flow fra *s* til *t* er $O(E^2)$, når algoritmen anvendes på en ikke orienteret graf $G = (V, E)$, hvorom det gælder at $|E| \geq |V|$.

Bevis for Sætning 2
Under forudsætningen $|E| \geq |V|$ kører BFS i tiden $O(E)$, da den almindelige køretid for BFS er $O(V + E)$ jf. afsnittet Breadth First Search. Hver iteration af Edmunds-Karp bringer mindst en kant i *E* ned på en overskydende kapacitet på 0. Da grafen er ikke-orienteret vides det, at når en kant er bragt ned på en overskydende kapacitet på 0, vil den aldrig igen kunne indgå i en sti fra *s* til *t*. Hver iteration af Edmunds-Karp kan køres i tiden $O(E)$ under ovenstående forudsætninger og der vil maksimalt foretages $O(E)$ iterationer, hvilket fører til køretiden $O(E^2)$.

■

Den modificerede version af Edmunds-Karp fjerner præcis én kant i hver iteration, så både Sætning 2 og beviset for Sætning 2 holder for den modificerede version af Edmunds-Karp. Forudsætningen $|E| \geq |V|$ kan med rimelighed antages at holde for grafer der repræsenterer elforsyningsnet, da det ikke er realistisk, at hver knude i grafen kun er forbundet til én anden knude.

Køretiden kan forbedres yderligere, hvis man tager højde for den specielle struktur af den graf der arbejdes med. Som beskrevet i afsnittet Elforsyningsnet består det samlede lavspændingsnet af mange små delnet, hvorfor grafrepræsentationen også bliver en skov af mange små undergrafer i stedet for én stor sammenhængende graf. Når BFS implementeres med hashtabeller, så initialisering kan undgås, kan en søgning i en delgraf $G' = (V', E')$ foretages i tiden $O(E')$ under samme forudsætninger som nævnt i Sætning 2. Desuden vil Edmunds-Karp også kun fjerne kanter fra *G'* hvorfor det maksimale antal iterationer ligeledes reduceres til $O(E')$. Med en BFS implementering baseret på hashtabeller kan Edmunds-Karp (og den modificerede Edmunds-Karp) altså bringes ned på en køretid på $O(E'^2)$, hvor $|E'|$ er antallet af kanter i den/de undergrafer i skoven, der rent faktisk indeholder fejl. Derved vil algoritmen automatisk se bort fra alle de undergrafer, der ikke indeholder fejl.

For at beregne startkapaciteterne for alle kanterne kræves det, at der laves en BFS søgning ud fra de fejlramte knuder, så afstanden til de fejlramte knuder kan indgå i beregningen. Under ovenstående forudsætninger kan BFS søgningen foretages i tiden $O(E)$ og efterfølgende, når alle afstande er beregnet, kan vægttildelingen også foretages i tiden $O(E)$. Derfor bliver den samlede tid for at beregne startkapaciteter $O(E)$. På

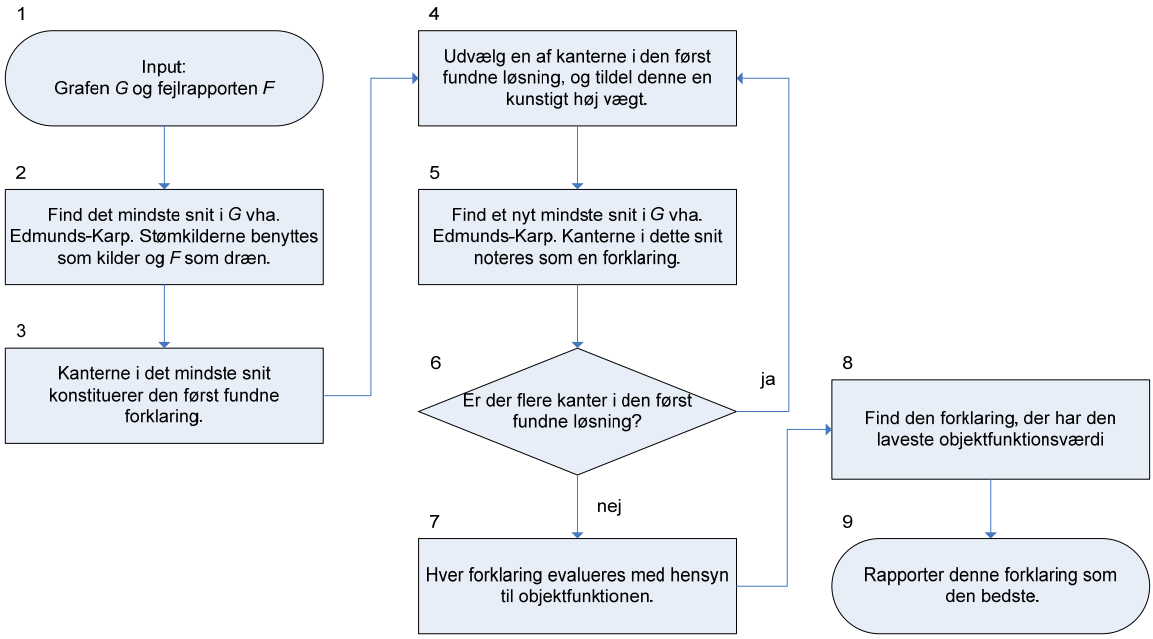
samme måde som nævnt ovenfor, kan BFS også implementeres med hashtabeller når der skal tildeles startkapaciteter, så køretiden kan bringes ned på $O(E')$.

I EOMS vil man oftest beregne et konstant antal alternative forklaringer, f.eks. ved at have 10 forskellige modeller for beregning af startkapaciteter, som bruges til at bestemme 10 alternative forklaringer. Bestemmelse af alternative forklaringer vil altså kun påvirke køretiden med en konstant faktor, der ikke fremgår af den asymptotiske køretid.

Derfor bliver den samlede køretid for EOMS $O(E^2)$ worst case når der regnes på en ikke orienteret graf hvorom det gælder at $|E| \geq |V|$. På en skov af grafer kan køretiden bringes ned på $O(E'^2)$, hvor $|E'|$ er antallet af kanter i de undergrafer i skoven, der indeholder fejlmeldte knuder.

9.3.10 Flowdiagram af hele EOMS algoritmen

Flowdiagrammet på Figur 33 viser hele forløbet af den Edmunds-Karp baserede algoritme, EOMS.



Figur 33: Flowdiagram over hele EOMS algoritmen.

9.4 Opsummering af fejlfindingsalgoritmer

I det foregående er der udviklet tre OMS algoritmer, nemlig ATOMS, BOMS og EOMS. ATOMS og BOMS anvender samme grundlæggende metode til at løse fejlfindingsproblemet, om end det må forventes, at ATOMS er væsentlig mere effektiv pga. de implementeringsmæssige fordele der kommer fra array teknologien.

EOMS bruger en radikalt anderledes måde at gribe problemet an på, ved at betragte elforsyningsnettet som et flownetværk og søge efter det mindste snit. Derved opnås en algoritme der på en effektiv måde kan drage nytte af de specielle strukturer der er i elforsyningsnettet.

For at omsætte de forklaringer der findes med OMS algoritmerne til konkrete arbejdsordre, er det hensigtsmæssigt at kunne identificere grupper af fejl. Derfor vil næste afsnit fokusere på forskellige metoder til gruppering af knuderne i en graf.

9.5 Gruppering af grafer

Når en af OMS algoritmerne bruges til at finde en forklaring på en fejlrapport, består resultatet af en mængde kanter, der med stor sandsynlighed er defekte. Hver forklaring omsættes til en eller flere konkrete arbejdsordrer, hvoraf det fremgår, hvilke kanter i nettet der skal efterses. For at kunne vurdere, om de knuder der indgår i fejlrapporten skal betragtes som én fejl og dermed blive til én arbejdsordre, eller om der skal genereres flere uafhængige arbejdsordrer, grupperes grafen omkring de fejlmeldte knuder. For hver gruppe i grafen der indeholder fejlmeldte knuder, genereres der én arbejdsordre.

Til gruppering (eng. clustering) af grafer findes der mange metoder. Geometriske metoder er meget udbredt, i disse metoder er hver knude i en graf repræsenteret af en vektor. Forskelligheden af to knuder beregnes oftest som afstanden mellem deres respektive vektorer. Denne fremgangsmåde forudsætter, at forskelligheden af to knuder afhænger af deres indbyrdes fysiske afstand og at knudernes koordinater er tilgængelige.

Grafer er imidlertid alsidige objekter, med mange andre karakteristika end knudernes indbyrdes afstand. Markov Clustering (MCL) benytter den måde, hvorpå grafen er forbundet til at bestemme grupperingen, hvilket ligger godt i forlængelse af OMS algoritmerne, der også anvender grafens topologi til at løse fejlfindingsproblemet. Markov Clustering er beskrevet i detaljer i [DONGEN] og er en både hurtig og matematisk elegant måde at gruppere en graf på.

9.5.1 Markov Clustering generelt

Markov Clustering bygger på en observation, der er gjort med hensyn til Markov kæder og problemstillingen ved at finde ”naturlige” grupper i en graf:

> **Observation 1**
> *Når en random walk[8] i grafen G støder ind i en tæt gruppering af knuder, er det usandsynligt at den vil forlade grupperingen før mange af dens knuder er besøgt.*

Den grundlæggende idé i Markov Clustering er at simulere flow i en graf. Under denne flow simulering ønsker man at styrke flowet hvor det er stærkt samtidig med at svække det hvor det er svagt. Hvis der eksisterer naturlige grupper i grafen, og Observation 1 holder stik, vil flowet mellem de naturlige grupper langsomt forsvinde helt og blotlægge grupperingen af grafen.

Selve det at simulere flow i en graf kan gøres ved at omdanne grafen til en Markov graf. I en Markov graf summerer vægtene af alle udgående kanter fra en knude til én. Flowet i grafen kan nu *udvides,* ved at sætte den til Markov grafen hørende Markov matrix i en potens. Flow simuleringen i sig selv er dog ikke nok til at afsløre den underliggende gruppering af grafen.

[8] En *random walk* er, som ordet antyder, en spadseretur på må og få en i graf. Begrebet bliver beskrevet i detaljer i det efterfølgende afsnit.

Vi er derfor nødt til at indføre en ny operator. Denne operator er ansvarlig for at styrke flowet hvor det er stærkt og samtidig svække flowet hvor det er svagt. Vi kalder denne operator for *inflation*[9]. Hvor *udvidelsen* af flowet i grafen kan beregnes ved hjælp af et matrix produkt, er flow *inflation* repræsenteret ved, at beregne Hadamard-Schur produktet[10] efterfulgt af en skalering af søjlerne i matricen.

Den resulterende proces kaldes for Markov Clustering processen eller MCL processen. Denne proces konvergerer kvadratisk mod en gruppe af såkaldt dobbelt idempotente[11] matricer. Dobbelt idempotente matricer er matricer der er uændrede både ved *udvidelse* og ved *inflation*. I denne tilstand, hvor matricen forbliver uændret, er flowet stabiliseret. Hvis der i denne tilstand eksisterer en naturlig gruppering, er denne synliggjort ved at knuderne i den dertil tilhørende graf nu enten kan kategoriseres som kilde eller et dræn. Hver naturlig gruppe består af et eller flere dræn samt et, typisk større, antal kilder.

9.5.2 Random Walk Modellen

For at forstå hvorfor Markov Clustering virker, er det nødvendigt at forstå hvad en *random walk* er og hvad en Markov kæde er. Da *random walk modellen* er baseret på Markov kæder er det nødvendigt først at forstå disse.

Definition 14

For et forløb der kan befinde sig i et endeligt antal tilstande $\{s_1, s_2, \ldots, s_n\}$ gælder det at en Markov kæde er defineret som en følge af sandsynligheder for at transitere mellem disse tilstande. Enhver Markov kæde kan repræsenteres som en stokastisk matrix M.[12]

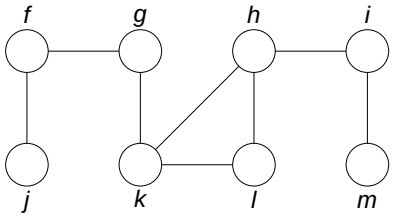
En følge af sandsynligheder kan kun beskrives som en Markov kæde hvis to betingelser er opfyldt:

1. For det første skal matricen *M* skal være stokastisk, dvs. at søjlerne skal summere til én. Dette opnås ved at søjle-normalisere matrixrepræsentationen af grafen *G*.
2. Desuden er Markov kæder per definition hukommelsesløse, dvs. at sandsynligheden for at skifte til en given tilstand, udelukkende afhænger af den tilstand man aktuelt befinder sig i. Da matrixrepræsentationen af grafen *G* udelukkende afhænger af grafens struktur er denne betingelse også opfyldt.

Enhver Markov kæde kan beskrives som en *random walk* på en retningsbestemt graf hvor hver knude $\{v_1, v_2, \ldots, v_n\}$ er en tilstand $\{s_1, s_2, \ldots, s_n\}$ i Markov kæden.

[9] Udvidelsen af flow i grafen kaldes på engelsk *expansion*, det er derfor meget naturligt at den nye operator bliver kaldt for *inflation*.
[10] Hadamard-Schur produktet er det elementvise produkt af matricerne, se http://planetmath.org/encyclopedia/HadamardProduct.html
[11] For en idempotent størrelse gælder det at den er uændret ved multiplikation med sig selv.
[12] En uddybende definition af Markov kæder kan findes i [MEYER] kapitel 8.4.

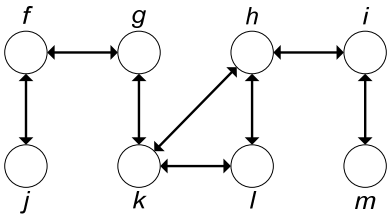
Random walk modellen kan nu beskrives ud fra et konkret eksempel.



$$T_{G_1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 34: Grafen G_1 og den dertil hørende matrixrepræsentation T_{G1}.

I dette eksempel benytter vi grafen G_1 og den dertil hørende matrixrepræsentation T_{G_1}, se Figur 34.



$$M = \left\| T_{G_1} \right\|_1 = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/3 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/3 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 35: Grafen G_1 omdannes til en Markov graf og en dertil hørende Markov matrix M.

Grafen G_1 omdannes nu til en retningsbestemt graf, dvs. hver kant erstattes af to kanter, - én i hver retning. De nye kanter vægtes så vægtene af alle udgående kanter fra en knude summerer til én. Grafen er nu en såkaldt Markov graf. Algebraisk fremkommer den til Markov grafen hørende Markov matrix, *M,* ved at normalisere søjlerne i T_{G_1} under 1-normen som vist på Figur 35.

$$\left\| T_{ij} \right\|_1 = T_{ij} \Big/ \left\| t_j \right\|_1 \qquad \text{hvor } t_j \text{ er den j-te søjle i } T. \tag{24}$$

1-normen er defineret som

$$\left\| \vec{v} \right\|_1 = \sum_{i=1}^{k} \vec{v}_i \qquad \text{hvor } \vec{v} = \{x_1, x_2, \ldots, x_k\} \tag{25}$$

Populært sagt foregår *random walken* nu ved at en knude tilfældigt udvælges som startknude og derefter bevæger man sig rundt i grafen. Dette gøres ved, at man ved hver knude vælger en af de udgående kanter, - med lige stor sandsynlighed. Vægtene i M er altså sandsynlighederne for at bevæge sig videre til de andre knuder i grafen. M_{ij} angiver sandsynligheden for at bevæge sig til knude *i*, givet at man befinder sig på knude *j*.

Denne proces må antages at være hukommelsesløs, da valget af den næste knude udelukkende afhænger af, hvilken knude man aktuelt befinder sig på.

Algebraisk foregår *random walken* ved, at Markov matricen *M* gentagne gange multipliceres med sig selv. Det er desuden nødvendigt at søjlenormalisere matricen efter hver iteration, da processen ellers dør ud. Det andet skridt i *random walken* er givet ved $M_2 = \|M \cdot M\|_1$, det tredje skridt er givet ved $M_3 = \|M_2 \cdot M_2\|_1$ osv. M_2 kan ses på Figur 36.

$$M_2 = \begin{bmatrix} 0{,}75 & 0 & 0 & 0 & 0 & 0{,}17 & 0 & 0 \\ 0 & 0{,}42 & 0{,}11 & 0 & 0{,}5 & 0 & 0{,}17 & 0 \\ 0 & 0{,}17 & 0{,}44 & 0 & 0 & 0{,}17 & 0{,}17 & 0{,}5 \\ 0 & 0 & 0 & 0{,}67 & 0 & 0{,}11 & 0{,}17 & 0 \\ 0 & 0{,}25 & 0 & 0 & 0{,}5 & 0 & 0 & 0 \\ 0{,}25 & 0 & 0{,}17 & 0{,}17 & 0 & 0{,}44 & 0{,}17 & 0 \\ 0 & 0{,}17 & 0{,}11 & 0{,}17 & 0 & 0{,}11 & 0{,}33 & 0 \\ 0 & 0 & 0{,}17 & 0 & 0 & 0 & 0 & 0{,}5 \end{bmatrix} \quad M_{100} = \begin{bmatrix} 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 \\ 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 \\ 0{,}21 & 0{,}21 & 0{,}21 & 0{,}21 & 0{,}21 & 0{,}21 & 0{,}21 & 0{,}21 \\ 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0{,}22 & 0{,}22 & 0{,}22 & 0{,}22 & 0{,}22 & 0{,}22 & 0{,}22 & 0{,}22 \\ 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

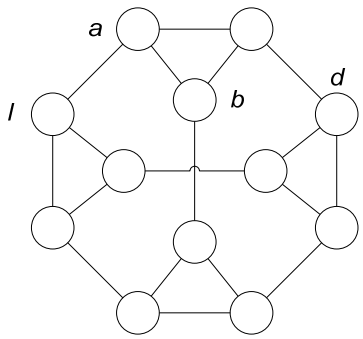
Figur 36: Markov matricen *M* efter hhv. to og et hundrede iterationer.

Efter et stort antal iterationer, i dette eksempel synes ét hundrede at være nok, stabiliserer Markov matricen *M* sig i en ligevægtstilstand, se Figur 36. Yderligere iterationer ændrer ikke resultatet. I denne tilstand er alle søjler i *M* identiske og de enkelte elementer i en søjle angiver den asymptotiske tid *random walken* har opholdt på en given knude. Vi noterer os at knuderne *h* og *k* er de mest besøgte knuder med hhv. 21 % og 22 % af den samlede tid. Man bør overbevise sig selv om det fornuftige i, at *random walken* ofte har besøgt knuderne *h* og *k*, ud fra den måde grafen *G* er forbundet.

De mest besøgte knuder virker umiddelbart som et fornuftigt gæt på, hvor gruppernes tyngdecentre, de førnævnte dræn, må være. Ud fra ligevægtstilstanden M_{100} på Figur 36 kan vi derfor gætte på at knuderne *h* og *k* kan fungere som dræn.

9.5.3 Random Walk Modellens begrænsninger

I langt de fleste eksempler vil en *random walk* i en graf producere resultater som i overstående eksempel. Ikke desto mindre er det muligt at konstruere grafer hvor *random walken* i dens nuværende form vil producere uventede resultater. Lad os se på et nyt eksempel.



Figur 37: Grafen G_2, der består af fire forbundne trekanter.

På Figur 37 ses grafen G_2. To eksempler på ekstreme grupperinger af denne graf er hvor hver knude er sin egen gruppe, eller hvor der kun er én gruppe, der indeholder samtlige knuder. Den eneste ikke-ekstreme gruppering er den, hvor hver trekant er en gruppe. Den til G_2 hørende Markov matrix $M = \left\| T_{G_2} \right\|_1$, hvor T_{G_2} er matrixrepræsentationen af G_2, er vist på Figur 38. Efter et hundrede iterationer af *random walk modellen* fremkommer Markov matricen M_{100}, som også er illustreret på Figur 38.

$$M = \begin{bmatrix} 0 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 \\ 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 \\ 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 \end{bmatrix} \quad M_{100} = \begin{bmatrix} 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 & 0{,}20 \\ 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 & 0{,}17 \\ 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 & 0{,}15 \\ 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 \\ 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 & 0{,}11 \\ 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 \\ 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 & 0{,}06 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 38: Markov matricerne M og M_{100} hørende til grafen G_2.

I M_{100} har Markov matricen stabiliseret sig i en ligevægtstilstand, hvor alle søjler i M_{100} er identiske og de enkelte elementer i en søjle angiver den asymptotiske tid *random walken* har opholdt på en given knude. Det ses ud fra matricen at knuden *a* er den mest besøgte knude med 20 % af den samlede tid. *Random walk modellen* synes i dette tilfælde at foretrække en gruppering omkring knuden *a*, - en gruppering der indeholder samtlige knuder i G_2. Dette er lidt uheldigt, da vi ønsker en gruppering, hvor hver trekant er en gruppe. For at forstå hvorfor *random walk modellen* foretrækker denne gruppering, bliver vi nødt til at se på noget man kalder *k-sti gruppering.*

k-sti gruppering. En *sti* er en sekvens af kanter, der leder fra én knude til en anden, Definition 2. Den samme kant må gerne indgå flere gange i en *sti*. *k-sti gruppering* er baseret på følgende observation.

> **Observation 2**
> *To knuder der tilhører den samme gruppe, må nødvendigvis have et større antal stier af længde k mellem sig, end to knuder der tilhører forskellige grupper.*

Observation 2 knytter sig til Observation 1, og ud fra Observation 2 kan vi forklare hvorfor *random walk modellen* ikke synes at producere de ønskede resultater for G_2.

Lad der være givet en graf $G=(V,E)$, hvor $V=\{v_1,v_2,\ldots,v_t\}$, og T_G er den til grafen hørende matrixrepræsentation. Lad k være et heltal, og $Z_{k,G}$ være en lighedskoefficient tilhørende G, over mængden af knuder V, givet ved $Z_{k,G}(v_i,v_j)=\infty$ hvor $i=j$ og

$$Z_{k,G}(v_i,v_j)=\left(T_G^k\right)_{ij}, \qquad i\neq j \tag{26}$$

Størrelsen $\left(T_G^k\right)_{ij}$ kan direkte læses som *antallet af stier med længde k mellem knuderne i og j*. Hvis G kan inddeles i grupper er det rimeligt at antage, at antallet af *stier* med længde k der begynder og ender i samme gruppe er stort, sammenlignet med antallet af *stier* med længde k der begynder og ender i forskellige grupper. Antagelsen gælder også for vægtede grafer, her er der blot tale om kapaciteter i stedet for deciderede *stier*, men for at holde eksemplet nogenlunde simpelt, vælger vi at se på ikke-vægtede grafer. Til grafen på Figur 37, G_2, indeholder T_{G_2} og $T_{G_2}^2$ følgende værdier.

$$T_{G_2}=\begin{bmatrix}
0&1&1&0&0&0&0&0&0&0&0&1\\
1&0&1&0&0&0&0&1&0&0&0&0\\
1&1&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&1&0\\
0&0&0&1&1&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&1&1&0&0&0\\
0&1&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&1&1&0&1&0&0\\
0&0&0&0&0&0&0&0&1&0&1&1\\
0&0&0&0&1&0&0&0&0&1&0&1\\
1&0&0&0&0&0&0&0&0&1&1&0
\end{bmatrix}
\qquad
T_{G_2}^2=\begin{bmatrix}
3&1&1&1&0&0&0&1&0&1&1&0\\
1&3&1&1&0&0&1&0&1&0&0&1\\
1&1&3&0&1&1&0&1&0&0&0&1\\
1&1&0&3&1&1&1&0&0&0&1&0\\
0&0&1&1&3&1&1&0&0&1&0&1\\
0&0&1&1&1&3&0&1&1&0&1&0\\
0&1&0&1&1&0&3&1&1&1&0&0\\
1&0&1&0&0&1&1&3&1&1&0&0\\
0&1&0&0&0&1&1&1&3&0&1&1\\
1&0&0&0&1&0&1&1&0&3&1&1\\
1&0&0&1&0&1&0&0&1&1&3&1\\
0&1&1&0&1&0&0&0&1&1&1&3
\end{bmatrix}$$

Figur 39: De til grafen G_2 hørende matricer.

