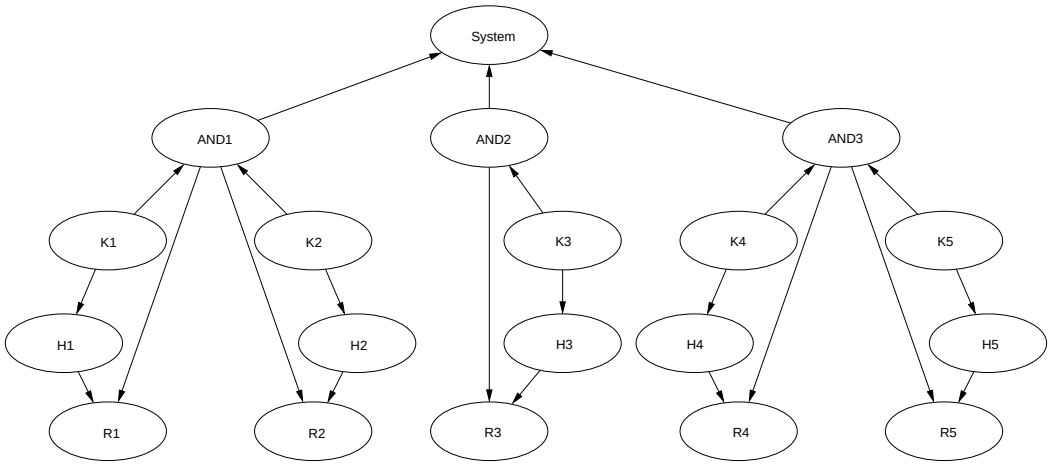
Beregning af sekvenser af reparationshandlinger til troubleshooting i domæner med afhængige fejl og afhængige handlinger



Gruppe E1-117, DAT6
Aalborg Universitet
Institut for datalogi
Fredrik Bajersvej 7E, 9220 Aalborg ø

Aalborg Universitet
Afdelingen for datalogi

Frederik Bajers Vej 7E, 9220 Aalborg Øst, Danmark

Title: Calculating Sequences of Repairactions for Troubleshooting in Domains Containing Dependent Faults and Dependent Actions

Groupmembers:

Lise Thorn Andersen
Frank G. Hahn

Supervisor:

Helge Langseth

Project period:

10/2 - 15/7 2001

Group:

E1-117

Printed reports:

7

Pages:

65

Abstract:
The purpose of this master thesis is to extend existing methods for calculating approximated optimal sequences of repair actions for systems containing multiple independent or dependent causes and/or dependent repair actions and to compare the results against the calculated optimal sequences.
The expressions for efficiency are developed for use in calculating sequences of repair actions. The expressions are an extension of existing expressions. We have added a weight (V) for the purpose of being able to handle dependent actions.
We experienced that using a weight was useless in itself and that the expression of efficiency $\frac{P}{C}$ in almost all situations yielded fine results. Moreover we discovered that a combination of the used heuristics based on $\frac{P}{C}$ and $\frac{P \cdot V}{C}$ gave even better results than the use of $\frac{P}{C}$ on its own for the models giving suboptimal results with $\frac{P}{C}$.

Aalborg Universitet
Afdelingen for datalogi

Frederik Bajers Vej 7E, 9220 Aalborg Øst, Danmark

Titel: Beregning af sekvenser af reparationshandlinger til troubleshooting i domæner med afhængige fejl og afhængige handlinger

Gruppemedlemmer:

Lise Thorn Andersen
Frank G. Hahn

Synopsis:
Formålet med dette speciale er at videreudvikle eksisterende metoder til beregning af tilnærmede optimale handlingssekvenser for systemer med multiple uafhængige eller afhængige fejlårsager og/eller afhængige reparationshandlinger, samt at vurdere løsningerne op mod beregnede optimale sekvenser.
Effektivitetsudtryk udvikles til brug ved beregning af sekvenser af reparationshandlinger. Udtrykkene er en udvidelse af de eksisterende effektivitetsudtryk, hvor vi har tilføjet en vægtning (V), med det formål at kunne håndtere afhængige handlinger.
Vi fandt ud af, at idéen med denne form for vægtning i sig selv var ubrugelig, og at effektivitetsudtrykket $\frac{P}{C}$ i næsten alle situationer gav fine resultater. Derudover fandt vi frem til at en kombination af heuristikkerne baseret på hhv. $\frac{P}{C}$ og $\frac{P \cdot V}{C}$ gav et samlet bedre resultat end $\frac{P}{C}$ alene, for de modeller, hvor $\frac{P}{C}$ gav suboptimale resultater.

Vejleder:

Helge Langseth

Projekt periode:

10/2 - 15/7 2001

Gruppe:

E1-117

Antal eksemplarer:

7

Antal sider:

65

Forord

Denne rapport er resultatet af et speciale, der omhandler sekvensberegning til fejlfinding. Formålet er at finde heuristikker, der kan beregne tilnærmede optimale troubleshootingsekvenser, idet en eksakt beregning er NP-hård. De systemer, vi ønsker at finde heuristikker til, indeholder afhængige reparationshandlinger og/eller multiple uafhængige eller afhængige fejl.

Der tages udgangspunkt i vores rapport på dat 5, som omhandlede automatisk generering af troubleshootingmodeller, der var baseret på single-fault antagelsen.

Tak til Hugin Expert for at stille værktøjet Hugin Professional til rådighed i projektperioden.

Kildehenvisninger i rapporten er angivet ved [kilde] og kan findes i litteraturoversigten bagerst i rapporten.

Aalborg Universitet, 15. juli, 2001

Lise Thorn Andersen

Frank G. Hahn

Indhold

Kapitel 1

Indledning

Dette projekt omhandler støttesystemer til fejlfinding og fejlretning. I mangel af et dækkende dansk ord, benytter vi fremover den engelske betegnelse "troubleshootere".

Vi beskæftiger os med videreudvikling af metoder til at beregne en optimal sekvens af handlinger til troubleshooting af et system. Vi videreudvikler metoderne således at det er muligt at finde en tilnærmelsesvis optimal sekvens af troubleshootinghandlinger for systemer med multiple fejl, både uafhængige[1] og afhængige fejl. Vi har lagt vægt på at finde en heuristik, der er rettet mod at finde sekvenser for modeller, der indeholder afhængige reparationshandlinger på tværs af antagelserne om fejlårsagerne. Afhængige handlinger i et system betyder at der er komponenter, som repareres af flere handlinger. Ved afhængige fejl forståes fejl, der ikke hver for sig kan forårsage at systemet er i fejltilstand. I stedet skal en hel gruppe af indbyrdes afhængige komponenter være i fejltilstand, før systemet er i fejltilstand. Dette svarer til replikerede ressourcer, hvor systemet først fejler når samtlige backup ressourcer fejler.

1.1. Tidligere arbejde

Vi har i et tidligere projekt "TOMGen" på dat 5 [Hahn and Andersen, 2001], beskæftiget os med den del af troubleshooting, der omfatter indsamling af information fra domæneeksperter og oprettelse af den til troubleshooting bagvedliggende matematiske (f.eks. bayesianske) model for det system, der ønskes troubleshootet. Her var hovedvægten lagt på (semi-)automatisk modelgenerering ud fra den indsamlede information. Vi lavede et værktøj, som automatisk kan generere en troubleshootermodel til et teknisk system. I denne sammenhæng er et teknisk system kendetegnet ved at det er opbygget eller sammensat af flere typer ensartede komponenter, og at den enkelte komponenttype går igen flere gange i systemet.

[1] dvs. en eller flere fejl, hvis tilstedeværelse ikke påvirker hinanden

Vi byggede dér videre på den erfaring, vi havde indsamlet i et tidligere projekt fra dat3[2], som omhandlede troubleshooting af computernetværk. Det var bl.a. konklusionerne fra dette projekt, der førte til kravet om at beskrive et system ved en topologi for sig og hver af komponenterne (beskrevet som typer eller klasser) for sig. Vi tog i [Hahn and Andersen, 2001] udgangspunkt i den fremgangsmåde, der er benyttet i BATS[3] systemet, som vi anvendte i dat3 projektet [Andersen et al., 1999].

TOMGen er et værktøj, til at generere bayesianske modeller til brug ved troubleshooting, ud fra en topologi over systemet, og en beskrivelse af de (komplekse) komponenter, der indgår i denne topologi.

De systemer, som dat5-projektet og det resulterende værktøj, TOMGen, var rettet mod, skulle opfylde følgende krav: Systemerne skulle bestå af klart identificerbare komponenter, som indgik i en overordnet topologi. Der måtte ikke være cykler i denne topologi, dvs. topologien skulle have træstruktur. Komponenterne skulle kunne beskrives og troubleshootes for sig. På et givet tidspunkt måtte der kun være én fejl tilstede i systemet, dette krav betegnes *single-fault*.

Et computernetværk er et nærliggende eksempel på et system, der ofte opfylder ovennævnte krav. Computernetværk består af en række klart identificerbare enheder (computere, switche, printere m.v.). Disse enheder er ordnet i en topologi, som er den struktur enhederne er forbundne i. Der kan opstå situationer, hvor topologien ændres, enkelte enheder udskiftes, eller der foretages en opgradering af maskinerne. Nye afdelinger kan kræve udvidelse af computernettet, eller omstrukturering af det eksisterende computernet.

Et af problemerne med at antage single-fault er at computernetværk ofte indeholder replikerede ressourcer, f.eks. backupmaskiner der tager over hvis en central server fejler. Dette er en af grundene til at vi valgte at undersøge afhængige fejl. Et andet problem er, at der i TOMGen er antaget uafhængige handlinger (dvs. højst en handling pr. fejlårsag), hvilket heller ikke er helt realistisk, idet der ofte findes flere løsningsmuligheder til et problem med forskellig grad af succes og forskellig omkostning. I dette projekt ønsker vi at undersøge hvorvidt det er muligt at finde gode sekvenser af handling for systemer, der ikke opfylder single-fault og uafhængige handlinger.

1.2. Fejlfinding

Troubleshooting – og metoder til design af troubleshooting

En "intelligent" troubleshooter guider brugeren gennem en række af troubleshootingtrin (handlinger og undersøgelser) i forsøg på hurtigst muligt at løse problemet. Der bruges mange penge på support af tekniske systemer – ikke mindst computersystemer. Support

[2] Udsnit af rapporten fra dat 3, der bl.a. viser modeller med replikerede ressourcer som DNS, kan ses i Bilag A

[3] Bayesian Automated Troubleshooting System, se [Jensen et al., 2001]

består typisk i telefonsupport og, om nødvendigt, teknikerbesøg. Hvis en intelligent automatisk troubleshooter stilles til rådighed, f.eks. over web, kan mange problemer løses uden menneskelig assistance. Hvis troubleshooteren ikke er i stand til at finde en løsning på et problem, kan den af troubleshooteren indsamlede information overføres til en supportmedarbejder, som så fortsætter troubleshootingen. Den indsamlede information kan spare supportmedarbejderens tid, da vedkommende kan springe over de troubleshootingtrin, som allerede er udført.

Modellen, som ligger til grund for troubleshooting, kan være baseret på beslutningstræer, bayesianske net eller helt andre metoder. Vi vælger at anvende bayesianske net, idet de er ideelle til formålet: At træffe beslutninger under usikkerhed ud fra ikke komplette informationer om et system, hvor der kan være usikkerhed om konsekvenserne af beslutningerne. Beslutningsprocessen mht. hvilke handlinger og undersøgelser, der skal udføres, er baseret på modellen. De handlinger og undersøgelser, der udføres, har til formål at lokalisere og reparere fejlen i systemet. Beslutningsprocessen består i at finde den sekvens af handlinger og undersøgelser, ud af de mulige sekvenser, der fører til den billigste løsning af problemet. Vi antager, at forudsætningen for at troubleshooting startes er, at der er identificeret at systemet er i fejltilstand. Ved handlinger forstår vi reparationshandlinger, som evt. eliminerer en eller flere årsager til fejl, med en fast omkostning og med en given sandsynlighed for at reparere hver enkelt årsag. Det er disse handlinger som skal løse problemet. Eksempel: Handlingen "udskift netkort" eliminerer (næsten) at en fejl i netkortet er skyld i systemets fejlfunktion.

1.3. Afgrænsning

Undersøgelser i form af observationer behandles ikke, da troubleshooting i sig selv er NP-hard, når observationer medtages. Vi fravælger at beskæftige os med observationer for ikke unødigt at komplicere heuristikkerne. Metoder til beregning af tilnærmede sekvenser af troubleshootingtrin, hvori der indgår observationer, er behandlet i bl.a. [Jensen et al., 2001].

Reparationshandlingen CS — tilkald af reparatør — medtages ikke i beregningen af sekvenser af reparationshandlinger. Grunden til ikke at tage denne handling med er at den skal behandles specielt, da den kan reparere alle fejl i systemet, og at håndtering af den også er behandlet i [Jensen et al., 2001]. Derudover gør den i sig selv sekvensberegningen NP-hard.

1.4. Problemformulering

Vi vil videreudvikle de eksisterende metoder til beregning af en optimal sekvens af reparationshandlinger[4], således at der også kan beregnes en tilnærmet optimal sekvens for et

[4] — eller, hvor en optimal sekvens ikke er mulig at beregne pga. NP-hardness, en sekvens, der er bedst mulig.

system med multiple uafhængige eller afhængige fejlårsager og/eller afhængige reparationshandlinger. Løsningerne vil vi vurdere i forhold til de optimale løsninger, der findes ved udtømmende søgning.

Kapitel 2

Teori

I dette kapitel beskriver vi teorien bag troubleshooting. Først beskrives en generel algoritme til at finde en sekvens af handlinger, der løser problemet som troubleshootes. Derefter beskrives udtrykket for effektivitet af handlinger, som benyttes i beregningen af sekvensen, under de forskellige antagelser om både årsagerne til fejl[1] og antagelser om de handlinger[2], der reparerer årsagerne[3].

Det første afsnit (2.1) er en generel indledning til bayesianske net og troubleshooting teorien, derunder algoritmer til at finde den optimale sekvens og heuristikker til at finde en tilnærmet optimal sekvens, og kompleksiteten af algoritmerne og af heuristikkerne.

De to efterfølgende afsnit (2.2 og 2.3) omhandler troubleshooting under antagelsen af hhv. single-fault og uafhængige fejl. Til disse problemstillinger findes der allerede kendte løsninger i de tilfælde, hvor handlingerne er uafhængige. Vi bygger videre på disse løsninger (eller dele af dem), for at finde løsninger til problemet med afhængige handlinger. I de kendte løsninger bruges et udtryk for handlingernes effektivitet til at afgøre i hvilken rækkefølge, det er hensigtsmæssigt at udføre handlingerne. Da sortering efter dette effektivitetsudtryk ikke nødvendigvis giver en optimal eller bare hensigtsmæssig rækkefølge af handlingerne i situationer med afhængige handlinger (og / eller afhængige fejlårsager), udvider vi dette effektivitetsudtryk med en vægtning, og anvender dette vægtede effektivitetsudtryk[4] til sortering af handlingerne. I det næste afsnit (2.4) udleder vi et vægtet effektivitetsudtryk for situationer, hvor der er afhængige fejlårsager.

Et effektivitetsudtryk er et forsøg på at måle hvor effektiv en handling H er, til at få systemet til at fungere igen, relativt til andre handlinger.

Tabel 2.1 viser en oversigt over de 6 forskellige problemsituationer som troubleshooting-problemstillingerne kan inddeles i.

[1] Antagelserne om årsager opdeles i single-fault, uafhængige fejl og afhængige fejl
[2] Antagelserne om handlingerne opdeles i uafhængige og afhængige handlinger
[3] Ved årsager forstår vi fejlårsager, vi benytter også betegnelsen komponenter
[4] Vi betragter et effektivitetsudtryk som et tal til at sortere reparationshandlingerne i sekvensen efter.

ÅRSAGER -> / HANDLINGER	Single-fault XOR	Uafhængige fejl OR	Afhængige fejl XOR+AND
Uafhængig (1:1)	(1) $\frac{P}{C}$	(3) $\frac{P}{C \cdot (1-P)}$	(5) $\frac{\sum_j P(AND_j^K) \cdot P}{C}$
Afhængig (n:m)	(2) $\frac{P \cdot V}{C}$	(4) $\frac{P \cdot V}{C \cdot (1-P)}$	?

Tabel 2.1.: Oversigt over effektivitetsudtryk i forskellige situationer

Troubleshootingmetoden, dvs. hvordan der findes en god sekvens af fejlfindingstrin (handlinger og spørgsmål), og hvordan information (årsager, handlinger og spørgsmål) repræsenteres i den bayesianske model, er beskrevet detaljeret i [Skaanning et al., 2000] og [Jensen et al., 2001].

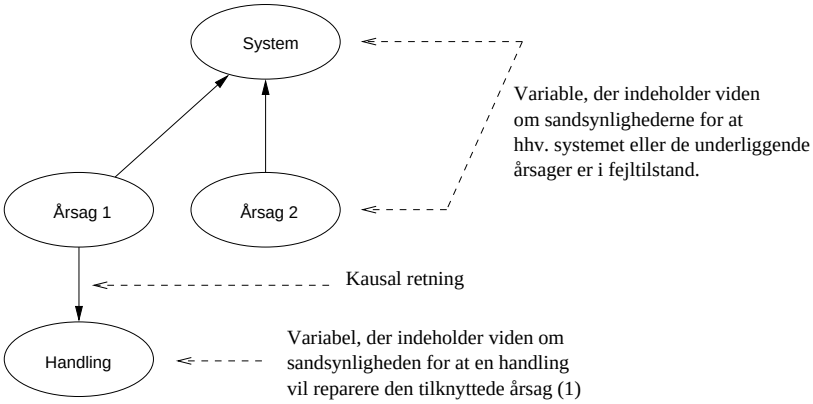
2.1. Troubleshooting

2.1.1. Bayesianske net og modellering

Bayesianske modeller giver en måde, hvorpå problemområder kan modelleres, ved at repræsentere betingede uafhængigheder og de kausale relationer i problemområdet vha. sandsynlighedsteori og grafteori. Dermed kan repræsentationen af problemet med et bayesiansk net bruges til at fremskaffe information om nogle variable givet andre. Et bayesiansk net består af en mængde af variable (knuder) og en mængde af retningsbestemte kanter, som forbinder variablerne. Hver variabel har en mængde af tilstande, som gensidigt udelukker hinanden, dvs. at en variabel til et givet tidspunkt er i én tilstand. Variablerne sammen med de retningsbestemte kanter udgør en retningsbestemt acyklisk graf (DAG). For hver variabel v med forældrene $w_1, \ldots . w_n$, er der specificeret en betinget sandsynlighedstabel $P(v|w_1, \ldots . w_n)$. Hvis v ikke har nogen forældre, reduceres denne tabel til en marginal sandsynlighedsfordeling $P(v)$. Yderligere information om bayesianske net kan findes i [Jensen, 2000].

Bayesianske net anvendes på mange områder, hvor der træffes beslutninger under usikkerhed, f.eks. medicinsk diagnose, analyse af husdyr og planlægning. Et stort anvendelsesområde for bayesianske net har været diagnose[5]. Systemer til diagnose udnytter de bayesianske modeller vældig godt — f.eks. ved at modellere underliggende faktorer som forårsager sygdomme eller fejlfunktioner, som igen forårsager symptomer, og ved at kunne repræsentere usikkerheder mht. de kausale sammenhænge. I dette projekt anvender vi bayesianske netværk til at modellere problemområder, der kan fejlfindes vha. de metoder der beskrives i dette kapitel.

[5]Nudansk ordbog: undersøgelse og afgørelse - bestemmelse af en sygdoms / fejlfunktions art.



Figur 2.1.: Struktur for en bayesiansk model, brugt til modellering af et problemdomæne og til at udtrække de nødvendige sandsynligheder for at kunne beregne en sekvens af handlinger og finde denne sekvens' forventede omkostning.

2.1.2. Troubleshooting med bayesianske net og Hugin

Antag, at vi ønsker at udføre troubleshooting på et fejlfungerende system med n mulige underliggende grunde til fejlfunktionen repræsenteret ved variablene $K_1, \ldots, K_n$ (K for komponent). I et computernetværk er komponenterne servere, segmenter og netkort i maskiner m.v. Antag at vi har defineret reparationshandlinger $H_1, \ldots, H_k$, som har mulighed for at løse problemet, og at hver reparationshandling H_i har sandsynligheden $P_i = P(H_i = \text{ja}|e)$ for at afhjælpe problemet givet den nuværende evidens[6] (e), og en omkostning eller pris C_i for at udføre handlingen. Indenfor f.eks. printersystemer og computernetværk er omkostningen ved handlinger ofte afhængig af fortiden. Hvis f.eks. en handling kræver at en komponent adskilles, vil andre handlinger, som kræver samme komponent adskilt, være billigere at udføre samtidig med. Men vi ser bort fra afhængige omkostninger, idet troubleshooting med afhængige omkostninger er NP-hard i sig selv [Jensen et al., 2001], [Sochorova and Vomlel, 2000]. Vi antager altså at omkostningen ved udførelse af en handling er uafhængig af historien, og dermed ikke ændrer sig. Omkostningen kan være kombineret af mange forskellige omkostningsfaktorer f.eks. den tid, det tager at udføre handlingen, og penge til at købe værktøj og materialer.

I domæner som printersystemer og computernetværk er der ofte kun en fejl på et givet tidspunkt (single-fault), da der oftest vil opdages en fejlfunktion straks når der opstår en fejl, og som følge heraf vil troubleshooting blive igangsat ved tilstedeværelse af én fejl.

En troubleshootingsekvens er en sekvens af troubleshootinghandlinger. Formålet med en troubleshootingsekvens er at reparere fejlen i systemet billigst muligt, dvs. med mindst mulig omkostning. Målet for, hvor effektiv en troubleshootingsekvens er, er den forventede

[6] Evidens er summen af den, på et givet tidspunkt, tilstedeværende faktiske viden om systemet, som er lagt ind i modellen. Denne viden er opnået gennem de udførte handlinger og spørgsmål.

reparationsomkostning, *expected cost of repair* (ECR).