Ud fra matricen $T_{G_2}^2$ på Figur 39 kan man læse at Observation 2 ikke holder i dette tilfælde. Det gælder ikke at antallet af *stier* af længe 2 mellem knuderne i en af trekanterne er større end antallet af stier mellem trekanterne. Denne uheldige egenskab

har noget at gøre med selve strukturen af grafen og specielt at grupperne er trekanter. For *stier* bestående af et lige antal kanter, f.eks. $k = 2$, er det *stier* som *ad*, på Figur 37 der ødelægger antagelsen. Ligeledes for *stier* bestående af et ulige antal kanter, f.eks. $k = 1$, er det stier som *al* der ødelægger antagelsen.

Generelt gælder det at *stier* af ulige paritet, dvs. *stier* af ulige længde, eksempelvis *stien* fra *ab*, ikke drager større fordel af at *stien* befinder sig i en trekant, end en *sti* som *ad* drager af at dens endepunkter befinder sig i en trekant. Lighedskoefficienten $Z_{k,G}$ er i meget høj grad afhængig af pariteten af k. Ulige potenser af T_{G_2} får sine værdier fra *stier* af ulige længde, imens lige potenser af T_{G_2} får sine værdier fra *stier* af lige længde. Den eneste undtagelse til denne regel er de *stier*, der indeholder et loop med et ulige antal kanter. Læg mærke til, at hver kant i grafen automatisk giver et loop af længden to.

Ovenfor nævnte undtagelse er samtidig løsningen på problemet; ved at tilføje et loop til hver eneste knude i grafen forsvinder lighedskoefficienten $Z_{k,G}$´s afhængighed af paritet. Ligesom hver kant i grafen producerer et loop med længden to, producerer hver knude nu et loop med længden én. Hermed er lighedskoefficientens afhængighed af paritet forsvundet. Rent algebraisk kan et loop til hver eneste knude tilføjes ved at addere T_{G_2} med identitetsmatrixen I.

$$T_{G_2}+I=\begin{bmatrix}1&1&1&0&0&0&0&0&0&0&0&1\\1&1&1&0&0&0&0&1&0&0&0&0\\1&1&1&1&0&0&0&0&0&0&0&0\\0&0&1&1&1&1&0&0&0&0&0&0\\0&0&0&1&1&1&0&0&0&0&1&0\\0&0&0&1&1&1&1&0&0&0&0&0\\0&0&0&0&0&1&1&1&1&0&0&0\\0&1&0&0&0&0&1&1&1&0&0&0\\0&0&0&0&0&0&1&1&1&1&0&0\\0&0&0&0&0&0&0&0&1&1&1&1\\0&0&0&0&1&0&0&0&0&1&1&1\\1&0&0&0&0&0&0&0&0&1&1&1\end{bmatrix}\quad\left(T_{G_2}+I\right)^2=\begin{bmatrix}4&3&3&1&0&0&0&1&0&1&1&2\\3&4&3&1&0&0&1&2&1&0&0&1\\3&3&4&2&1&1&0&1&0&0&0&1\\1&1&2&4&3&3&1&0&0&0&1&0\\0&0&1&3&4&3&1&0&0&1&2&1\\0&0&1&3&3&4&2&1&1&0&1&0\\0&1&0&1&1&2&4&3&3&1&0&0\\1&2&1&0&0&1&3&4&3&1&0&0\\0&1&0&0&0&1&3&3&4&2&1&1\\1&0&0&0&1&0&1&1&2&4&3&3\\1&0&0&1&2&1&0&0&1&3&4&3\\2&1&1&0&1&0&0&0&1&3&3&4\end{bmatrix}$$

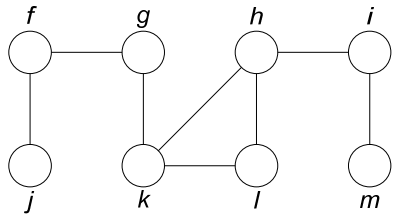
Figur 40: De til grafen G_2 hørende matricer adderet med identitetsmatricen.

På Figur 40 illustreres det, hvordan vi ved at tilføje loops til samtlige knuder i grafen G_2, kan neutralisere lighedskoefficienten $Z_{k,G}$´s afhængighed af paritet. Med denne modifikation fungerer *k-sti gruppering* på grafen G_2 og dermed holder Observation 2 også i dette, noget specielle, tilfælde. Ved at modificere *random walk modellen* på samme måde, kan også den forbedres med hensyn til dette specielle tilfælde. Vi vil kalde denne forbedrede model for den **modificerede random walk model**.

Efter gennemgangen af flowsimulering vha. den *modificerede random walk model*, kan det forklares hvordan man ved hjælp af *inflation* operatoren kan afsløre grupperingen af grafen.

9.5.4 Inflation

Der vil nu blive defineret en ny operator. Den nye operator hedder *inflation*, og benyttes i forbindelse med den *modificerede random walk model*, og dermed flow simulering, til at styrke flowet hvor det er stærkt og samtidig svække flowet hvor det er svagt.



Figur 41: Grafen G_1, der også er at se på Figur 34.

I dette afsnit benytter vi os igen af grafen G_1 som illustreret på Figur 41. Nøjagtig som i afsnittet **Random Walk Modellen**, konstruerer vi en Markov matrice ud fra grafen. Som beskrevet i afsnittet **Random Walk Modellens begrænsninger**, vælger vi denne gang at tilføje loops til grafen, ved at addere matrixrepræsentationen T_{G_1} med identitetsmatricen *I*. Endelig søjlenormaliseres den resulterende matrix.

$$M = \left\| T_{G_1} + I \right\|_1 \tag{27}$$

Vi ser nu på et udvalg af iterationer af *random walken* på denne Markov matrix. Matricerne M, M_2, M_3 og M_{100} er vist på Figur 42.

$$M = \begin{bmatrix} 1/3 & 1/3 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 1/3 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 1/4 & 1/3 & 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 1/4 & 0 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 1/4 & 0 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 1/2 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0{,}39 & 0{,}22 & 0 & 0 & 0{,}42 & 0{,}08 & 0 & 0 \\ 0{,}22 & 0{,}31 & 0{,}06 & 0 & 0{,}17 & 0{,}15 & 0{,}08 & 0 \\ 0 & 0{,}08 & 0{,}29 & 0{,}19 & 0 & 0{,}21 & 0{,}28 & 0{,}17 \\ 0 & 0 & 0{,}15 & 0{,}36 & 0 & 0{,}06 & 0{,}08 & 0{,}42 \\ 0{,}28 & 0{,}11 & 0 & 0 & 0{,}42 & 0 & 0 & 0 \\ 0{,}11 & 0{,}19 & 0{,}21 & 0{,}08 & 0 & 0{,}29 & 0{,}28 & 0 \\ 0 & 0{,}08 & 0{,}21 & 0{,}08 & 0 & 0{,}21 & 0{,}28 & 0 \\ 0 & 0 & 0{,}08 & 0{,}28 & 0 & 0 & 0 & 0{,}42 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0{,}33 & 0{,}22 & 0{,}03 & 0 & 0{,}37 & 0{,}09 & 0{,}04 & 0 \\ 0{,}22 & 0{,}20 & 0{,}09 & 0{,}03 & 0{,}21 & 0{,}14 & 0{,}11 & 0{,}01 \\ 0{,}04 & 0{,}11 & 0{,}23 & 0{,}22 & 0{,}01 & 0{,}21 & 0{,}24 & 0{,}20 \\ 0 & 0{,}03 & 0{,}16 & 0{,}29 & 0 & 0{,}09 & 0{,}11 & 0{,}35 \\ 0{,}25 & 0{,}14 & 0 & 0 & 0{,}31 & 0{,}04 & 0 & 0 \\ 0{,}12 & 0{,}18 & 0{,}21 & 0{,}12 & 0{,}08 & 0{,}22 & 0{,}24 & 0{,}07 \\ 0{,}04 & 0{,}11 & 0{,}18 & 0{,}11 & 0{,}01 & 0{,}18 & 0{,}20 & 0{,}07 \\ 0 & 0 & 0{,}10 & 0{,}23 & 0 & 0{,}04 & 0{,}05 & 0{,}30 \end{bmatrix} \quad M_{100} = \begin{bmatrix} 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 \\ 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 & 0{,}14 \\ 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 \\ 0{,}12 & 0{,}12 & 0{,}12 & 0{,}12 & 0{,}12 & 0{,}12 & 0{,}12 & 0{,}12 \\ 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 & 0{,}08 \\ 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 & 0{,}19 \\ 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 & 0{,}13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 42: Et udsnit af random walken på grafen G_1.

Det ses på Figur 42 at M_{100}, dvs. *random walken* efter et hundrede iterationer, har stabiliseret sig i en ligevægtstilstand *e*, hvor *e* er en søjlevektor. Alle søjler i M_{100} har

værdien *e*. Elementerne i *e* repræsenterer, som før nævnt, den asymptotiske tid *random walken* har opholdt sig på en given knude i G_1. Den samlede masse i M_{100} synes i det store hele at være jævnt fordelt ud over matricen. Vi kan af matricen M_{100} aflæse hvor tyngdecentrene, de førnævnte dræn, må være (knuderne *h* og *k*,) men ikke hvilke knuder der hører til hvilket tyngdecenter.

Lad os derimod se på matricerne M_2 og M_3. I disse tidlige iterationer opfører Markov matricen sig i høj grad på samme måde som $\left(T_{G_2} + I\right)^2$ gjorde det på Figur 40. I M_2 og M_3 gælder det at sandsynligheden for at bevæge sig fra en knude til en anden er stor hvis de to knuder tilhører den samme naturlige gruppe.

På baggrund af ovenstående observationer, ser det ud til at forholde sig sådan, at flowet er stærkt i de naturlige grupper gennem de første iterationer, men at denne effekt på langt sigt synes at dø ud. Derfor vil det være oplagt at styrke den førstnævnte effekt. Dette opnås ved at ændre hver søjle i Markov matricen så sandsynlighederne for at bevæge sig til de populære naboknuder favoriseres yderligere på bekostning af de mindre populære naboer. Denne effekt kan opnås ved at sætte de enkelte elementer i hver søjle i en potens større end én, (potensen kunne f.eks. være to,) og derefter normalisere søjlen. Vi kalder denne operator for *inflation* og giver den symbolet Γ (stort gamma.)

$$\text{Givet } v = \begin{pmatrix} 0 \\ 3 \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1/2 \\ 0 \\ 1/3 \\ 1/6 \end{pmatrix} \quad \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \\ 0 \end{pmatrix}$$

$$\Gamma_2 v = \begin{pmatrix} 0 \\ 9/10 \\ 0 \\ 1/10 \end{pmatrix} \quad \begin{pmatrix} 9/14 \\ 0 \\ 4/14 \\ 1/14 \end{pmatrix} \quad \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \\ 0 \end{pmatrix}$$

Figur 43: Effekten af inflation på forskellige vektorer.

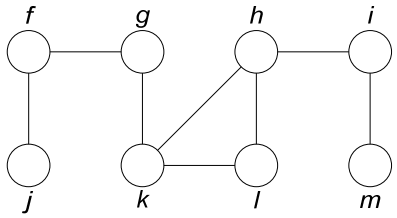
På ovenstående figur, Figur 43, er vist et par eksempler på, hvilken effekt *inflation* har på en vektor *v*. Γ_2 angiver at den benyttede potens er 2. Den benyttede potens bestemmer styrken af *inflationen*.

I det første eksempel er styrkeforholdet mellem ikke-nul-elementerne i *v* 3:1, efter *inflation* er styrkeforholdet 9:1 - det i forvejen favoriserede element favoriseres yderligere. I det andet eksempel summerer elementerne i *v* til én, og *v* kunne derfor være en søjle i en søjlestokastisk matrix, som f.eks. en Markov matrix. Her er resultatet det samme, - de i forvejen mest sandsynlige udfald favoriseres yderligere på bekostning af de mindre sandsynlige udfald. Det tredje eksempel viser, at lige sandsynlige udfald forbliver lige sandsynlige, altså 1:1, under *inflation*.

Udvidelse og *inflation* af flowet i en Markov matrix kan forstås som modsatrettede processer. *Udvidelsen* af flowet udglatter ujævnheder i Markov matricen, og fordeler sandsynlighederne for at bevæge sig mellem de forskellige tilstande en smule. *Inflation* er derimod en proces der trækker konturer op og skærper de karakteristiske træk i Markov matricen.

9.5.5 Markov Clustering Eksempel

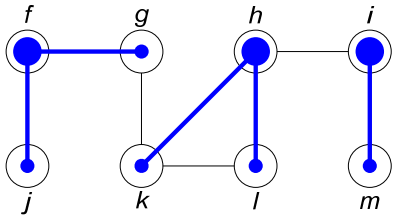
I dette afsnit gennemgår vi et gennemløb af Markov Clustering ud fra et konkret eksempel. Markov Clustering processen er en iterativ proces, hvor man gennem hver iteration påtrykker de to operatorer *udvidelse* og *inflation*. Til dette eksempel benytter vi grafen G_1 på Figur 44. Markov matricen konstrueres som i afsnittet **Random Walk Modellens begrænsninger** ved at søjlenormalisere summen af matrixrepræsentationen T_{G_1} og identitetsmatricen I. Altså $M = \left\| T_{G_1} + I \right\|_1$. Markov matricen kan ligeledes ses på Figur 44.



$$M = \begin{bmatrix} 1/3 & 1/3 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 1/3 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 1/4 & 1/3 & 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 1/4 & 0 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 1/4 & 0 & 0 & 1/4 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 1/2 \end{bmatrix}$$

Figur 44: Grafen G_1, og den dertil hørende Markov matrix M.

Vi gennemløber nu et antal iterationer hvor vi først *udvider* flowet, for derefter at påtrykke *inflation*.



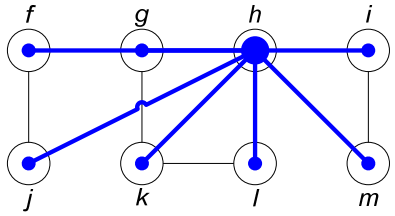
$$M_{13} = \begin{bmatrix} 1{,}00 & 1{,}00 & 0 & 0 & 1{,}00 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1{,}00 & 0 & 0 & 1{,}00 & 1{,}00 & 0 \\ 0 & 0 & 0 & 1{,}00 & 0 & 0 & 0 & 1{,}00 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 45: Gruppering af grafen G_1 samt Markov matricen efter 13 iterationer af Markov Clustering processen.

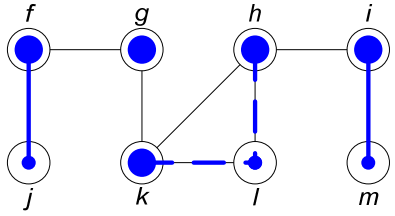
Efter 13 iterationer fås den på Figur 45 viste matrix. M_{13} tilhører gruppen af dobbelt idempotente matricer, dvs. matricer der forbliver uændrede både under *udvidelse* og *inflation*. Markov Clustering processen må derfor siges at have stabiliseret sig i en ligevægtstilstand. I denne ligevægtstilstand kan grupperingen direkte aflæses af matricen.

Grupperingen aflæses på følgende måde. Hver søjle i Markov matricen svarer til en knude i grafen. Hvis en søjle har et ettal på diagonalen er den pågældende knude et dræn, dvs. et tyngdecenter for en gruppe. Disse dræn er markeret på grafen i Figur 45 som en stor blå prik. De resterende søjler er kilder, hver kilde løber over i et dræn. Hvilket dræn en kilde løber over i kan aflæses ud fra hvilken række i den pågældende søjle ettallet befinder sig i, f.eks. løber *k* (den sjette søjle) over i *h* (den tredje række.)

Ovenstående gruppering er opnået ved at benytte en potenskoefficient på 2. Potenskoefficienten bestemmer styrken af inflationen og dermed densiteten af grupperingen.



$$M_{23} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1{,}00 & 1{,}00 & 1{,}00 & 1{,}00 & 1{,}00 & 1{,}00 & 1{,}00 & 1{,}00 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



$$M_{4} = \begin{bmatrix} 1{,}00 & 0 & 0 & 0 & 1{,}00 & 0 & 0 & 0 \\ 0 & 1{,}00 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1{,}00 & 0 & 0 & 0 & 0{,}50 & 0 \\ 0 & 0 & 0 & 1{,}00 & 0 & 0 & 0 & 1{,}00 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1{,}00 & 0{,}50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 46: Gruppering af grafen med potenskoefficient 1,3 (øverst) og potenskoefficient 4,0 (nederst.)

På Figur 46 ses resultatet af grupperingen ved brug af potenskoefficient 1,3 og 4,0. En lav potenskoefficient betyder at Markov Clustering processen forholdsvis langsomt (23 iteration) stabiliserer sig i en ligevægtstilstand og resultatet er få relativt store grupper. Omvendt betyder en høj potenskoefficient at Markov Clustering processen forholdsvis hurtigt (4 iterationer) stabiliserer sig i en ligevægtstilstand og resultatet er mange relativt små grupper. Bemærk specielt at knuden *l* på Figur 46 (nederst) tilhører to grupper, hvilket aflæses som værdien 0,5 i matricen.

Potenskoefficienten har stor indflydelse på Markov Clustering processens opførsel, og kan benyttes til at finjustere algoritmen.

9.5.6 Formel beskrivelse af Markov Clustering

I de foregående afsnit har vi fået en forståelse for, hvordan de individuelle komponenter i Markov Clustering agerer. Vi er derfor nu i stand til at beskrive Markov Clustering formelt.

Definition 15
For enhver simpel graf $G = (V, E)$, hvor $V = \{v_0, v_1, \ldots v_n\}$ og E er alle par af knuder der forbinder grafen, findes der en matrixrepræsentation $T \in \Re^{n \times n}$ hvor

$$T_{ij} = \begin{cases} 1 & hvis\ (v_i, v_j) \in E \vee (v_j, v_i) \in E \\ 0 & ellers \end{cases} \quad (28)$$

Definition 16
Givet en matrixrepræsentation $T \in \Re^{n \times n}$ af en simpel graf, er den dertil hørende Markov matrix $M \in \Re^{n \times n}$, defineret som $M = \|T + I\|_1$ hvor I er identitetsmatricen.

Definition 17
*Lad $M \in \Re^{n \times n}$ være en matrix. **Udvidelsen** af flowet i M er defineret som matrixproduktet. $\Re^{n \times n} \times \Re^{n \times n} \to \Re^{n \times n}$.*

Definition 18
*Givet en matrix $M \in \Re^{n \times n}$, hvor $M_{ij} \geq 0$ for $i = 1,2,\ldots,n$ og $j = 1,2,\ldots,n$, samt et ikke-negativt tal r, er **inflation** resultatet af at genskalere hver søjle i M med potenskoefficienten r. Inflation af matricen M skrives som $\Gamma_r M$ og Γ_r kaldes inflations operatoren med potenskoefficient r. Lidt mere formelt er $\Gamma_r : \Re^{n \times n} \to \Re^{n \times n}$ defineret som*

$$\Gamma_r M_{ij} = (M_{ij})^r \Big/ \sum_{k=1}^{n} (M_{kj})^r \quad (29)$$

Implementeringen af Markov Clustering er simpel og elegant, dette er en af styrkerne ved algoritmen. Algoritmen kan beskrives i pseudokode som angivet i Boks 10.

```
G er en graf.
T er den til G tilhørende matrixrepræsentation.
M1 = T + I                    // Hvor I er identitetsmatricen.
M1 = |M1|₁                    // Søjlerne i M1 normaliseres.
r = r₀                        // En passende potenskoefficient
                              // vælges.
while (change)
{
      M2 = M1 * M1            // Udvidelse
      M1 = Γr M2              // Inflation
      change = Difference(M1, M2)
}

set CLUSTERING til de enkelte komponenter i M1
```

Boks 10: Pseudokode for Markov Clustering algoritmen.

I Markov Clustering algoritmen er den eneste parameter man kan justere *r* – potenskoefficienten. Potenskoefficienten kaldes undertiden også for granulariteten, og beskriver den ønskede densitet af grupperne. En lav værdi af *r* betyder, at man ønsker få store grupper, mens en høj værdi angiver, at man ønsker mange små grupper.

Med hensyn til konvergens, kan man vise at Markov Clustering processen konvergerer[13] kvadratisk mod en såkaldt dobbelt idempotente matrix, dvs. en matrix der er uændret både ved *udvidelse* og *inflation*. I praksis betyder dette, at algoritmen har stabiliseret sig i en ligevægtstilstand efter 5 – 30 iterationer.

Ved analyse af den asymptotiske køretid af Markov Clustering processen er det de tre operatorer inde i loopet der er interessante. Både *inflation* og forskellen mellem to matricer kan udregnes i $O(V^2)$ og derfor er det *udvidelsen*, matrixproduktet, der er den dominerende faktor. Den asymptotiske køretid af matrixproduktet, og dermed den asymptotiske køretid af Markov Clustering processen, er $O(V^3)$. Det skal dog siges, at det er muligt at forbedre ydeevnen af matrixproduktet væsentligt, ved at benytte en sparse matrix repræsentation, se Appendiks 1: Matrix repræsentation og Appendiks 2: Matrix multiplikation.

9.5.7 Flowsimulering med fejlbehæftede knuder

Ved gruppering i en graf hvor en eller flere knuder er fejlbehæftede, vil det være hensigtsmæssigt at kunne inddrage denne viden når grupperingen finder sted. Vi har derfor modificeret den traditionelle Markov Clustering proces til at tage højde for fejlmeldte knuder.

I afsnittet Random Walk Modellens begrænsninger, beskrev vi hvordan vi ved at tilføje loops til grafen kunne neutralisere lighedskoefficientens afhængighed af paritet. Ved selektivt at tilføje yderligere loops til grafen kan vi dirigere flowet, så dets tyngdecenter bliver omkring de fejlbehæftede knuder. Det er attraktivt at skabe en gruppering, der har tyngdecenter omkring de fejlbehæftede knuder, da vi ønsker at splitte grafen op således, at den topologiske afstand mellem de steder i nettet der arbejdes, maksimeres. Læg mærke til at vi stadig simulerer flow i den fuldstændige graf; information vedrørende fejlbehæftede knuder bruges udelukkende til at dirigere tyngdecentrets lokalitet. Selektiv tilføjelse af loops kan algebraisk beskrives som en matrix, der adderes til matrixrepræsentationen af grafen.

Definition 19

Givet en graf $G = (V, E)$*, hvor* $V = \{v_0, v_1, \ldots, v_n\}$ *og en vektor af ikke-negative heltal s med* $|s| = n$*, kan selektiv tilføjelse af loops defineres som en matrix* $S \in \Re^{n \times n}$ *for hvilket det gælder* $S_{ij} = 0$ *for* $i \neq j$ *og* $S_{ij} = s_i$ *for* $i = j$*.*

Vi benytter samme graf som i tidligere eksempler, G_1. I dette eksempel er knuden *k* fejlbehæftet og vi ønsker i forbindelse med gruppering derfor et tyngdecenter omkring

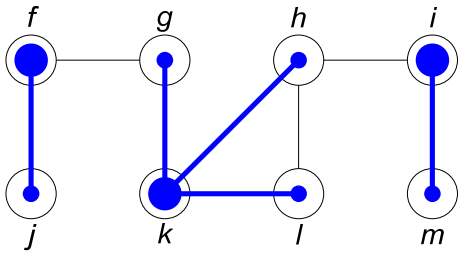
[13] Beviset for konvergens kan læses i [DONGEN] kapitel 6.5.

knuden *k*. På Figur 47 har vi tilføjet yderligere 4 loops til knuden *k*. Da alle knuder i forvejen havde fået tilføjet et loop, betyder det at *k* nu har 5 loops. (Se sjette søjle, sjette række.)

$$T_{G_1} + I = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{G_1} + I + S = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 5 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figur 47: Venstre: Matrixrepræsentation af grafen hvor der er tilføjet loops. Højre: Yderligere loops er tilføjet til knuden *k* for at skabe en gruppering med tyngdecenter i *k*.

Når vi efterfølgende benytter Markov Clustering processen til at bestemme en gruppering af grafen, får vi det på Figur 48 viste resultat. Dette resultat skal sammenlignes med Figur 45, hvorved man tydeligt ser at tyngdecenteret, dvs. drænet, der befandt sig omkring knuden *h* i Figur 45 nu i stedet befinder sig omkring knuden *k* i Figur 48.



$$M_7 = \begin{bmatrix} 1{,}00 & 0 & 0 & 0 & 1{,}00 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1{,}00 & 0 & 0 & 0 & 1{,}00 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1{,}00 & 1{,}00 & 0 & 0 & 1{,}00 & 1{,}00 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figur 48: Gruppering af grafen G_1 samt Markov matricen efter 7 iterationer af Markov Clustering processen. Yderligere loops var tilføjet til knuden *k*.

Denne metode tilføjer yderligere fleksibilitet til Markov Clustering processen. Selektiv tilføjelse af loops samt potenskoefficienten udgør de værktøjer, vi har til rådighed med henblik på at skabe en optimal gruppering af grafen.

9.5.8 Markov Clusterings begrænsninger

Som det ses af ovenstående, er Markov Clustering en meget fleksibel metode til at finde grupper i en graf. Dog må man forvente, at den på meget store grafer kører for langsomt. De største delnet i lavspændingsnettet er på ca. 3000 knuder, hvilket godt kan behandles med Markov Clustering, men algoritmen kan ikke anvendes på DONG Energys samlede elforsyningsnet. For at identificere de enkelte delnet, i det samlede net, beskrives i det næste afsnit en simpel form for gruppering kaldet Sektionering.

9.6 Sektionering af grafer

Som et alternativ til den grupperingsalgoritme, der er beskrevet i forrige afsnit, indføres en algoritme til sektionering af en graf. Sektionering skal forstås som en simpel gruppering, hvor en del af dynamikken fra den rigtige gruppering er fjernet. Gruppering finder selv et fornuftigt forhold mellem antallet af grupper, størrelsen af grupperne og den indbyrdes afstand mellem grupperne. Sektionering er en meget mere låst algoritme, hvor man angiver den ønskede indbyrdes afstand mellem sektionerne som eneste parameter. Således kan man angive, at alle knuder med en indbydes afstand på eksempelvis 3 eller mindre, skal sammensættes i en sektor, hvorefter sektioneringsalgoritmen bruger dette som eneste kriterium for, om to knuder ligger i samme sektion eller ej. Formålet med sektionering er at ofre en del af fleksibiliteten fra grupperingsalgoritmen, for derved at konstruere en algoritme, der pga. den simplere beregning, kan anvendes på væsentligt større problemer.

I det konkrete tilfælde vil sektionering, i lighed med gruppering, blive anvendt til at finde klynger af fejl i netværket, således at disse kan behandles separat af fejlfindingsalgoritmerne og så man kan oprette præcis én arbejdsordre pr. klynge af fejl. Sektionering vil også blive anvendt i kombination med Markov Clustering til først at identificere delnet, for derefter at anvende Markov Clustering til at identificere naturlige grupper.

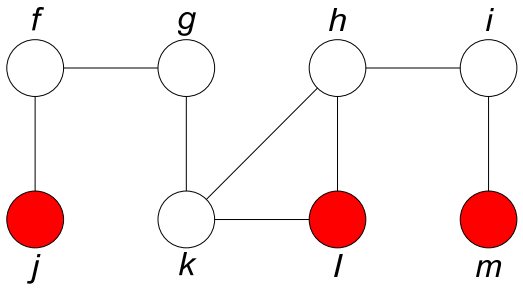
Sektionering er baseret på adskillige kald til en udvidet version af BFS se [CORMEN].

```
Sektionér(G, Vu, b)      //G = graph, Vu= liste af knuder der skal
                         //sektioneres, b = sektionsbuffer

      For hver knude v i Vu
            Hvis v ikke tilhører en sektion
                  sektion = ny sektion bestående af v
                  stop = falsk

                  Do While Not Stop
                        BFSResult = Knuder fundet via bfs fra alle
                        knuder i sektion indenfor en maksimal afstand
                        på b

                        Hvis |BFSResult ∩ Vu|=|sektion|  //Ingen nye
                              stop = sand       // knuder fundet
                        Ellers
                              sektion = BFSResult ∩ Vu
                  Loop

            Tilføj sektion til listen over sektioner
      Next

      Returner liste af sektioner
```

Boks 11: Pseudokode for sektioneringsalgoritmen

Som det ses af pseudokoden i Boks 11 består modifikationen af BFS i, at den kan afbryde søgningen, når en bestemt afstand fra kilderne er opnået.

På figuren herunder illustreres, hvordan sektionerne ændrer sig, når man retter på sektionsbufferen.



Figur 49: Eksempel på sektionering. De fejlramte knuder der skal sektioneres er V_u={j, l, m}

Hvis sektionering køres på eksemplet i Figur 49 med en sektionsbuffer på 3 opdeles V_u i to sektioner:

- Sektion 1 = *{l, m}*
- Sektion 2 = *{j}*,

da afstanden mellem l og m er 3, mens afstanden fra j til både *l* og *m* er større end 3. Hvis sektionsbufferen sættes op til 4 holdes hele V_u sammen i en sektion, så resultatet bliver

- Sektion 1 = *{j, l, m}*,

da afstanden mellem *j* og *l* er 4 og afstanden mellem *l* og *m* er 3.

Bemærk at sektionering, i modsætning til gruppering, ikke finder et tyngdecenter (dræn) for hver sektor.

9.6.1 Køretid

Køretiden for sektioneringsalgoritmen er domineret af kaldene til BFS, da det kan afgøres i konstant tid, $O(1)$, om en knude i V_u er tilføjet en sektion, hvis sektioner implementeres som hashtabeller.

Som det fremgår af pseudokoden i Boks 11, vil der i værste fald blive udført en BFS søgning for hver knude i V_u. Da køretiden for BFS, jf., afsnittet Breadth First Search er $O(V+E)$, er worst case køretid for sektionering $O(V_u \cdot (V+E))$.

Strukturen af den graf, der repræsenterer elforsyningsnettet gør, at sektionering kan køre på selv meget store problemer. Da graden af grafen er lav, dvs. $|E| = k \cdot |V|$ hvor k er en lille konstant, kan den forventede køretid på BFS udtrykkes som $O(V)$, hvorfor den

forventede køretid kan udtrykkes som $O(V_u \cdot V)$. Da man ikke vil forvente en situation, hvor alle forbrugere i netværket rapporterer fejl, kan det med rimelighed antages, at $|Vu| << |V|$ hvilket yderligere styrker den forventede køretid.

Faktisk vil størrelsen af sektionsbufferen, samt størrelsen af delgraferne i skoven også påvirke den forventede køretid. Hvis sektionsbufferen er lille, eller grafen er delt op i mange så delgrafer, vil BFS køre langt hurtigere end $O(V)$. I det konkrete tilfælde skal sektionsbufferen være forholdsvis stor, da man ønsker at generere et lille antal arbejdsordrer, men opdelingen af grafen i mange små delgrafer vil påvirke den forventede køretid positivt.

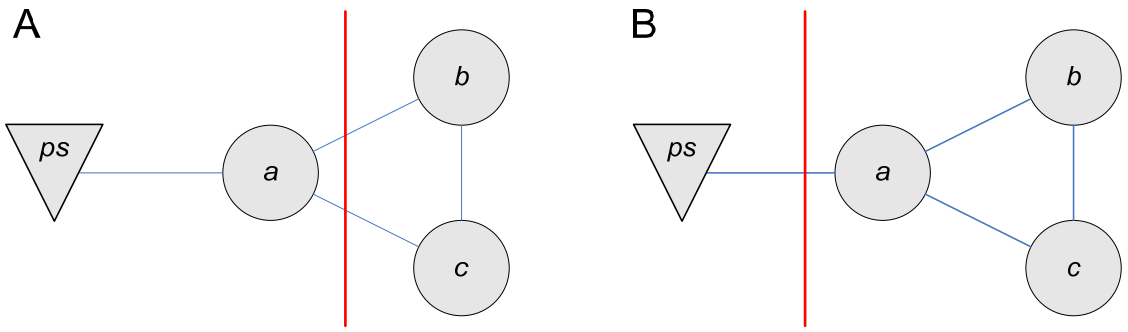
For en sammenligning af køretiderne for sektionering og gruppering, se afsnittet Benchmarking.

9.7 Indlæring

Parametrene α og β i objektfunktionen er meget afgørende for, hvor godt de arbejdsordrer der genereres af OMS systemet, stemmer overens med virkeligheden. Parametrene kan ikke på forhånd fastlægges ud fra en matematisk betragtning, da de afhænger af, hvor ofte forskellige typer af fejl opstår i nettet. α og β vil højst sandsynligt også variere mellem de forskellige typer delnet. Derfor må parametrene fastlægges empirisk på baggrund af erfaringer med, hvor i nettet fejlene typisk opstår. Til dette formål anvendes den konfigurationsalgoritme der beskrives i dette afsnit.

9.7.1 Indsamling af data

Som beskrevet i afsnittet Algoritmer beregner hver af de tre OMS algoritmer en række mulige forklaringer til hver fejlrapport, og vælger ved hjælp af objektfunktionen den mest sandsynlige. Derefter omsættes en eller flere forklaringer til en arbejdsordre og en tekniker sendes ud for at udbedre fejlen. For at kunne estimere parametrene i objektfunktionen på baggrund af empiriske data kræves det, at systemet udvides, så teknikeren melder tilbage med den præcise placering af fejlen, når arbejdet i marken afsluttes[14]. Det er denne rapport, sammenholdt med de alternative forklaringer OMS algoritmen beregner, der bruges som grundlag for indlæringsalgoritmen.



Figur 50: Tænkt eksempel, hvor der er forskel mellem arbejdsordren fra OMS (A) og den faktiske placering af fejlen (B).

Figur 50 viser et eksempel på uoverensstemmelse mellem den arbejdsordre der genereres og placeringen af den faktiske fejl. Med fejlrapporten $F=\{b, c\}$ vil OMS algoritmerne finde to mulige forklaringer, nemlig $S_1=\{(a, b),(a, c)\}$ (Figur 50 A) og $S_2=\{(ps, a)\}$ (Figur 50 B). Ved eksemplet ovenfor forestiller man sig, at α og β i objektfunktionen er indstillet således, at S_1 vælges og teknikeren derfor sendes ud for at efterse forbindelserne (a, b) og (a, c). Teknikeren konstaterer imidlertid, at forklaringen S_2 faktisk er den korrekte, og melder dette tilbage til systemet. Bemærk at den arbejdsordre der blev genereret på baggrund af S_1 ikke er nytteløs; selvom den ikke ramte den præcise

[14] Da DONG Energy i forvejen er forpligtet til at dokumentere forsyningssikkerheden, skal den præcise placering af alle fejl alligevel meldes tilbage til systemet, så der kan foretages en konsekvensberegning.

placering af fejlen, bliver teknikeren stadig sendt ud til et sted i den umiddelbare nærhed af fejlen. Den tilbagemelding systemet modtager kan beskrives således:

> *Givet fejlrapporten F={b, c} i det delnet der illustreres på* Figur 50, *skal α og β indstilles således, at S_2 er den mest sandsynlige forklaring, og arbejdsordren derfor baseres på denne forklaring.*

9.7.2 Simplificeret repræsentation af forklaringer

Som tidligere beskrevet i afsnittet Objektfunktionen, simplificeres hver forklaring, når den behandles af objektfunktionen, så det kun er $|S|$ og $|K(S)|$ dvs. antallet af defekte kanter og antallet af knuder uden strøm, der indgår i beregningen. I eksemplet fra Figur 50 vil de to mulige forklaringer derfor blive repræsenteret således:

	$\|S\|$	$\|K(S)\|$
S_1	2 (kanterne (*a*, *b*) og (*a*, *c*))	2 (knuderne *b* og *c*)
S_2	1 (kanten (*ps*, *a*))	3 (knuderne *a*, *b* og *c*)

Med en objektfunktion af formen

$$f(S) = \alpha \cdot |K(S)| + \beta \cdot |S| \qquad (30)$$

er problemet altså at bestemme alle kombinationer af α og β, således at $f(S_1) > f(S_2)$ hvorved forklaringen S_2 vil blive betragtet som den mest sandsynlige og dermed være den forklaring, der danner grundlag for arbejdsordren.

9.7.3 Konvergens

Der findes intet argument for, at værdierne af α og β vil konvergere mod faste værdier over tid. Derimod er ideen, at man ved at udnytte informationer om, hvor i nettet de reelle fejl opstår, kan konfigurere parametrene til, i flest mulige tilfælde at vælge den korrekte forklaring - eller i det mindste en forklaring der ligger tæt på den korrekte. I eksemplet fra Figur 50 er der f.eks. intet i vejen for, at der på et senere tidspunkt igen kan påstå en fejl i nettet med præcis samme fejlrapport, $F=\{b, c\}$, men hvor fejlen faktisk skyldes forklaringen S_1. Derved vil det datagrundlag der bruges til at estimere α og β indeholde to direkte modstridende rapporter, hvorfor α og β må bestemmes som en vægtning mellem alle rapporterne.

9.7.4 Trivielle og hårde problemer

Udover de direkte modstridende rapporter, kan der også opstå trivielle kombinationer af forklaringer, hvor alle valg af $\alpha, \beta \in \Re^+$ vil resultere i, at den ønskede forklaring vælges. Hvis den korrekte løsning betegnes S_k med den tilhørende konsekvens $K(S_k)$ og den fejlagtige forklaring betegnes S_f med konsekvensen $K(S_f)$ vil det trivielle tilfælde opstå, hvis det gælder at:

$$|S_k| \leq |S_f| \quad \wedge \quad |K(S_k)| \leq |K(S_f)| \qquad (31)$$

under forudsætning af, at der anvendes en objektfunktion som (30).

Da den mest sandsynlige forklaring findes ved at minimere objektfunktionen, kan der endvidere opstå hårde problemer, der aldrig kan løses som et minimeringsproblem, når det gælder at:

$$|S_k| \geq |S_f| \quad \wedge \quad |K(S_k)| \geq |K(S_f)| \tag{32}$$

Både de trivielle såvel som de hårde problemer udelades fra beregningen af α og β, da ingen af disse problemer bidrager med information om, hvilke værdier af α og β der er hensigtsmæssige.

9.7.5 Flere alternative forklaringer

Oftest vil en fejlrapport ikke kun resultere i to forklaringer, som det var tilfældet på Figur 50, men derimod typisk 3-10 mulige forklaringer, hvoraf kun én er den korrekte. Antallet af alternative forklaringer hænger nøje sammen med graden af grafen samt størrelsen af fejlrapporten, dvs. jo flere fejlmeldte knuder des flere forklaringer.

Modellen må derfor kunne behandle flere alternative forklaringer. Hvis man ser bort fra eksistensen af direkte modstridende forklaringer, som beskrevet ovenfor, kan problemet beskrives som et Constraint Satisfaction Problem (CSP), se afsnittet Konfigurationsproblemer. Dette CSP kan opstilles som vist i (33).

Bestem α og β under hensyntagen til begrænsningerne

$$\begin{aligned} &\alpha|K(S_k)| + \beta|S_k| \leq \alpha|K(S_{f1})| + \beta|S_{f1}| \\ &\alpha|K(S_k)| + \beta|S_k| \leq \alpha|K(S_{f2})| + \beta|S_{f2}| \\ &\qquad\vdots \\ &\alpha|K(S_k)| + \beta|S_k| \leq \alpha|K(S_{fn})| + \beta|S_{fn}| \end{aligned} \tag{33}$$

hvor S_{f1}-S_{fn} betegner de n alternative forklaringer, og $K(S_{f1})$-$K(S_{fn})$ betegner de respektive konsekvenser af disse alternative forklaringer. S_k og $K(S_k)$ betegner stadig hhv. den korrekte forklaring og konsekvensen af denne.

Hvis to af de alternative forklaringer viser sig at være modstridende, eller hvis blot en af de alternative forklaringer falder i kategorien af hårde problemer, vil der ikke eksistere nogen værdier af α og β der opfylder begrænsningerne i CSP problemet.

Da man således, i de fleste tilfælde, ikke kan forvente at der findes et sæt værdier til α og β der løser hele CSP problemet, er man nødt til at lede efter det sæt værdier, der overholder flest mulige begrænsninger - dvs. det sæt værdier der vælger flest af de forkerte forklaringer fra. Derfor brydes ovenstående CSP problem op i n mindre

problemer, der hver består af den korrekte forklaring og netop én alternativ forklaring. De nye CSP problemer har formen:

Bestem α og β under hensyntagen til begrænsningen

$$\alpha\left|K(S_k)\right| + \beta\left|S_k\right| \leq \alpha\left|K(S_f)\right| + \beta\left|S_f\right|, \qquad (34)$$

dvs. hver linie i det samlede CSP problem, (33), skilles ud som et selvstændigt problem. Ved hjælp af de kriterier der beskrives i formlerne (31) og (32) kan det hurtigt afgøres, om der for hver af de ovenstående *n* CSP problemer er tale om et hårdt eller et trivielt problem, hvorved den pågældende alternative forklaring blot kan ignoreres.

I alle andre tilfælde kan CSP problemet løses og intervaller med lovlige kombinationer af α og β kan bestemmes.

9.7.6 Typiske/atypiske fejl

Det er kun de mere typiske fejl der skal anvendes til indlæring, da meget atypiske fejl alligevel ikke vil kunne placeres korrekt af OMS algoritmerne. Hvis atypiske fejl medtages i indlæringen, vil de således kun være med til at forvrænge estimatet af α og β. Det er ikke altid muligt at afgøre maskinelt, om der er tale om en typisk fejl eller ej, hvorfor det overlades til teknikeren at melde tilbage til systemet, om den pågældende fejl kan betragtes som typisk.

9.7.7 Arraymodel

For lettere at kunne finde løsninger til uligheden (34) samples α og β i passende intervaller. Hvis man endvidere indfører en øvre grænse på $|S|$ og $|K(S)|$ opnås en diskret ulighed, der kan løses ved hjælp af arrayteknologi. $|S|$ og $|K(S)|$ er i deres natur heltallige og da man kun ønsker at anvende de typiske fejlscenarier til indlæring, er der ingen problemer i at indføre en øvre grænse på disse to variable - hvis én af værdierne overskrider grænsen er der ikke tale om et typisk fejlscenarium.

Den arraymodel der anvendes til at løse ulighed (34) er vist i Boks 12.

```
Variabletypes
  Parameter              : integer (1..10);
  ObjectiveValue         : integer (1..400);
  NodeCount              : integer (1..20);
  EdgeCount              : integer (1..20);

Variables
  GoodNodeCount          : input, NodeCount;
  GoodEdgeCount          : input, EdgeCount;
  BadNodeCount           : input, NodeCount;
  BadEdgeCount           : input, EdgeCount;
  BadObjectiveValue      : interim, ObjectiveValue;
  GoodObjectiveValue     : interim, ObjectiveValue;
  Alpha                  : output, Parameter;
  Beta                   : output, Parameter;

Relations
  GoodObjectiveValue =
     (GoodNodeCount*Alpha+GoodEdgeCount*Beta);
  BadObjectiveValue =
     (BadNodeCount*Alpha + BadEdgeCount*Beta);
  GoodObjectiveValue < BadObjectiveValue;
```

Boks 12: Arraymodel til at løse ulighed (34)

Modellen i Boks 12 anvendes ved at påtrykke alle fire inputparametre en værdi, hvorefter en derived relation[15] beregning over α og β giver alle lovlige kombinationer af disse parametre. Inputværdierne påtrykkes således:

- GoodNodeCount = $|K(S_k)|$
- GoodEdgeCount = $|S_k|$
- BadNodeCount = $|K(S_f)|$
- BadEdgeCount = $|S_f|$

jf. notationen fra ulighed (34).

Da man på forhånd benytter formel (32) til at sikre sig, at der ikke er tale om et hårdt problem, vil der altid findes mindst et par af α og β der overholder begrænsningerne.

[15] Se afsnittet Derived Relation.

9.7.8 Flere begrænsninger

Arraymodellen løser problemet med én begrænsning - se formel (34). For at finde den bedst mulige løsning til det kombinerede problem, formel (33), anvendes arraymodellen til at løse alle de delproblemer (33) kan splittes op i. Derved beregnes et antal lovlige kombinationer af α og β for hver af de alternative forklaringer.

Hvis der findes en eller flere kombinationer af α og β, der tilfredsstiller begrænsningerne for alle alternative forklaringer, er der fundet en løsning der tilfredsstiller det samlede CSP problem udtrykt ved (34). Der er dog som førnævnt stor sandsynlighed for, at en eller flere af begrænsningerne i (34) vil være i modstrid med hinanden, hvilket vil komme til udtryk ved, at der ikke er noget overlap mellem de værdier af α og β der løser to af delproblemerne. Man er derfor nødt til at indføre en vægtning, der udtrykker hvor mange af begrænsningerne i (34) et givet par af α og β tilfredsstiller. Det vælges at udtrykke vægtningen (eng. rating), af de enkelte værdipar, som en stokastisk størrelse, således at ratingen ligger mellem 0 og 1. *ratingen* for et værdipar findes således:

$$rating(\alpha,\beta) = \frac{Antal\ begrænsninger\ overholdt\ af\ \alpha\ og\ \beta}{Samlet\ antal\ begrænsninger\ i\ problemet}$$

Har man f.eks. *rating(1,2)* = 4/5 udtrykker det, at man ved at sætte α = 1 og β = 2 kan overholde 4 ud af 5 begrænsninger i et CSP problem med strukturen fra formel (33). Med andre ord kan man sige, at man fravælger 4 ud af de 5 forkerte forklaringer - den sidste forklaring kan ikke fravælges pga. modstrid i data.