Den forventede reparationsomkostning $ECR(S)$ for en given sekvens af fejlfindingshandlinger $S =< H_1, H_2, \ldots, H_k >$ beregnes i det generelle tilfælde med udtrykket 2.1, for single-fault kan dette udtryk skrives simplere (udtryk 2.2).

$$\begin{aligned} ECR(S) &= C_1 + \\ &\quad P(H_1 = nej|sys = f) \cdot C_2 + \\ &\quad P(H_1 = nej, H_2 = nej|sys = f) \cdot C_3 + \\ &\quad \ldots + \\ &\quad P(H_1 = nej, \ldots, H_{k-1} = nej|sys = f) \cdot C_k \\ &= C_1 + \sum_{i=2}^{n} P(H_1 = nej, \ldots, H_{i-1} = nej|sys = f) \cdot C_i, \end{aligned} \tag{2.1}$$

hvor $P(H_1|sys = f)$ er sandsynligheden for at systemet stadig er i fejltilstand efter udførelse af handling H_1 og dermed at handling H_2 skal udføres med omkostningen C_2. Det at systemet er i fejltilstand ($sys = f$) er en forudsætning for at foretage og fortsætte troubleshooting. Handling H_i udføres efter alle de forudgående handlinger $(H_1, \ldots, H_{i-1})$. Sandsynligheden for at H_i skal udføres er $P(H_1, \ldots, H_{i-1}|sys = f)$. Der propageres evidens efter hver udført handling H_j. At udføre en handling svarer til at vi indsætter evidens på $H = nej$, idet vi observerer at systemet stadig er i fejltilstand. Sandsynligheden $P(H_1, \ldots, H_n|sys = f)$ kan ifølge den fundamentale regel[7] findes som

$$P(H_1, \ldots, H_n|sys = f) = \frac{P(H_1, \ldots, H_n, sys = f)}{P(sys = f)}$$

Sandsynlighederne $P(H_1, \ldots, H_i, sys = f)$ og $P(sys = f)$ kan nemt udtrækkes af Hugin, hvilket er det værktøj vi benytter til at håndtere propagering m.v. i bayesianske netværk. $P(e)$ er sandsynligheden for den propagerede evidens. I Hugin kan denne værdi udtrækkes med funktionen $h\_domain\_get\_normalization\_constant$, når der anvendes sum-propagering [Hugin, 2000, side 56], eller den kan aflæses manuelt som $P(All)$.

$P(sys = f)$ er sandsynligheden for at systemet er i fejltilstand før der indsættes evidens på udførte handlinger i nettet (kan aflæses som $P(e)$ efter at evidens om at systemet er i fejltilstand er propageret), ligeledes er $P(H_1, sys = f)$ sandsynligheden $P(e)$ efter evidens på at handling H_1 er udført er propageret (stadig med evidens på at systemet er i fejltilstand).

Under antagelsen af single-fault bliver beregningen af ECR enklere. Sandsynligheden $P(H_i = nej|Sys = f)$ for at systemet stadig er i fejltilstand efter udførelse af H_i kan nu beregnes som $(1 - P_i)$, hvor P_i er sandsynligheden for at H_i reparerer systemet, da vi pga. single-fault antagelsen ved at *netop* én af komponenterne fejler, og summen af alle

[7] Den fundamentale regel er $P(A, B) = P(A|B)P(B) \Leftrightarrow P(A|B) = \frac{P(A,B)}{P(B)}$.

P_i er 1.

$$\begin{aligned} ECR(S) &= C_1 + \\ &\quad (1 - P_1) \cdot C_2 + \\ &\quad (1 - P_1 - P_2) \cdot C_3 + \\ &\quad \ldots + \\ &\quad (1 - P_1 - P_2 - \ldots - P_{n-1}) \cdot C_n \\ &= C_1 + \sum_{i=2}^{n} (1 - P_1 - \ldots - P_{i-1}) \cdot C_i, \end{aligned} \tag{2.2}$$

hvor C_i er omkostningen ved at udføre handling H_i i sekvensen S og P_i er sandsynligheden for at handling H_i reparerer systemets fejlfunktion. P_i beregnes som $P(K_j) \cdot P(H_i|K_j)$, hvor $P(K_j)$ er sandsynligheden for at komponent K_j er i fejltilstand (og dermed er årsag til at systemet er i fejltilstand), og $P(H_i|K_j)$ er sandsynligheden for at H_i reparerer K_j givet at K_j er i fejltilstand.

Værktøjet Hugin og den tilhørende API er beskrevet i afsnit 3.1.3.

Bagstopperen:
Hvis der er ikke-perfekte handlinger i sekvensen, kan det ikke garanteres, at udførelse af sekvensen efterlader systemet i *ok* tilstand. Derfor kan en speciel handling kaldet bagstopperen medtages. Bagstopperen ("Call Service" - CS) kan være et serviceeftersyn, dvs. tilkaldelse af udefra kommende assistance til at reparere systemet. Omkostningen for handlingen CS svarer til den høje omkostning ved at tilkalde ekstern hjælp til løsningen af problemet. Handlingen CS har sandsynligheden 1 for at reparere enhver fejlårsag, den reparerer alle komponenter og efterlader dermed systemet i *ok* tilstand.

Det, at CS reparerer alle fejl gør den speciel. Den er en afhængig handling, i den forstand at den reparerer samme komponent(er) som en (flere) andre handlinger, og dermed bliver troubleshootingopgaven NP-hard (se afsnit 2.1.4). Men da vi ved, at udførelsen af CS med 100% sikkerhed reparerer systemet, ved vi også, at der *efter* udførelse af CS ikke skal udføres flere handlinger. Derfor kan en rimelig placering af CS i den beregnede sekvens S beregnes på følgende måde:

1. Beregn sekvensen af reparationshandlinger $S =< H_1, H_2, \ldots, H_n >$
2. Beregn $ECR_0 = ECR(< CS >)$
3. Beregn for 1 til n $ECR_i = ECR(< H_1, \ldots, H_i, CS >)$
4. Sekvensen med den laveste ECR vælges som den bedste (den er ikke nødvendigvis den optimale).

Kompleksiteten af denne beregning er (maksimalt) n beregninger af ECR, dvs. $O(n^2) \cdot O(Prob)$. Dvs. at med denne tilnærmede beregningsmetode er troubleshooting med CS beregnelig.

Vi medtager ikke handlingen CS i det videre arbejde, da den i sig selv gør troubleshooting NP-hard. Da det ved ikke-perfekte handlinger uden CS ikke kan garanteres at systemet efterlades i *ok* tilstand, kan vi i princippet ikke længere beregne $ECR(S)$ (Expected Cost of *Repair*) af en sekvens af ikke-perfekte handlinger. Det vi så beregner er den forventede omkostning ved udførelse af sekvensen S, $EC(S)$, så snart der indgår ikke-perfekte handlinger i sekvensen. For nemheds skyld fortsætter vi dog med at bruge betegnelsen $ECR(S)$ vel vidende at der ikke er garanti for reparation af systemet uden perfekte handlinger eller CS.

En anden konsekvens vil være at alle handlinger vil blive udført, hvis der bare er den mindste sandsynlighed for at en endnu ikke udført handling kan reparere systemet. Hvis CS var med, ville en handling med meget lille reparationssandsynlighed og med høj omkostning ofte ikke blive udført før der blev tilkaldt assistance.

2.1.3. Heuristik

Algoritme til beregning af sekvens:

1. Beregn effektiviteten $Ef(H_i)$ for alle handlinger (H_i).
2. Vælg den bedste handling (den med højest effektivitet), og tilføj den til sekvensen, hvis $Ef(H_i) > 0$. Har den bedste handling H_i $Ef(H_i) = 0$, betyder det at systemet ikke længere er fejlbehæftet.
3. Propager info om handlingens "udførelse", således at sandsynlighederne for fejl i de enkelte komponenter opdateres.
4. Fjern handlingen H_i fra listen af handlinger, der kan udføres.
5. Gentag fra trin 1, indtil der ikke er flere handlinger H_i (med $Ef(H_i) > 0$) i listen af handlinger, som kan udføres.

Denne algoritme svarer overordnet til den, som bruges i [Jensen et al., 2001] BATS-troubleshooteren. I dette projekt ændrer vi på hvordan handlingernes effektivitet beregnes, med det formål at den beregnede sekvens af handlinger har den lavest mulige ECR, også når systemet, som skal troubleshootes, indeholder afhængige fejlårsager og/eller afhængige handlinger.

Kompleksiteten af denne algoritme til beregning af den bedste sekvens af reparationshandlinger er (hvis $Ef(H_i)$ kan beregnes uden ekstra propageringer):

$$O(n) \cdot O(\text{prop. af evidens}), \text{ hvor } n \text{ er antallet af handlinger} \tag{2.3}$$

I tilfældet, hvor der er single-fault og uafhængige handlinger, kan ovenstående algoritme forenkles som følger:

1. Beregn effektiviteten $Ef(H_i)$ for alle handlinger (H_i).

2. Sorter handlingerne efter faldende effektivitet.

I afsnit 2.2 beskrives det nærmere hvordan denne algoritme fremkommer.

2.1.4. Kompleksitet af troubleshooting

Ofte kan handlinger reparere mere end en fejl, og dermed er det ikke altid muligt at finde den optimale sekvens ved at sortere mht. effektivitet. Den forventede reparationsomkostning bliver så mere kompleks at beregne i disse tilfælde. Problemet er at udtrykket for effektivitet i sin simple form ikke tager højde for at der er afhængige fejl. At finde den optimale sekvens ved afhængige fejl er NP-hard og dermed skal der anvendes en heuristik, der begrænser søgerummet ved at finde rimelige, tilnærmede sekvenser. Tilnærmede algoritmer for at finde en god sekvens og for at beregne dens ECR præsenteres i [Jensen, 2000], [Skaanning et al., 2000] og [Jensen et al., 2001]. Effektiviteten og enkelheden af repræsentationen og algoritmerne, som beskrives her, opnås under antagelse af single-fault, dvs. at der kun er én fejl i systemet på et givet tidspunkt.

Nondeterministisk Polynomiel kompleksitet

Det, at et problem er af *Nondeterministisk Polynomiel kompleksitet*, vil sige at det ville kunne løses i polynomiel tid af en nondeterministisk algoritme. En nondeterministisk algoritme er en algoritme som, hver gang der er flere muligheder, ved på magisk vis hvilken mulighed, der skal vælges. Denne egenskab betyder, at det ikke er nødvendigt at regne alle mulighederne igennem for at finde den bedste. Et problem P siges at være NP-hard, hvis et NP-complete problem i polynomiel tid kan reduceres til P.

Vi har beskrevet opgaven, som vi ønsker at finde en løsning til. Desværre er det et NP-hard problem at finde den bedste sekvens af handlinger til reparation af et system [Sochorova and Vomlel, 2000].

Kompleksiteten af den generelle troubleshootingopgave er undersøgt og beskrevet i artiklen [Sochorova and Vomlel, 2000]. De har ved at reducere et kendt NP-hard problem "Exact cover by 3-sets" til en troubleshooting opgave bevist, at det generelle troubleshootingproblem med at finde en sekvens S af reparationshandlinger med forventet reparationsomkostning $ECR(S) \leq K$, hvor $K \in \Re^+$ er en konstant, er NP-complete under antagelse af afhængige handlinger og single-fault eller uafhængige fejl. Dermed vil troubleshootingopgaven være NP-hard under antagelse af *afhængige* fejl. De har også bevist at troubleshooting er NP-hard under antagelse af afhængige omkostninger - selv under antagelse af uafhængige handlinger. Troubleshooting med spørgsmål og uafhængige handlinger er også NP-hard [Jensen et al., 2001], [Sochorova and Vomlel, 2000].

På denne baggrund er det relevant at søge efter gode heuristikker til at beregne en tilnærmet sekvens, i stedet for en eksakt. Begrundelsen er at den korrekte sekvens ikke er beregneligt i det generelle tilfælde, men kun kan findes ved at begrænse modellen (single-fault eller hierarkiske uafhængige fejl) eller ved at tilnærme beregningen ved andre

metoder (heuristikker med begrænset look-ahead og/eller interaktiv beregning af næste handling.

2.2. Single-fault

I dette afsnit beskriver vi først fejlfinding i et system, hvor der antages single-fault og *uafhængige* handlinger. Med denne baggrund undersøger vi derefter, hvordan der kan fejlfindes i et system med single-fault og *afhængige* handlinger.

Single-fault er en relativt nem måde at modellere simple problemstillinger på. Den er velegnet til meget stramt strukturerede problemsituationer uden interaktivitet mellem komponenterne i systemet. Hvis de tilhørende reparationshandlinger kan betragtes som uafhængige, er det muligt at beregne en optimal sekvens af reparationshandlinger i polynomiel tid. Hvis der tillades afhængige handlinger, bliver beregningen af den optimale sekvens NP-hard.

I et system, hvor de enkelte komponenters funktion ikke er afhængige af andre komponenters funktion, og hvor der troubleshootes så snart der opstår en fejlfunktion i systemet, er det rimeligt at antage, at der kun er én fejl i systemet.

2.2.1. Uafhængige handlinger

Uafhængige handlinger er handlinger, som reparerer *forskellige* fejlårsager (komponenter). Hvis to handlinger reparerer samme komponent, er de *afhængige*.

Definition 1 *Handlinger H_i og H_j er uafhængige hvis og kun hvis $Pa(H_i) \cap Pa(H_j) = \emptyset$. $Pa(H)$ er mængden af forældre til knuden H i den bayesianske model. (Handlingerne er repræsenterede som børn af fejlårsagerne.) [Langseth and Jensen, 2001]*

Følgende gennemgang af grundlæggende troubleshooting er baseret på [Jensen, 2000], [Heckerman et al., 1995], [Breese and Heckerman, 1995] og [Kalagnanam and Henrion, 1990].

Antag at vi allerede har udført nogle handlinger og at den information, vi har fået fra disse handlinger, giver sandsynlighederne $P_i = P(H_i = \text{ja}|e)$, for at handlingen H_i vil afslutte sekvensen med yderligere omkostning C_i. Vi har altså udført handlingerne $< H_1, H_2, \ldots, H_{i-1} >$ i sekvensen $< H_1, H_2, \ldots, H_{i-1}, H_i, H_j, H_{j+1}, \ldots, H_n >$, hvor $j = i + 1$. Betragt to kandidathandlinger H_i og H_j. Vi undersøger nu følgende to tilfælde:

1. først udføres H_i, og hvis den ikke afhjælper fejlen udføres H_j
2. først udføres H_j, og hvis den ikke afhjælper fejlen udføres H_i

I det første tilfælde er *bidraget* fra handlingerne H_i og H_j til den forventede reparationsomkostning, ECR:

$$C_i + P(H_i = \text{nej}|e)C_j$$

hvor C_j er omkostningen ved at udføre H_j, efter at H_i er udført. Bidraget til ECR er i det andet tilfælde:

$$C_j + P(H_j = \text{nej}|e)C_i$$

Pga. forudsætningen om fast omkostning ændres omkostningen ved den efterfølgende troubleshooting ikke af H_i og H_j's rækkefølge. Så hvis

$$C_i + P(H_i = \text{nej}|e)C_j < C_j + P(H_j = \text{nej}|e)C_i \tag{2.4}$$

vil det være bedst at udføre H_i først, evt. efterfulgt af H_j. Det forudsættes her at handlingssekvensen, som kommer efter $< H_i, H_j, >$ og $< H_j, H_i, >$ er den samme.

Ligning 2.4 kan omskrives sådan:

$$\begin{aligned}
C_i + P(H_i = \text{nej}|\text{e})C_j &< C_j + P(H_j = \text{nej}|\text{e})C_i \\
&\Downarrow \\
C_i + (1 - P(H_i = \text{ja}|e))C_j &< C_j + (1 - P(H_j = \text{ja}|e))C_i \\
&\Updownarrow \\
C_i + (1 - P_i)C_j < C_j + (1 - P_j)C_i, &\text{ hvor } P_k \equiv P(H_k = \text{ja}|e), \\
&\Updownarrow \\
-P_iC_j &< -P_jC_i \\
&\Updownarrow \\
\frac{P_i}{C_i} &> \frac{P_j}{C_j}.
\end{aligned} \tag{2.5}$$

Da vi endnu ikke har brugt antagelse om at handlingerne er uafhængige, gælder ligning 2.5, når blot der gælder single-fault og handlingerne har fast omkostning. Men ligning 2.5 forudsætter at H_i og H_j kommer umiddelbart efter hinanden. Ligning 2.5 gælder for de første to handlinger i S efter e.

Med andre ord siger ligning 2.5 at det ikke er optimalt at udføre et par af handlinger $< H_i, H_j >$ som nr. 1 og 2 i sekvensen hvis ikke

$$\frac{P(H_i)}{C_i} \geq \frac{P(H_j)}{C_j}$$

Er der uafhængige handlinger vil denne lokale (partielle) ordning også, som der vises herefter, give en total ordning. I tilfælde med afhængige handlinger er det kun ordningen af første par af handlinger der holder. En sortering vil ikke nødvendigvis give den optimale sekvens.

Værdien $\frac{P_i}{C_i}$ er et mål for effektiviteten af handling H_i. Bemærk at effektiviteten af en handling varierer med udførte handlinger og gjorte observationer, da P_i'erne ændres af evidensen.

En fristende tilgang til at finde en optimal sekvens ville være altid at udføre den handling, som har den højeste effektivitet. Desværre garanterer denne metode ikke, at man herved finder en handlingssekvens med lavest mulig ECR, da ligning 2.5 forudsætter at H_i og H_j er naboer i sekvensen S. Men hvis vi yderligere kan antage, at under udførelsen

af handlingerne *ændres sandsynlighederne P_i for alle de resterende handlinger med den samme faktor*, så vil alle effektiviteterne også ændres med en konstant faktor. I dette tilfælde kan vi starte fejlfindingsopgaven ved at sortere handlingerne efter aftagende effektivitet, og vi får herved en sekvens med lavest mulig ECR. Denne ekstra antagelse er opfyldt hvis vi antager at:

1. der er kun én fejl (single-fault) eller uafhængige fejl, og
2. uafhængige handlinger. Denne antagelse bevirker at effektiviteterne kun skal beregnes én gang og så kan sorteres for at finde sekvensen (Kompleksiteten er $O(1) \cdot O$(prop. af evidens) plus kompleksiteten ved beregning af Ef'erne).

Bevis 1 *Lad H_m være en handling, som ikke reparerede fejltilstanden. Vi beregner $P(H_i = ja|H_m = nej, e)$. På grund af single-fault antagelsen har vi at $P(H_m = nej|H_i = ja, e) = 1$. Ved at bruge Bayes regel får vi*

$$P(H_i = ja|H_m = nej, e) = \frac{P(H_m = nej|H_i = ja, e)P(H_i = ja|e)}{P(H_m = nej|e)} = \frac{P(H_i = ja|e)}{P(H_m = nej|e)}.$$

Det vil sige at $P(H_m = nej|e)$ blot er en normaliseringskonstant for de resterende handlinger, og den relative størrelsesorden af effektiviteterne bevares.

□

Dermed har vi følgende effektivitetsudtryk, som handlingerne kan sorteres efter:

$$Ef(H) = \frac{P(H = ja)}{C_H} = \frac{P(H|K) \cdot P(K)}{C_H} \tag{2.6}$$

Hvis disse antagelser er opfyldte, er ECR for en optimal sekvens $S =< H_1, \ldots, H_k >$:

$$ECR(S) = C_1 + (1 - P_1)C_2 + (1 - P_1 - P_2)C_3 + \ldots + (1 - \sum_{i=1}^{k-1} P_i)C_k \tag{2.7}$$

Forklaring: H_1 skal altid udføres med omkostningen C_1, da troubleshooting kun startes hvis systemet er i fejltilstand. Med sandsynligheden P_1 reparerer H_1 systemet, og da der er forudsat single-fault, er sandsynligheden for at H_2 også skal udføres $(1 - P_1)$. Dvs. at med sandsynligheden $(1 - P_1)$ må vi også tage omkostningen C_2 ved at udføre H_2, som har sandsynligheden P_2 for at reparere systemet. Herefter er sandsynligheden for at systemet stadig er i fejltilstand $((1 - P_1) - P_2)$ osv.

2.2.2. Afhængige handlinger

Fra definitionen af uafhængige handlinger, definition 1, fås at afhængige handlinger er handlinger, som reparerer samme komponent. Dvs. at handlinger H_i og H_j er afhængige

når $Pa(H_i) \cap Pa(H_j) \neq \emptyset$. Dermed er det ikke længere givet at $P(H_j = nej | H_i = ja) = 1$, da det er muligt at H_i og H_j reparerer samme komponent.

Idet handlinger ikke altid i praksis er knyttede til en enkelt fejlårsag, fjerner vi antagelsen om uafhængige handlinger og tillader afhængige handlinger. Udvidelsen med afhængige handlinger giver mulighed for en mere realistisk modellering af virkeligheden. Konsekvensen af at fjerne antagelsen om uafhængige handlinger er, at der ikke længere er nogen sikkerhed for at effektiviteterne på de resterende handlinger ændrer sig med en fast faktor ved udførelse af en handling, da forudsætning 2 side 14 ikke er opfyldt. Dermed er det ikke længere muligt at finde den optimale sekvens ved at sortere mht. effektivitet som $Ef(H_i) = \frac{P(H_i=\text{ja})}{C_i}$, og den forventede reparationsomkostning kan ikke beregnes med ligning 2.7. En eksakt beregning af den optimale sekvens er NP-hard, og derfor skal en anden beregningsmetode anvendes. Vi har valgt at søge efter en heuristik, der giver en god tilnærmet sekvens.

Et klassisk eksempel er, at en fejl i en komponent ofte både vil kunne repareres med en handling, der afhjælper den specifikke fejl, og med handlinger, der udskifter en større eller mindre del af komponenten (evt. hele komponenten).

I eksempel 1 vises et eksempel. Tilnærmede algoritmer for at finde en god sekvens og for at beregne dens ECR for systemer, hvor der findes handlinger, som kan reparere mere end én fejl, præsenteres i [Jensen, 2000], [Skaanning et al., 2000] og [Jensen et al., 2001].

En handling, der kan reparere flere komponenter, har effektiviteten:

$$Ef(H) = \frac{P(H = ja)}{C_H} = \frac{\sum_i P(H = ja | K_i) \cdot P(K_i)}{C_H}, \tag{2.8}$$

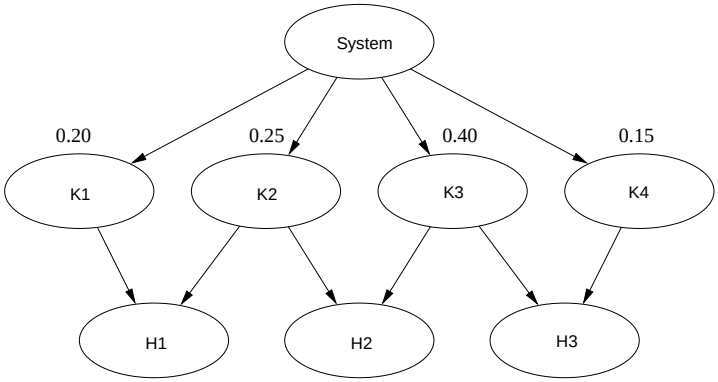
hvor K_i er de komponenter, som repareres af handling H, da H's positive effekt på systemet udgøres af summen af H's effekter på de komponenter, den virker på, og hvor $P(K_i)$ er sandsynligheden for at årsag K_i ligger til grund for problemet og hvor $P(H|K_i)$ er sandsynligheden for at handling H reparerer årsag K_i (givet at K_i er i årsag til fejltilstand).