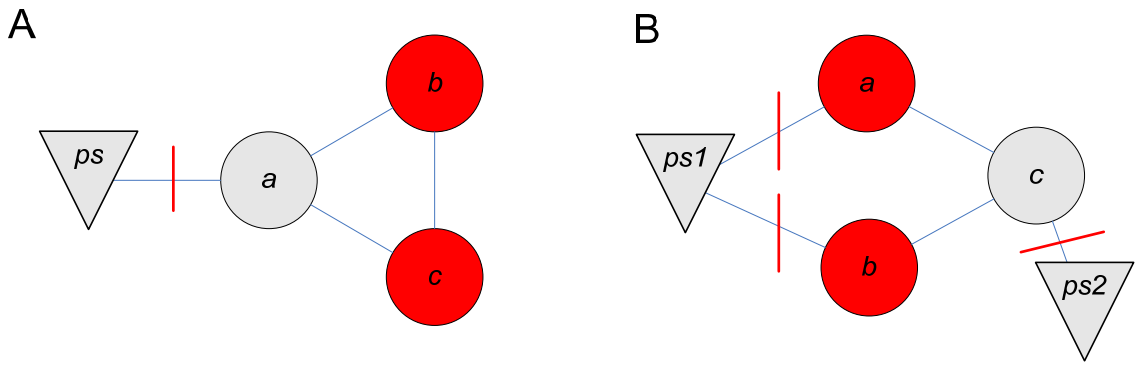
9.7.9 Flere fejlrapporter

Det er målet, at indlæringsalgoritmen ikke kun skal kunne håndtere en enkelt tilbagemelding fra en tekniker, men at man kan akkumulere en lang række tilbagemeldinger og på baggrund af alle disse bestemme det sæt af α og β, der fravælger flest mulige forkerte forklaringer.

Da den *rating* der beregnes for de enkelte α og β par for hver rapport er normaliseret, dvs. værdien altid ligger mellem 0 og 1, kan *ratings* over en række fejlrapporter direkte summeres og de α og β par der opnår den højeste *rating* er dem, der vil fravælge flest muligt forkerte forklaringer i alle de akkumulerede fejlrapporter. Havde man ikke valgt at normalisere *ratings*, ville rapporter med mange alternative forklaringer veje tungere i det samlede regnskab, end forklaringer med at lille antal alternative forklaringer. I denne algoritme vægtes alle fejlrapporter ens, men man kunne forestille sig, at man ud fra teknikerens angivelse af, hvor typisk denne fejl er, kunne lade *ratings* for de forskellige rapporter indgå i summen med forskellig vægt. På den måde kunne meget typiske fejl indgå i indlæringen med høj vægt, mens de mindre typiske fejl ikke har så meget indflydelse på estimatet af α og β.

9.7.10 Eksempel

På Figur 51 herunder er vist to forskellige fejlscenarier, der skal bruges til estimering af α og β. De forbindelser i nettet, der i teknikerens rapport er angivet som den reelle årsag til fejlene, er markeret med en rød streg på figuren.



Figur 51: To forskellige fejlscenarier. De røde knuder er fejlmeldt.

I fejlscenario A findes disse mulige forklaringer:

S	*|S|*	*K(S)*	*|K(S)|*
{(ps, a)}	1	*{a, b, c}*	3
{(a, b),(a, c)}	2	*{b, c}*	2

Den første forklaring, *S={(ps, a)}*, er af teknikeren rapporteret som den korrekte.

I fejlscenario B findes disse forklaringer:

S	*|S|*	*K(S)*	*|K(S)|*
{(ps1, a),(ps1, b),(a, c),(b, c)}	4	*{a, b}*	2
{(ps1,a),(ps1,b),(ps2,c)}	3	*{a, b, c}*	3

I dette tilfælde forestiller vi os, at teknikeren har rapporteret den sidste forklaring, *S={(ps1,a),(ps1,b),(ps2,c)}* som værende den korrekte.

Når tallene fra de to eksempler køres gennem indlæringsalgoritmen, findes en lang række værdier af α og β med en rating på 2, hvilket vil sige, at den korrekte løsning vælges i begge tilfælde. Der er altså ingen modstrid i de data der regnes på i eksemplet. Det første af de mulige værdipar det findes er $(\alpha, \beta) = (1, 2)$. For en komplet liste af alle de værdipar, der opfylder begrænsningerne i dette eksempel, overlades det til læseren at indsætte værdierne i prototypen og gennemføre beregningen.

9.8 Delkonklusion

I første del af dette afsnit blev tre OMS algoritmer til løsning af fejlfindingsproblemet beskrevet. Alle tre algoritmer finder sandsynlige forklaringer på en fejlrapport, forskellen består i køretiden. Den asymptotiske worst case køretid for de tre algoritmer afviger ikke meget, men den forventede køretid af EOMS er væsentligt bedre end de to andre, da den udnytter nogle af de specielle strukturer i grafen.

Hvor stor den faktiske forskel i ydeevne mellem de tre algoritmer er, vil blive belyst i afsnittet Benchmarking.

Dernæst er to forskellige metoder til gruppering af grafen opstillet. Først den meget fleksible men også forholdsvis langsomme algoritme Markov Clustering og derefter den mere begrænsede men meget hurtige Sektionering. De to algoritmer tænkes anvendt som en hybrid, så Sektionering først kan identificere delnet i det samlede elforsyningsnet, hvorefter Markov Clustering kan finde naturlige grupper af fejl i de enkelte delnet, for at generere arbejdsordrer.

Den sidste del af afsnittet, ser på en metode til indlæring af parametrene α og β på baggrund af tilbagemeldinger fra teknikerne i marken. Dette problem angribes som et konfigurationsproblem og løses derfor ved hjælp af array teknologi. Ved testkørsel på genererede data ser indlæring ud til at virke korrekt, men hvorvidt metoden konvergerer, kan ikke fastslås før den køres på data fra DONG Energy.

I næste afsnit gives en kort beskrivelse af, hvilke teknologier der er anvendt til at implementere algoritmerne fra dette afsnit.

10 Implementering

I dette afsnit vil vi beskrive, hvilke tanker vi har gjort os med hensyn til selve implementeringen. Der argumenteres for valg af udviklingsplatform samt udviklingsmodel. Afsnittet indeholder desuden en brugervejledning, der kort beskriver hvordan prototypen benyttes.

10.1 Valg af udviklingsplatform

På ethvert softwareprojekt skal man undersøge, hvilke udviklingsplatforme der er tilgængelige samt hvilke fordele og ulemper de har. Specielt bør man lægge vægt på:

- Udviklingsplatformens abstraktionsniveau
- De tilgængelige programmeringssprog
- Performance

Da det i rapporten beskrevne softwareprojekt er en prototype, er abstraktionsniveauet af særlig høj interesse. Under implementeringen af en prototype er man specielt interesseret i at afprøve forskellige hypoteser og specifikke implementeringer af disse.

I tidligere projekter har vi benyttet os af forskellige programmeringssprog, især C, C++ og C#. Derfor er det oplagt at vælge en udviklingsplatform, der understøtter et af netop disse sprog.

På baggrund af ovenstående punkter falder vores valg på Microsofts .NET 2.0 platform, da man her er på et højt abstraktionsniveau og har mange gængse datastrukturer til rådighed i .NET frameworket.

10.2 Udviklingsmodel

Under udviklingsforløbet har vi benyttet os af udviklingsmodellen Extreme Programming [BECK]. Grundprincippet i Extreme Programming er en iterativ proces. I første iteration identificeres de mest basale funktionaliteter, hvorefter disse funktionaliteter på kort tid implementeres i en prototype. I de efterfølgende iterationer opstilles der et nyt mål for implementeringen, igen et mål der på kort tid kan realiseres, hvorefter dette mål implementeres i prototypen.

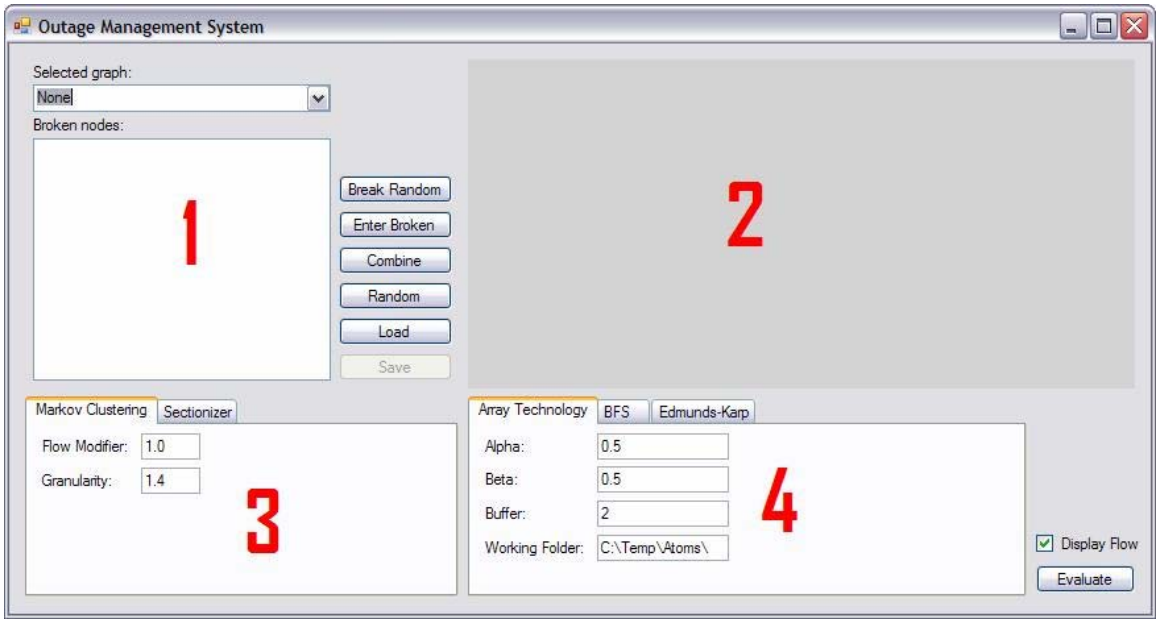
Extreme Programming står i stærk kontrast til vandfaldsmodellen, hvor alle detaljer planlægges og designes før de implementeres. I Extreme Programming designer man i et antal trin, iterationer, hvor hvert trin består af både analyse, design og implementering.

I Extreme Programming er det væsentligt, at en stor del af koden er dækket af Unit Tests; disse bør være både White Box og Black Box. Unit Tests benyttes i Extreme Programming til at verificere, at funktionalitet der er blevet tilføjet ikke får uhensigtsmæssige konsekvenser for den eksisterende kodebase.

På små softwareprojekter er Extreme Programming en stor fordel, da man undgår en unødvendigt beaukratisk forretningsgang. På store software projekter kan Extreme Programming til gengæld være en ulempe, da det kan være næsten umuligt at skabe sig et overblik over projektet.

10.3 Brugervejledning

Prototypen kan eksekveres direkte fra Cd'en ved at dobbeltklikke på filen **oms.exe**. Så snart prototypen er startet op ses hovedvinduet, Figur 52.



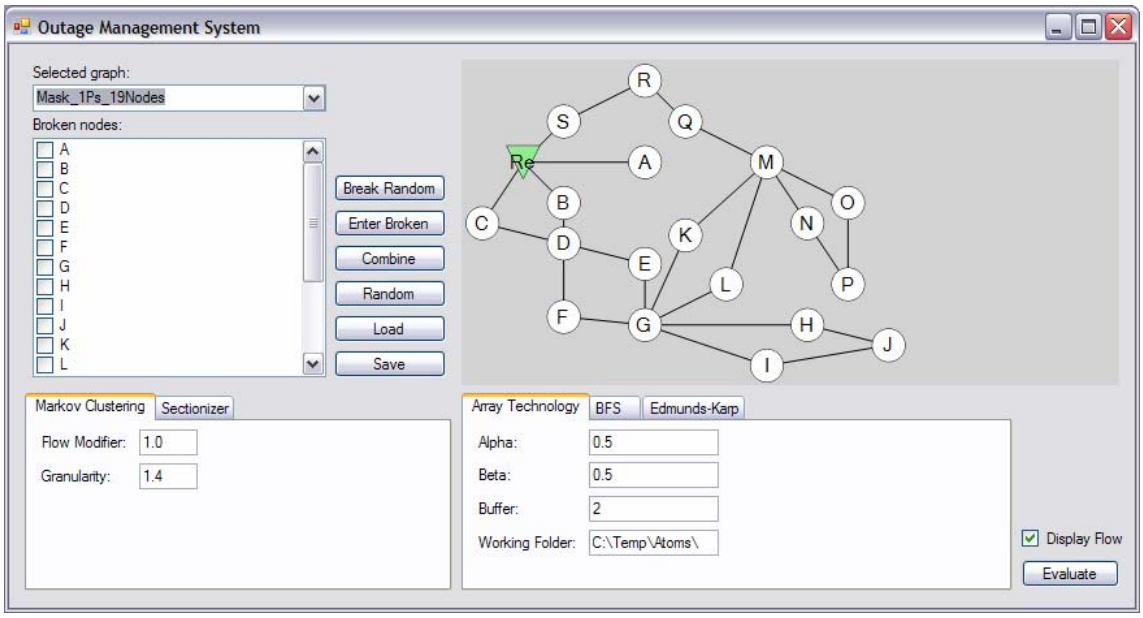
Figur 52: Prototypens hovedvindue.

Hovedvinduet består af fire dele:

1. Et værktøj til udvælgelse af grafen for et elforsyningsnet og markering af et antal knuder som fejlmeldte.
2. En grafisk repræsentation af grafen for det valgte elforsyningsnet.
3. Nederst i venstre hjørne vælges hvilken grupperingsalgoritme der skal anvendes.
4. Til løsning af selve fejlfindingsproblemet vælges, nederst i højre hjørne, en af de tre OMS algoritmer.

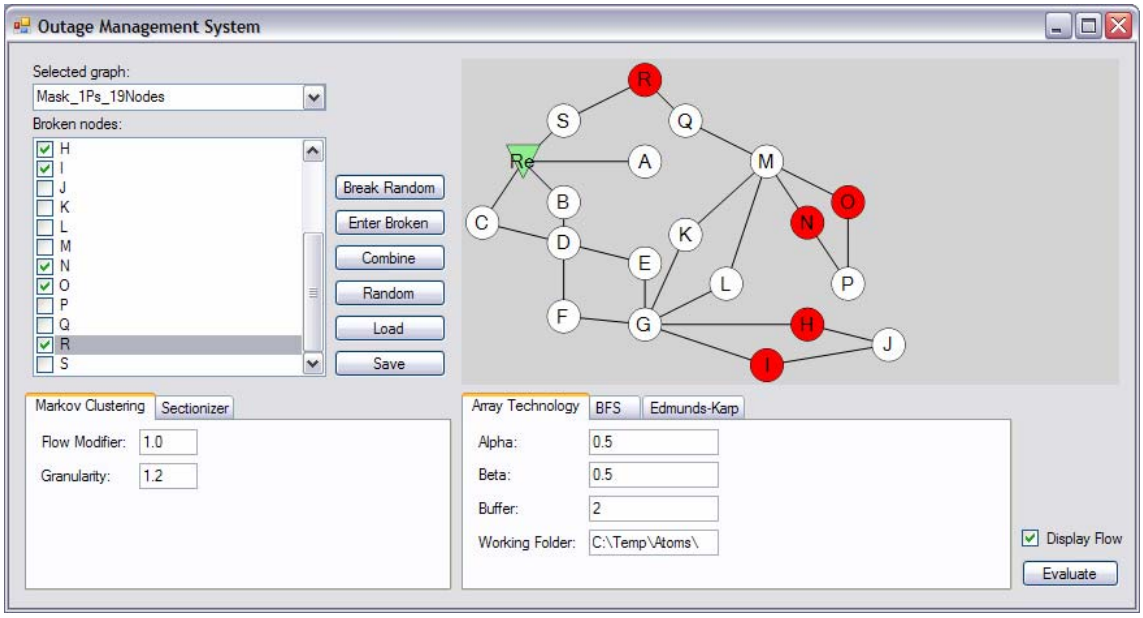
Det efterfølgende er en trinvis gennemgang af brugen af Outage Management System.

Første trin er at vælge grafen for et elforsyningsnet. Grafen vælges i rullegardinet under punkt 1, hvorefter den valgte grafs knuder bliver vist under rullegardinet. Desuden bliver en grafisk repræsentation af grafen vist under punkt 2, se Figur 53.



Figur 53: Grafen for et elforsyningsnet vælges i rullegardinet øverst i venstre hjørne.

I næste trin fejlmeldes et antal knuder ved at sætte flueben i boksen til venstre. På Figur 54 er knuderne *R*, *H*, *I*, *N* og *O* således fejlmeldt. Fejlmeldte knuder er markeret med rød i den grafiske repræsentation.

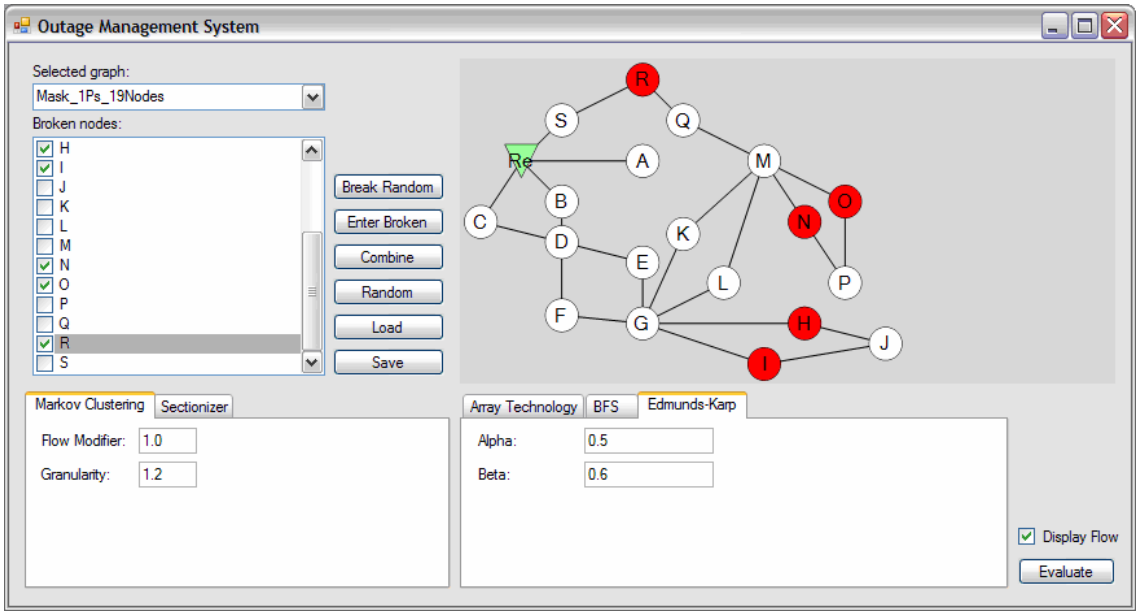


Figur 54: Fem knuder fejlmeldes: R, H, I, N og O.

På tredje trin vælges et grupperingsværktøj samt parametre for dette. Under punkt 3 (Figur 52) findes to grupperingsværktøjer: *Markov Clustering* og *Sectionizer*. På Figur 54 er Markov Clustering valgt og den dertil hørende parameter *Granularity* er angivet til 1,2.

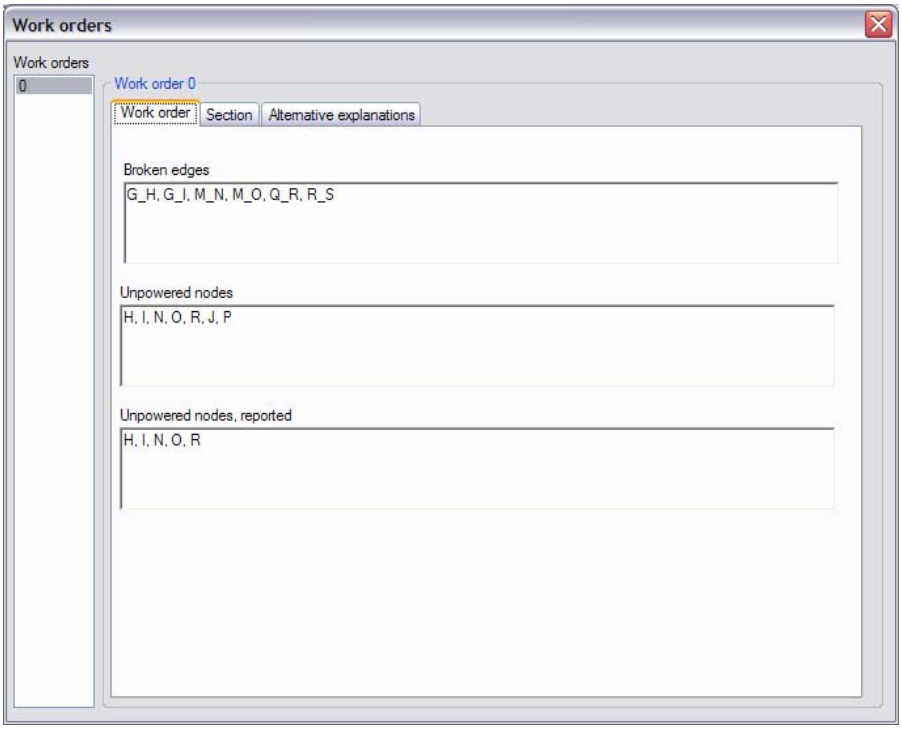
Fjerde trin er at vælge et fejlfindingsværktøj, dette gøres vha. fanebladene under punkt 4 på Figur 52. De tre tilgængelige værktøjer er: *Array Technology (ATOMS)*, *BFS (BOMS)*

og *Edmunds-Karp (EOMS)*. På Figur 55 er Edmunds-Karp valgt og de dertil hørende parametre er sat: $\alpha = 0{,}5$ og $\beta = 0{,}6$.



Figur 55: Valg af OMS algoritme.