Hvis en handling reparerer mere end én komponent, er det tilstrækkeligt pga. single-fault antagelsen at angive én sandsynlighed for hver fejlårsag, den reparerer, dvs. sandsynligheden for at handlingen løser problemet, under antagelse af at årsagen er den aktuelle tilgrundliggende fejlårsag. Pga. single-fault antagelsen er det ikke nødvendigt at overveje kombinationer af fejlårsagerne.

Vi afprøver nu heuristikken for single-fault og uafhængige handlinger (den grådige strategi) på eksemplet fra [Skaanning et al., 2000].

Et eksempel (Figur 2.2) på de problemer, som følger med afhængige handlinger

Eksempel 1 *[Den grådige strategi] På figur 2.2 er vist et eksempel på et system med afhængige handlinger. $K_1, \ldots, K_4$ er fire mulige årsager til fejlfunktion i systemet. Vi antager at præcis én af disse årsager er til stede, og at a priori sandsynlighederne for deres tilstedeværelse er hhv. 0.20, 0.25, 0.40 og 0.15.*



Figur 2.2.: Eksempel på system med afhængige handlinger.

Der er tre reparationshandlinger tilknyttet fejlårsagerne, og hver handling kan reparere de fejlårsager hvorfra der på figuren er angivet en kausal forbindelse. Antag at alle handlinger er perfekte, dvs. $P(H|K_i) = 1$*, og antag omkostningen* $C_H = 1$ *ved deres udførelse.*

Hvis vi bruger metoden fra single-fault og uafhængige handlinger (beskrevet i afsnit 2.2.1) til at finde en sekvens af handlinger til at reparere systemet i dette eksempel, får vi, som det fremgår af det følgende, ikke den bedste sekvens:

Det "oprindelige" udtryk for effektivitet fra ligning 2.6 er:

$$Ef(H) = \frac{P(H|K) \cdot P(K)}{C_H}, \text{ hvor } K \text{ repareres af handling } H$$

hvor $P(H|K_i) = 1$*, da vi her har antaget perfekte handlinger.*

Sekvensen af reparationshandlinger beregnes udfra metoden i afsnit 2.2.1. Først udregnes alle effektiviteterne, hvorefter hele sekvensen af handlinger stilles op efter faldende effektivitet. Handlingernes effektiviteter, givet udtryk 2.8, bliver:

$Ef(H_1) = \frac{P(K_1)+P(K_2)}{1} = 0.20 + 0.25 = 0.45$
$Ef(H_2) = \frac{P(K_2)+P(K_3)}{1} = 0.25 + 0.40 = 0.65$
$Ef(H_3) = \frac{P(K_3)+P(K_4)}{1} = 0.40 + 0.15 = 0.55$

Hermed er den grådige sekvens $< H_2, H_3, H_1 >$*. Dens ECR udregnes i følge ligning 2.1 som:*
$ECR(< H_2, H_3, H_1 >) = C_2 + C_3 \cdot P(H_2 = nej) + C_1 \cdot P(H_2 = nej, H_3 = nej)$*, dvs. at*
$ECR(< H_2, H_3, H_1 >) = 1 + 1(\frac{0.35}{1}) + 1(\frac{0.20}{1}) = 1.55.$

Men sekvensen $< H_1, H_3 >$*, som er den optimale sekvens (fundet ved udtømmende søgning*[8]*), har* $ECR = 1.45$*. Det viser at den fundne sekvens* $< H_2, H_3, H_1 >$ *ikke er optimal. Bemærk at* H_2 *ikke skal udføres i den optimale sekvens, da det antages at alle*

[8] exhaustive search

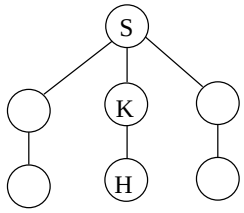
handlinger er perfekte, og alle komponenter dermed allerede er repareret af H_1 og H_3.

Det som her gik galt var, at den grådige algoritme ikke tog højde for at alle komponenter kan repareres uden brug af H_2, og at K_1 og K_4 kun kan repareres af hhv. H_1 og H_3. I situationen, hvor H_2 allerede er udført, tager metoden heller ikke højde for at i den nye situation, hvor kun K_1 eller K_4 er i fejltilstand, er effektiviteterne for H_1 og H_3 reelt ændrede, da ingen af dem længere har mulighed for at reparere K_2 hhv. K_3. Ifølge den anvendte grådige strategi beregnes handlingernes effektiviteter kun ud fra situationen ved troubleshootingens start. Der udføres ikke propagering af den ny evidens (evidensen at en handling er udført og ikke reparerede systemet).

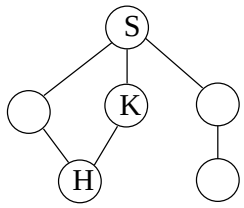
Håndtering af afhængige handlinger under single-fault

Vi belyser her nogle forskellige modelstrukturer med forskellige karakteristika. Strukturerne i modellen består af en systemknude (S) der indikerer at systemet er i fejltilstand og en række komponenter (K) der kan repareres af handlinger (H). Disse forskelligartede strukturer er senere udgangspunkt for en test af heuristikkerne.

På figur 2.3 og figur 2.4 er handlingerne uafhængige.



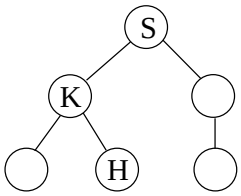
Figur 2.3.: Uafhængige handlinger - én årsag med én handling



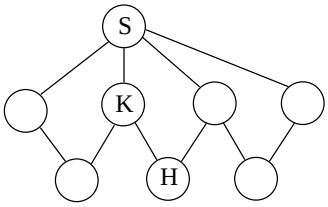
Figur 2.4.: Uafhængige handlinger - flere årsager med en handling

På figur 2.5 og figur 2.6 er handlingerne afhængige. Strukturen på figur 2.6 er en kombination af strukturen på de to foregående figurer.

Vi ønsker at finde en heuristik baseret på et effektivitetsudtryk, der giver en bedre sekvens af handlinger for strukturer som på figur 2.6. Fra [Jensen et al., 2001] vidste vi at



Figur 2.5.: Afhængige handlinger - én årsag med flere handlinger.



Figur 2.6.: Flere årsager med flere handlinger.

$\frac{P}{C}$ ikke altid giver optimale sekvenser for denne struktur. Samtidig ønsker vi at effektivitetsudtryket giver mindst ligeså god en sekvens for strukturer som på figur 2.3 og figur 2.4 (samt figur 2.5) som den grådige metode.

I det følgende eksempel bruger vi den generelle algoritme til beregning af sekvens af reparationshandlinger fra afsnit 2.1.3 på den samme model som i eksempel 1. Vi anvender stadig det samme effektivitetsudtryk, som i eksempel 1.

Eksempel 2 *[SACSO/propagerings-heuristik] Her kommer eksemplet fra eksempel 1 igen, men med den ændring, at der nu propageres evidens efter hver udført handling, før den næste handling vælges. Metoden til at finde en sekvens af reparationshandlinger er nu:*

1. *Effektiviteterne udregnes - stadig som* $\frac{\sum P}{C}$.
2. *Den bedste handling udføres.*
3. *Der propageres evidens om handlingens udførelse, og handlingen fjernes fra listen af handlinger, der kan udføres.*
4. *Indtil der ikke er flere handlinger med en positiv effekt, gentag fra punkt 1.*

Der antages fortsat at alle handlinger er perfekte, dvs. $P(H|K_i) = 1$, *og har omkostningen* $C_H = 1$ *ved deres udførelse.*

Effektivitetsudtrykket fra eksempel 1:

$$Ef(H) = \frac{P(H = ja)}{C_H} = \frac{\sum_i P(H|K_i) \cdot P(K_i)}{C_H},$$

bruges stadig, og nu det giver følgende effektiviteter:

$Ef(H_1) = \frac{P(K_1)+P(K_2)}{1} = 0.20 + 0.25 = 0.45$
$Ef(H_2) = \frac{P(K_2)+P(K_3)}{1} = 0.25 + 0.40 = 0.65$
$Ef(H_3) = \frac{P(K_3)+P(K_4)}{1} = 0.40 + 0.15 = 0.55$

H_2 udtages som den optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med en sandsynlighed på 0,35.

Anden runde af beregning (efter propagering) giver:
$Ef^1(H_1) = \frac{P(K_1)}{1} = 0.20$
$Ef^1(H_3) = \frac{P(K_4)}{1} = 0.15$
H_1 indsættes dermed i sekvensen som den næste handling. Nu er systemet med sandsynligheden 0,15 stadig i fejltilstand.

Hermed er den bedste sekvens beregnet med propagering efter hver valgt / udført handling: $< H_2, H_1, H_3 >$. *Dens ECR udregnes som:*
$ECR(< H_2, H_1, H_3 >) = C_2 + C_1 \cdot P(H_2 = n) + C_3 \cdot P(H_2 = n, H_1 = n)$, *dvs. at*
$ECR(< H_2, H_1, H_3 >) = 1 + 1(\frac{0.35}{1}) + 1(\frac{0.15}{1}) = 1.50$.
Dette er bedre end for sekvensen, som vi fandt i eksempel 1, hvor $ECR = 1.55$. *Sekvensen* $< H_1, H_3 >$ *har* $ECR = 1.45$, *og dermed er den fundne sekvens* $< H_2, H_1, H_3 >$ *stadig ikke optimal.*

Udtryk for effektivitet

Heller ikke ovenstående metode, hvor der propageres i den bayesianske model og effektiviterne udregnes igen for hver udført handling, giver den optimale sekvens af handlinger. I dette eksempel er problemet, at metoden ikke tager højde for at handling H_1 og H_3 tilsammen kan overflødiggøre handling H_2. Der skal et fuldstændigt lookahead[9] til, for fuldstændigt at tage højde for dette. Vi leder derfor efter en heuristik til beregning af handlingssekvensen, som bedre tager højde for sådan et tilfælde, hvor der er afhængige handlinger.

Vi har tidligere (i eksempel 1) beskrevet hvordan effektiviteten for en handling, der kan reparere flere komponenter, kan beregnes som summen af effektiviteterne for at reparere de enkelte komponenter. Dvs.:

$$Ef(H) = \frac{P(H = ja)}{C_H} = \frac{\sum_i P(H|K_i) \cdot P(K_i)}{C_H} \tag{2.9}$$

Hermed får en handling, der kan reparere flere komponenter, i praksis tildelt summen af de effektiviteter, som den har ved at reparere hver enkelt af disse komponenter, da:

$$Ef(H) = \frac{\sum_i P(H|K_i) \cdot P(K_i)}{C_H} = \sum_i \frac{P(H|K_i) \cdot P(K_i)}{C_H}$$

[9] beregning af ECR for alle mulige sekvenser af handlinger

Dette virker hensigtsmæssigt, da handlingen har en reel større sandsynlighed for at reparere systemet, end den ville have, hvis den f.eks. kun reparerede én af komponenterne.

Mens denne måde at beregne handlingernes effektiviteter fint tager højde for tilfældet, hvor en handling reparerer flere komponenter, så tager den ikke højde for tilfældet, hvor flere handlinger reparerer samme komponent. I dette tilfælde får hver af de handlinger, som kan reparere komponent K, fuld kredit for sin mulighed for at reparere K ved beregningen af deres effektiviteter. Resultatet er, at værdien af at reparere en komponent K bliver talt med det antal gange, som der er reparationshandlinger tilknyttet K. Dette er uhensigtsmæssigt, da K kun skal repareres én gang, for at den er ok, og dermed ikke forårsager at systemet er i fejltilstand.

Derfor ønsker vi at *fordele* værdien af at reparere komponenten mellem de tilknyttede handlinger. Dvs. vi ønsker at fordele den del af $Ef(H)$, som tilføres af en komponents reparation, mellem de handlinger, som kan reparere samme komponent. Dette gør vi ved at udvide formlen til beregning af en handlings effektivitet med en vægning. Vi fordeler *værdien af at reparere en komponent* mellem handlinger, som kan reparere samme komponent. Vi får så et "vægtet effektivitetsudtryk"[10] :

$$\begin{aligned} Ef(H) &= \frac{\sum_i P(H|K_i) \cdot P(K_i) \cdot V_i}{C_H} \text{ og} \\ V_i &= \frac{P(H|K_i)}{\sum_j P(H_j|K_i)}, \end{aligned}$$

hvor H_j er de handlinger, som reparerer K_i.

Hermed får vi effektivitetsudtrykket

$$Ef(H) = \frac{\sum_i P(H|K_i) \cdot P(K_i) \cdot \frac{P(H|K_i)}{\sum_j P(H_j|K_i)}}{C_H} \tag{2.10}$$

hvor $P(K_i)$ er sandsynligheden for at årsag K_i ligger til grund for problemet og hvor $P(H|K_i)$ er sandsynligheden for at handling H reparerer årsag K_i (givet at K_i er i årsag til fejltilstand). Med $P(H_j|K_i)$ tages kun de handlinger H_j i betragtning som reparerer årsag K_i.

Hvis en handling kan reparere flere komponenter, vil reparation af komponenter, der også repareres af andre handlinger være mindre vigtige. Dette tager vi højde for ved at vægte reparation af disse komponenter.

Effekten af vores vægtning er i hovedtræk at reparere systemet med færrest muligt handlinger. Dvs. *først* udføre de handlinger, der alligevel skal udføres, derefter alternative handlinger, der reparerer de fejlårsager, hvor de først valgte handlinger fejlede.

Grunden, til overhovedet at vægte, er at lave et hack, for at løse et NP-hard problem med en kendt heuristik, så der opnåes en (god) tilnærmet sekvens - beregnet i polynomiel tid.

[10] For nemheds skyld fortsætter vi med at kalde det for handlingens effektivitet, $Ef(H)$.

Udtrykket 2.10 giver den "normale" effektivitet vægtet med forholdet mellem de tilknyttede handlingers reparationssandsynlighed (vægtningen $\frac{P(H|K_i)}{\sum_j P(H_j|K_i)}$ er korrekt 1, hvis der kun er en handling på en fejlårsag). Ulempen er at fejlårsager med flere handlinger i nogle strukturkombinationer "straffes" i forhold til årsager med kun én handling. Dermed bliver reparationen af disse fejlårsager ofte skubbet til senere i sekvensen.

Målet med denne vægtning er at eliminere de handlinger fra sekvensen , som kan undværes, dvs. at undgå at udføre handlinger, som kan overflødiggøres ved udførelse af (kombinationer af) andre handlinger, dvs. at minimere ECR ved at minimere antallet af udførte handlinger. Dette opnås ved at lægge en prioritering på handlingerne, således at de handlinger, der alligevel skal udføres — idet der ikke er noget alternativ til dem — får en højere prioritet end handlinger, der reparerer komponenter, hvortil der er flere reparationsmuligheder.

Vi opnår denne prioritering ved at betragte reparation af en komponent som en værdi, der indgår i effektiviteten af en handling. Den værdi, det har at reparere en komponent, fordeles mellem de handlinger, der kan reparere komponenten - i forhold til sandsynlighederne for at handlingerne reparerer komponenten.

En konsekvens af denne vægtning, ved strukturer af typen som SACSO-eksemplet, er at de yderste handlinger ofte vil få en relativt højere prioritet, idet de hver især reparerer en komponent, der ikke repareres af andre handlinger.

Vægtningen skal være netop $\frac{P(H|K_i)}{\sum_j P(H_j|K_i)}$ med det formål at blandt handlinger, der reparerer samme komponent, skal de handlinger, der har størst reparationssandsynlighed prioriteres højest.

Dvs. $\frac{P(H|K_i)}{\sum_j P(H_j|K_i)}$ er en vægtning, der angiver en handlings effektivitet på en årsag i forhold til de andre handlinger, der også kan reparere denne årsag.

Eksempel 3 *Nu gentager vi eksempel 1 og 2, men med den ændring at vi bruger det vægtede effektivitetsudtryk fra 2.10, og den overordnede algoritme (som i eksempel 2) til beregningen af sekvensen af reparationshandlinger. Beregningen foretages ligesom i eksempel 2 ét skridt ad gangen, da handlingerne kan være afhængige. Først beregnes alle handlingernes effektiviteter, den bedste udtages, og evidens på denne handlings udførelse propageres. Dette gentages indtil systemet ikke længere er i fejltilstand.*

Der antages fortsat at alle handlinger er perfekte, dvs. $P(H|K_i) = 1$*, og har omkostningen* $C_H = 1$ *ved deres udførelse. Dermed bliver vægten* $V_i = \frac{P(H|K_i)}{\sum_j P(H_j|K_i)} = \frac{1}{Antal_H(K_i}$.

Følgende resultat fås:

$$Ef^0(H_1) = \frac{\frac{P(K_1)}{1}+\frac{P(K_2)}{2}}{1} = \frac{0.20}{1} + \frac{0.25}{2} = 0.200 + 0.125 = 0.325$$

$$Ef^0(H_2) = \frac{\frac{P(K_2)}{2}+\frac{P(K_3)}{2}}{1} = \frac{0.25}{2} + \frac{0.40}{2} = 0.125 + 0.200 = 0.325$$

$$Ef^0(H_3) = \frac{\frac{P(K_3)}{2}+\frac{P(K_4)}{1}}{1} = \frac{0.40}{2} + \frac{0.15}{1} = 0.200 + 0.150 = 0.350.$$

Handling H_3 *udtages som den optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med en sandsynlighed på 0,45.*

Anden runde af beregningen efter propagering, hhv. nulstilling af $P(K_3)$ og $P(K_4)$, der jo nu er repareret giver:
$Ef^1(H_1) = \frac{\frac{P(K_1)}{1}+\frac{P(K_2)}{2}}{1} = 0.200 + 0.125 = 0.325$
$Ef^1(H_2) = \frac{\frac{P(K_2)}{2}+\frac{P(K_3)}{2}}{1} = 0.125 + 0.000 = 0.125.$
Herefter vil H_1 blive udført, da den har den største effektivitet. H_1 reparerer komponenterne K_1 og K_2, og da vi stadig forudsætter perfekte handlinger, vil alle systemets komponenter - og dermed systemet - nu fungere, og dermed vil troubleshootingen stoppe.

Hermed er den bedste sekvens beregnet med propagering efter hver valgt / udført handling: $< H_3, H_1 >$. Dens ECR udregnes som:
$ECR(< H_3, H_1 >) = C_3 + C_1 \cdot P(H_3 = n)$, *dvs. at*
$ECR(< H_3, H_1 >) = 1 + 1(\frac{0.45}{1}) = 1.45.$
Denne ECR er lavere end for sekvensen, som vi fandt i eksempel 1 og 2, hvor $ECR = 1.55$, hhv. 1.50. Og i dette tilfælde finder vi den sekvens, som [Skaanning et al., 2000] og vi ved afsøgning af alle mulige sekvenser har fundet som den optimale - dvs. den med lavest ECR.

Udvidelse med usikkerhed

Perfekte handlinger er i praksis ikke altid realistiske. F.eks. kan genstart af en server i et computernet ofte løse et problem med at serveren hænger og dermed ikke svarer. Men genstart løser ikke i alle tilfælde problemet. Erfaringen kan have vist f.eks. at genstart løser problemet i 95% af tilfældene. Dermed er $P(Genstart = ja|\text{hænger}) = 0,95$. Vi ønsker derfor mulighed for at lade en handling være usikker, dvs. vi ønsker at kunne angive sandsynligheden for at en handling reparerer en given komponent, givet at denne komponent er i fejltilstand ($0 < P(H|K) \leq 1$).

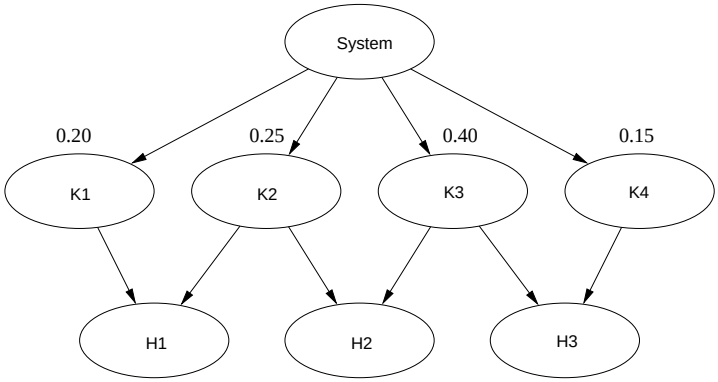
Fjernes antagelsen om at handlinger reparerer 100%, bruges stadig det oprindelige effektivitetsudtryk fra ligning 2.6. Blot er $P(H|K)$ ikke længere lig med 1, men i stedet er $0 < P(H|K) \leq 1$.

Dermed kan vi fortsat bruge det vægtede effektivitetesudtryk fra ligning 2.10:

$$Ef(H) = \frac{\sum_i P(H|K_i) \cdot P(K_i) \cdot \frac{P(H|K_i)}{\sum_j P(H_j|K_i)}}{C_H}$$

Vi gentager nu eksemplet fra eksempel 3, blot med den ændring at handlingerne ikke er perfekte.

Eksempel 4 *Vi bruger igen det vægtede effektivitetsudtryk fra 2.10, og den overordnede algoritme (som i eksempel 2) til beregningen af sekvensen af reparationshandlinger. Beregningen foretages ligesom i eksempel 2 ét skridt ad gangen, da handlingerne kan være afhængige. Først beregnes alle handlingernes effektiviteter, den bedste udtages, og evidens på denne handlings udførelse propageres. Dette gentages indtil systemet ikke længere er i fejltilstand.*



Figur 2.7.: Eksempel på system med afhængige handlinger.