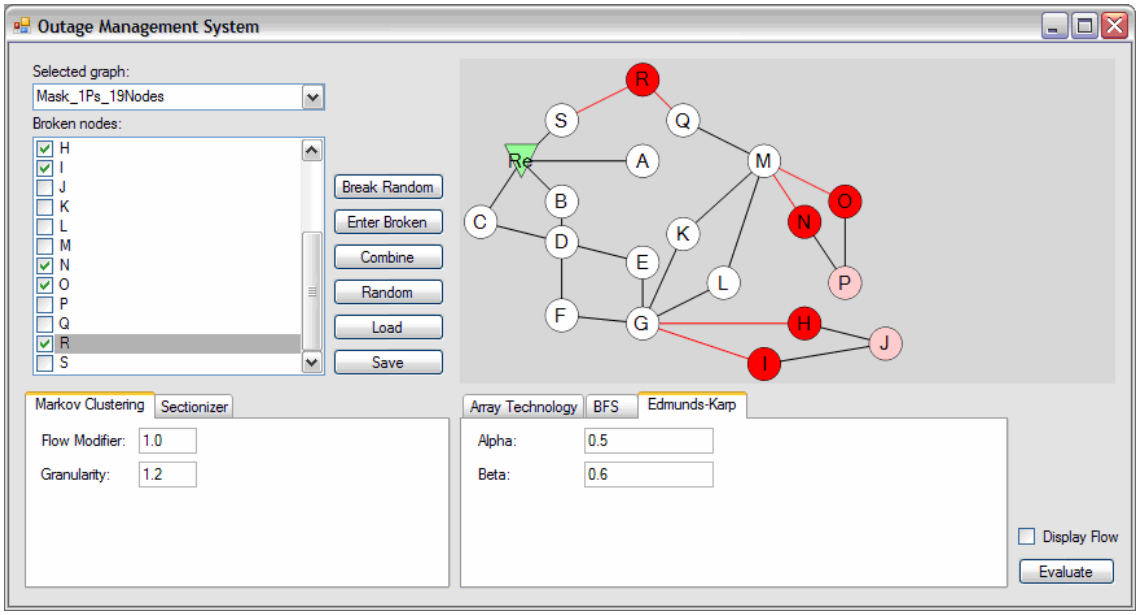
Det femte og sidste trin er at trykke på knappen *Evaluate*, hvorefter et antal arbejdsordre bliver genereret og vist i et nyt vindue, se Figur 56.



Figur 56: En genereret arbejdsordre.

En arbejdsordre viser den bedste forklaring på det første faneblad. Den bedste forklaring er et antal defekte kanter samt et antal afbrudte knuder. De to øvrige faneblade indeholder information om, hvilken gruppe denne arbejdsordre er tildelt, samt hvilke alternative forklaringer der er blevet vurderet.

Den grafiske repræsentation viser desuden den bedste forklaring som vist på Figur 57. Afbrudte knuder, der ikke oprindelig var fejlmeldt er markeret med lyserød, desuden er defekte kanter markeret med rød.



Figur 57: Den bedste forklaring bliver vist i den grafiske repræsentation.

I dette eksempel er den bedste forklaring at kanterne *GH*, *GI*, *MN*, *MO*, *QR* og *RS* er defekte. Knuder *J* og *P* er afbrudte udover de i forvejen fejlmeldte knuder.

Ved at kombinere de forskellige grupperings- og fejlfindingsværktøjer, samt dertil hørende parametre, kan man undersøge deres styrker og svagheder.

10.3.1 Liste over grafer

I rullegardinet under punkt 1 findes følgende grafer:

- Mask_1Ps_19Nodes – Et maskenet med 19 knuder og én strømforsyning.
- Mask_1Ps_50Nodes – Et maskenet med 50 knuder og én strømforsyning.
- Mask_1Ps_69Nodes – Et maskenet med 69 knuder og én strømforsyning.
- Mask_1Ps_100Nodes – Et maskenet med 100 knuder og én strømforsyning.
- Mask_2Ps_50Nodes – Et maskenet med 50 knuder og to strømforsyninger.
- Mask_2Ps_69Nodes – Et maskenet med 69 knuder og to strømforsyninger.
- Mask_2Ps_100Nodes – Et maskenet med 100 knuder og to strømforsyninger.
- Mask_3Ps_100Nodes – Et maskenet med 69 knuder og tre strømforsyninger.
- Radial_1Ps_19Nodes – Et radialnet med 19 knuder og én strømforsyning.

- Radial_1Ps_50Nodes – Et radialnet med 50 og én strømforsyning.
- Radial_1Ps_69Nodes – Et radialnet med 69 og én strømforsyning.
- Radial_1Ps_100Nodes – Et radialnet med 100 og én strømforsyning.

10.3.2 Generering af testdata

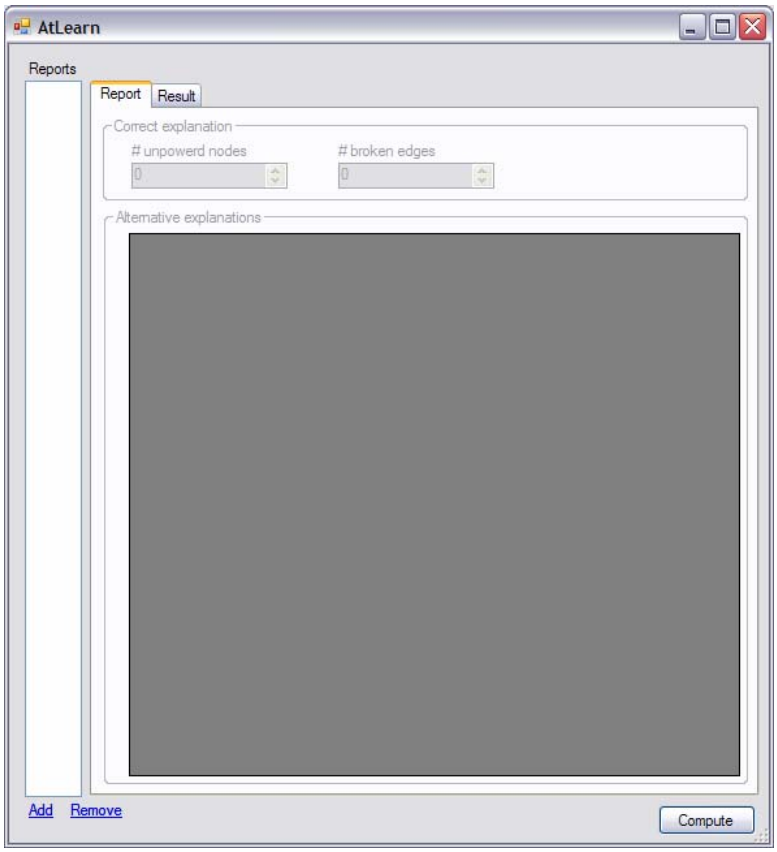
Med prototypen følger en lang liste af grafer, derudover er der rig mulighed for at konstruere egne grafer.

Ved hjælp at knappen **Combine** under punkt 1 er det muligt at kombinere de eksisterende grafer til en ny graf. Det er muligt at angive hvor mange grafer (delnet) der ønsket kombineret, samt en sandsynlighed for at et delnet er forbundet til dets nabo.

Knappen **Random** genererer en tilfældig graf. Det er muligt at angive et antal knuder og et antal strømforsyninger. Desuden kan den tilfældigt genererede grafs gennemsnitlige grad, og dermed antallet af kanter, indtastes.

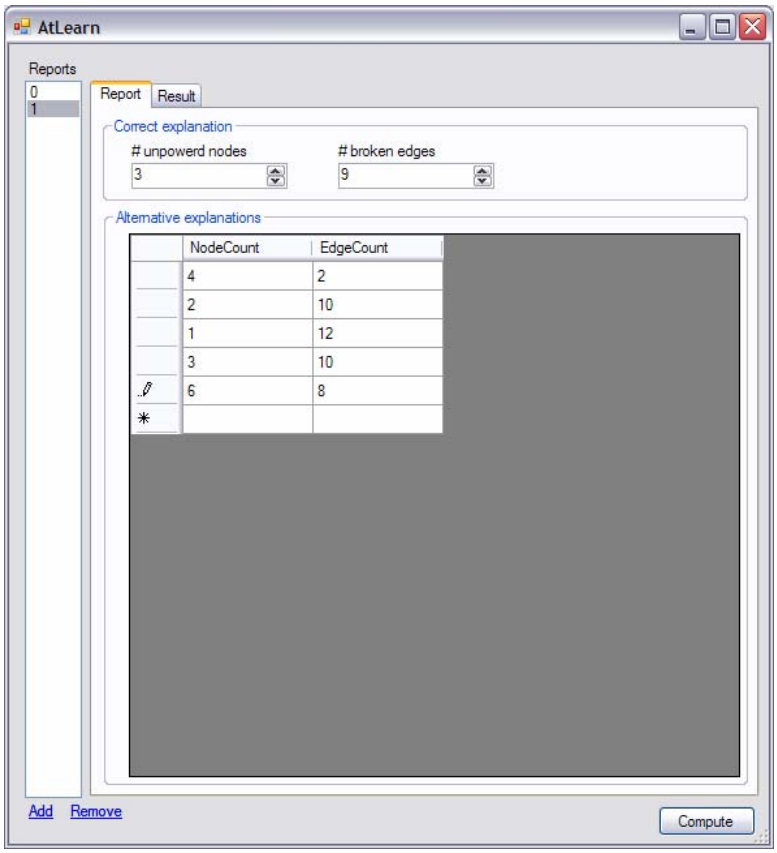
10.3.3 Brug af indlæring

Indlæringsmodulet kan startes fra hovedmenuen, ved at vælge **Tools -> Learning…** hvorved indlæringsmodulet toner frem på skærmen. Se Figur 58.



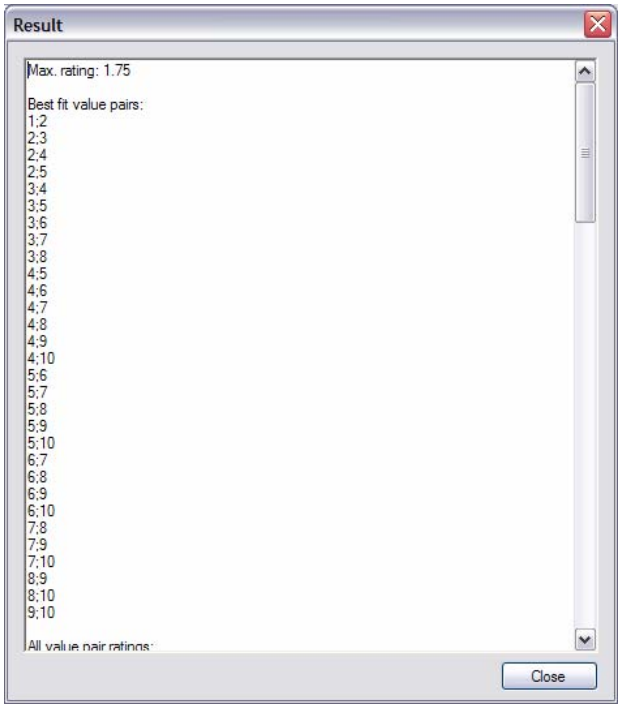
Figur 58: Indlæringsmodulet.

I indlæringsmodulet er det muligt at tilføje et vilkårligt antal fejlrapporter, til brug ved indlæringsprocessen. En ny fejlrapport tilføjes ved at klikke **Add** nederst i venstre hjørne på Figur 58. Hver rapport består af en korrekt forklaring, samt et antal alternative forklaringer. Nye forklaringer tilføjes ved at klikke på nederste række i tabellen. På Figur 59 er to fejlrapporter tilføjet til indlæringsmodulet.



Figur 59: To fejlrapporter er tilføjet til indlæringsmodulet.

Ved at klikke på fanebladet **Result** evalueres den aktive fejlrapport; af resultatet fremgår det hvilke værdipar af α og β der fravælger det største antal alternative forklaringer. Knappen **Compute** nederst i højre hjørne på Figur 59 evaluerer *alle* fejlrapporter og sammenholder resultaterne. Af denne evaluering fremgår det hvilke værdipar af α og β, der globalt set fravælger det største antal alternative forklaringer. Et eksempel på en sådan evaluering er illustreret på Figur 60.



Figur 60: Et eksempel på en global evaluering.

11 Benchmarking

I dette afsnit udføres konkrete målinger af ydeevnen af både OMS-algoritmerne og grupperingsalgoritmerne. Det vil blive undersøgt, hvilke egenskaber og strukturer i grafen, der påvirker køretiden for de enkelte algoritmer, og der vil blive undersøgt, hvor store elforsyningsnet hver af algoritmerne kan anvendes på.

Første del af afsnittet omhandler OMS-algoritmerne mens sidste del koncentrerer sig om gruppering.

Alle testdata i dette afsnit er genereret vha. den i afsnittet Implementering gennemgåede funktion til generering af tilfældige grafer.

11.1 Benchmarking af BOMS

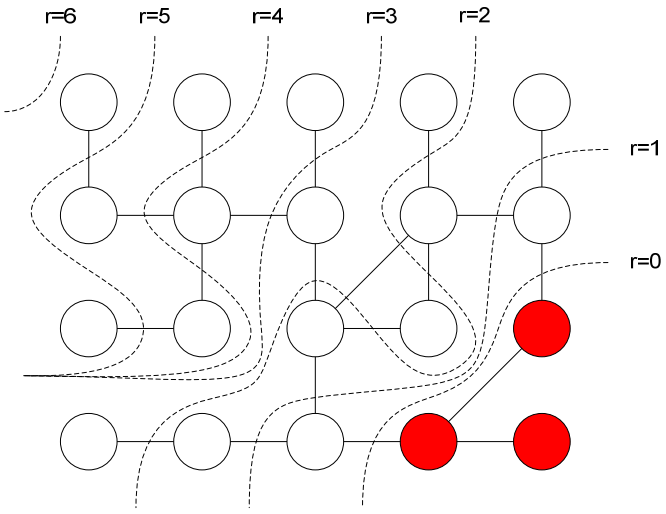
I afsnittet BOMS har vi gennemgået hvordan vi ved hjælp af almindelig grafteori og i særdeleshed Breadth First Search (BFS) algoritmen kan løse fejlfindingsproblemet.

En egenskab ved BOMS er hvorledes vi, efter at have fundet den umiddelbare forklaring S_U, evaluerer alternative forklaringer der befinder sig nær den umiddelbare forklaring. Disse alternative forklaringer bliver beregnet ved at udvælge knuder der befinder sig inden for en fast defineret afstand fra de fejlmeldte knuder. Alle kombinationer af de udvalgte knuder, sammenholdt med fejlrapporten, udgør de alternative forklaringer.

Udvælgelsen af knuder til alternative forklaringer foregår på følgende måde. De fejlramte knuder, *F*, benyttes som kilder i BFS til at finde alle knuder, *W*, der ligger maksimalt *r* kanter væk fra mindst en knude i *F*. Det gælder da at $|W| \leq |V| - |F|$. I praksis er det ofte et fåtal af det samlede antal knuder der er fejlramt, dvs. $|F| << |V|$. Dette betyder at W potentielt kan være næsten alle knuder i grafen. I forbindelse med disse test er det derfor rimeligt at antage $|W| \approx |V|$ i værste fald. Den faktiske størrelse af *W* afhænger af to ting:

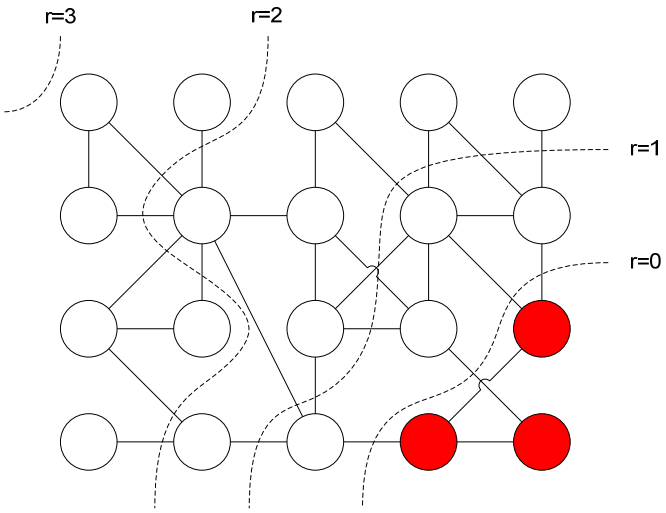
1. $|W|$ afhænger af *r*, det maksimale antal kanter en udvalgt knude $w \in W$ må ligge fra en fejlramt knude $f \in F$.
2. $|W|$ afhænger af grafens struktur og især grafens *grad*.

Figur 61 viser hvorledes størrelsen af W afhænger af *r*.



Figur 61: Udvælgelsen af knuders afhængighed af den maksimale afstand, *r*, fra en fejlramt knude.

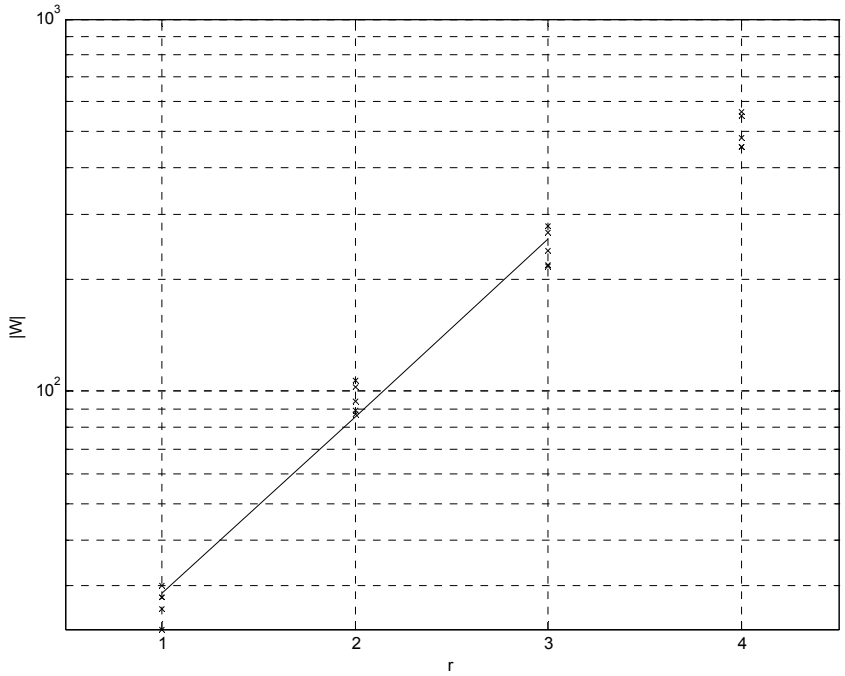
Størrelsen af *W* afhænger også af grafens *grad*, dvs. det gennemsnitlige antal kanter fra hver knude. Grafen på Figur 62 er af højere grad end grafen på Figur 61. Sammenholdes størrelsen af *W* for ens værdier af *r* på de to grafer, ses det at størrelsen af *W* for en given værdi af *r* er betydeligt større på Figur 62.



Figur 62: Udvælgelsen af knuders afhængighed af grafens grad.

11.1.1 Afhængighed af *r*

Ved at generere et antal testgrafer, hver med 1000 knuder og et gennemsnit på tre kanter mellem hver knude, dvs. *grad* tre, kan man vise at $|W|$ afhænger af *r*. I hver graf vælges 10 knuder tilfældigt til at være fejlmeldte, hvorefter antallet af udvalgte knuder findes for forskellige værdier af *r*. Resultatet kan ses på Figur 63. Det fuldstændige datasæt kan desuden ses i Appendiks 3.



Figur 63: Antallet af udvalgte knuder som funktion af en fast defineret afstand.

På Figur 63 er antallet af udvalgte knuder indtegnet som funktion af en fast defineret afstand til en fejlramt knude i et logaritmisk koordinatsystem. Det ses tydeligt, at udvælgelsen af knuder afhænger af den maksimale afstand disse knuder må befinde sig fra en fejlramt knude, og at denne afhængighed er eksponentiel. Et bedste fit er desuden indtegnet på Figur 63 for de første tre målepunkter.

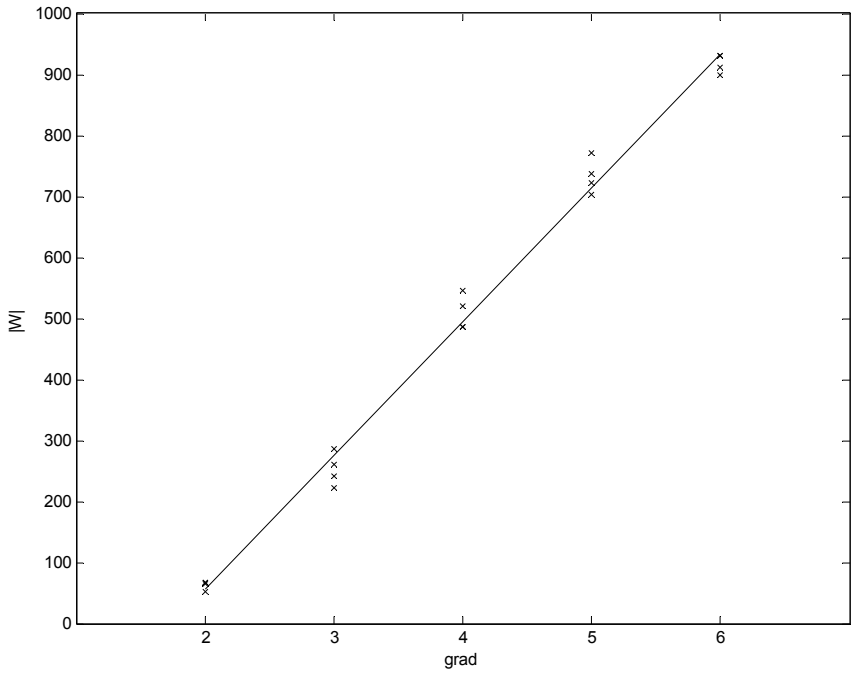
Grunden til at målepunkterne synes at bøje af og det sidste målepunkt ikke er medtaget til beregning af et bedste fit er følgende: Udvælgelsen af knuder foregår vha. BFS og da grafen kun har en endelig størrelse, vil man når r vokser "støde ind i kanten" af grafen, jævnfør Figur 61. Antallet af udvalgte knuder vokser altså eksponentielt, så længe man ikke rammer grafens ydre grænse. Derfor baseres generaliseringen kun på de første målepunkter.

Testdata viser desuden at $|W|$ er cirka $2/3$ af $|V|$ allerede for $r = 4$ i disse tilfældigt genererede grafer. Det maksimale antal kanter en udvalgt knude må ligge fra en afbrudt knude skal altså ikke være særlig højt, før man udvælger samtlige knuder i grafen. Samtidig er det muligt at konstruere grafer, hvor dette ikke er tilfældet, eksempelvis vil et maskenet med store loops ikke have denne egenskab.

11.1.2 Afhængighed af *grad*

Den anden egenskab, nemlig at størrelsen af W afhænger af grafens struktur, og i særdeleshed grafens *grad*, vises ved at generere et nyt sæt testgrafer. Igen har hver graf 1000 knuder men det gennemsnitlige antal kanter mellem hver knude varierer fra to til seks. Vi benytter denne gang en konstant værdi for $r = 3$. Som i foregående test vælges

10 knuder tilfældigt til at være fejlmeldt. Resultatet kan ses på Figur 64. Det fuldstændige datasæt kan ses i Appendiks 4.



Figur 64: Antallet af udvalgte knuder som funktion af grafens grad.

Ud fra grafen på Figur 64 ser der ud til at være en lineær sammenhæng mellem antallet af udvalgte knuder og grafens grad. Denne sammenhæng kan forklares ud fra at antallet af naboer til én fejlmeldt knude stiger lineært med grafens *grad*.

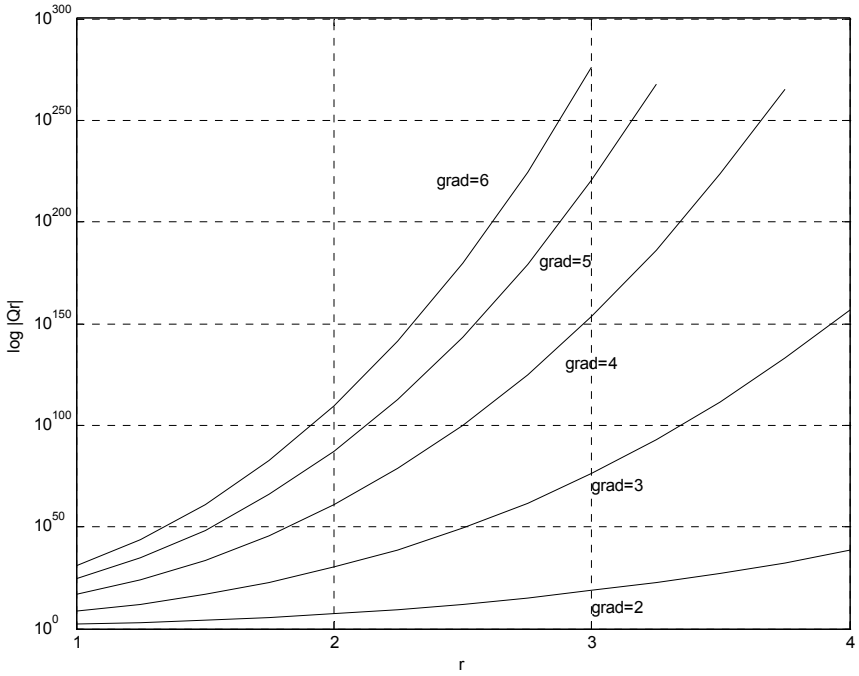
11.1.3 Alternative forklaringer

I de to foregående afsnit har vi gennemgået hvilke faktorer der har indflydelse på antallet af udvalgte knuder, $|W|$. I dette afsnit vil vi undersøge hvordan antallet af alternative forklaringer afhænger af $|W|$.

Når et antal knuder er udvalgt til at danne basis for de alternative forklaringer, er det næste skridt at danne disse alternative forklaringer. Dette gøres ved at udregne alle permutationer af de udvalgte knuder og kombinere hver enkelt permutation med de fejlramte knuder. Hver enkelt af disse kombinationer danner basis for en ny forklaring. Disse forklaringer evalueres i forhold til objektfunktionen, for at finde den bedste forklaring.

Vi generer et antal testgrafer, hver med 100 knuder og et gennemsnitligt antal kanter fra hver knude på to. Ved at vælge værdier for *r* og derefter vælge tre knuder tilfældigt der markeres som fejlmeldte, kan vi finde $|W|$. Knuderne i W benyttes nu sammen med

knuderne i F til at danne mængden af alternative forklaringer Q_r. Det fuldstændige datasæt kan ses i Appendiks 5.



Figur 65: Logaritmen til antallet af alternative forklaringer som funktion af en fast defineret afstand til en fejlmeldt knude, *r*.

Figur 65 viser logaritmen til antallet af alternative forklaringer som funktion af r for grafer af forskellig *grad*. Antallet af alternative forklaringer er beregnet som samtlige permutationer af knuderne i W. Altså $|Q_r| = 2^{|W|}$. Som grafen tydeligt viser, eksploderer antallet af alternative forklaringer når en højere værdi for r vælges, eksempelvis findes der ca. 67 mill. forklaringer for $r = 1$, $grad = 3$, blot i dette lille net. Dette er en meget uheldig egenskab ved BOMS.

11.2 Benchmarking af ATOMS

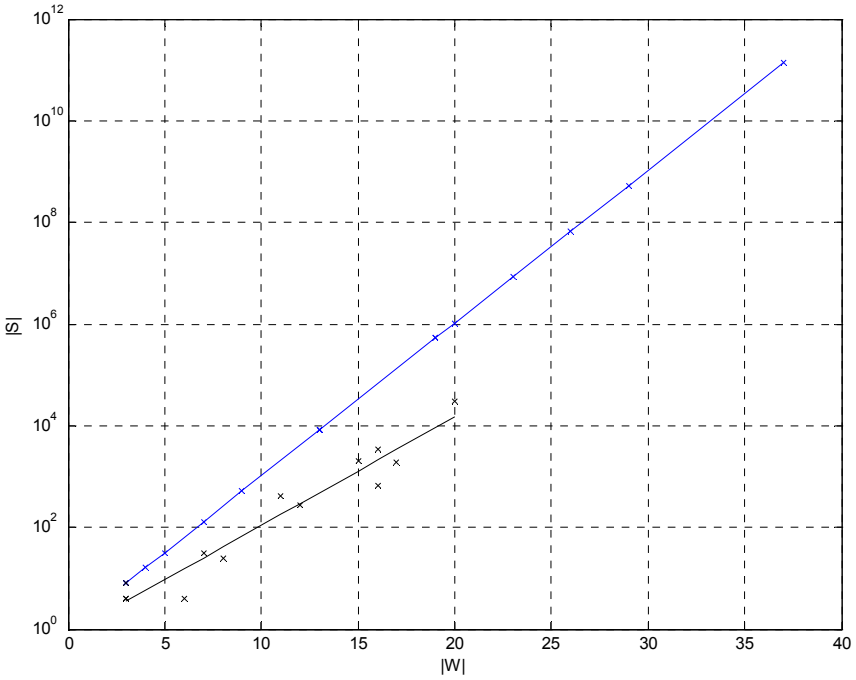
I foregående afsnit har vi gennemgået hvordan fejlfindingsproblemet kan løses ved hjælp af BOMS algoritmen. BOMS algoritmen virker et langt stykke hen af vejen godt, men vi har i foregående afsnit vist, at BOMS algoritmen evaluerer et antal alternativer forklaringer der afhænger eksponentielt af antallet af udvalgte knuder, der igen afhænger eksponentielt af den fast definerede afstand til en fejlmeldt knude, *r*.

Det overordnede flow i ATOMS algoritmen er meget lig det i BOMS algoritmen. ATOMS algoritmen benytter sig af arraymotoren til at bestemme den umiddelbare løsning S_U hvor BOMS algoritmen benytter sig af BFS søgninger. Desuden bestemmer ATOMS også alternative forklaringer vha. derived relations i arraymotoren.

11.2.1 Antallet af alternative forklaringer

Den store forskel på BOMS algoritmen og ATOMS algoritmen er den måde hvorpå alternative forklaringer bliver evalueret. ATOMS algoritmen benytter arraymotoren til at bestemme alle *lovlige* kombinationer af udvalgte knuder, hvorefter disse kombinationer bliver evalueret med hensyn til objektfunktionen. BOMS algoritmens store svaghed er, at den ukritisk evaluerer samtlige permutationer af udvalgte knuder. Derfor er ATOMS algoritmen i denne sammenhæng en væsentlig forbedring.

For at underbygge denne påstand, benytter vi de samme testgrafer som ved undersøgelse af antallet af alternative forklaringer for BOMS algoritmen. Hver testgraf har 100 knuder og et gennemsnitligt antal kanter fra hver knude på to. Nøjagtig som ved tidligere undersøgelse vælger vi værdier for *r* og måler derefter sammenhørende værdier af $|W|$ og $|Q|$. Det fuldstændige datasæt kan ses i Appendiks 6 og resultatet er sammenholdt med resultatet for BOMS algoritmen på Figur 66.



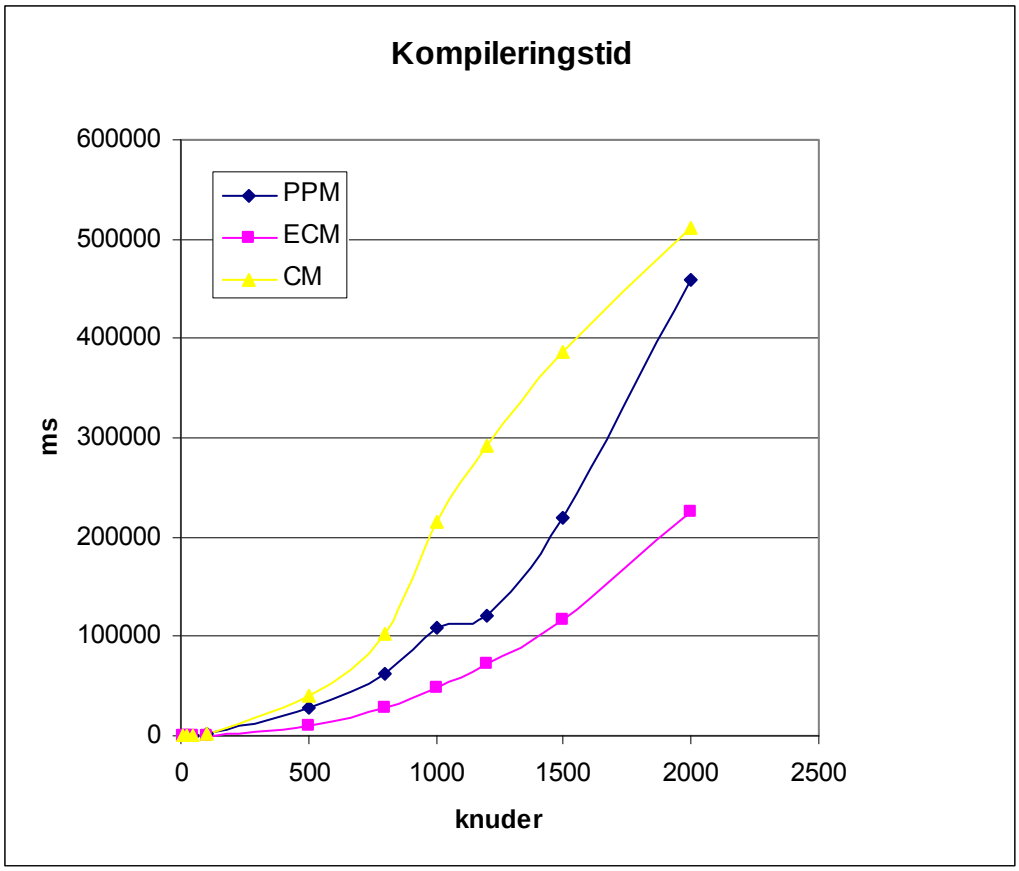
Figur 66: Logaritmen til antallet af alternative forklaringer som funktion af antallet af udvalgte knuder for ATOMS (sort.) Resultatet for BOMS er vist som reference (blå.)

Som Figur 66 illustrerer, afhænger antallet af alternative forklaringer stadig eksponentielt af antallet af udvalgte knuder. ATOMS algoritmen forbedrer BOMS algoritmen med en konstant faktor. Det bør understreges, at selvom forbedringen ved brug at ATOMS algoritmen er en konstant faktor, er der stadig tale om en væsentlig forbedring. Eksempelvis evaluerer ATOMS algoritmen 29.000 alternative forklaringer for $|W| = 20$ mens BOMS algoritmen evaluerer over 100.000 alternative forklaringer.

11.2.2 Kompileringstid

Udover antallet af alternative forklaringer, ATOMS algoritmen skal evaluere, er en anden begrænsning størrelsen af de modeller arraymotoren kan kompilere.

For at undersøge hvor store modeller arraymotoren kan kompilere, har vi målt gennemsnitlige værdier af kompileringstiden for de arraymodeller ATOMS algoritmen benytter sig af, hhv. PPM, ECM og CM. Resultatet af disse målinger kan ses på Figur 67. Hver køretid er beregnet som et gennemsnit af kompileringstiden for tre forskellige grafer med det samme antal knuder. Datasættet kan ses i Appendiks 7.



Figur 67: Kompileringstiden (ms) for de tre arraymodeller ATOMS algoritmen benytter sig af som funktion af antallet af knuder.

Grafen på Figur 67 illustrerer flere interessante egenskaber ved arraykompileren. Kompileringen af PPM synes at knække efter 1000 knuder, hvilket muligvis kan forklares ved at arraymotoren aktiverer bestemte optimeringsmetoder når modellen når en hvis størrelse. Da der er benyttet gennemsnitlige værdier, til at generere graferne på Figur 67, er det usandsynligt, at der blot er tale om et uheldigt gennemløb.

Alle tre grafer synes at være en eksponentiel funktion af antallet af knuder for knudeantal mellem 0 og 1000. Ved knudeantal over 1000 knækker PPM og CM graferne, CM grafen knækker dog ikke så voldsomt som PPM. For knudetal over 1500 synes PPM og ECM graferne igen at være eksponentialfunktioner, men det er usikkert, hvorvidt dette også gælder for CM grafen.

Andre test har vist, at grænsen for kompilering af arraymodellerne går ved ca. 3000 knuder. For større modeller løber arraykompileren tør for systemresurser. Da det absolut største delnet i DONG Energys elforsyningsnet er på ca. 3000 knuder, vil ATOMS algoritmen godt kunne anvendes på det faktiske elforsyningsnet, det vil dog kræve, at arraymodellerne kan kompileres på forhånd og lagres i binær form.

11.2.3 ATOMS algoritmens begrænsninger

ATOMS algoritmen er en klar forbedring af BOMS algoritmen, men har dog begrænsninger. Det vil ikke være muligt at kompilere hele DONG Energys elforsyningsnet med 400.000 knuder. ATOMS algoritmen kan derfor kun løse fejlfindingsproblemet i de isolerede delnet, og kan ikke benyttes til fejlfinding på kryds af delnet.

Ligesom BOMS algoritmen er ATOMS algoritmen stærkt afhængig af det maksimale antal kanter en udvalgt knude må ligge fra en fejlramt knude, r, ved udvælgelsen af knuder til alternative forklaringer. Både BOMS algoritmen og ATOMS algoritmen er afhængige af grafens grad.

11.3 Benchmarking af EOMS

I de foregående afsnit har vi gennemgået BOMS- og ATOMS algoritmerne. Disse algoritmer er begrænset af antallet af knuder, der udvælges til at evaluere alternative forklaringer. EOMS algoritmens virkemåde er fundamentalt anderledes. I stedet for at udvælge knuder og kombinere disse med knuderne i den umiddelbare forklaring for at finde alternative forklaringer, erstatter EOMS algoritmen blot en kant i den først fundne forklaring for at finde en alternativ forklaring. Med denne tilgang til problemet kan antallet af alternative forklaringer, der skal evalueres, holdes konstant, hvilket er en dramatisk forbedring af BOMS- og ATOMS algoritmens opførsel.

Som beskrevet i afsnittet Algoritmer finder EOMS algoritmen en god forklaring i $O(E^2)$ hvor E er antallet af kanter i den pågældende graf. I praksis er EOMS algoritmen væsentlig hurtigere end $O(E^2)$. Vi vil i dette afsnit undersøge, hvilke faktorer der spiller ind i EOMS algoritmens faktiske køretid.

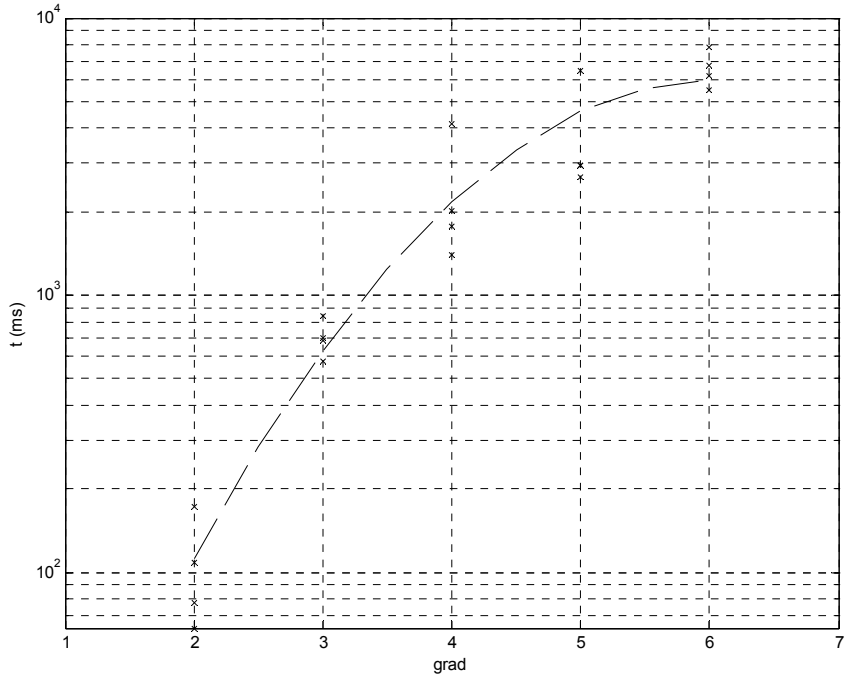
Grundet EOMS algoritmens virkemåde er der to faktorer, man må forvente har indflydelse på algoritmens køretid.

1. Grafens *grad* har indflydelse på antallet af kanter i det mindste snit. Det mindste snit kan i princippet være alle kanter i grafen, E, men i konkrete elforsyningsnet vil det være langt mindre.
2. Den gennemsnitlige afstand fra en fejlmeldt knude til en strømkilde har indflydelse på, hvor mange niveauer EOMS algoritmen skal gennemsøge før den finder en sti fra en fejlmeldt knude til en strømkilde.

I de to efterfølgende afsnit vil vi ved hjælp af konkrete testeksempler efterprøve effekten af disse faktorer.

11.3.1 Afhængighed af grad

For at illustrere EOMS algoritmens afhængighed af grafens *grad*, måler vi EOMS algoritmens køretid på et antal testgrafer af forskellig *grad*. Grafsættet er det samme som ved undersøgelse af BOMS algoritmens afhængighed af grafens *grad*. Hver graf har 1000 knuder og et gennemsnitligt antal kanter mellem hver knude fra to til seks. 10 knuder udvælges tilfældigt hvorefter en løsning findes med EOMS algoritmen. Det fuldstændige datasæt kan ses i Appendiks 8. Resultatet kan ses på Figur 68.



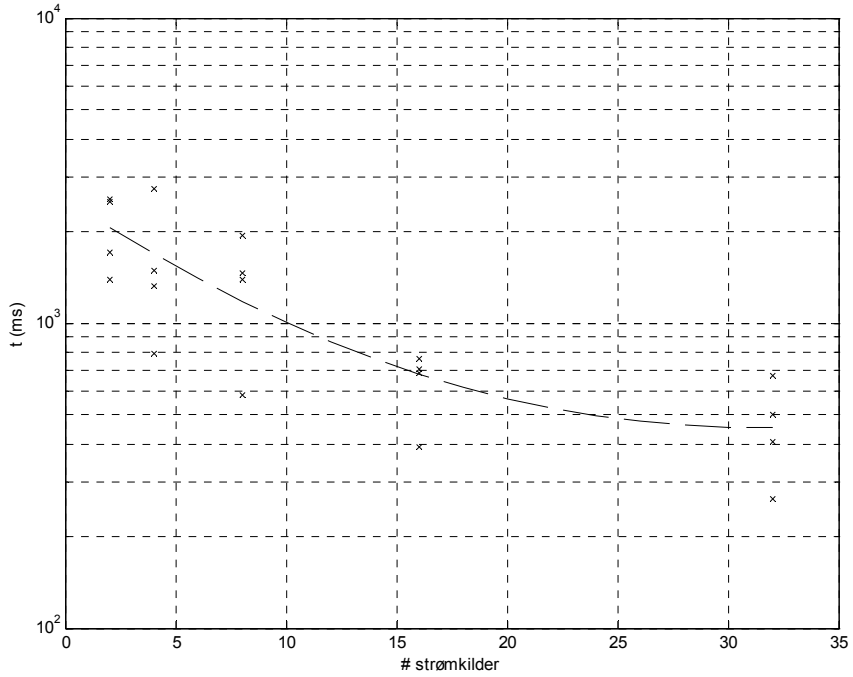
Figur 68: Grafen viser EOMS algoritmens køretid i ms som funktion af grafens grad.

Grafen på Figur 68 viser, at køretiden af EOMS algoritmen stiger når grafens *grad* forøges. På baggrund af de bagved liggende data er det ikke muligt, entydigt at bestemme køretiden af EOMS algoritmens afhængighed af grafens *grad*, men en eksponentiel funktion er indtegnet på figuren for at antyde tendensen. Afhængigheden kunne lige så vel være et polynomium af høj grad.

Det væsentlige i denne sammenhæng, er at *graden* af det faktiske elforsyningsnettet højst sandsynlig befinder sig et sted mellem 2 og 4, derfor er grafens *grad* ikke en hindring for EOMS algoritmen.

11.3.2 Afstand til strømkilde

For at undersøge, om den gennemsnitlige afstand fra en fejlbehæftet knude til en strømkilde har indflydelse på køretiden af EOMS algoritmen, laver vi endnu et eksperiment. Vi konstruerer et nyt grafsæt med 1000 knuder. Graferne i sættet er alle af grad 3, men antallet af strømkilder varierer fra hhv. 2 til 32. Det fuldstændige datasæt kan ses i Appendiks 9. 10 knuder udvælges tilfældigt og markeres som fejlmeldte, hvorefter en forklaring findes ved hjælp af EOMS algoritmen. Den tid EOMS algoritmen forbruger noteres. Resultatet kan ses på Figur 69.



Figur 69: EOMS algoritmens køretid i ms som funktion af antallet af strømkilder i grafen.

Grafen på Figur 69 viser, at køretiden af EOMS algoritmen som forventet falder med antallet af strømkilder i grafen. Dette synes meget fornuftigt, da et stort antal strømkilder nødvendigvis betyder, at EOMS algoritmen gennemsøger færre niveauer før den finder en strømkilde og derfor terminerer. En svagt eksponentiel funktion er indtegnet på grafen for at antyde tendensen.

Slutteligt har vi testet EOMS på meget store problemer, for at undersøge om algoritmen kan bruges på det samlede elforsyningsnet. Til simulering af stormtilstanden er fejlfindingsproblemet løst på en graf af *grad* 3 med 343.000 knuder, hvor 400 knuder er markeret som fejlmeldte. Denne situation kan EOMS gennemregne på ca. 10 minutter. Til simulering af normaltilstanden er der konstrueret en graf med i alt 540.000 knuder fordelt på 8000 delnet og med 4 fejlmeldte knuder. For at løse fejlfindingsproblemet på denne tilstand kræver EOMS kun 11 sekunder. EOMS er altså, som den eneste af de tre OMS-algoritmer vi har undersøgt, i stand til at løse fejlfindingsproblemet på det samlede elforsyningsnet.

11.4 Delkonklusion: Benchmarking af fejlfindingsalgoritmer

Her i benchmarking afsnittet er det blevet gennemgået, hvilke parametre der påvirker køretiden for de tre OMS algoritmer. Kort sagt er ATOMS og BOMS meget påvirket af grafens grad samt hvor stor afstand der skal søges efter alternative forklaringer i. EOMS er i høj grad påvirket af afstanden til nærmeste strømkilde samt grafens grad.

Alle OMS algoritmerne er meget følsomme overfor antallet af fejl i nettet samt nettets størrelse. Det viser sig, ikke uventet, at EOMS er den af algoritmerne der har langt den bedste ydeevne. Faktisk er forskellen i ydeevne så stor, at det ikke kan illustreres fornuftigt i en graf. I Tabel 6 ses en sammenligning af køretiderne for ATOMS/BOMS og EOMS på en graf med 100 knuder. Det ses at EOMS påvirkes svagt af antallet af fejl, mens ATOMS/BOMS stiger meget voldsomt. ATOMS og BOMS er slået sammen til et datasæt, da deres køretider i dette eksempel ligger meget tæt på hinanden.