Der antages at handlingerne har følgende sandsynligheder for at reparere systemet:
$P(H_1 = ja|K_1) = 0,95$ *og* $P(H_1 = ja|K_2) = 0,95$,
$P(H_2 = ja|K_2) = 0,92$ *og* $P(H_2 = ja|K_3) = 0,90$,
$P(H_3 = ja|K_3) = 0,96$ *og* $P(H_3 = ja|K_4) = 0,94$,
og omkostningen $C_H = 1$ *ved deres udførelse.*

Følgende resultat fås:

$$Ef^0(H_1) = \frac{P(H_1=ja|K_1)P(K_1)\frac{P(H_1|K_1)}{\sum_j P(H_j|K_1)}+P(H_1=ja|K_2)P(K_2)\frac{P(H_1|K_2)}{\sum_j P(H_j|K_2)}}{1}$$
$$= 0,95 \cdot 0.20\frac{0,95}{0,95} + 0,95 \cdot 0.25\frac{0,95}{0,95+0,92} = 0.190 + 0.121 = 0.311$$

$$Ef^0(H_2) = \frac{P(H_2=ja|K_2)P(K_2)\frac{P(H_2|K_2)}{\sum_j P(H_j|K_2)}+P(H_2=ja|K_3)P(K_3)\frac{P(H_2|K_3)}{\sum_j P(H_j|K_3)}}{1}$$
$$= 0,92 \cdot 0.25\frac{0,92}{0,95+0,92} + 0,90 \cdot 0.40\frac{0,90}{0,90+0,96} = 0.113 + 0.174 = 0.287$$

$$Ef^0(H_3) = \frac{P(H_3=ja|K_3)P(K_3)\frac{P(H_3|K_3)}{\sum_j P(H_j|K_3)}+P(H_3=ja|K_4)P(K_4)\frac{P(H_3|K_4)}{\sum_j P(H_j|K_4)}}{1}$$
$$= 0,96 \cdot 0.40\frac{0,96}{0,90+0,96} + 0,94 \cdot 0.15\frac{0,94}{0,94} = 0.198 + 0.141 = 0.339.$$

Handling H_3 *udtages som den optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med en sandsynlighed på 0,475. Efter propagering af udførelsen af* H_3 *er:*
$P(K_1) = 0,4211$
$P(K_2) = 0,5263$
$P(K_3) = 0,0337$ *og*
$P(K_4) = 0,0189$.

Anden runde af beregningen giver så:

$$Ef^1(H_1) = \frac{P(H_1=ja|K_1)P(K_1)\frac{P(H_1|K_1)}{\sum_j P(H_j|K_1)}+P(H_1=ja|K_2)P(K_2)\frac{P(H_1|K_2)}{\sum_j P(H_j|K_2)}}{1}$$
$$= 0,95 \cdot 0.4211\frac{0,95}{0,95} + 0,95 \cdot 0.5263\frac{0,95}{0,95+0,92} = 0.400 + 0.254 = 0.654$$

$$Ef^1(H_2) = \frac{P(H_2=ja|K_2)P(K_2)\frac{P(H_2|K_2)}{\sum_j P(H_j|K_2)}+P(H_2=ja|K_3)P(K_3)\frac{P(H_2|K_3)}{\sum_j P(H_j|K_3}}{1}$$
$$= 0,92 \cdot 0.5263\frac{0,92}{0,95+0,92} + 0,90 \cdot 0.0337\frac{0,90}{0,90} = 0.2589 + 0.0303 = 0.2892$$

Herefter vil H_1 blive udført, da den har den største effektivitet. Systemet er nu i fejltilstand med sandsynligheden 0,0475. Efter propagering af udførelsen af H_1 er:
$P(K_1) = 0,2105$
$P(K_2) = 0,2632$
$P(K_3) = 0,3368$ *og*
$P(K_4) = 0,1895$.

Nu er der kun handling H_2 tilbage at udføre. Den har nu effektiviteten
$Ef^2(H_2) = \frac{P(H_2=ja|K_2)P(K_2)\frac{P(H_2|K_2)}{\sum_j P(H_j|K_2)} + P(H_2=ja|K_3)P(K_3)\frac{P(H_2|K_3)}{\sum_j P(H_j|K_3)}}{1}$
$= 0,92 \cdot 0.2632\frac{0,92}{0,92} + 0,90 \cdot 0.3368\frac{0,90}{0,90} = 0.5453$
Efter udførelsen af H_2 er systemet med sandsynligheden 0,0216 stadig i fejltilstand.

Hermed er den bedste sekvens beregnet med propagering efter hver udført handling: $< H_3, H_1, H_2 >$. Dens ECR udregnes som:
$ECR(< H_3, H_1H_2 >) = C_3 + C_1 \cdot P(H_3 = n) + C_2 \cdot P(H_3 = n, H_1 = n)$, *dvs. at*
$ECR(< H_3, H_1H_2 >) = 1 + 1(\frac{0.475}{1} + 1(\frac{0.0475}{1}) = 1.5225$.
Ved udtømmende søgning fandt vi også denne sekvens som den optimale.

For at vurdere vægtningens effekt, gentages eksemplet uden vægtning:

Eksempel 5 *Vi bruger nu effektivitetsudtryket fra 2.9:*

$$Ef(H) = \frac{P(H = ja)}{C_H} = \frac{\sum_i P(H|K_i) \cdot P(K_i)}{C_H},$$

og stadig den overordnede algoritme (som i eksempel 2) til beregningen af sekvensen af reparationshandlinger.

Der antages stadig at handlingerne har reparationssandsynlighederne:
$P(H_1 = ja|K_1) = 0,95$
$P(H_1 = ja|K_2) = 0,95$
$P(H_2 = ja|K_2) = 0,92$
$P(H_2 = ja|K_3) = 0,90$
$P(H_3 = ja|K_3) = 0,96$
$P(H_3 = ja|K_4) = 0,94$,
og omkostningen $C_H = 1$ ved deres udførelse.

Følgende resultat fås:
$Ef^0(H_1) = \frac{P(H_1=ja|K_1)P(K_1)+P(H_1=ja|K_2)P(K_2)}{1}$
$= 0,95 \cdot 0.20 + 0,95 \cdot 0.25 = 0.4275$
$Ef^0(H_2) = \frac{P(H_2=ja|K_2)P(K_2)+P(H_2=ja|K_3)P(K_3)}{1}$
$= 0,92 \cdot 0.25 + 0,90 \cdot 0.40 = 0.5900$
$Ef^0(H_3) = \frac{P(H_3=ja|K_3)P(K_3)+P(H_3=ja|K_4)P(K_4)}{1}$
$= 0,96 \cdot 0.40 + 0,94 \cdot 0.15 = 0.5250$.

Handling H_2 udtages som den optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med en sandsynlighed på 0,41. Efter propagering af udførelsen af H_2 er:
$P(K_1) = 0,4878$
$P(K_2) = 0,0488$
$P(K_3) = 0,0976$ *og*
$P(K_4) = 0,3659$.

Anden runde af beregningen giver så:
$Ef^1(H_1) = \frac{P(H_1=ja|K_1)P(K_1)+P(H_1=ja|K_2)P(K_2)}{1}$
$= 0,95 \cdot 0.4878 + 0,95 \cdot 0.0488 = 0.5098$
$Ef^1(H_3) = \frac{P(H_3=ja|K_3)P(K_3)+P(H_3=ja|K_4)P(K_4)}{1}$
$= 0,96 \cdot 0.0976 + 0,94 \cdot 0.3659 = 0.4376$.

Herefter vil H_1 blive udført, da den har den største effektivitet. Systemet er stadig i fejltilstand med sandsynligheden 0,201. Efter propagering af udførelsen af H_1 er:
$P(K_1) = 0,0498$
$P(K_2) = 0,0050$
$P(K_3) = 0,1990$ *og*
$P(K_4) = 0,7463$.

Nu er der kun handling H_3 tilbage at udføre. Den har nu effektiviteten
$Ef^2(H_3) = \frac{P(H_3=ja|K_3)P(K_3)+P(H_3=ja|K_4)P(K_4)}{1}$
$= 0,96 \cdot 0.1990 + 0,94 \cdot 0.7463 = 0.8926$.
Efter udførelsen af H_3 er systemet med sandsynligheden 0,0216 stadig i fejltilstand.

Hermed er den bedste sekvens: $< H_2, H_1, H_3 >$. Dens ECR udregnes som:
$ECR(< H_2, H_1, H_3 >) = C_2 + C_1 \cdot P(H_2 = n) + C_3 \cdot P(H_2 = n, H_1 = n)$, *dvs. at*
$ECR(< H_2, H_1, H_3 >) = 1 + 1(\frac{0.41}{1} + 1(\frac{0.201}{1}) = 1.611$.
Dermed er denne sekvens dårligere, end den vi fandt med det vægtede effektivitetsudtryk.

2.3. Uafhængige fejl

Uafhængige fejl[11] med uafhængige handlinger er detaljeret gennemgået i [Srinivas, 1995]. Vi udvider denne fremgangsmåde til at omfatte handlinger der er usikre[12], og vi forsøger at udvide den med afhængige handlinger. Uafhængige handlinger er ifølge definition 1 side 12 handlinger, som reparerer *forskellige* fejlårsager (komponenter).

Vi starter med at beskrive forudsætningerne og antagelserne for uafhængige fejl (bl.a. i forhold til single-fault antagelsen). Derefter udvider vi med at reparationshandlinger ikke reparerer med 100% sikkerhed og til sidst forsøger vi at udvide handlinger til at kunne være afhængige.

[11] dvs. en eller flere fejl, hvis tilstedeværelse ikke påvirker hinanden
[12] Usikre handlinger er handlinger, der ikke reparerer de tilknyttede fejlårsager med 100% sandsynlighed

2.3.1. Fejlfinding med uafhængige fejl og uafhængige handlinger

Dette afsnit giver et overblik over hvordan troubleshooting i et system med uafhængige fejlårsager og uafhængige og perfekte reperationshandlinger kan håndteres på en beregningsmæssigt effektiv måde. Afsnittet er baseret på [Srinivas, 1995].

[Srinivas, 1995] har fundet og beskrevet en metode til at beregne en sekvens af handlinger, der med den lavest mulige ECR reparerer et system med multiple uafhængige fejl. Beregningsmetoden er eksakt og udføres i lineær tid. Beregningen af den optimale sekvens (den med den totalt laveste ECR) bliver af polynomiel kompleksitet, hvis observationer tilføjes, men dem behandler vi, som nævnt tidligere, ikke i dette projekt.

Metoden til at finde den optimale sekvens baseres på følgende antagelser om strukturen af systemmodellen: I [Srinivas, 1995] er der til hver fejlårsag tilknyttet netop en handling. Handlingerne reparerer den tilknyttede fejlårsag med 100% sikkerhed (hvilket svarer til at udskifte komponenten med en ny). Omkostningen for en handling er uafhængig af historien, dvs. uafhængig af handlingens placering i sekvensen, og fejlårsagerne er uafhængige. Metoden kan dog ligeså godt udledes med uafhængige handlinger (én handling kan evt. reparere flere komponenter), pga. antagelsen om perfekte handlinger. Når $P(H|K) = 1$, er $P(H) = P(K)$, da sandsynligheden for at handling H vil reparere komponent K er $P(H) = P(H|K) \cdot P(K)$.

Vi benytter bayesianske netværk til at repræsentere modellerne, idet de tillader mere frihed med hensyn til modelleringen af et problemområde, end det komponenthierarki, som [Srinivas, 1995] benytter. Vi ønsker at udnytte denne frihed til at udvide metoden til at omfatte afhængige handlinger, og i afsnit 2.4 afhængige fejl. Bayesianske net er langt mere udtryksfulde end de i [Srinivas, 1995] anvendte komponenthierarkier. Afhængige handlinger og afhængige årsager kan modelleres med bayesianske net, men ikke i komponenthierarkiet.

Nu viser vi effektivitetsudtrykket, som bruges i metoden i [Srinivas, 1995], men med udgangspunkt i den repræsentation og notation, som vi anvender i dette projekt.

Antag at vi ser på et system med n komponenter, dvs. komponenterne K_i for $1 \leq i \leq n$. Hver komponent kan være enten ok eller i fejltilstand, $K_i = \{ok, f\}$. Vi har en fælles sandsynlighedsfordeling for komponenterne $P(K_1, K_2, \ldots, K_n)$ givet i systemmodellen (der som tidligere beskrevet er repræsenteret som et bayesiansk net). Vi har én reparationshandling H_i pr. komponent K_i (svarende til udskiftning af komponenten), som reparerer komponenten med 100% sikkerhed. Generelt er sandsynligheden for at handling H vil reparere komponent K lig $P(H) = P(H|K) \cdot P(K)$. Pga. antagelsen om perfekte handlinger, dvs. at $P(H|K) = 1$, er $P(H) = P(K)$. Vi har også en fast omkostning C_i forbundet med udførelsen af hver enkelt handling H_i.

Hvis alle komponenter er ok, er systemet ok, $Sys = ok$. Hvis en eller flere komponenter er i fejltilstand, er systemet i fejltilstand, $Sys = f$. Som ved single-fault observeres systemets tilstand efter hver udført handling, og troubleshootingen stoppes, når systemet ikke længere er i fejltilstand.

Betragt en troubleshootingstrategi, som er en sekvens af sekvens af reparationshandlingerne $S =< H_1, H_2, \ldots, H_n >$. Den forventede reparationsomkostning ved brug af denne handlingssekvens er ifølge ligning 2.1 side 8

$$ECR(S) = C_1 + \sum_{k=2}^{n} P(H_1, \ldots, H_{k-1}|sys = f) \cdot C_k.$$

$P(H_1, \ldots, H_{k-1}|sys = f)$ er sandsynligheden for at systemet stadig er i fejltilstand efter at handlingerne $H_1, \ldots, H_{k-1}$ er udført. Da vi forudsætter, at systemet stadig er i fejltilstand (ellers ville troubleshootingen stoppe her), ved vi at der stadig må være mindst én fejlfungerende ikke repareret komponent i systemet. Pga. forudsætningen om perfekte handlinger, ved vi at komponenterne $\{K_1, K_2, \ldots, K_{k-1}\}$ alle er ok. Altså må mindst én af komponenterne $\{K_k, K_{k-1}, \ldots, K_n\}$ være i fejltilstand. Derfor er $P(H_1, \ldots, H_{k-1}|sys = f) = (1 - P(K_k = ok, K_{k+1} = ok, \ldots, K_n = ok)$. Vi bruger nu notationen $P(K_{[k,n]} = ok)$ for $P(K_k = ok, K_{k+1} = ok, \ldots, K_n = ok)$. Vi kan nu skrive $ECR(S)$ som

$$ECR(S) = C_1 + \sum_{k=2}^{n} C_k \cdot (1 - P(K_{[k,n]} = ok)).$$

Dermed er

$$ECR(S) = \sum_{k=1}^{n} C_k \cdot (1 - P(K_{[k,n]} = ok)), \tag{2.11}$$

da $P(K_{[1,n]} = ok) = 0$, ellers var systemet ikke i fejltilstand ved troubleshootingens start. Dette ECR-udtryk gælder under forudsætning af at handlingerne er perfekte.

Nu viser vi betingelsen, som skal sikre at strategien er optimal, dvs. at den har lavest mulig ECR.

Betragt sekvensen $S^j =< H_1, H_2, \ldots, H_j, H_{j+1}, \ldots, H_n >$. Lad sekvensen S^{j+1} være identisk med S^j undtagen at handlingerne H_j og H_{j+1} er byttet om. Vi sammenligner nu ECR for S^j og S^{j+1}.

$$\begin{aligned} & ECR(S^j) - ECR(S^{j+1}) \qquad (2.12) \\ = \; & (C_j(1 - P(K_{[j,n]} = ok)) + C_{j+1}(1 - P(K_{[j+1,n]} = ok))) \\ & -(C_{j+1}(1 - P(K_{[j,n]} = ok)) + C_j(1 - P(K_j = ok, K_{[j+2,n]} = ok))) \end{aligned}$$

Sekvens S^j er billigere end S^{j+1} hvis $ECR(S^j) - ECR(S^{j+1}) \leq 0$. Vi bruger nu udtrykket

R_{ok} for $K_{[j+2,n]} = ok$. Nu får vi

$$\begin{aligned}
&(C_j(1 - P(K_j = ok, K_{j+1} = ok, R_{ok})) + C_{j+1}(1 - P(K_{j+1} = ok, R_{ok})) \\
&-(C_{j+1}(1 - P(K_j = ok, K_{j+1} = ok, R_{ok})) + C_j(1 - P(K_j = ok, R_{ok}))) \leq \quad 0 \\
&\Updownarrow \\
&C_j(1 - P(K_j = ok, K_{j+1} = ok, R_{ok})) - C_j(1 - P(K_j = ok, R_{ok})) \\
&+C_{j+1}(1 - P(K_{j+1} = ok, R_{ok})) - C_{j+1}(1 - P(K_j = ok, K_{j+1} = ok, R_{ok})) \leq \quad 0 \\
&\Updownarrow \\
&C_j(-P(K_j = ok, K_{j+1} = ok, R_{ok}) + P(K_j = ok, R_{ok})) \\
&+C_{j+1}(-P(K_{j+1} = ok, R_{ok}) + P(K_j = ok, K_{j+1} = ok, R_{ok})) \leq \quad 0 \\
&\Updownarrow \\
&C_j(P(K_j = ok, R_{ok}) - P(K_j = ok, K_{j+1} = ok, R_{ok})) \\
&\leq C_{j+1}(P(K_{j+1} = ok, R_{ok}) - P(K_j = ok, K_{j+1} = ok, R_{ok})) \qquad (2.13)
\end{aligned}$$

Hermed kan vi, givet en sekvens af reparationshandlinger S og en sandsynlighedsfordeling over fejlsandsynlighederne for systemets komponenter $P(K_1, K_2, \ldots, K_n)$, kontrollere hvorvidt strategien er et lokalt optimum ved at kontrollere om ulighed 2.13 holder for alle par af nabohandlinger i sekvensen. Hvis den holder, vil det altid føre til en højere ECR at ombytte to nabohandlinger.

Vi antager, at hver komponent K_i kan fejle uafhængigt med sandsynligheden P_i ($= P(K_i)$), dvs. at $P(K_i = f) = P_i$. Se nu på første led i ligning 2.13. Da vi har multiple uafhængige fejl, har vi at

$$P(K_j = ok, R_{ok}) = P(K_j = ok) \cdot P(R_{ok}) = (1 - P_j) \cdot P(R_{ok}).$$

På samme måde får vi at

$$P(K_j = ok, K_{j+1} = ok, R_{ok}) = (1 - P_j)(1 - P_{j+1}) \cdot P(R_{ok}).$$

Ligning 2.13 kan nu omskrives til

$$\begin{aligned}
&C_j((1 - P_j)P(R_{ok}) - (1 - P_j)(1 - P_{j+1})P(R_{ok})) \\
&\leq C_{j+1}((1 - P_{j+1})P(R_{ok}) - (1 - P_j)(1 - P_{j+1})P(R_{ok})) \\
&\Updownarrow \\
&P(R_{ok})C_j((1 - P_j) - (1 - P_j)(1 - P_{j+1})) \\
&\leq P(R_{ok})C_{j+1}((1 - P_{j+1}) - (1 - P_j)(1 - P_{j+1})) \\
&\Updownarrow \\
&P(R_{ok})C_j(1 - P_j)(1 - (1 - P_{j+1})) \leq P(R_{ok})C_{j+1}(1 - P_{j+1})(1 - (1 - P_j)) \\
&\Updownarrow \\
&P(R_{ok})C_j(1 - P_j)P_{j+1} \leq P(R_{ok})C_{j+1}(1 - P_{j+1})P_j
\end{aligned}$$

Dermed kan ligning 2.13 reduceres til

$$C_j \frac{1-P_j}{P_j} \leq C_{j+1} \frac{1-P_{j+1}}{P_{j+1}}. \tag{2.14}$$

Dvs. at når der gælder uafhængige fejl og perfekte uafhængige handlinger, så er sekvens S^j billigere end S^{j+1} hvis ulighed 2.14 er opfyldt.

Vha. ligning 2.14 kan vi beregne den globalt[13] optimale strategi ved at sortere reparationshandlingerne H_j efter efter stigende $\frac{P_j}{(1-P_j)C_j}$.

I [Srinivas, 1995] betyder udførelse af en handling, at komponenten udskiftes (dvs. at handlinger reparerer 100%). Dvs. i modellen angives sandsynligheden for at komponenten er i fejltilstand og dette bliver så sandsynligheden for at handlingen vil reparere denne komponent. I effektivitetsudtrykket $Ef(H) = \frac{P}{(1-P)C}$, er P sandsynligheden for at handlingen reparerer komponenten. Dette svarer i [Srinivas, 1995] til $P(K)$, idet $P(H|K) = 1$ og fordi $P = P(H|K) \cdot P(K)$.I domæner, hvor det er muligt at antage uafhængige handlinger, er det en stor fordel af bygge på denne antagelse, da det er nemt at bevise og regne på, og gør det muligt at finde sekvensen eksakt. Ved problemsituationer, hvor komponenter (subkomponenter) blot udskiftes, er det tilstrækkeligt at modellere handlingerne som uafhængige.

Udvidelse med usikkerhed

Det er ikke realistisk at forvente, at alle reparationshandlinger 100% sikkert reparerer de tilhørende komponenter, som angivet i [Srinivas, 1995]. Derfor ønsker vi at kunne angive en sandsynlighed, $P(H|K)$, hvormed handlingen H reparerer komponenten K, givet at komponenten er i fejltilstand.

Da vi antog single-fault, kunne vi stadig bruge effektivitetsudtrykket $\frac{P}{C}$, når vi antog ikke perfekte handlinger, da vi kunne udlede $Ef(H) = \frac{P}{C}$ uden at antage sikre handlinger. Det samme er ikke tilfældet her. For at nå frem til ECR-udtrykket i ligning 2.11, som er grundlaget for udledningen af $Ef(H_i) = \frac{P_i}{(1-P_i)C_i}$ for systemer med multiple uafhængige fejl, brugte vi antagelsen om perfekte handlinger. Dermed ved vi ikke om effektivitetsudtrykket holder, når vi erstatter $P(H_i) = 1 \cdot P(K_i)$ (fra side 26) med $P(H_i|K_i) \cdot P(K_i)$, hvor $0 < P(H_i|K_i) \leq 1$. Vi tror at det stadig holder, men vi kan ikke for nuværende bevise det. Ifølge vores tests fremkommer altid optimale sekvenser ved brug af $\frac{P}{(1-P)C}$ ved multiple uafhængige fejl og usikre uafhængige handlinger. Derfor antager vi, at det holder, og bruger effektivitetsudtrykket

$$Ef(H) = \frac{P(H|K) \cdot P(K)}{(1 - P(H|K) \cdot P(K))C} \tag{2.15}$$

Hvis dette effektivitetsudtryk modsat vores forventning ikke er korrekt med usikre handlinger, vil det stadig være tæt på korrekt når $P(H|K)$ er tæt på 1. Og i praksis vil der sjældent blivet medtaget handlinger med lave reparationssandsynligheder i et troubleshootingsystem (de vil normalt kun komme med, hvis de er meget billige at udføre).