Antal fejlmeldte knuder	ATOMS/ BOMS	EOMS
1	0,031	0,016
2	4,094	0,016
3	120,000	0,031
4	n/a	0,031
5	n/a	0,047
6	n/a	0,047
7	n/a	0,062
8	n/a	0,094
9	n/a	0,094
10	n/a	0,094

Tabel 6: Køretider for ATOMS/BOMS og EOMS målt i sekunder.

Resultaterne i Tabel 6 betyder, at EOMS er den eneste af OMS-algoritmerne der vil virke i stormtilstanden, hvor mange knuder i nettet er fejlmeldte. I den almindelige tilstand, hvor få knuder i hvert delnet er fejlmeldt, vil man kunne anvende en hvilken som helst af de tre OMS-algoritmer.

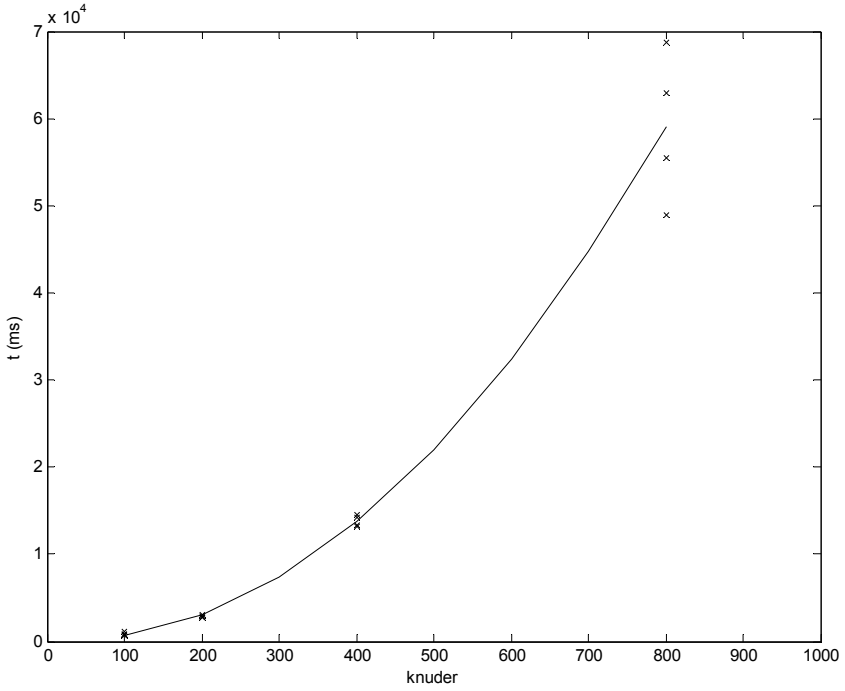
11.5 Benchmarking af Markov Clustering og Sektionering

Til gruppering af grafen og delnet har vi benyttet os af Markov Clustering og Sektionering. I dette afsnit vil vi ved hjælp af konkrete testresultater sammenligne de to algoritmer. Datasættet der er benyttet i dette afsnit kan ses i Appendiks 10.

11.5.1 Markov Clustering

I afsnittet Algoritmer har vi beskrevet hvordan ydelsen af Markov Clustering afhænger af antallet af knuder i grafen. I Appendiks 2: Matrix multiplikation vises det at den asymptotiske køretid for Markov Clustering er $O\left(V^2 \cdot k\right)$ hvor n er antallet af knuder i grafen og k er det maksimale antal ikke-nul-elementer i en enkelt række eller søjle i matrixrepræsentationen af grafen.

For at undersøge ydelsen af Markov Clustering måler vi algoritmens køretid på et antal testgrafer med varierende antal knuder. På hver graf er 10 knuder markeret som fejlmeldt. Grafen på Figur 70 viser algoritmens køretid som funktion af antallet af knuder i grafen.



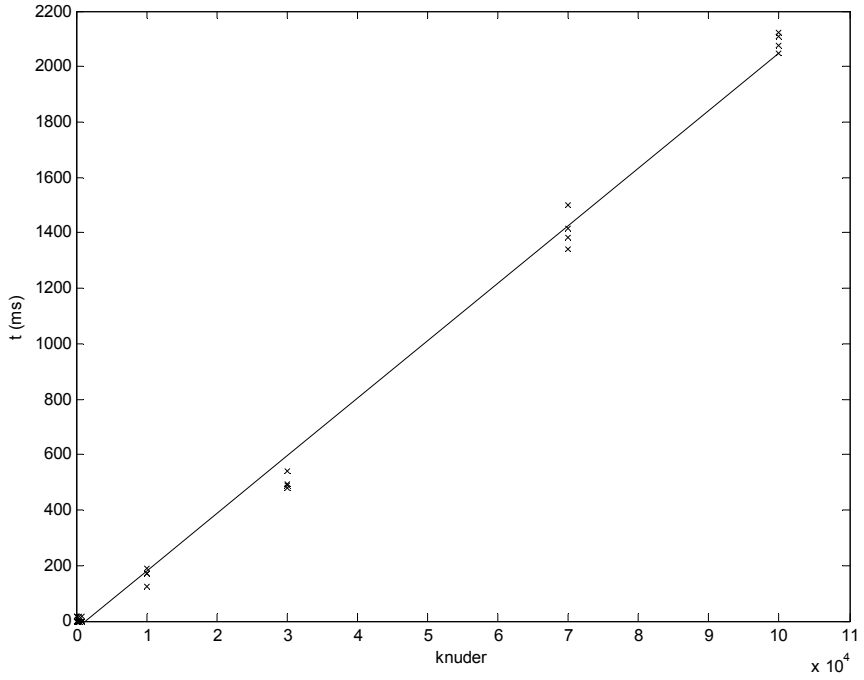
Figur 70: Markov Clusterings køretid som funktion af antallet af knuder i grafen.

Køretiden synes at være et polynomium af grad to, hvilket også var forventet. Figur 70 viser også at for en graf med 800 knuder er køretiden cirka 1 minut. Der er derfor ingen tvivl om, at Markov Clustering er uegnet til inddeling af grafen for det fuldstændige elforsyningsnet, med dets 400.000 knuder. Til gengæld er Markov Clustering velegnet til at generere arbejdsordrer af høj kvalitet i de større delnet.

11.5.2 Sektionering

I afsnittet Sektionering af grafer beskrev vi en sektioneringsalgoritme, for hvilken køretiden er $O(V_U \cdot V)$ hvor V_U er antallet af fejlmeldt knuder og V er antallet af knuder i grafen. Modsat Markov Clustering er den forventede køretid af Sektionering væsentligt lavere end den asymptotiske worst case køretid.

For at undersøge den faktiske ydelse af Sektionering, måler vi algoritmens køretid på et antal testgrafer med et varierende antal knuder. Ligesom under testen af Markov Clustering er 10 knuder markeret som fejlmeldt på hver graf. Grafen på Figur 71 viser algoritmens køretid som funktion af antallet af knuder i grafen.



Figur 71: Køretiden af Sektionering som funktion af antallet af knuder i grafen.

På figuren ser vi, at køretiden af Sektionering er lineært afhængig af antallet af knuder i grafen, hvilket står i stærk kontrast til køretiden af Markov Clustering. På figuren kan vi også se, at Sektionering er i stand til at håndtere 100.000 knuder på kun 2 sekunder, hvilket gør algoritmen særdeles velegnet til at inddele grafen for det fuldstændige elforsyningsnet i delnet. Sektioneringsalgoritmens afhængighed af antallet af fejlramte knuder, V_U, vil ikke blive undersøgt i dette afsnit, det skal blot nævnes, at V_U typisk er meget mindre end V, og dets indflydelse derfor er begrænset.

11.6 Delkonklusion: Benchmarking af Markov Clustering og Sektionering

De to algoritmer vi har undersøgt i dette afsnit har tydeligvis både fordele og ulemper. Markov Clustering er en meget fleksibel algoritme med hvilken man kan generere grupper der følger grafens naturlige struktur, til gengæld er den for langsom til at kunne benyttes på hele elforsyningsnettet. Sektionering ofrer noget af denne fleksibilitet til fordel for en uovertruffen køretid.

Den mest attraktive løsning med hensyn til gruppering af graferne må derfor være en hybridløsning, der benytter Sektionering til at inddele grafen for det fuldstændige elforsyningsnet i delnet, og Markov Clustering til at generere arbejdsordrer i de større delnet.

12 Perspektivering

Dette afsnit omhandler udvidelser af selve beregningsmodellen samt udvidelser af anvendelsesområdet for denne type system. De udvidelser til beregningsmodellen der beskrives her, er ikke implementeret i prototypen, da de enten falder udenfor det testdata der har været til rådighed i projektet eller ligger indenfor de begrænsninger der er opsat i afsnittet Afgrænsning.

Alle algoritmerne i denne rapport er meget afhængige af, at deres respektive parametre er indstillet fornuftigt. Når man sammenligner optimale parameterværdier for stormtilstanden med normaltilstand er der intet der tyder på, at disse værdier skal være ens. Eksempelvis vil man i stormtilstanden tillade, at grupperingsalgoritmen danner større grupper, da det er sandsynligt, at større områder faktisk er uden strøm.

Der kan også være behov for at operere med forskellige parameterværdier afhængigt af, hvilken type delnet man regner på. Det er sandsynligt, at man vil opnå bedre resultater, hvis man arbejder med forskellige parameterværdier for maske-, ø- og radialnet.

Det vil derfor være oplagt, at opbygge systemet så der kan opereres med flere sæt parametre. I nogle tilfælde kan systemet selv afgøre, hvilke paramtere der skal anvendes, eksempelvis på baggrund af delnet typen, mens det i andre tilfælde vil være brugerne, der skifter parameterværdierne når systemet f.eks. overgår til stormtilstand.

12.1 Udvidelse af modellen for sandsynligheder

Alle algoritmer i dette projekt anvender topologien i nettet, til at beregne den mest sandsynlige placering af fejl. Udover topologien indgår mange andre parametre i sandsynligheden for, at en given komponent er i stykker. Herunder gives en række eksempler på de oplysninger, der vil kunne supplere topologien, for at opnå en endnu mere præcis beregning. Listen er ikke udtømmende, der findes formentlig mange andre paramtere, der med fordel kan medtages i beregningen.

- Ledningstyper
- Vejarbejde/anlægsarbejde
- Komponenttype
- Bebyggelsestype (eksempelvis beboelse, industri mv.)
- Teknikernes vurdering

Ledningstyper skal forstås som eksempelvis luftledninger og jordledninger. Man kan forestille sig, at der efter en storm er øget risiko for fejl på luftledninger, hvorfor man ønsker at modificere fejlsandsynlighederne i systemet, så fejl med større sandsynlighed placeres på luftledningerne. Det samme gælder for vejarbejde og anlægsarbejde - alle jordledninger der går gennem et område, hvor der udføres gravearbejde, har større risiko for fejl, grundet deres eneste naturlige fjende, - gravemaskinen. Da både kommuner og entreprenører har elektroniske kort over, hvor der aktuelt udføres gravearbejde og DONG

Energy ligeledes har elektroniske kort over deres elforsyningsnet, vil man ved hjælp af overlay analyse[16] i et GIS system, let kunne finde alle ledninger, der går gennem de kritiske områder.

Ud fra komponenttype vil det også være muligt at beregne en sandsynlighed for fejl. Modellen kan evt. udvides til også at omfatte alder og stand for de enkelte komponenter, således at ældre komponenter i dårlig stand har den største sandsynlighed for fejl. På samme måde som beskrevet i afsnittet Indlæring, kan tilbagemeldinger fra teknikerne bruges til at bestemme fejlfrekvensen for de enkelte komponenttyper over tid, således at hver gang en tekniker rapporterer en fejl på en bestemt komponenttype, stiger fejlsandsynligheden på denne komponenttype.

Bebyggelsestypen i bestemte områder kan påvirke, hvor sikre fejlrapporter man kan forvente at modtage. Man kan f.eks. ikke forvente, at alle forbrugere i et beboelsesområde melder fejl ved afbrydelser midt på dagen eller om natten. Som tidligere beskrevet udtrykker parameteren α i objektfunktionen sandsynligheden for, at forbrugere faktisk er uden strøm, men ikke har rapporteret det. Man kan derfor forudse, at α i høj grad afhænger af bebyggelsestype og tidspunkt på dagen. I denne situation vil et GIS system med et digitalt kort over bebyggelsestyper kunne anvendes, til at justere α i forhold til den konkrete fejlrapport.

Sidst men ikke mindst ligger der stor viden om fejlenes placering ved de erfarne teknikere, der vedligeholder nettet. I nogle tilfælde vil de umiddelbart kunne se, at en del af en arbejdsordre fra systemet virker usandsynlig. Man kunne derfor indføre en mulighed for, at en tekniker kan tvinge algoritmen til at fjerne en eller flere forbindelser fra arbejdsordren, og derefter gennemregne situationen igen og udstede en ny arbejdsordre.

Af de OMS algoritmer der er udviklet i dette projekt, egner specielt EOMS sig godt til de udvidelser der her omtales. Når EOMS initialiseres, beregnes en fejlsandsynlighed på alle forbindelser i nettet ud fra topologien, dvs. en sandsynlighed der kun afhænger af afstanden til den nærmeste fejlmeldte knude, se afsnittet EOMS. Hvis EOMS skulle inkorporere en eller flere af de ekstra parametre i sandsynligheden, skulle sandsynligheden for fejl på hver forbindelse beregnes som en vægtet sum af de parametre, man ønsker at benytte. Der er således intet i vejen for, at benytte samtlige ovenstående parametre og topologien på samme tid.

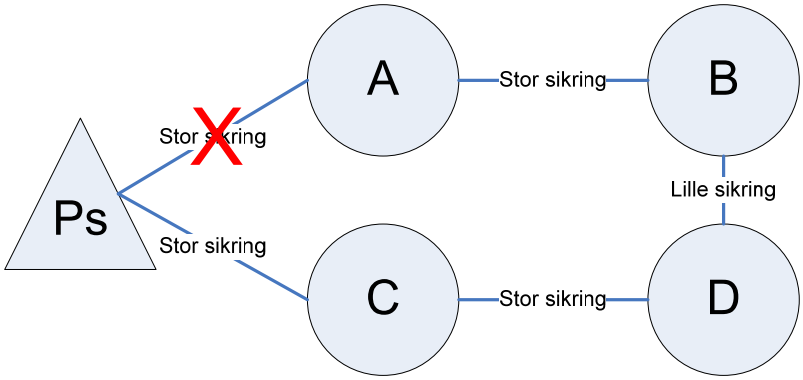
12.2 Elektrotekniske beregninger

Fælles for de tre fejlfindingsalgoritmer der er beskrevet i dette projekt er, at de bestemmer placeringen af fejl i nettet, ved at fjerne én forbindelse af gangen, uden hensyntagen til, hvilke konsekvenser det har for belastningen i de resterende forbindelser i nettet.

[16] Ved overlay analyse i et GIS system findes alle objekter i et datasæt, der geografisk set ligger indenfor objekterne i et andet datasæt [LONGLEY].

Beregningen kunne forfines ved at inddrage elektrotekniske parametre, så primære fejl og følgefejl bedre kan beregnes. Man kan f.eks. forestille sig, at hvis størrelse og placering af alle sikringer i nettet er kendt, kan følgefejl af at fjerne en forbindelse fra nettet i nogle tilfælde beregnes. På Figur 72 ses et eksempel på et net, hvor placering og størrelse af sikringer er angivet. Sikringen på forbindelsen *PsA* antages at være defekt. Hvis det kun er forbruger *A* der rapporterer fejlen, vil en rent topologisk beregning placere fejl på forbindelserne *PsA* og *AB*. Med hensyntagen til sikringernes størrelse, vil man derimod forvente, at opstår der fejl på forbindelsen *PsA* opstår der højest sandsynligt en følgefejl på forbindelsen *BD*, da der her kun sidder en lille sikring.

Der findes mange andre elektrotekniske fænomener, der kan anvendes til at beregne følgefejl i nettet. Indførsel af elektrotekniske beregninger vil dog gribe radikalt ind i den metode, der i denne rapport anvendes til at finde fejl i nettet, da sandsynligheden for fejl på de enkelte forbindelser antages at være uafhængig af andre forbindelser i nettet. Derfor vil det kræve en modifikation af algoritmerne at medtage sådanne beregninger. Endvidere må man forventes, at kompleksiteten af elktrotekniske beregninger på et større elforsyningsnet hurtigt bliver så voldsom, at det vil betyde en væsentlig forringelse af ydeevnen for algoritmerne. Det vil derfor kræve nærmere analyse, om sådanne beregninger kan gennemføres i praksis.



Figur 72: Eksempel på en følgefejl, der kan beregnes vha. elektrotekniske parametre.

12.3 Udvidelser af Arraymodellen

Som beskrevet i afsnittet ATOMS, er der udviklet en algoritme, der ved hjælp af tre forskellige arraymodeller kan løse fejlfindingsproblemet. Den primære grund til, at der benyttes tre forskellige modeller, i stedet for blot én, er at fejlfindingsproblemet når det betragtes som et konfigurationsproblem, er et Constraint Satisfaction Optimization Problem (CSOP). Arraymodellen i dens nuværende form retter sig mod almindelige Constraint Satisfaction Problems (CSP). Forskellen på de to typer problemer er, hvorvidt man leder efter en vilkårlig løsning, der overholder begrænsningerne i problemet (CSP), eller man ønsker at tilknytte en godhed til alle lovlige løsninger, og finde den løsning, der maksimerer godheden (CSOP). For en udførlig gennemgang af CSP og CSOP se [TURNER].

Hvis arraymotoren udvides, så man får mulighed for at definere en funktion der udtrykker godheden af en løsninger (objektfunktion) i modellen, og man derefter kan finde den bedste af de lovlige løsninger, vil fejlfindingsproblemet direkte kunne løses ud fra CM modellen. En sådan udvidelse vil kræve, at der både indføres en standardiseret måde at beskrive objektfunktionen på, formentlig som en udvidelse til AMS og/eller AML sproget, samt at der indføres algoritmer i runtime miljøet, der kan tage højde for objektfunktionen, når der slås op i arraymodellen. CSOP problemer opstår i mange sammenhænge, så det er ikke kun i forbindelse med OMS systemer at en sådan udvidelse vil kunne finde anvendelse.

I forbindelse med indlæring, skal man løse et CSP problem, som før eller siden ender i en modstrid, hvor der ikke længere findes en løsning, der opfylder begrænsningerne. Derfor har man brug for løbende at ændre på modellen, så man kan starte med en model med meget få begrænsninger og gradvist indføre nye, indtil der ikke længere findes lovlige løsninger. Som arraymotoren virker i dag, kan dette godt lade sig gøre, men det kræver at modellen rekompileres hver gang der foretages rettelser i AMS koden. Når modellen kompilerers, finder arraymotoren alle lovlige løsninger, hvilket naturligvis tager noget tid. I den samenhæng modellen anvendes i ved indlæring, er man ikke interesseret i alle lovlige løsninger, men blot om der findes mindst én lovlig løsning.

Med algoritmer som eksempelvis backtracking, suppleret med heuristikker til lookahead, value ordering og variable ordering [TURNER], [SADEH], kan man afgøre, om der findes mindst én lovlig løsning til en model. Disse metoder er ofte hurtigere end en komplet gennemregning af modellen. Da Array Technology allerede har sprog og parser til at beskrive og behandle et CSP, ville det være en stor styrke, at udvide produktporteføljen med en back tracking algoritme i runtime miljøet, der direkte kan anvendes på en AMS/AML fil, uden først at kompilere den til AMB. En anden mulighed er, at modificere kompileren, så nye begrænsninger kan indføres, uden at genkompilere hele modellen.

12.4 Andre anvendelser

Udfaldshåndtering er ikke kun interessant i forbindelse elforsyningsnet. Andre typer distrubutionsnet, f.eks. vandforsyningen, naturgasnettet eller fjernvarme, vil også kunne drage nytte af teknologien. I bredere forstand kan algortimerne også anvendes i forbindelse med håndtering af kloarkeringsnettet, computernetværk og analyse af forsyningslogistik i større organisationer.

Algoritmerne vil ikke kræve nogen særlige modifikationer, for at finde anvendelse i ovenstående områder, da fejlsøgning ved hjælp af topologi er en generel metode, der kan benyttes bredt. Som det er beskrevet ovenfor, kan modellen for fejlfinding i elforsyningsnettet, suppleres med flere sandsynlighedsparametre og elektrotekniske beregninger. Det samme gør sig gældende for de øvrige anvendelsesområder, hvor information om, hvordan fejl forplanter sig i eksempelvis naturgasnettet, vil kunne forbedre modellen.

13 Konklusion

Der er i dette projekt fundet frem til algoritmer der løser både fejlfindingsproblemet, udstedelse af arbejdsordre og indlæring. Med udgangspunkt i prototypen fra dette projekt har man således en solid algoritmebasis for at implementere Outage Management Systemer i bredeste forstand.

Første trin i procssen var at skabe en repræsentation af det fysiske elforsyningsnet, der reducerer kompleksiteten af problemtet så det kan behandles af en computer. Der blev valgt en grafrepræsentaion, der bibeholder information om nettets topologi men skjuler mange andre detaljer.

På baggrund af grafrepræsentationen er problemet ridset op. Det er præciseret, at lokalisering af fejl i elforsyningsnettet, dvs. fejlfindingsproblemet, samt gruppering af de fundne fejl og generering af arbejdsordrer er oplagte kandidater for automation.

Selvom den formelle matematiske definition af problemet var en stor og meget tidskrævende udfordring, har det vist sig at være et uundværligt værktøj i udarbejdelsen af algoritmerne - ikke mindst for at kunne argumentere for korrektheden af disse.

Der er implementeret tre forskellige metoder til løsning af selve fejlfindingsproblemet samt to metoder til gruppering, såvel som en metode til indlæring. Gennem analysen er det blevet klart, at problemstillingen indeholder elementer fra både grafteoretiske problemer og konfigurationsproblemer. Derfor har samspillet mellem de grafalgoritmer der er implementeret i projektet og Array Studio vist sig som en stærk kombination.

Selve fejlfindingsproblemet ligger i sin natur tæt op af det klassiske grafteoretiske problem at finde mindste snit i en graf. Gennem analysen af de tre forskellige fejlfindingsalgoritmer, der er udviklet i projektet, ATOMS, BOMS og EOMS, har netop EOMS, der er baseret på mindste snit algoritmer, vist sig at være en utrolig kraftfuld og stabil algoritme. ATOMS, som er baseret på array teknologi, er en meget fleksibel algoritme, der grundet den array teknologiske fundament let kan modificeres til at inddrage yderligere relationer end blot grafens topologi. BOMS algoritmen er den simpleste af de tre algoritmer, men på mindre net og med lav grad kan denne algoritme sagtens levere tilfredsstillende resultater.

En stor fordel ved ATOMS er, at det virkeligt tunge beregningsarbejde ligger i kompileringsfasen, som passende kan udføres når der ikke er så høj belastning på systemet og kun kræver genberegning når koblingstilstanden i nettet ændres. Som nævnt i perspektiveringen vil en mulighed for at løse CSOP problemer med Array teknologi give et løft til ydeevnen af ATOMS og endvidere vil man kunne reducere kompleksiteten af selve algoritmen kraftigt. Hvis Array runtime miljøet direkte kunne løse CSOP, ville det kun være nødvendigt med en enkelt array model, i stedet for de tre der anvendes nu.