[13] globalt da enhver ombytning af handlinger er mulig vha. parvise naboombytninger.

2.3.2. Afhængige handlinger

Som beskrevet i afsnit 2.2.2, er en eksakt beregning af den optimale sekvens af reparationshandlinger for et system med afhængige handlinger[14] NP-hard.

Udvidelse fra uafhængige handlinger

Udtrykket $Ef(H) = \frac{P}{(1-P)C}$ for uafhængige handlinger, hvor $P = P(H = ja)$, udvider vi først til at omfatte at en handling evt. reparerer flere komponenter. P kan for en handling, der reparerer to komponenter, K_1 og K_2, beregnes som

$$\begin{aligned} P(H = ja) \quad = \quad & P(H = ja|K_1 = f, K_2 = f) \cdot P(K_1 = f) \cdot P(K_2 = f) \\ & + P(H = ja|K_1 = ok, K_2 = f) \cdot P(K_1 = ok) \cdot P(K_2 = f) \\ & + P(H = ja|K_1 = f, K_2 = ok) \cdot P(K_1 = f) \cdot P(K_2 = ok) \end{aligned} \quad (2.16)$$

under forudsætning af at K_1 og K_2 er indbyrdes uafhængige, denne forudsætning er opfyldt sålænge der ikke er evidens på systemknuden. Ved test har det vist sig, at sekvenser fundet hhv. med og uden evidens på *System* er ens. Evidens på system er nødvendig ved beregning af ECR. Vi benytter udtryk 2.16 ved udledning af det vægtede udtryk, og bruger $P(H = ja)$ fra Hugin i eksemplerne. For en handling, der reparerer flere end to komponenter, beregnes $P(H = ja)$ på samme måde: som en sum over alle de led, hvor mindst én af komponenterne er i fejltilstand (hvis alle komponenter er ok, er $P(H = ja|K_1 = ok, \ldots K_n = ok) = 0$, og leddet bliver dermed 0). Hvert led består af $P(H = ja|K_1 =?, \ldots K_n =?)$ ganget med de tilhørende $P(K_1 =?) \cdots P(K_n =?)$ (hvor $? \in \{ok, f\}$ er den tilstand, som den enkelte komponent er i, i den givne permutation af leddet).

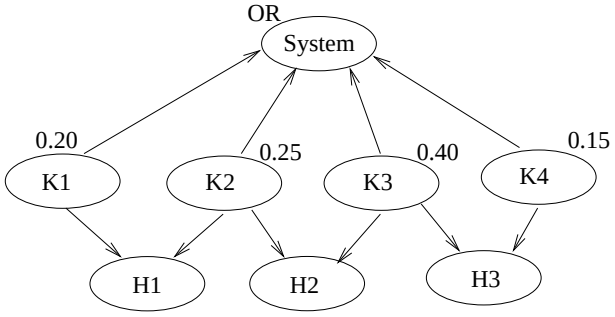
Hermed får vi, for en handling H, der reparerer K_1 og K_2, effektivitetsudtrykket

$$Ef(H) = \frac{P(H = ja)}{(1 - P(H = ja))C}. \quad (2.17)$$

Da en eksakt beregning af den optimale sekvens af reparationshandlinger for et system med afhængige handlinger, som lige nævnt, er NP-hard, kan ligning 2.17 kun være et mere eller mindre godt tilnærmet effektivitetsudtryk. For at få et første indtryk af hvordan det fungerer, bruger vi det på et eksempel. Vi genbruger SACSO-eksemplet fra tidligere, dog med den ændring at vi nu har modelleret det med multiple uafhængige fejl (se figur 2.8). Denne modellering medfører at komponenterne K_1, K_2, K_3 og K_4 er forældre til knuden System. For nemheds skyld har vi ikke omregnet komponenternes betingede sandsynligheder, som i de tidligere eksempler var opgivet som $P(K_i = f|Sys = f)$ til marginale sandsynligheder $P(K_i = f)$. I stedet bruger vi værdierne fra de tidligere betingede sandsynligheder: { 0.20, 0.25, 0.40, 0.15 }, men nu som marginale sandsynligheder. Det er ikke væsentligt at bruge de samme fejlsandsynligheder for komponenterne, som vi brugte

[14] Ifølge definition 1: handlinger, som reparerer samme fejlårsag.

under single-fault, da vi ikke ønsker at sammenligne dette eksempel med single-fault eksemplerne. I stedet for at beregne effektiviteterne manuelt med ligning 2.16 udtrækker vi $P(H = ja)$ vha. Hugin, og indsætter dem i ligning 2.17.



Figur 2.8.: Eksempel på system med afhængige handlinger.

Eksempel 6 *Vi bruger nu effektivitetsudtrykket fra ligning 2.17 sammen med den generelle heuristik beskrevet i afsnit 2.1.3 på den ovenfor viste model. Vi antager perfekte reparationshandlinger, dvs. at* $P(H|K) = 1$*, og at omkostningerne for handlingerne* H_1 *og* H_3 *er 1 samt at handling* H_2 *har omkostningen 1.1.*

Med evidens på $Sys = f$ *er* $P(e^0) = 0.694$

Fra starten, hvor vi kun har evidens på at $Sys = f$*, har handlingerne følgende effektiviteter:*

$Ef^0(H_1) = \frac{P(H_1=ja)}{(1-P(H_1=ja))\cdot C_{H_1}} = \frac{0.576}{(1-(0.576))\cdot 1} = 1.358$

$Ef^0(H_2) = \frac{P(H_2=ja)}{(1-P(H_2=ja))\cdot C_{H_2}} = \frac{0.793}{(1-(0.793))\cdot 1.1} = 3.483$

$Ef^0(H_3) = \frac{P(H_3=ja)}{(1-P(H_3=ja))\cdot C_{H_3}} = \frac{0.706}{(1-(0.706))\cdot 1} = 2.401$

Handling 2 udvælges som den bedste og indsættes i sekvensen. Systemet er stadig i fejltilstand med sandsynligheden $\frac{P(e^1)}{P(e^0)} = \frac{0.144}{0.694} = 0.207$*.*

Nu er H_1 *og* H_3*'s effektiviteter er*

$Ef^1(H_1) = \frac{P(H_1=ja)}{(1-P(H_1=ja))\cdot C_{H_1}} = \frac{0.625}{(1-(0.625))\cdot 1} = 1.666$

$Ef^1(H_3) = \frac{P(H_3=ja)}{(1-P(H_3=ja))\cdot C_{H_3}} = \frac{0.531}{(1-(0.531))\cdot 1} = 1.132$

Handling 1 udføres som den (næste) optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med sandsynligheden $\frac{P(e^2)}{P(e^0)} = \frac{0.0545}{0.694} = 0.0778$

Nu har handling H_3 *effektiviteten*

$Ef^2(H_3) = \frac{P(H_3=ja)}{(1-P(H_3=ja))\cdot C_{H_3}} = \frac{1.0}{(1-(1.0))\cdot 1} = \frac{1}{0}$ *(uendelig stor),*

og udføres. Nu er systemet med sandsynligheden $\frac{P(e^2)}{P(e^0)} = \frac{0}{0.694} = 0$ *stadig i fejltilstand.*

Den fundne sekvens af reparationshandlinger er $< H_2, H_1, H_3 >$ *med forventet repara-*

tionsomkostning:
$ECR(< H_2, H_1, H_3 >) = C_2 + C_1 \cdot P(H_2 = n) + C_3 \cdot P(H_2 = n, H_1 = n) = 1.1 + 1 \cdot 0.207 + 1 \cdot 0.0778 = 1.385$.

Den optimale sekvens finder vi med udtømmende søgning som $< H_3, H_1 >$ *med forventet reparationsomkostning*
$ECR(< H_3, H_1 >) = C_3 + C_1 \cdot P(H_3 = n) + C_2 \cdot P(H_3 = n, H_1 = n) = 1 + 1 \cdot 0.294 + 1 \cdot 0 = 1.294$.

Dermed fandt vi ikke, med dette effektivitetsudtryk, den optimale sekvens i dette tilfælde. Som i eksempel 2 tager dette effektivitetsudtryk heller ikke højde for at handling H_1 og H_3 tilsammen kan overflødiggøre handling H_2. Igen skal der et fuldstændigt look-ahead til, for helt at tage højde for dette.

Vi vil derfor udvide effektivitetsudtrykket med den samme vægtning, som vi brugte i single-fault tilfældet i afsnit 2.2.2. "Værdien" af at reparere en komponent fordeles med vægten $\frac{P(H|K_i)}{\sum_j P(H_j|K_i)}$ imellem de handlinger, der kan reparere komponenten. Men vi kan ikke vægte med den vægtning på effektivitetsudtrykket 2.17 ($Ef(H) = \frac{P(H=ja)}{(1-P(H=ja))C}$).

I stedet erstatter vi $P(H = Ja)$ med udtrykket fra 2.16:

$$\begin{aligned} P(H = ja) \quad = \quad & P(H = ja|K_1 = f, K_2 = f) \cdot P(K_1 = f) \cdot P(K_2 = f) \\ & + P(H = ja|K_1 = ok, K_2 = f) \cdot P(K_1 = ok) \cdot P(K_2 = f) \\ & + P(H = ja|K_1 = f, K_2 = ok) \cdot P(K_1 = f) \cdot P(K_2 = ok), \end{aligned}$$

og ganger derefter vægtene på sandsynlighederne for at komponenterne er i fejltilstand $P(K_i = f)$. Dette er det udtryk, der anvendes ved den maskinelle sammenligning af udtrykkene (i testafsnittet), hvor der testes med det vægtede Srinivas udtryk.

Da effektivitetsudtrykket fra 2.17 i forvejen er et tilnærmet udtryk, vælger vi i næste eksempel (eksempel 7) at simplificere udtrykket for $P(H = ja)$ til $\sum_i P(K_i) \cdot P(H|K_i)$. Med handlinger, som reparerer flere komponenter, bliver fejlen ved denne approximation, at handlingens sandsynlighed for at reparere kombinationer af komponenterne ikke medregnes. Dermed bliver fejlen i princippet større, jo flere komponenter en handling kan reparere.

Hermed får vi effektivitetsudtrykket:

$$Ef(H) = \frac{\sum_i P(K_i) \cdot P(H|K_i) \cdot \frac{P(H|K_i)}{\sum_j P(H_j|K_i)}}{(1 - \sum_i P(K_i) \cdot P(H|K_i) \frac{P(H|K_i)}{\sum_j P(H_j|K_i)}) \cdot C_H} \tag{2.18}$$

Argumenterne for vægtningen er de samme som i afsnit 2.2.2.

Målet med denne vægtning er stadig at eliminere de handlinger fra sekvensen, som kan undværes, dvs. at undgå at udføre handlinger, som kan overflødiggøres ved udførelse af (kombinationer af) andre handlinger.

En konsekvens af denne vægtning, ved strukturer af typen som SACSO-eksemplet, er at de yderste handlinger ofte vil få en relativt højere prioritet, idet de hver især reparerer en komponent, der ikke repareres af andre handlinger.

Vi gentager nu eksempel 6 under brug af effektivitetsudtrykket fra ligning 2.18.

Eksempel 7 *Vi antager stadig perfekte reparationshandlinger, dvs. at* $P(H|K) = 1$*, og at omkostningerne for handlingerne* H_1 *og* H_3 *er 1, handling* H_2 *har omkostningen 1.1.*

Med evidens på $Sys = f$ *er* $P(e^0) = 0.694$ *og*
$P(K_1 = f|e^0) = 0.288$,
$P(K_2 = f|e^0) = 0.360$,
$P(K_3 = f|e^0) = 0.576$ *og*
$P(K_4 = f|e^0) = 0.216$.

Fra starten, hvor vi kun har evidens på at $Sys = f$*, har handlingerne følgende effektiviteter:*

$$Ef^0(H_1) = \frac{P(K_1)P(H_1|K_1)\frac{P(H_1|K_1)}{P(H_1|K_1)} + P(K_2)P(H_1|K_2)\frac{P(H_1|K_2)}{P(H_1|K_2)+P(H_2|K_2)}}{(1-(P(K_1)P(H_1|K_1)\frac{P(H_1|K_1)}{P(H_1|K_1)} + P(K_2)P(H_1|K_2)\frac{P(H_1|K_2)}{P(H_1|K_2)+P(H_2|K_2)}))\cdot C_{H_1}}$$

$$= \frac{P(K_1)1\frac{1}{1} + P(K_2)1\frac{1}{1+1}}{(1-(P(K_1)1\frac{1}{1} + P(K_2)1\frac{1}{1+1}))\cdot C_{H_1}} = \frac{0.288+0.360\cdot\frac{1}{2}}{(1-(0.288+0.360\cdot\frac{1}{2}))\cdot 1} = 0.884$$

$$Ef^0(H_2) = \frac{P(K_2)\cdot\frac{1}{2} + P(K_3)\cdot\frac{1}{2}}{(1-(P(K_2)\cdot\frac{1}{2} + P(K_3)\cdot\frac{1}{2}))\cdot C_{H_2}} = \frac{0.360\cdot\frac{1}{2} + 0.576\cdot\frac{1}{2}}{(1-(0.360\cdot\frac{1}{2} + 0.576\cdot\frac{1}{2}))\cdot 1.1} = 0.802$$

$$Ef^0(H_3) = \frac{P(K_3)\cdot\frac{1}{2} + P(K_4)\cdot\frac{1}{1}}{(1-(P(K_3)\cdot\frac{1}{2} + P(K_4)\cdot\frac{1}{1}))\cdot C_{H_3}} = \frac{0.576\cdot\frac{1}{2} + 0.216\cdot\frac{1}{1}}{(1-(0.576\cdot\frac{1}{2} + 0.216\cdot\frac{1}{1}))\cdot 1} = 1.017$$

Handling 3 udvælges nu som den bedste og indsættes i sekvensen. Systemet er stadig i fejltilstand med sandsynligheden $\frac{P(e^1)}{P(e^0)} = \frac{0.204}{0.694} = 0.294$.

Nu er
$P(K_1 = f|e^1) = 0.500$,
$P(K_2 = f|e^1) = 0.625$,
$P(K_3 = f|e^1) = 0.0$ *og*
$P(K_4 = f|e^1) = 0.0$,

og H_1 *og* H_2*' effektiviteter er nu*

$$Ef^1(H_1) = \frac{P(K_1)V_1 + P(K_2)V_2}{(1-(P(K_1)V_1 + P(K_2)V_2))\cdot C_{H_1}} = \frac{0.500\frac{1}{1} + 0.625\frac{1}{1+1}}{(1-(0.500\frac{1}{1} + 0.625\frac{1}{1+1}))\cdot 1} = 4.333$$

$$Ef^1(H_2) = \frac{P(K_2)V_2 + P(K_3)V_3}{(1-(P(K_2)V_2 + P(K_3)V_3))\cdot C_{H_2}} = \frac{0.625\frac{1}{1+1} + 0\frac{1}{1}}{(1-(0.625\frac{1}{1+1} + 0\frac{1}{1}))\cdot 1.1} = 0.455.$$

Nu udføres handling 1 som den optimale og indsættes i sekvensen. Systemet er stadig i fejltilstand med sandsynligheden $\frac{P(e^2)}{P(e^0)} = \frac{0.0}{0.694} = 0.0$*, da alle komponenter er blevet repareret med 100% effektive reparationshandlinger, og handling* H_2 *udføres derfor ikke.*

Den fundne sekvens af reparationshandlinger er $< H_3, H_1 >$ *med forventet reparationsomkostning:*
$ECR(< H_3, H_1 >) = C_3 + C_1\cdot P(H_3 = n) + C_2\cdot P(H_3 = n, H_1 = n) = 1 + 1\cdot 0.294 + 1\cdot 0 = 1.294$,

er den, som vi med udtømmende søgning har fundet som den optimale sekvens .

Ud af dette eksempel ser denne metode med vægtning ud til at finde udmærkede sekvenser i strukturer af typen som SACSO-eksemplet. Vores tests i næste kapitel, kapitel 3, vil vise hvor god den er mere generelt. Den maskinelle test med det vægtede ikke reducerede udtryk gav også den optimale sekvens.

2.4. Afhængige fejl

Hidtil har vi begrænset modelleringen med antagelser om single-fault eller uafhængige fejl. Disse antagelser havde det tilfælles, at systemet var i fejltilstand når bare én af fejlårsagerne var i fejltilstand. Dette gælder ikke for afhængige fejl. Afhængige fejl afspejler problematikken med replikerede ressourcer i f.eks. computernetværk, hvor de afhængige fejlårsager er delt op i grupper, og systemet er i fejltilstand hvis en af grupperne er i fejltilstand. En gruppe er kun i fejltilstand, hvis samtlige indeholdte komponenter er i fejltilstand. Et eksempel på afhængige fejl taget fra computernetværksdomænet er DNS-servere, hvor der ofte er backupressourcer tilstede: DNS fejler kun hvis samtlige DNS-servere er i fejltilstand, dvs. systemet er ikke i fejltilstand hvis bare én af serverne fungerer.

I dette afsnit beskriver vi en måde at modellere afhængige fejl med bayesianske netværk. Vi beskriver hvordan uafhængige og afhængige handlinger modelleres i dette netværk, og hvordan effektiviteter af handlinger og ECR af sekvensen beregnes.

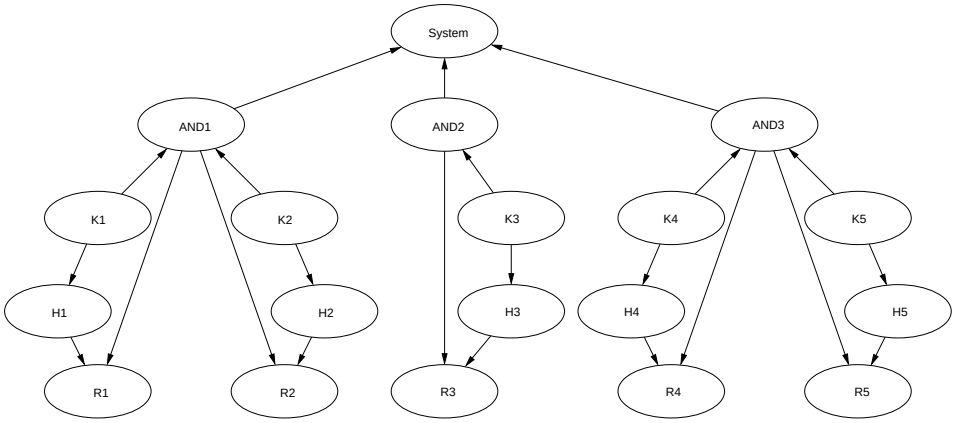
2.4.1. Modellering af afhængige fejl

Vi antager for nemheds skyld at der er uafhængige handlinger i dette afsnit og viser en model med afhængige fejlårsager og tilknyttede handlinger. De afhængige fejlårsager er grupperet under AND-knuder, der kun er i fejltilstand, når alle deres forældre (fejlårsagerne) er i fejltilstand. Vi antager at systemet er i fejltilstand hvis én eller flere af disse AND-knuder er i fejltilstand[15].

Handlingerne er stadig direkte børn af de årsager, handlingerne reparerer, dog er der til hver handling tilknyttet netop én *result*-knude, der benyttes til at aflæse sandsynligheden for at handlingen vil kunne medvirke til at reparere systemet. En result-knude har tilstandene $\{Ja, Nej\}$ og er barn af en handling og af alle AND-knuderne for de fejlårsager handlingen reparerer. Result-knuden er i tilstanden Ja når handlingen er i tilstand Ja og én eller flere af de AND-knuder, den er barn af, er i fejltilstand. Dette følger af at handlingen kun vil være nødvendig at udføre, når den er istand til at reparere en AND-knude (ved at reparere en af fejlårsagerne under AND-knuden). Dvs. at en handling udføres ikke nødvendigvis selvom den kan repare en komponent, som er

[15] Under undersøgelserne har det vist sig at beregningen af sekvenser ser ud til at give korrekte resultater uanset om vi antager single-fault eller uafhængige fejl mellem AND-knuderne

i fejltilstand. Den udføres kun hvis komponenten er i en gruppe, hvor alle komponenter er i fejltilstand. Sandsynligheden for at handlingen vil reparere systemet er således $P(H|e) \cdot \sum_i P(AND_i)$, hvor AND_i er de AND-knuder, som handlingen kan reparere via komponentreparationer. Denne sandsynlighed er repræsenteret i result-knudens tabel.



Figur 2.9.: Eksempel på en model med afhængige fejl og uafhængige handlinger

2.4.2. Beregning af effektivitet og ECR

Beregningen af en handlings effektivitet udføres på samme måde som for de andre typer modeller, dog benyttes sandsynligheden for at result-knuden er i tilstand Ja i stedet for sandsynligheden for den tilknyttede handling. Ligeledes indsættes evidens på at en handling er udført på result-knuden. Vi benytter udtrykket $\frac{P}{C}$ som effektivitetsudtryk, hvor $P = P(Result = Ja|e)$.

$$P(Result = Ja|e) = P(H|e) \cdot \sum_i P(AND_i),$$

så

$$\frac{P}{C} = \frac{P(H|e) \cdot \sum_i P(AND_i)}{C_H} = \frac{P(Result|e)}{C_H}$$

For ECR benytter vi den generelle beregningsform, hvor evidens indsættes på resultknuderne i stedet for på handlinger. Dvs. vi aflæser $P(Result = No|Sys = Fejl)$ i stedet for $P(Handling = No|Sys = Fejl)$

2.4.3. Modellering af afhængige handlinger

Definitionen af en afhængig handling er mere flydende her, idet handlingerne vil være indirekte afhængige pga. de afhængige årsager, idet handlinger, der reparerer forskellige fejlårsager under samme AND-knude er indirekte afhængige gennem denne AND-knude.

Vi beholder dog den oprindelige definition på hvad en egentlig afhængig handling er (side 14 under afsnit 2.2.2).

Modelleringen er lidt mere kompliceret. Umiddelbart vil en handling, der kan reparere flere fejlårsager, være i tilstanden Ja, når den reparerer mindst en af fejlårsagerne, idet den så vil have en positiv effekt. Den tilknyttede result-knude er i tilstand Ja når handlingen er i tilstand Ja og mindst en af de til fejlårsagerne tilknyttede AND-knuder i fejltilstand.

Vi udfører tests på hvor gode sekvenserne fundet ved $\frac{P}{C}$ er i forhold til de optimale sekvenser, før vi beslutter om det er nødvendig (gavnlig) at medtage en modificeret heuristik med vægtning. Og hvordan en vægtningen skal se ud for at være gavnlig.

Kapitel 3

Undersøgelser og test af heuristikker

I dette kapitel undersøger vi de forskellige måder at finde en sekvens af reparationshandlinger. Undersøgelserne udføres på et udvalg af modeller under de forskellige antagelser. For årsager er antagelserne single-fault, uafhængige fejl samt afhængige fejl, for handlinger er det perfekte og ikke-perfekte uafhængige eller afhængige handlinger.

Formålet er at få viden om, hvornår de forskellige fremgangsmåder giver en optimal sekvens eller en bedre sekvens i forhold til hinanden. For de mindre modeller (færre end 10 handlinger) finder vi den optimale sekvens ved udtømmende søgning.

3.1. Opsætning

Her beskriver vi hvilke undersøgelser vi udfører, og hvordan resultaterne repræsenteres. Derefter beskrives hvordan de de forskellige problemsituationer modelleres under de forskellige antagelser. Til sidst beskrives det programmel, vi anvender til at udføre undersøgelserne.

3.1.1. Beskrivelse af undersøgelse

En undersøgelse består af en sekvens af tests udført på den samme model. Dvs. at modellens struktur er den samme for alle testene, mens sandsynlighederne for årsager og handlinger varieres tilfældigt for hver test. I hver test beregnes den optimale sekvens (ved udtømmende søgning blandt alle de mulige sekvenser), samt sekvenserne fundet ved brug af heuristikken beskrevet i afsnit 2.1.3, med hhv. den normale og den vægtede effektivitet (for single-fault er disse $Ef = \frac{P}{C}$ og $Ef = \frac{P \cdot V}{C}$). Resultatet af en test er ECR for de fundne sekvenser, hvor den ECR af den optimale sekvens er sat til 100%, og for de to andre sekvenser angives ECR som procentvise afvigelse fra den optimale.

Vi tester 3 forskellige heuristikker og sammenligner sekvenserne fundet med den optimale

sekvens. Heuristikkerne benævnes *pc*, *pvc* og *alt*. De to førstnævnte er heuristikkerne baseret på hhv. $\frac{P}{C}$ og $\frac{P}{(1-P)C}$, heuristikken *alt* er en kombination af disse to heuristikker, hvor der findes en sekvens med *pc* og *pvc*, hvorefter den bedste af de to vælges.

3.1.2. Beskrivelse af modellering

Modellerne repræsenteres ved bayesianske netværk, der oprettes i Hugin. Modelleringen tager højde for om der er antaget single-fault, uafhængige fejl eller afhængige fejl. Dette resulterer i tre forskellige typer af modeller. Handlingerne repræsenteres direkte i modellerne, og udformningen af deres sandsynlighedstabeller beskrives herefter.

Alle modellerne har følgende karakteristika tilfælles: Knuden *System* er den problemdefinerende knude, dvs. her angives hvorvidt systemet er i fejltilstand eller ej. Idet vi har antaget, at vi kun troubleshooter når systemet er i fejltilstand, sættes denne knude til altid at være i fejltilstand. Handlingerne har fejlårsagerne som forældre, og de indeholder information om sandsynligheden for at reparere de tilknyttede fejlårsager (i tabellen) samt hvilken omkostning, der er tilknyttet udførelsen af handlingen (givet som en attribut).

I modellerne for *single-fault* indeholder *System* tilstandene $\{Ok, K_1, \ldots, K_n\}$, hvor K_i er en af de underliggende komponenter. Her er komponenterne børn af systemknuden. Sandsynligheden for tilstand Ok sættes til 0, og sandsynligheden for tilstand $K_i = P(K_i|Sys = f)$. Komponenterne, der ligger til grund for systemets fejltilstand, er børn af *System* og er kun i fejltilstand, når *System* er i tilstanden svarende til komponenten. Herved sikres at der på komponentniveau er single-fault. Der er kun et niveau af fejlårsager i modellerne. Flere nivauer (underårsager) kræver en anderledes modellering for fortsat at sikre single-fault.

I modellerne for *uafhængige fejl* har *System* samme tilstande som fejlårsagerne, dvs. $\{Ok, Fejl\}$. Systemknuden sættes ved start af troubleshooting til at være i tilstanden $Fejl$. Fejlårsagerne er forældre til *System*knuden, og har en marginal sandsynlighedsfordeling, der angiver sandsynligheden for at komponenten er i fejltilstand.

For *afhængige fejl* er strukturen noget anderledes, idet der på øverste niveau under *System* er antaget single-fault (dvs. XOR) mellem AND-knuderne. Hver mængde af indbyrdes afhængige fejlårsager er samlet under en AND-knude. Et eksempel på modellering af et system medm afhængige fejl ses på figur 2.9 side 35. Der er 2 niveauer af årsager, hvor niveauet under *System* består af AND-knuderne og niveauet under disse er de "faktiske" årsager. Sandsynlighedsfordelingen for AND-knuderne svarer til at de kun er i fejltilstand, når alle underliggende komponenter er i fejltilstand. De underliggende komponenter har en marginal sandsynlighedsfordeling. Handlingerne med reparationssandsynligheder er stadig børn af årsagerne, dog er der yderligere tilføjet et underliggende niveau af "handlinger", resultatknuder, der indikerer sandsynligheden for at en handling vil reparere systemet. Dette modelleres ved at tage sandsynligheden for fejlårsagernes tilhørende AND-knude(r) med i betragtning vha. en kausal forbindelse, så result-knuden er barn af både handlingen, og af de til handlingens komponenter hørende AND-knuder.

Resultatknuderne vil kun være i tilstand Ja, når handlingen er i tilstanden Ja og de(n) overliggende AND-knuder er i tilstanden $Fejl$. For uddybende beskrivelse se afsnit 2.4.

Det er kun for modellerne med afhængige fejl, at vi har medtaget modeller med uafhængige handlinger. De uafhængige handlinger har en sandsynlighedsfordeling svarende til nedenstående tabel (3.1):

	K=Ok	K=Fejl
H=Ja	0	Reparationssandsynligheden $P(H = Ja \mid K = Fejl)$
H=Nej	1	$1 - P(H = Ja \mid K = Fejl)$

Tabel 3.1.: Sandsynlighedsfordeling for en handling, der reparerer årsag K

Ved handlinger, der reparerer flere fejlårsager, angives sandsynligheden for at handlingen reparerer hver årsag, givet at denne årsag er i fejltilstand. De øvrige sandsynligheder i tabellen kan beregnes ud fra disse to informationer, givet at handlingen reparerer fejlårsagerne uafhængigt af hinanden. Vi har antaget at en handling, der reparerer flere årsager, reparerer disse uafhængigt af hinanden. Derfor bliver $P(H = j \mid K_i = f, K_j = f) = P(H = j \mid K_i = f) \cdot P(H = j \mid K_j = f)$, hvor K_i og K_j er de komponenter, som repareres af H.

Eksempel 1 *Udregning af tabel-indhold for handling, der reparerer flere komponenter*
Givet et eksempel med to komponenter K1 og K2 og en handling H, der reparerer disse.

K2 =	Ok	Ok	Fejl	Fejl
K1 =	Ok	Fejl	Ok	Fejl
H = Ja	0	P1	P2	1-(1-P1)*(1-P2)
H = Nej	1	1-P1	1-P2	(1-P1)*(1-P2)

Tabel 3.2.: Beregning af tabel for handling H, der reparerer K1 og K2

Vi har nu at handlingen reparerer K1 med sandsynligheden P1 givet at denne er i fejltilstand (samme gælder for P2 og K2). Med disse informationer, og antagelsen om at handlingen reparerer årsagerne K1 og K2 uafhængigt af hinanden fås en tabel (svarende til de indtastede i Hugin) som 3.2.

3.1.3. Beskrivelse af programmel

Til modelleringen benytter vi programmet Hugin, der er udviklet af firmaet Hugin Expert og stillet gratis til rådighed for studerende. Hugin gør det nemt at oprette bayesianske modeller og muliggør propagering af evidens og udtrækning af sandsynligheder i modellen. Hugin kan hentes i en Light-udgave fra hjemmesiden *http://www.hugin.com*. Sammen med programmet Hugin (den grafiske overflade), medfølger der en API, der gør det muligt at udføre beregninger og propagering i modeller fra egne programmer. Vi benytter denne API til at lave sekvensberegningen.

Det program vi udviklede til formålet er skrevet i C++ og indeholder følgende funktionalitet:

- Beregning af sekvens ved heuristik med $\frac{P}{C}$ (sekvensen kaldes *pc* i tabellerne).
- Beregning af sekvens ved heuristik med $\frac{P \cdot V}{C}$ (sekvensen kaldes *pvc* i tabellerne).
- Alternativ beregning af sekvens, ved at vælge den bedste sekvens af to sekvenser beregnet med ovenstående to heuristikker. En sekvens beregnet med den alternative (kombinerede) heuristik betegnes *alt* i tabellerne.
- Beregning af optimal sekvens ved udtømmende søgning (betegnet *opt* i tabellerne).
- Oprettelse af et antal tilfældige modeller – sandsynlighederne er tilfældige for handlinger og fejlårsager, strukturen for modellen er givet til hver situation. For hver model beregnes en sekvens med hver af de tre ovenstående nævnte metoder. Og resultateterne gemmes, hvis ECR af de fundne sekvenser afviger fra den ECR af den optimale sekvens.

Beregningerne af sekvenser følger den viste algoritme i afsnit 2.1.3. For hver sekvens beregnes ECR, der beregnes ud fra udtryk 2.1 side 8 og følger fremgangsmåden i følgende algoritme:

```
Sæt evidens på at systemet er i fejltilstand og propager
ECR = Omkostning for Handling(0)
Indsæt evidens på at Handling(0) er udført og propager

FOR h = 1 to Antal Handlinger i sekvens
  Beregn P = P(Handling(0)=Fejlede, ..., Handling(h-1)=Fejlede | System=Fejl)
  ECR = ECR + P * Omkostning for Handling(h)
  Indsæt evidens på at Handling(h) er udført og propager
ENDFOR
```

Omkostningen for første handling (0) i sekvens er altid regnet med, idet denne handling altid udføres. Derefter tilføjes omkostningerne for de resterende handlinger enkeltvist, vægtet med sandsynligheden for at det bliver nødvendigt at udføre dem. Efter at omkostningen for hver handling er medtaget i ECR, propageres evidens om at handlingen er blevet udført og har fejlet.

Den udtømmende søgning foretages ved, at generere alle mulige sekvenser indeholdende samtlige handlinger og beregne ECR for disse sekvenser. Sekvensen med den laveste ECR og det mindste antal handlinger vælges som den optimale. Hvis der er flere sekvenser med samme ECR og antal handlinger, vælges en tilfældig (i vores tilfælde vælges den først fundne).

De tilfældige modeller oprettes ved at tage udgangspunkt i en givet model, som beskriver strukturen og så indlægge tilfældige sandsynligheder for at handlinger reparerer de individuelle fejlårsager samt tilfældige omkostninger tilknyttet udførelsen af handlingerne. Sandsynlighederne for fejlårsagerne genereres også tilfældigt.

For nogle af modellerne blev der i en forudgående undersøgelse antaget perfekte handlinger. Resultaterne af disse tests er kort opsummeret for disse test.

3.2. Test med single-fault

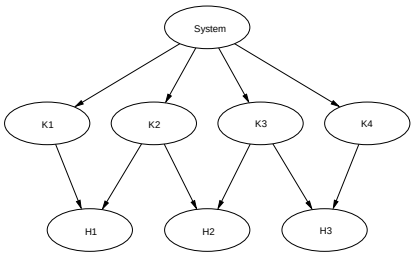
For single-fault benyttes udtrykket $\frac{P}{C}$ til at finde sekvensen pc, for sekvensen pvc benyttes den vægtede udgave af udtrykket.

Vurderingerne af modellerne for single-fault er baseret på 1.000 - 10.000 tilfældigt genererede modeller, med den for hver model viste struktur. Antallet af modeller for hvert forsøg er afgjort af antallet af handlinger, 10.000 gennemløb for modeller med 3 handlinger til 1.000 gennemløb for modeller med 6 handlinger.

Sandsynlighederne for årsagerne vælges tilfældigt i intervallet [0.01-0.99] (tilsvarende intervallet for uafhængige fejl), reparationssandsynlighederne for handlingerne vælges tilfældigt i intervallet [0.90-1.00] og deres omkostning ligger i intervallet [1.00, 5.00].

For hvert forsøg er der angivet hvor mange handlinger, der er udført, dette er i tabellerne angivet med tallet efter "Sammenligning". Antallet af sekvenser, der afviger, er angivet som % i stedet for absolutte værdier. Kun afvigelser for de ikke-optimale sekvenser er medtaget ved beregningen af de maksimale og gennemsnitlige afvigelser i ECR.

3.2.1. Model 1: SACSO-type med afhængige handlinger



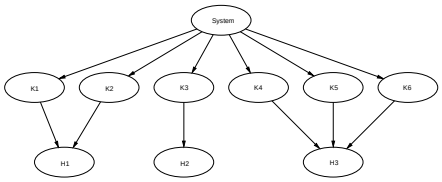
Sammenligning (10.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	2.89%	18.02%	4.14%
$ECR(pvc) > ECR(opt)$	23.5%	22.01%	4.77%
$ECR(alt) > ECR(opt)$	0.72%	5.08%	1.86%
$ECR(pc) > ECR(pvc)$	2.57%	18.02%	4.13%

Denne struktur taget fra SACSO [Jensen et al., 2001], var udgangspunktet for at vælge den vægtning, vi kom frem til. Det ses at selvom sekvenserne fundet ved det vægtede effektivitetsudtryk alene ikke er anvendelige, så giver kombinationen af heuristikkerne baseret på det oprindelige udtryk ($\frac{P}{C}$) og på det tilsvarende vægtede udtryk, hvor den bedste sekvens vælges, en betydelig lavere afvigelse fra den optimale sekvens, både hvad angår maksimal afvigelse og den gennemsnitlige afvigelse.

Det tyder på at de sekvenser, hvor $\frac{P}{C}$ giver dårlige resultater, giver bedre resultater med $\frac{P \cdot V}{C}$. Yderligere undersøgelser af tilsvarende strukturer, med 4 og 5 handlinger viser at

den kombinerede heuristik stadig giver væsentlig bedre resultater end $\frac{P}{C}$ alene.

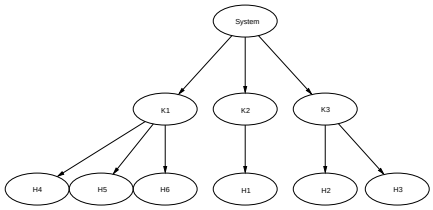
3.2.2. Model 2: 1-3 årsager pr. handling



Sammenligning (10.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	0%	0%	0%
$ECR(pvc) > ECR(opt)$	0%	0%	0%
$ECR(alt) > ECR(opt)$	0%	0 %	0%
$ECR(pc) > ECR(pvc)$	0%	0%	0%

I dette tilfælde findes de optimale sekvenser ved udtrykket $\frac{P}{C}$ og $\frac{P \cdot V}{C}$. Der er dog heller ingen afhængige handlinger, hvorfor $\frac{P \cdot V}{C}$ reduceres til $\frac{P}{C}$. Testen er medtaget for at vise at for uafhængige handlinger giver begge heuristikker optimale sekvenser.

3.2.3. Model 3: 1-3 handlinger pr. årsag

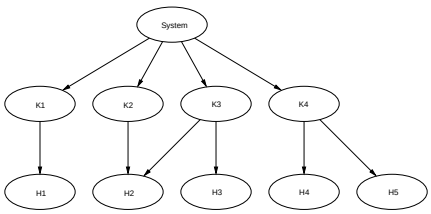


Sammenligning (1.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	4.7%	8.43%	1.00%
$ECR(pvc) > ECR(opt)$	65.2%	33.60%	8.23%
$ECR(alt) > ECR(opt)$	3.0%	3.39%	0.75%
$ECR(pc) > ECR(pvc)$	2.0%	8.14%	1.23%

Antages der ikke perfekte handlinger, giver kombinationen af de to heuristikker ($\frac{P}{C}$ og $\frac{P \cdot V}{C}$) bedre resultater end $\frac{P}{C}$ alene, dog er forskellgen ikke videre stor for den gennemsnitlige afvigelse, men det ser ud til at de tilfælde, hvor $\frac{P}{C}$ giver sekvenser, der afviger væsentligt fra den optimale, fanges ved brug af udtrykket $\frac{P \cdot V}{C}$.

For denne struktur er $\frac{P \cdot V}{C}$ ubrugelig, når der antages perfekte handlinger (ikke vist i tabel), idet den "straffer" de årsager, der repareres af flest handlinger. Dvs. den giver andre sekvenser end dem fundet ved $\frac{P}{C}$, der finder de optimale sekvenser, på trods af at der er afhængige handlinger.

3.2.4. Model 4: En kombination af disse typer modeller



Sammenligning (1.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	8.0%	11.34%	2.32%
$ECR(pvc) > ECR(opt)$	31.9%	14.57%	3.35%
$ECR(alt) > ECR(opt)$	1.3%	4.21%	1.44%
$ECR(pvc) \geq ECR(pc)$	7.0%	11.34%	2.39%

Igen ses at kombinationen af de to heuristikker giver et væsentligt bedre resultat end $\frac{P}{C}$ alene. $\frac{P \cdot V}{C}$ er ikke anvendelig alene, idet den ofte giver ikke-optimale sekvenser, i tilfælde hvor $\frac{P}{C}$ finder de optimale sekvenser.

Denne test er også udført under antagelse af perfekte handlinger, hvor $\frac{P}{C}$ giver de optimale sekvenser, og sekvenser fundet ved $\frac{P \cdot V}{C}$ afviger i 22% af tilfældende.

3.2.5. Vurdering af resultater for single-fault

Beregning af sekvenser med udtrykket $\frac{P \cdot V}{C}$ er relevant for modeller med afhængige handlinger, hvor der ikke er perfekte handlinger. For modeller med strukturer lignende model 1 er $\frac{P \cdot V}{C}$ også relevant, når der antages perfekte handlinger. Sekvenserne fundet ved $\frac{P \cdot V}{C}$ alene er ikke praktisk anvendelige, idet afvigelsen fra de optimale sekvenser er for høj. Men kombinationen, hvor den bedste af sekvenserne fundet ved både $\frac{P}{C}$ og $\frac{P \cdot V}{C}$ udvælges, er et udmærket alternativ til sekvenser fundet ved $\frac{P}{C}$ alene.

3.3. Test med uafhængige fejl

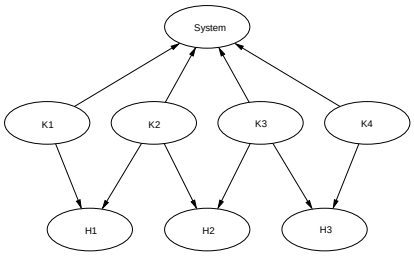
For test af modeller under antagelse af uafhængige fejl benyttes udtrykket $\frac{P}{C}$ til at finde sekvensen *pc*, for sekvensen *pvc* benyttes den vægtede udgave af udtrykket.

Vi har valgt ikke at benytte udtrykket fra Srinivas, idet det viste sig at det gav dårligere resultater end $\frac{P}{C}$. En sammenligning af de benyttede udtryk for effektivitet kan ses i afsnit 3.5

Testene er udført under de samme forudsætninger som for single-fault.

Det forventes at resultaterne for test med uafhængige handlinger ikke afviger væsentlig fra resultaterne fundet under single-fault antagelsen.

3.3.1. Model 1: SACSO-type med afhængige handlinger

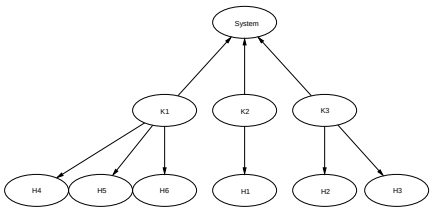


Sammenligning (10.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	0.78%	6.52%	1.66%
$ECR(pvc) > ECR(opt)$	33.08%	103%	13.22%
$ECR(alt) > ECR(opt)$	0.18%	5.64%	1.39%
$ECR(pc) > ECR(pvc)$	0.62%	6.52%	1.68%

Under antagelse af uafhængige fejl, giver $\frac{P}{C}$ næsten optimale sekvenser (kun 0.78% af sekvenserne var ikke-optimale) og $\frac{P \cdot V}{C}$ giver dårligere resultater end under den tilsvarende test ved single-fault.

Dog giver kombinationen af de to heuristikker stadig det bedste resultat.

3.3.2. Model 2: 1-3 handlinger pr. årsag

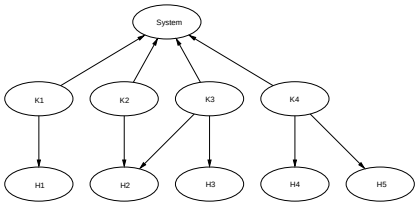


Sammenligning (1.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	3.4%	3.22%	0.71%
$ECR(pvc) > ECR(opt)$	71.1%	144%	15.62%
$ECR(alt) > ECR(opt)$	2.5%	3.22%	0.53%
$ECR(pc) > ECR(pvc)$	1.1%	2.89%	0.98%

Denne type struktur ser ud til at håndteres dårligt af heuristikken, der anvender det vægtede udtryk. Forbedringen ved brug af kombinationen af heuristikkerne er tilstede, dog ikke nær så stor som ved de andre typer af modeller.

Ved perfekte handlinger findes de optimale sekvenser ved $\frac{P}{C}$.

3.3.3. Model 3: En kombination af disse typer modeller



Sammenligning (1.000)	Antal	Maks. afv.	Gns.
$ECR(pc) > ECR(opt)$	2.4%	7.38%	0.82%
$ECR(pvc) > ECR(opt)$	55.9%	113%	13.86%
$ECR(alt) > ECR(opt)$	1.3%	1.12%	0.38%
$ECR(pc) > ECR(pvc)$	1.1%	7.38%	1.35%

Ved ikke-perfekte handlinger giver den kombinerede heuristik igen væsentligt bedre resultater end heuristikken med $\frac{P}{C}$ alene. Dvs. brug af den kombinerede heuristik er også relevant for modeller med denne struktur.