Selve processen, det at modellere fejlfindingsproblemet i Array Studio, har givet et stort udbytte og en dyb indsigt i, hvilke aspekter af fejlfindingsproblemet der har den

beregningsmæssigt største kompleksitet. Det viser sig, at bestemmelse af umiddelbare forklaringer og konsevkensberegninger ikke kræver overvældende beregningsresourcer, mens bestemmelse af alternative forklaringer er den store udfordring i fejlfindingsproblemet. En mere effektiv metode til bestemmelse af alternative forklaringer er også den primære årsag til hastighedsforøgelsen af EOMS i forhold til ATOMS og BOMS.

De tre fejlfindingsalgoritmer er alle i stand til at løse fejlfindingsproblemet på delnet i Dong Energys elforsyningsnet, dvs. grafer på op til 3000 knuder, indenfor en rimelig tidsramme. Typisk tager det få sekunder at løse fejlfindingsproblemet på et delnet, men ved mange fejlrapporter kan køretiden komme op på nogle minutter. Alle algoritmerne holder sig altså under den acceptable tidsgrænse på 10 min.

EOMS er endvidere i stand til at løse fejlfindingsproblemet på det fuldstændige elforsyningsnet, - grafer med over 400.000 knuder, indenfor samme tidsramme. Med 540.000 knuder i nettet og 4 fejl tager gennemregningen kun 11 sekunder og i en stormtilstand med 50 fejl kan gennemregnes på under 5 minutter.

For at svaret fra en af de tre fejlfindingsalgoritmer kan omsættes til en eller flere arbejdsordrer skal fejlene grupperes efter deres topologiske afstand. På grund af både begrænsede ressourcer og arbejdssikkerhed ønsker man at kombinere fejl der ligger tæt på hinanden, topologisk set, så de kun resulterer i én arbejdsordre.

Der er implementeret to algoritmer til at løse denne del af problemstillingen: Markov Clustering og Sektionering. Markov Clustering er en utroligt fleksibel algoritme til bestemmelse af naturlige grupper i en graf. De naturlige grupper oprettes så antallet af forbindelser mellem grupperne er lavt. Af hensyn til arbejdssikkerheden er det vigtigt, at der findes så få forbindelser som muligt mellem de steder i netværket, hvor teknikerne arbejder, derfor er denne algoritme særdeles velegnet til at oprette arbejdsordrer i de enkelte delnet. Sektionering er en særdeles hurtig algoritme, der benytter sig af den indbyrdes afstand mellem fejlramte knuder til at afgøre, om de skal betragtes som værende i samme sektor eller ej. Sektionering er meget velegnet til at inddele det fuldstændige elforsyningsnet i delnet.

Endelig er der udarbejdet en algoritme til indlæring. Algoritmen tager udgangspunkt i konfigurationsteknikker prøver at estimere parameterværdierne til objektfunktionen. På genereret testdata[17] har denne algoritme vist sig at være i stand til at estimere fornuftige værdier af de parametre fejlfindingsalgoritmerne benytter.

Under benchmarking af de tre fejlfindingsalgoritmer er der kastet lys på hvilke faktorer der spiller ind i de respektive algoritmers ydeevne, udover antallet af knuder i grafen. Det gælder for ATOMS og BOMS at grafens grad, samt hvor mange niveauer af knuder der udtages til udregning af alternative forklaringer har stor indflydelse på ydelsen af disse

[17] Det er desværre ikke lykkedes at fremskaffe testdata fra DONG Energy. Derfor er projektets tests gennemført å genereret data, dog efter nøje overvejelser omkring hvordan data kan genereres, så strukturen ligger tæt op af DONG Energys data.

algoritmer. Endvidere er det vist at antallet af strømkilder i grafen indvirker på EOMS algoritmens ydeevne.

Generelt har projektet være både spændende og udfordrende, det har været nødvendigt at undersøge og anvende teknologier, der på forhånd ikke syntes at have nogen relation til OMS problemet. Specielt skal det fremhæves, at samspillet mellem Array Studios konfigurationsteknologi og den klassiske grafteori har vist sig både udfordrende og kraftfuld. Der er ingen tvivl om, at denne kombination indeholder potentiale til at løse mange andre problemstillinger, både indenfor udfaldshåndtering og mange andre områder.

14 Referencer

[BECK] Kent Beck, *Extreme Programming Explained*, Forlaget Addison-Wesley, (2000)

[BUTZE] Aske Butze-Ruhnenstierne, Svend K. Johannsen, *Verdens Største Egenværdiberegning*, Polyteknisk Trykkeri, (2004)

[CORMEN] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, MIT Press, (1990)

[DONGEN] Stijn van Dongen, *Graph Clustering by Flow Simulation*, PrintPartners Ipskamp, Enschede, (2000)

[HELLESEN] Bjarne Hellesen, Mogens O. Larsen, *Matematik for ingeniører Bind 3*, Danmarks Tekniske Universitet, (2000)

[LONGLEY] Paul A. Longley, Michael F. Goodchild, David J. Maguire, David W. Rhind, *Geographic Information Systems and Science,* John Wiley & Sons, Ltd, (2001)

[MEYER] Carl D. Meyer, Amy N. Langville, *Deeper Inside PageRank*, N. Carolina State University, (2004)

[NÆRFØR] Nærføringsudvalget, *Håndbog om Nærføring*, Online: http://www.naerfoering.dk/haandbog/Haandbog_om_Naerfoering.pdf (2006), Hentet: 05-04-2007

[SADEH] Norman Sadeh, Katia Sycara, Yalin Xiong, *Backtracking Techniques for the Joh Shop Scheduling Constraint Satisfaction Problem,* Carnegie Mellon University (1994)

[TURNER] Christopher Turner, *Constraint Based Reasoning with Constraint Logic Programming and Array Based Logic*, Queen's University, (1996)

[VATTER] Vince Vatter, *Graphs, Flows, and the Ford-Fulkerson Algorithm.* University of St Andrews (2004)

15 Appendiks

15.1 Appendiks 1: Matrix repræsentation

Ydeevnen af Markov Clustering processen er som før nævnt stærkt afhængig af ydeevnen af matrixproduktet. Da den Markov matrix Markov Clustering processen benytter repræsenterer en graf er det rimeligt at antage, at matricen er meget tyndt befolket, dvs. langt de fleste elementer er nul.

I en sparse matrixrepræsentation lagrer man kun de elementer der ikke er nul. På den måde kan man opnå store besparelser med hensyn til hvor meget hukommelse matrixrepræsentation benytter, men endnu vigtigere er det, at man med en sparse matrixrepræsentation kan forbedre ydeevnen af matrixproduktet og dermed ydeevnen af hele Markov Clustering processen. Såvel *inflations*-, som forskelsoperatoren drager også nytte af en sparse matrixrepræsentation, men da disse operatorer ikke er de dominerende faktorer i Markov Clustering processen, anses dette blot som en positiv sideeffekt.

$$M = \begin{bmatrix} 0 & 1 & 0 & 2 \\ 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 4 & 0 & 2 & 0 \end{bmatrix}$$

$$M_{dense} = [[0 \quad 1 \quad 0 \quad 2] \quad [3 \quad 0 \quad 0 \quad 0] \quad [0 \quad 5 \quad 0 \quad 0] \quad [4 \quad 0 \quad 2 \quad 0]]$$

$$M_{sparse} = [\{(1,1) \quad (3,2)\} \quad \{(0,3)\} \quad \{(1,5)\} \quad \{(0,4) \quad (2,2)\}]$$

Figur 73: En matrix M og en dense såvel som en sparse repræsentation af denne.

15.1.1 Dense matrix repræsentation

På Figur 73 ses to forskellige repræsentationer af matricen *M*. I dense matrixrepræsentationen lagres matricen som et "array af arrays." Dense matrixrepræsentationen kan enten tolkes som et array af rækker hvor hver række er et array af elementer, alternativt kan dense matrixrepræsentationen tolkes som et array af søjler hvor hver søjle er et array af elementer. Det første alternativ kaldes en rækkeorienteret dense matrixrepræsentationen, mens det andet alternativ kaldes en søjleorienteret dense matrixrepræsentationen. I disse to dense matrixrepræsentationer gælder det, at man *enten* kan udtrække en række i konstant tid, $O(1)$, *eller* en søjle, - men ikke begge dele.

Det skal i den forbindelse nævnes, at hvis man implementerer sine matricer samt operationer på disse i et sprog der er forholdsvis tæt på maskinkode, som f.eks. C eller C++, bestemmer ens matrixrepræsentation direkte hvordan de enkelte elementer lagres i den fysiske hukommelse. Dette betyder, at man ved valg af dense matrix repræsentation samtidig vælger hvordan matricen kan tilgås optimalt mht. til *caching*, dette gælder især for store matricer. Hvis man eksempelvis vælger af lagre matricen som en rækkeorienteret dense matrix, dvs. et array af rækker, vil man ved at tilgå de enkelte

elementer i en række opnå langt bedre udnyttelse af *cachen*, end hvis man tilgår de enkelte elementer i en søjle – der vil resultere i en lang række af *cache misses*.

15.1.2 Sparse matrix repræsentation

Af forskellige sparse matrix repræsentationer bør *Sparse Coordinate Format (SCF)* og *Harwell-Boeing (HB)* nævnes, se [BUTZE]. Disse formater er primært designet til at udnytte den tilgængelige hukommelse optimalt.

I SCF lagres en matrice som elementer, hvor hvert element består af to indices, række- og søjleindeks, samt en værdi. Disse elementer kan så sorteres efter række- eller søjleindeks og man opnår derved en rækkeorienteret sparse matrixrepræsentation hhv. søjleorienteret sparse matrixrepræsentation.

I HB lagres en matrice igen som elementer, men i denne repræsentation består et element blot af ét indeks, række- eller søjleindeks, samt en værdi. Indekset vælges selvfølgelig konsistent. Desuden benyttes et array af indices til at holde styr på, hvornår en ny række, eller søjle alt efter orienteringen, begynder i element listen. HB er en smule mere kompakt end SCF, desværre er det væsentlig mere kompliceret at konvertere en HB lagret matrix fra en rækkeorienteret til en søjleorienteret repræsentation.

Til brug i Markov Clustering er vi primært interesserede i en repræsentation, hvor vi kan udtrække rækker og søjler i konstant tid, $O(1)$, til brug ved matrixmultiplikation. SCF opfylder dette krav. I praksis gøres dette ved at holde ventresiden af matrixproduktet i rækkeorienteret SCF mens højresiden konverteres til søjleorienteret SCF. Alternativt kan man vælge at lagre begge repræsentationer til hver en tid, hvilket selvfølgelig betyder at hukommelsesforbruget fordobles. Dette kan dog være acceptabelt for problemer op til en hvis størrelse.

I praksis kan sparse matrixrepræsentationen f.eks. lagres som et ”array af hashtabeller.” I den rækkeorienterede matrixrepræsentation betyder dette, at hver række i matricen er repræsenteret af en hashtabel.

Sparse matrixrepræsentationer er kun interessante for matricer der faktisk er sparse, - dvs. hvis matricen indeholder mange nul-elementer. Hvor mange ikke-nul-elementer der skal være før det ikke kan betale sig, rent hukommelsesmæssigt, at benytte en sparse matrixrepræsentation afhænger af repræsentationen.

Vi benytter matricer til at repræsentere en graf. Det er i denne sammenhæng værd at lægge mærke til, at hver søjle i matricen svarer til en knude i grafen. Hver knude i grafen er forbundet til et antal andre knuder i grafen, typisk et sted mellem en og fem andre knuder. Dette betyder, at der kun vil være et til fem ikke-nul-elementer i hver søjle i matricen. Hvis vi f.eks. har en graf med 3000 knuder vil det altså være en enorm besparelse at lagre, og regne på, en til fem ikke-nul-elementer frem for alle 3000.

15.2 Appendiks 2: Matrix multiplikation

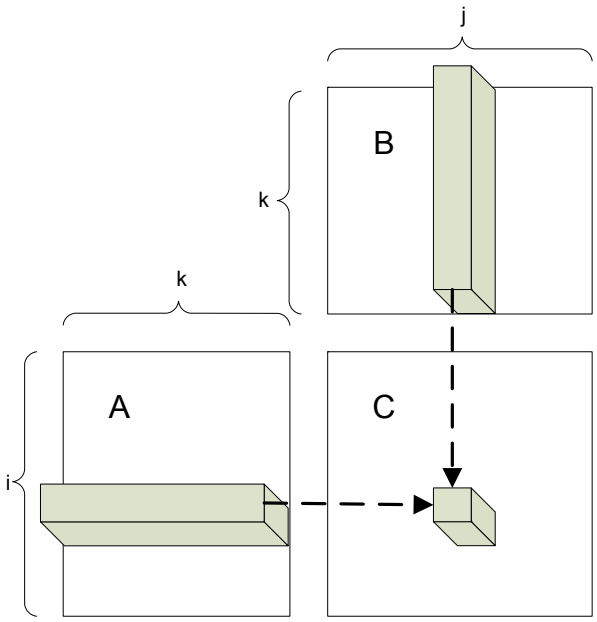
I dette afsnit vil vi forklare, hvordan matrixmultiplikation kan optimeres ved at benytte en sparse matrixrepræsentation.

Givet to matricer $A \in R^{i \times k}$ og $B \in R^{k \times j}$ kan deres produkt $C \in R^{i \times j}$ udregnes [HELLESEN]. Hvis A, B og C er dense matricer kan den traditionelle matrixmultiplikationsalgoritme, se [CORMEN], benyttes.

```
A, B og C er dense matricer.

DenseMatrixMatrixMultiply(A, B)
{
      for i=1 to A.Rows {
            for j=1 to B.Columns {
                  C[i, j] = 0
                  for k=1 to A.Columns {
                        C[i, j] += A[i, k] * B[k, j]
                  }
            }
      }
      return C
}
```

Boks 13: Pseudokode for den traditionelle matrixmultiplikationsalgoritme.

Den asymptotiske køretid for den traditionelle matrixmultiplikationsalgoritme er $O(n^3)$, hvor $n = \max(i, j, k)$. Køretiden for den traditionelle matrixmultiplikationsalgoritme er vores referencepunkt. Hvert element i matricen C er et vektorprodukt mellem en række i A og en søjle i B, se Figur 74. Hvis A og B indeholder mange nul-elementer foretages der unødigt mange multiplikationer. Dette kan der rodes bod på, ved brug af sparse matrixmultiplikation.



Figur 74: Matricerne A og B samt deres matrixprodukt C.

Hvis *A*, *B* og *C* i stedet er sparse matricer kan vektorprodukterne udregnes mere snildt. Der vælges en række i *A* og en søjle i *B* nøjagtig som i den traditionelle matrixmultiplikationsalgoritme, det første ikke-nul-element fra rækken i *A* vælges og der undersøges nu om søjlen i *B* indeholder et element med selv samme indeks. Hvis dette er tilfældet multipliceres de to tal og lægges til vektorproduktet, ellers vælges det næste ikke-nul-element fra rækken i *A*, der undersøges om søjlen i B indeholder et element med samme indeks, osv. osv. Alt dette kan gøres ved blot at erstatte det inderste loop i den traditionelle matrixmultiplikationsalgoritme. Da vi har valgt at lagre vores sparse matrix som et "array af hashtabeller" kan vi i konstant tid undersøge om et givent element i en række i *A* falder sammen med et element i den pågældende søjle i *B*.

```
A, B og C er sparse matricer.

SparseMatrixMatrixMultiply(A, B)
{
      for i=1 to A.Rows {
            for j=1 to B.Columns {
                  C[i, j] = 0
                  for k=0 to A[i].Keys.Count {
                        int key = A[i].Keys[k]
                        if (B[j].Contains(key)) {
                              C[i, j] += A[i, k] * B[k, j]
                              }
                  }
            }
      }
      return C
}
```

Boks 14: Pseudokode for sparse matrixmultiplikationsalgoritme.

Den asymptotiske køretid for sparse matrixmultiplikationsalgoritmen er $O\left(n^2 \cdot k\right)$, hvor $n = \max(i, j)$ og k er det størst forekommende antal ikke-nul-elementer i en række i A. I praksis vil en række i A typisk have én til fem ikke-nul-elementer i hver række og sparse matrixmultiplikationsalgoritmen er dermed en drastisk forbedring.

15.3 Appendiks 3

Fem testgrafer g_1, g_2, g_3, g_4 og g_5 hver med 1000 knuder og et gennemsnitligt antal kanter ud fra hver knude på tre, dvs. hver graf har cirka 1500 kanter. På hver af graferne udvælges 10 knuder tilfældigt og disse 10 knuder markeres som fejlramte.

r	$\lvert W \rvert\ (g_1)$	$\lvert W \rvert\ (g_2)$	$\lvert W \rvert\ (g_3)$	$\lvert W \rvert\ (g_4)$	$\lvert W \rvert\ (g_5)$	$\lvert W \rvert_{avg}$
1	28	28	26	30	23	**27**
2	103	94	89	107	87	**96**
3	266	238	217	277	218	**243**
4	552	478	454	562	454	**500**

Tabel 7: Antallet af udvalgte knuder i en graf som funktion af det maksimale antal kanter en udvalgt knude må ligge fra en fejlramt knude.

15.4 Appendiks 4

Fem testgrafer h_2, h_3, h_4, h_5 og h_6 hver med 1000 knuder men af variabel *grad*. Indekset angiver *graden* af hver graf. h_2 har gennemsnitligt to kanter mellem hver knude, h_3 har gennemsnitligt tre kanter mellem hver knude osv. 10 knuder udvælges tilfældigt og markeres som fejlramte, *r* er konstant $r = 3$.

	$\|W\|$ (h_2)	$\|W\|$ (h_3)	$\|W\|$ (h_4)	$\|W\|$ (h_5)	$\|W\|$ (h_6)
	64	286	520	722	931
	66	242	487	771	899
	52	261	545	703	912
	66	222	487	737	931
Gennemsnit	**62**	**253**	**510**	**733**	**918**

Tabel 8: Antallet af udvalgte knuder som funktion af grafens grad, dvs. det gennemsnitlige antal kanter mellem to knuder.

15.5 Appendiks 5

Fire testgrafer i_1, i_2, i_3 og i_4 hver med 100 knuder og et gennemsnitligt antal kanter ud fra hver knude på to, dvs. graferne har cirka 100 kanter hver. På hver af graferne udvælges tre knuder tilfældigt, disse tre knuder markeres som fejlramte.

	i_1		i_2		i_3		i_4	
r	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$
1	3	8	5	32	4	16	3	8
2	9	512	13	8.192	13	8.192	7	128
3	19	524.288	20	104.576	23	8.388.608	13	8.192
4	29	536.870.912	26	67.108.864	37	137.438.953.472	19	524.288

Tabel 9: Antallet af alternative forklaringer som funktion af antallet af udvalgte knuder.

15.6 Appendiks 6

Fire testgrafer $i1$, $i2$, $i3$ og $i4$ hver med 100 knuder og et gennemsnitligt antal kanter ud fra hver knude på to, dvs. graferne har cirka 100 kanter hver. På hver af graferne udvælges tre knuder tilfældigt, disse tre knuder markeres som fejlramte.

	$i1$		$i2$		$i3$		$i4$	
r	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$	$\|W\|$	$\|S\|$
1	3	8	3	4	6	4	3	4
2	7	32	8	24	16	648	11	420
3	12	272	15	2.016	29	$< 2^{29}$	16	3.360
4	17	1.908	23	$< 2^{23}$	39	$< 2^{39}$	20	29.600

Tabel 10: Antallet af alternative forklaringer som funktion af antallet af udvalgte knuder.

15.7 Appendiks 7

Et stort antal testgrafer med et knudeantal mellem 10 og 2000 samt et gennemsnitligt antal kanter ud fra hver knude på to, dvs. antallet af kanter er cirka lig antallet af knuder. Kompileringstiden for de tre arraymodeller PPM, ECM og CM er målt som et gennemsnit af tre kompileringer.

# knuder $\lvert N \rvert$	**Komp.tid PPM (ms)**	**Komp.tid ECM (ms)**	**Komp.tid CM (ms)**
10	125	78	93
20	280	109	250
50	953	234	619
100	2473	619	2.546
500	27.598	10.155	39.879
800	62.796	28.239	103.010
1.000	109.296	48.286	214.598
1.200	120.348	71.765	292.609
1.500	219.994	116.583	386.447
2.000	459.739	226.489	510.505

15.8 Appendiks 8

Fem testgrafer h_2, h_3, h_4, h_5 og h_6 hver med 1000 knuder men af variabel *grad*. Indekset angiver *graden* af hver graf. h_2 har gennemsnitligt to kanter mellem hver knude, h_3 har gennemsnitligt tre kanter mellem hver knude osv. 10 knuder udvælges tilfældigt og markeres som fejlramte.

	t (ms) (h_2)	t (ms) (h_3)	t (ms) (h_4)	t (ms) (h_5)	t (ms) (h_6)
	109	844	4172	2953	7828
	63	688	1391	2656	6172
	78	578	1766	2938	6750
	172	703	2016	6453	5484
Gennemsnit	**106**	**703**	**2336**	**3750**	**6559**

Tabel 11: Køretiden af EOMS algoritmen på grafer af forskellig grad, dvs. det gennemsnitlige antal kanter mellem to knuder.

15.9 Appendiks 9

Fem testgrafer p_2, p_4, p_8, p_{16} og p_{32} hver med 1000 knuder og af *grad* 3. Indekset angiver hvor mange strømkilder der befinder sig i grafen, p_2 har to strømkilder, p_4 har fire osv. 10 knuder udvælges tilfældigt og markeres som fejlramte.

	t (ms) (p_2)	t (ms) (p_4)	t (ms) (p_8)	t (ms) (p_{16})	t (ms) (p_{32})
	2547	2750	1391	688	500
	1391	1484	1938	703	406
	1703	1328	1453	766	672
	2500	797	578	391	266
Gennemsnit	**2035**	**1590**	**1340**	**637**	**461**

Tabel 12: Køretiden af EOMS algoritmen på grafer med hhv. 2 til 32 strømkilder.

15.10 Appendiks 10

Seks testgrafer c_1, c_2, c_3, c_4, c_5 og c_6 med et variabelt antal knuder hhv. 100, 200, 400, 800, 10.000 og 100.000. Alle graferne er af *grad* 3. På hver graf udvælges 10 knuder tilfældigt og markeres som fejlramte.

	c_1		c_2		c_3		c_4		c_5		c_6	
	MCL	Sek	MCL	Sek	MCL	Sek	MCL	Sek	MCL	Sek	MCL	Sek
	1063	16	2906	0	14484	16	62891	0	-	172	-	2078
	875	16	2703	0	13375	0	55469	16	-	125	-	2109
	813	0	2813	0	14156	0	48922	0	-	172	-	2125
	656	0	3031	0	14234	0	68750	0	-	188	-	2047
Avg.	**852**	**8**	**2863**	**0**	**13812**	**4**	**59008**	**4**	**-**	**164**	**-**	**2090**

Tabel 13: Køretiden (ms) af Markov Clustering og Sektionering på seks grafer med et variabelt antal knuder.