Igen findes de optimale sekvenser ved $\frac{P}{C}$ under antagelse af perfekte handlinger, hvor sekvenserne fundet ved $\frac{P \cdot V}{C}$ afviger i 47% af tilfældene.

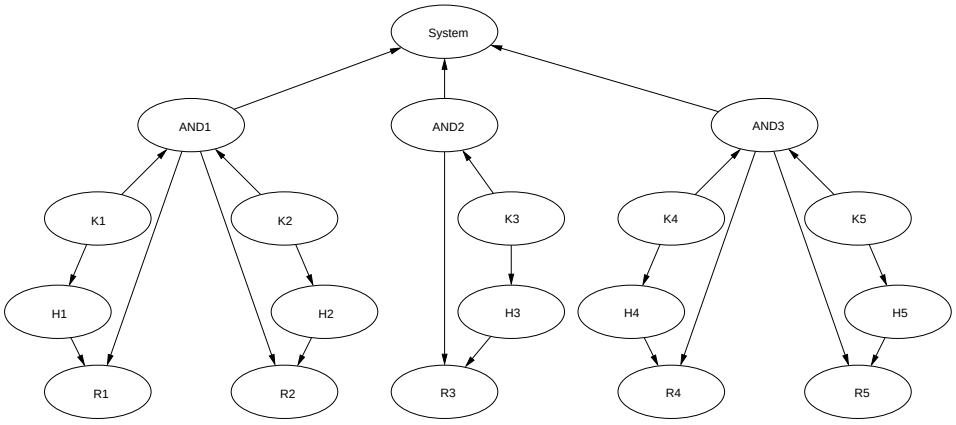
3.3.4. Vurdering af resultater for uafhængige fejl

Som forventet svarer resultaterne til dem fundet ved undersøgelserne af modeller med samme struktur under single-fault antagelsen. Igen ses det, at der er en klar forbedring ved at bruge kombinationen af de to heuristikker, og det ses at heuristikken med det vægtede udtryk alene giver langt dårligere resultater end $\frac{P}{C}$. Generelt kan vi sige, at antallet af modeller, hvor der findes ikke-optimale sekvenser, halveres (eller bedre i forhold til pc) ved brug af den kombinerede heuristik. Begge heuristikker har samme tidskompleksitet, så udførelsen af den kombinerede heuristik vil kun blive en fordobling af køretiden, og giver dermed ikke en nævneværdig forøgelse. Dette gælder generelt for denne form for heuristikker, dvs. hvis der findes andre heuristikker, der finder bedre sekvenser (under andre forudsætninger), kan disse kombineres på samme måde, med mulighed for at reducere ECR yderligere.

3.4. Test med afhængige fejl

Testmodellerne følger beskrivelsen af modelleringen beskrevet i afsnit 2.4, dette gælder også beregning af effektiviteter og ECR for sekvenserne. Til beregning af ECR benytter vi den generelle algoritme (fra afsnit 2.1.2).

3.4.1. Model 1: AND med uafhængige handlinger



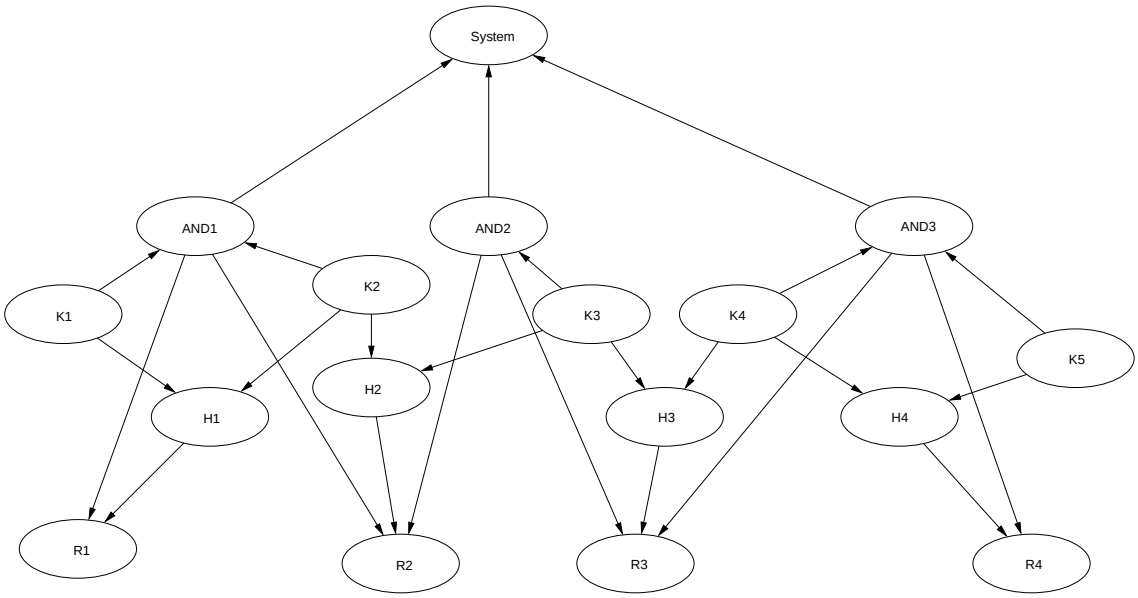
Figur 3.1.: Model med multiple afhængige fejl og uafhængige handlinger

Vi tester denne model med heuristikken, der anvender effektivitetsudtrykket $\frac{P}{C}$. Vi ønsker at afgøre hvor gode sekvenserne bliver i forhold til de optimale (der findes ved udtømmende søgning).

Vi har lavet modellen i to udgaver, én hvor der antages single-fault på øverste niveau (mellem AND-knuderne) og én hvor der antages uafhængige AND-knuder. Ligeledes blev disse undersøgelser foretaget både med perfekte og ikke-perfekte handlinger.

Resultatet af undersøgelsen blev, at heuristikken med $\frac{P}{C}$, i alle disse situationer, fandt den optimale sekvens.

Ovenstående model svarer meget godt til strukturen i et netværk med replikerede ressourcer (f.eks. DNS i CS-netværket), idet der meget sjældent er afhængige handlinger. Idet $\frac{P}{C}$ -heuristikken finder de optimale sekvenser, vil det også være muligt at udvide modellerne fra vores Dat 5 projekt til at indeholde replikerede ressourcer, og derved ikke længere være afhængig af single-fault antagelsen. Det er dog stadig et problem at modellen bliver meget omfattende (idet den ikke kan simplificeres til en naiv baysiansk model, når der ikke er antaget single-fault).



Figur 3.2.: Model med multiple afhængige fejl og afhængige handlinger

3.4.2. Model 2: AND med afhængige handlinger - Udvidet SACSO-model

For denne model giver heuristikken baseret på $\frac{P}{C}$ ikke længere de optimale sekvenser. Ud af 10.000 udførte forsøg, hvor handlingernes omkostning og reparationssandsynlighed, samt årsagernes fejlsandsynlighed blev tilfældigt valgt, resulterede heuristikken i 1.5% af tilfældene i ikke-optimale sekvenser med en maks. afvigelse fra ECR på 6.2% og en gennemsnitlig afvigelse på 0.9%.

Så for denne type struktur er $\frac{P}{C}$ stadig et godt, brugbart bud på et udtryk for effektivitet.

3.5. Afprøvning af alternative heuristikker

Ved forsøg udført under antagelsen af uafhængige handlinger, fandt vi frem til at udtrykket fra Srinivas ($\frac{P}{(1-P)C}$) gav langt dårligere resultater end dem fundet ved det normale udtryk ($\frac{P}{C}$).
Vi besluttede derfor at udføre nogle undersøgelser, hvor vi beregnede sekvenser baseret på de 4 forskellige udtryk vi var kommet frem til: $\frac{P}{C}$ og $\frac{P}{(1-P)C}$ samt de tilsvarende vægtede udtryk. Derudover tilføjede vi følgende 4 alternative (kombinerede) heuristikker, hvor vi kombinerede *pc* med de andre tre udtryk, hver for sig, samt en heuristik bestående af en kombination af alle udtrykkene. Resultatet var overraskende, og illustreres i nedenstående tabel 3.3, hvor vi sammenholder resultaterne af de 8 forskellige heuristikker, med den fundne optimale sekvens.

Heuristik baseret på	Ikke-optimale sekvenser
$\frac{P}{C}$	0.64%
$\frac{PV}{C}$	31.70%
$\frac{P}{(1-P)C}$	43.46%
$\frac{PV}{(1-PV)C}$	45.37%
ALT($\frac{PV}{C}$)	0.26%
ALT($\frac{P}{(1-P)C}$)	0.27%
ALT($\frac{PV}{(1-PV)C}$)	0.32%
ALT(alle)	0.11%

Tabel 3.3.: Oversigt over resultater udført ved adskillige heuristikker og kombinationer af disse

Denne test blev udført på Model 1: SACSO, under antagelse af uafhængige fejl, men lignende resultater blev opnået ved at teste på den kombinerede model 4. For at få mere præcise målinger, blev sekvenserne beregnet for 100.000 modeller med nævnte strukturer.

Et af de overraskende resultater var, hvor dårlige resultater Srinivas-udtrykket for uafhængige fejl giver i forhold til P/C. Det mest overraskende er, at en kombineret heuristik baseret på $\frac{P}{C}$ og $\frac{P}{(1-P)C}$ gav næsten lige så gode resultater som kombinationen med det vægtede P/C-udtryk.

3.6. Vurdering

Sekvenserne fundet ved brug af $\frac{P}{C}$ er udmærkede, optimale i de fleste tilfælde, der er dog en forbedring at hente ved at kombinere *pc*- og *pvc*-sekvenserne, idet *pvc* finder bedre sekvenser i nogle af de tilfælde, hvor *pc* fejler.

De alternative heuristikker, derunder den vægtede heuristik, er kun relevante under antagelse af ikke-perfekte afhængige handlinger. For perfekte handlinger har den vægtede heuristik dog også en gavnlig virkning for modeller, der har en struktur lignende SACSO-strukturen.

Det er overraskende at $\frac{P}{C}$ giver de bedste resultater for modeller under alle de forskellige antagelser om årsagernes afhængighed, og at den giver bedre resultater under antagelse af uafhængige fejl end det tilsvarende udtryk fra Srinivas [Srinivas, 1995].

Af den sidste undersøgelse, hvor samtlige anvendte og testede effektivitetsudtryk blev sammenlignet, ser vi at kombinationen af flere heuristikker, baseret på lignende effektivitetsudtryk, ser ud til at give bedre resultater end nogen enkelt heuristik for sig. Det at

kombinationen af ikke-optimale heuristikker giver bedre resultater er ikke i sig selv overraskende. Det overraskende var at kombinationen af to udtryk, rettet mod den samme type model med uafhængige handlinger (her $\frac{P}{C}$ og $\frac{P}{(1-P)C}$) giver næsten ligeså gode resultater for afhængige handlinger, som kombinationen af $\frac{P}{C}$ og $\frac{P \cdot V}{C}$. Derudover forbedres resultatet af en kombineret heuristik med antallet af de heuristikker, der kombineres.

Undersøgelsen kan udvides til at teste de anvendte heuristikker, hvor der anvedes 2-step look-ahead, for at se om det bedre kan betale sig at benytte denne, end at bruge kombinationer af heuristikker.

Kapitel 4

Konklusion

Vi udviklede effektivitetsudtryk til brug ved beregning af sekvenser. Udtrykkene er en udvidelse af de eksisterende ef-udtryk, hvor vi har tilføjet en vægtning, med det formål at kunne håndtere afhængige handlinger.

Vi fandt ud af at idéen med vores form for vægtning i sig selv var ubrugelig, og at effektivitetsudtrykket $\frac{P}{C}$ i næsten alle situationer gav gode resultater. Derudover fandt vi frem til at en kombination af heuristikkerne baseret på hhv. $\frac{P}{C}$ og $\frac{P \cdot V}{C}$ gav et samlet bedre resultat end $\frac{P}{C}$ alene, for de modeller, hvor $\frac{P}{C}$ gav suboptimale resultater. Det sidste følger af at $\frac{P \cdot V}{C}$ gav gode resultater for nogle af de modeller, hvor $\frac{P}{C}$ var suboptimal.

Det viste sig også at udtrykket $\frac{P}{(1-P)C}$ kun er anvendeligt under antagelse af uafhængige handlinger, og at vi ikke kunne udvide det til at være anvendelig ved afhængige handlinger.

4.1. Videre undersøgelser og udvidelsesmuligheder

Undersøgelserne vi har foretaget er baseret på modeller med få handlinger, da vi ønskede at vurdere resultaterne op mod fundne optimale sekvenser. Derudover er manuel modellering af større modeller en meget tidskrævende opgave, specielt at finde og beregne sandsynlighederne.

For at gøre undersøgelsen mere komplet foreslår vi følgende udvidelser:

- Undersøge på større strukturer (som de testede) og større kombinationer med flere af disse strukturer.
- Undersøge et realistisk eksempel, som f.eks. et udsnit af CS-netværket, hvor fundne sekvenser sammenlignes indbyrdes, idet optimale sekvenser ikke er beregnelige for mere end 10-12 handlinger.

- Undersøge hvor store modeller, der kan håndteres vha. bayesianske net. Derunder pladsforbrug og kompleksitet af propagering.
- Overveje andre heuristikker, der tager højde for de situationer, hvor $\frac{P}{C}$ og $\frac{P \cdot V}{C}$ gik galt, derunder
  - undersøge hvorfor de forskellige worst-case-modeller gik galt, for at udlede hvorledes effektivitetsudtrykkene kan forbedres, og om heuristikkerne evt. kan kombineres på udtryks-niveau (kombinere udtrykkene *i* én heuristik, i stedet for at kombinere heuristikkerne).
  - overveje andre heuristikker, derunder 2-step look-ahead samt kombinationer.
  - undersøge heuristikker, der baserer sig på undersøgelser af modellers struktur, og foretager Bottom-up eller Top-down baserede sekvensberegninger.
  - undersøge bedre kombinationer af heuristikker, der kan benyttes på samme model, for at finde en god sekvens.
- Undersøge alternative repræsentationer af modeller, som erstatning for bayesianske modeller, således at repræsentationen mere aktivt understøtter beregning af sekvens og gør denne beregning billigere end den tilsvarende propagering af indsamlet evidens i BN.

Bilag A

Oversigt over CS-netværket

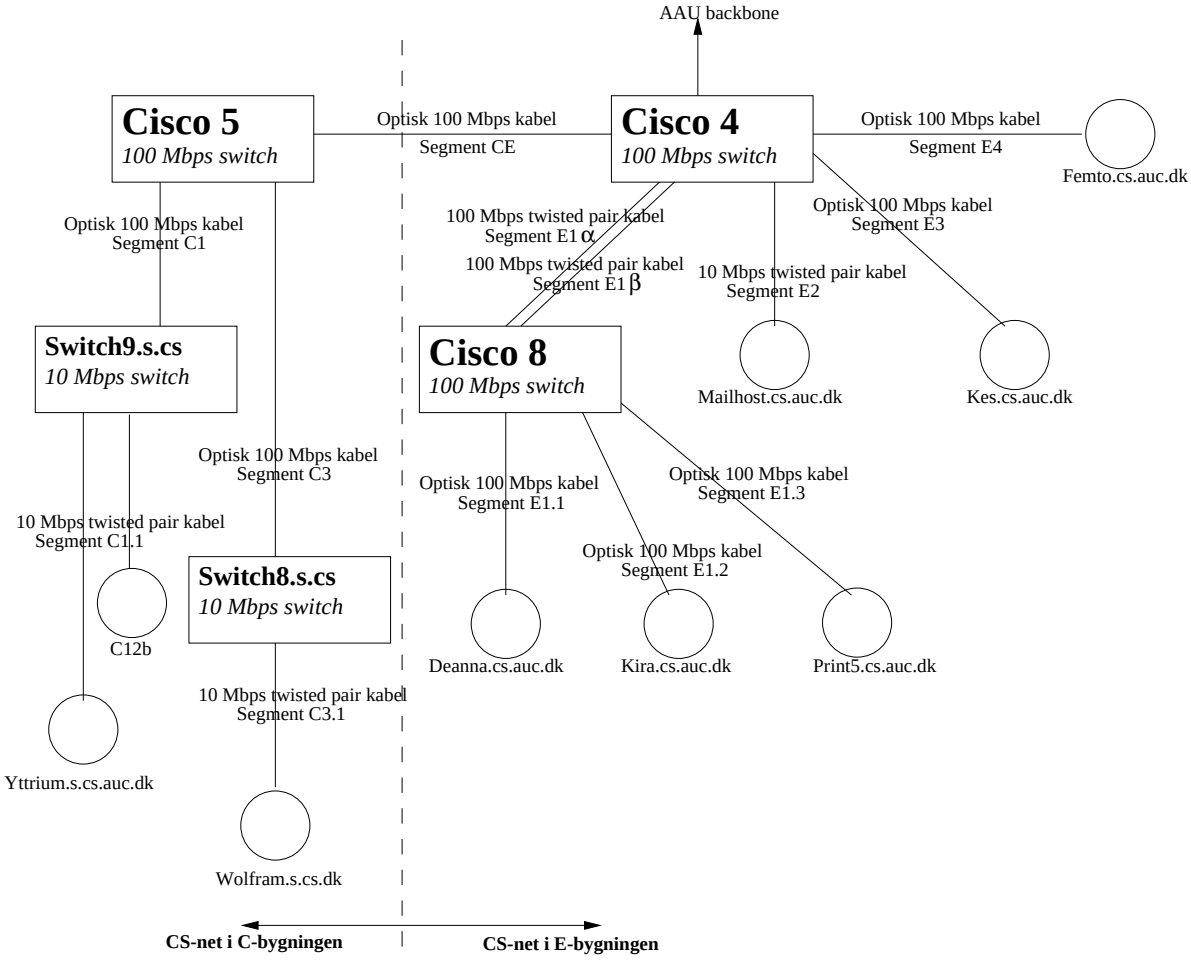
A.1. CS-netværksarkitekturen

Universitetsnetværket som helhed er organiseret omkring et AAU[1] backbone[2]. CS-netværket er forbundet til dette backbone via en 100 Mbps Cisco switch (Cisco4) placeret i E-bygningen[3] ved Institutet for Elektroniske Systemer. Overordnet fungerer CS-netværket som et Fast Ethernet, hvor der kun anvendes TCP/IP som netværksprotokol.

Beskrivelse af CS-netværket[4] er afgrænset til at omfatte dele af E-bygningen, hvor de store applikationsservere er placeret sammen med Domain Name Servere (DNS), Network File Servere (NFS) og mailserveren. Af netværksarkitekturen fremgår kun nogle af NFS serverne og enkelte Ultra/1 workstations. Endvidere omfatter netværksarkitekturen DNS-serverne og nogle NT-workstations (se figur A.1). Vi har kun medtaget de servere og workstations, som er nødvendige for at modellere problemområderne. Endelig omfatter beskrivelsen også en del af S-netværket, bestående af fire segmenter[5] (C1, C3, C1.1, C3.1) fra C-bygningen. Samt de to switche (switch 8, switch 9) og maskinerne (wolfram og yttrium).

Seks segmenter (E1α, E1β, E2, E3, E4 og CE) er forbundet til cisco4. Den primære DNS-server `femto.cs.auc.dk`, er placeret i segment E4 og forbundet til cisco4 via et optisk 100 Mbps kabel. Ligeledes er NFS'en `kes.cs.auc.dk`[6] placeret i segment E3 og forbundet via et optisk 100 Mbps kabel. Mailhost.cs.auc.dk, fungerer som mailserver og sekundær DNS-server, er placeret i segment E2 og forbundet via et 10 Mbps twisted

[1] Forkortelse for Ålborg Universitet.
[2] Højhastigheds forbindelse; ofte mellem segmenter eller fysiske lokationer.
[3] E-bygningen er en del af Institutet for Elektroniske Systemer, Institut 8.
[4] Hvis der ikke er andet angivet, vil ordet CS-netværk, fremover referere til både CS-netværket, og den del af S-netværket som indgår i beskrivelsen.
[5] Segment: en klynge af maskiner forbundet til samme switch eller en kabelforbindelse mellem to switche.
[6] Home server: her er brugerens home directories placeret



Figur A.1.: CS-netværksarkitekturen

pair kabel[7] til Cisco4. Segment E1α og E1β består af hver sin 100 Mbps twisted pair forbindelse til en 100 Mbps cisco switch (Cisco8), og segment CE er en optisk 100 Mbps forbindelse til en 100 Mbps cisco switch (Cisco5) placeret i C-bygningen ved Institutet for Elektroniske Systemer.

Tre segmenter (E1.1, E1.2 og E1.3) der alle er forbundet til Cisco8 er optiske 100 Mbps kabler. Segment E1.3 forbinder printserveren `print5.cs.auc.dk` til Cisco8. Segment E1.2 omfatter Ultra/1 workstationen `kira.cs.auc.dk` og ligeledes forbinder segment E1.1 en Ultra/1 workstation, `deanna.cs.auc.dk` til Cisco8. To segmenter (C1 og C3) er forbundet til Cisco5, som er placeret i C-bygningen.

Segment C1 er et 100 Mbps optisk kabel forbundet til switchen switch9.s.cs, og segment C3 er forbundet til switch8.s.cs og er et 100 Mbps optisk kabel.

[7] Twisted pair er en kabeltype bestående af to flettede kabler

Segment C1.1 er forbundet til switch9.s.cs. som er et 10 Mbps twisted pair kabel. På segment C1.1 er bl.a. yttrium.s.cs.auc.dk, forbundet som er en NT-workstation. Segment C3.1 er forbundet til switch8.s.cs vha. et 10 Mbps twisted pair kabel. Segment C3.1 omfatter wolfram.s.cs.auc.dk, som ligeledes er en NT-workstation.

A.1.1. Ingen kommunikation

Ingen kommunikation beskriver en problemstilling, hvor brugeren af nettet ikke kan etablere en forbindelse til en server.

DNS-server

Det kan hurtigt blive et problem i et større netværk at huske ip-adresserne på de maskiner, der kommunikeres med. Maskinerne i nettet tildeles derfor et navn, der så bruges istedet. DNS-serverens opgave er at afbilde navne til ip-adresser, der bruges serverne imellem, så forespørgsler ender hos den rette maskine. Afbildning gemmes i en hosttabel på DNS-serveren. Hvis der er et problem med DNS-serveren, medfører det, at afbildning ikke kan udføres, hvorved kommunikationen maskinerne imellem umuliggøres. Der findes dog i de fleste tilfælde mindst én sekundær DNS-server, som kan tage over i tilfælde af, at primær DNS-serveren ikke giver svar på forespørgsler. Som det kan ses af beskrivelsen af CS-nettet, omfatter denne også en primær og en sekundær DNS-server. Dermed ophører muligheden for at få forbindelse til en server, hvis der ikke kan kommunikeres med hverken primær- eller sekundær-serveren. Hvis DNS-serverne ikke fungerer, og der derfor ikke kan foregå kommunikation ved hjælp af navne, kan der stadig foretages kommunikation vha. arbejdsstationens ip-adresse. Dvs. at hvis en bruger af systemet angiver arbejdsstationens ip-adresse, vil kommunikationen stadig kunne foregå.

Der er flere forskellige årsager, der kan medføre at DNS-serveren ikke kan udføre sin opgave.

Software fejl

DNS-serveren kan have en software fejl, som medfører at serveren går ned. Software fejlen er en fejl i programmeringen af softwaren og ikke en forkert anvendelse af softwaren. I software generelt kan der være uopdagede fejl[8] og ved en bestemt brug af softwaren kan det udløse de uopdagede fejl. På de to DNS-servere mailhost og femto kører der DNS-server software. Det er denne software, som håndterer afbildningen mellem navne og ip-adresser. Hvis der under brug af DNS-softwaren fremprovokeres en software fejl, kan det få softwaren til at gå ned. DNS-softwaren skal genstartes, for at den igen udfører sin funktion i netværket.

DNS-serveren mailhost bruges desuden som mailserver og der kører derfor også mailserver software på mailhost. Udløses der en software fejl i mail softwaren vil dette dog ikke få DNS-serveren til at gå ned.

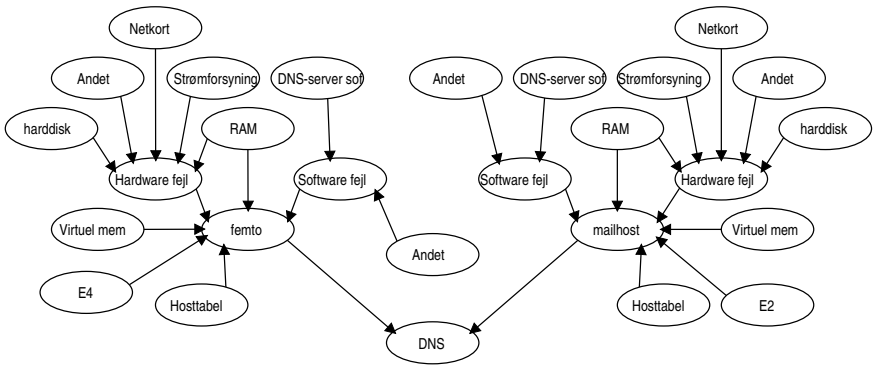
[8] det lader sig ikke praktisk gøre at udvikle 100% korrekt software [Pressman, 1997]

Hardware fejl
Der kan også optræde hardwarefejl, såsom at strømforsyningen, netkortet, RAM-moduler eller harddisken kan være defekt. Foruden de fire nævnte hardwareenheder, som er de enheder, hvor der forekommer hyppigst fejl, kan der også optræde fejl i andet hardware (Cpu, bundkort m.m. er eksempler på dette).

Andre årsager
En paritetsfejl i DNS-serverens RAM eller en korrupt virtual memory kan få serveren til at gå ned. Uden virtual memory vil DNS-serveren heller ikke kunne fungere.

En fejl i hosttabellen kan ligeledes være skyld i, at DNS-serveren ikke kan udføre sin opgave. Hvis der f.eks. slettes i en hosttabel, kan det medføre at pakker aldrig når frem til den tiltænkte modtager.

Figur A.2 viser et kausalt netværk for DNS-serverne.



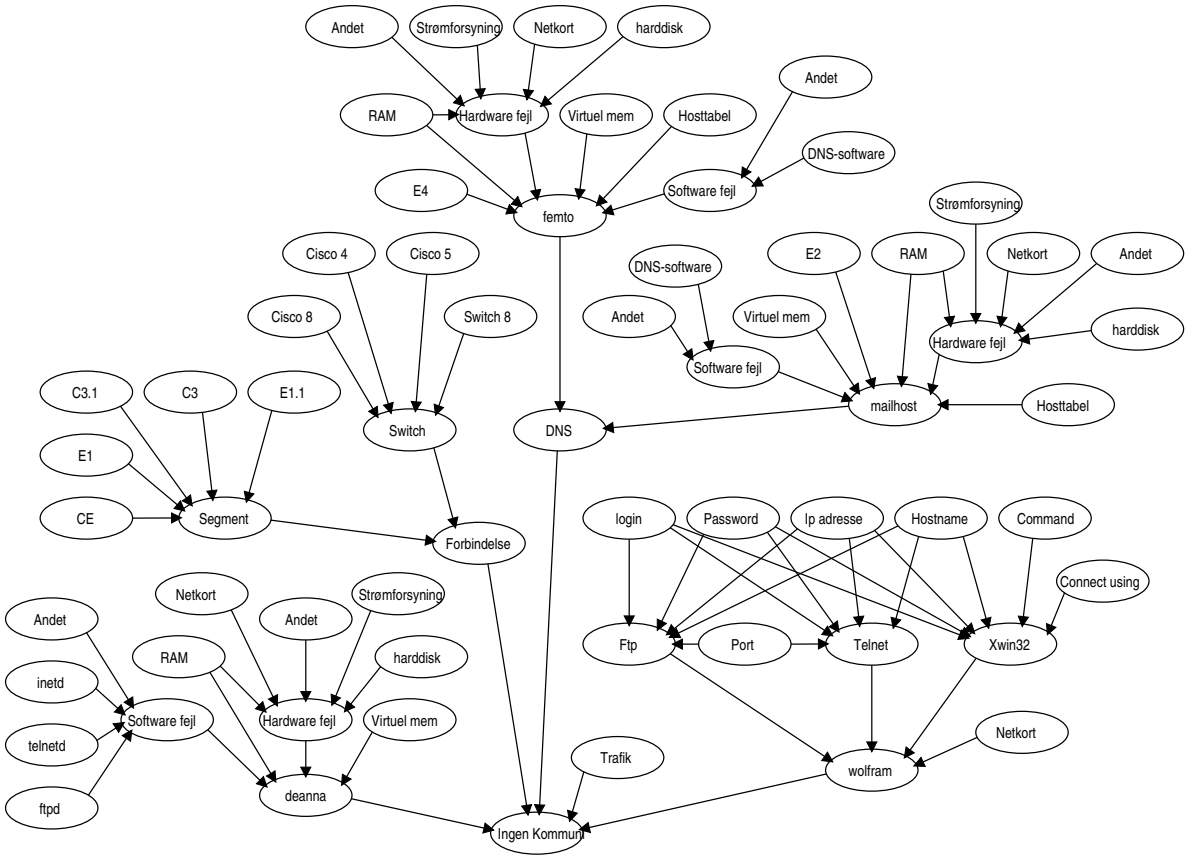
Figur A.2.: Kausalt netværk for de to DNS-servere femto og mailhost i CS-nettet

Figur A.3 viser et samlet kausalt netværk over forbindelsen mellem *arbejdsstationen wolfram* og serveren *deanna*

A.1.2. Kan ikke læse mail

Problemområdet, kan ikke læse mail, begrænses til at omfatte maillæsning med Pine[9] fra NT-maskinen wolfram via telnet eller Xwin32 til Unix-maskinen deanna (se figur A.1). Denne afgrænsning er hensigtsmæssig for at opnå et overskueligt problemområde. Det er muligt at ændre afgrænsningen ved f.eks. at erstatte Pine med Outlook[10] eller Messenger[11], og foretage andre justeringer, som gør, at problemområdet kan generaliseres til et samlet problemområde, som dækker flere former for maillæsning. Wolfram, som arbejdsstation, kan ligeledes erstattes med andre NT-maskiner på CS-netværket.

[9] Pine er en tekstbaseret Unix-mailreader udviklet på University of Washington.
[10] Windows baseret maillæsningsprogram fra Microsoft.
[11] Maillæsningsclient fra Netscape.



Figur A.3.: Kausalt netværk over kommunikationen mellem *arbejdsstationen wolfram* og serveren *deanna*

Brugerne på CS-nettet arbejder ofte fra deres grupperum. I den største del af grupperummene findes der flest NT-maskiner. Når brugeren så ønsker at benytte forskellige programmer på unix-systemet, etableres der en forbindelse til en unix maskine. Dvs. at brugeren i de fleste tilfælde har et terminal vindue op mod en unix maskine, når der arbejdes på NT-maskinen. Det virker derfor ofte naturligt at bruge Pine i et terminal vindue til at læse mail, når der nu alligevel arbejdes i et terminal vindue.

Idet 'Kan ikke læse mail' er det umiddelbare problem, en bruger oplever, antages WindowsNT på wolfram at køre og det antages, at en telnet eller Xwin forbindelse er korrekt etableret til deanna. De to antagelser bygger på, at en bruger på det tidspunkt han/hun oplever 'Kan ikke læse mail' som et umiddelbart problem, da vil brugeren have en korrekt etableret telnet eller Xwin forbindelse til deanna. I modsat fald vil de umiddelbare problemer have været 'wolfram virker ikke' eller 'ingen kommunikation'.

Årsagerne til 'Kan ikke læse mail'-problemet beskrives i de efterfølgende afsnit.

Brugerfejl

En årsag til ikke at kunne læse mail, er problemer, som forårsages af brugerfejl. Brugerfejlene sker under anvendelse af mailprogrammet Pine.

Mailhost

En anden årsag til problemer med at læse mail er mailhost. Mailhost er en speciel maskine i problemområdet, da den både er mailserver og DNS-server. Vi ser derfor på mailhosts funktioner hver for sig.

Mailhost mailserver
: Mailserveren kan være årsag, til at der ikke kan læses mail, hvis mailserver programmet er gået ned på grund af en software fejl.

Mailhost DNS-server
: Hvis der sker en sofware fejl i den software som udgør DNS-serveren, kan det få DNS-server programmet til at gå ned. Foruden en fejl i softwaren kan en fejlbehæftet hosttabel udgøre et problem.

Fælles årsager til mailserver og DNS-server
: Mailhost mailserver og mailhost DNS-server er fysisk den samme maskine. Derfor er der en række problemer, som kan være årsag til at der ikke kan læses mail, som gør sig gældene for begge.

  Hvis der er en hardware defekt, f.eks RAM, harddisk, strømforsyning eller netkort på mailhost, vil det få betydning for både DNS-serveren og mailserveren. Ingen af serverne ville kunne udføre deres arbejdsopgaver, og dermed ville brugeren heller ikke kunne læse mail.

  Korrupt virtual memory kan få betydning for både mailserveren og DNS-serveren på mailhost. Korrupt virtual memory kan få begge servere til at gå ned. Virtual memory på mailhost betragtes derfor, som en fælles enhed i problemområdet.

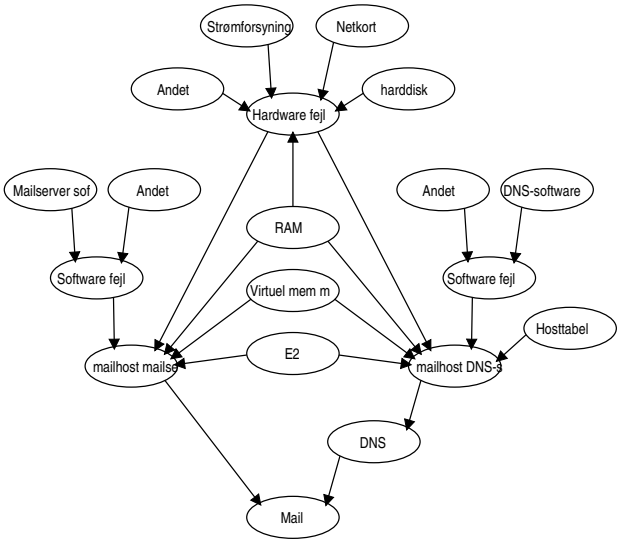
  En paritets fejl, kan også få både mailserveren og DNS-serveren på mailhost til at gå ned. Derfor betragtes RAM som en fælles enhed i problemområdet.

Nedenstående figur A.4 illustrerer det kausale netværk for mailhost.

Kes

Hvis der opstår problemer på filserveren kes (se figur A.1), hvor alle brugernes[12] hjemmekataloger ligger, kan det have indflydelse på mailserveren. Hvis filserveren er i en

[12]Med bruger menes de studerende, som har adgang til cs-netværket, professorer og ph.d mv. er tilknyttet en anden filserver.

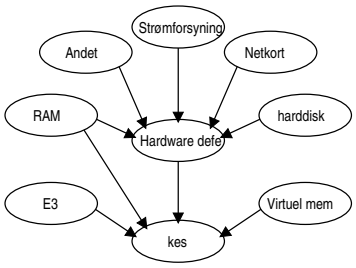


Figur A.4.: Kausalt netværk for mailhost.

fejltilstand, og mailserveren forsøger at læse forward-filen[13] i brugerens hjemmekatalog på kes, vil mailserveren ikke modtage noget svar. Forsøger mailserveren at læse forward-filen, for for mange brugere indenfor for kort tid, kan det resultere i, at mailserveren går ned, da den bliver overbelastet af ventende jobs.

Hardware defekter på kes eller korrupt virtual memory kan være årsag til, at kes ikke kan udføre sin opgave som filserver. Af mulige defekte hardware enheder, kan der nævnes RAM, strømforsyning, netkort og harddisk. Hvis det sker at virtual memory bliver korrupt eller at der opstår en paritets fejl i RAM, kan det få kes til at gå ned.

Det kausale netværk for kes kommer til at se ud som på figur A.5.



Figur A.5.: Kausalt netværk for filserveren kes.

[13]Med forward filen menes der den fil som indeholder mailadresser hvortil der skal forwardes mail

Forbindelse

En anden direkte årsag til mail er problemer med den fysiske forbindelse. Problemer med et defekt segment vil påvirke forbindelsen, hvilket igen kan betyde, at brugeren ikke længere kan læse mail. Ligeledes hvis en switch er defekt eller gået ned som følge af f.eks. en softwarefejl, kan det være grund til problemer med forbindelsen, og derigennem årsagen til, at brugeren ikke kan læse sin mail.

I konfigurationen af Pine er der angivet navnet på mailserveren, som Pine skal hente mails fra. Når navnet på mailserveren bruges, er det en forudsætning at DNS-serveren kan bruges. Hvis det sker at ingen af DNS-serverne er til rådighed, hvad enten det er et kabelbrud eller et andet problem, vil Pine ikke kunne få forbindelse til mailserveren.

Mailhost
: Da problemområdet er afgrænset til at være maillæsning via deanna, vil segmenter og switche, der er en del af forbindelsen mellem wolfram og deanna blive antaget at være i en normaltilstand. Dermed er forbindelsen begrænset til at være segment E2 mellem Cisco 4 og mailserveren mailhost (se figur A.1). Segment E2 kan afskære forbindelsen til mailserveren og dermed være en årsag til, at der ikke kan læses mail.

DNS
: Segmenterne E2 og E4 udgør forbindelsen til DNS-serverne og kabelbrud kan dermed afskære forbindelsen til DNS-serverne. Derfor kan E2 og E4 være en årsag til, at der ikke kan læses mail.

Kes
: Forbindelsen fra mailserveren til kes i CS-nettet udgøres af segment E2 og E3. Hvis der sker et kabelbrud i segment E3, kan mailhost ikke læse forward-filen på kes, og det kan medføre at mailserveren går ned.

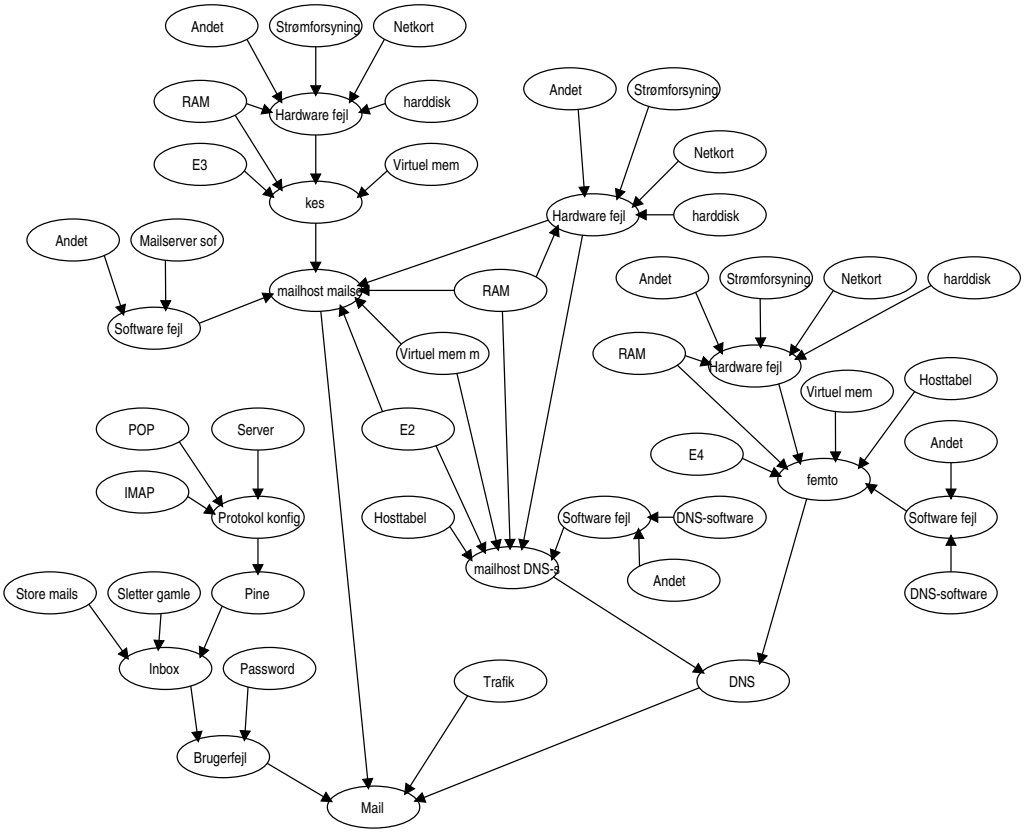
Trafik

En høj datatrafik på CS-nettet kan være årsag til, at der ikke kan logges på mailserveren. Den høje datatrafik kan udløse en timeout på brugerens forspørgsel om at logge på mailserveren. Hvis der ikke kan logges på mailserveren, kan der heller ikke læses mail.

Hvis de ovenfor beskrevne netværk samles i et kausalt netværk, kommer det til at se ud som på figur A.6.

A.1.3. Kan ikke udskrive

I dette afsnit beskrives en situation, hvor en bruger ikke kan udskrive. Problemet med at udskrive kan skyldes forskellige årsager, såsom printeren der udskrives til, brugerenskonto, printserveren, arbejdsstationen og den fysiske forbindelse, herunder DNS.

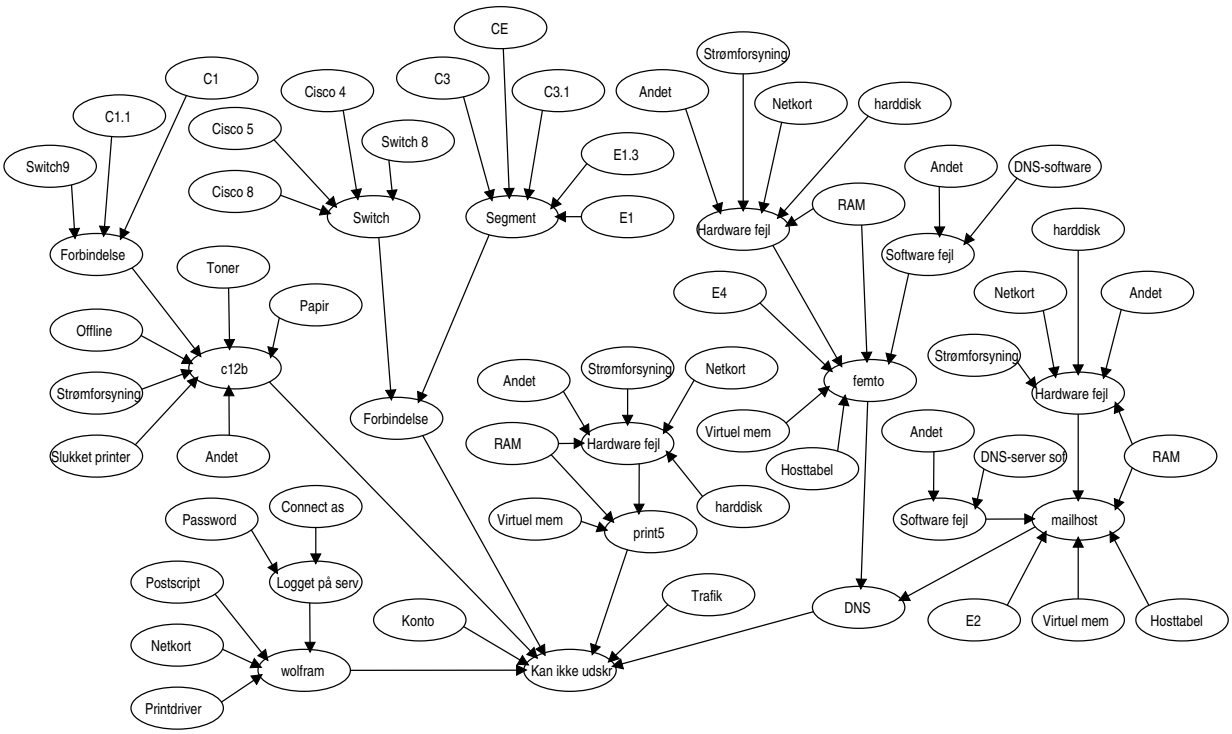


Figur A.6.: Kausalt netværk for problemområdet mail.

Vi vil tage udgangspunkt i tilfældet, hvor brugeren forsøger at udskrive fra arbejdsstationen wolfram på printer c12b. Deres placering i netværket kan ses på figur A.1 i afsnit A.1.

Som ved beskrivelsen af problemområdet ingen kommunikation og kan ikke læse mail er der gjort en række antagelser. Det antages igen, at WindowsNT på wolfram har en korrekt opsætning, der muliggør, at brugeren kan logge på printerserveren. Desuden antages det, at brugeren kan starte wolfram op og logge på, før der forsøges at lave en udskrift. Der ses heller ikke på software og hardware fejl på wolfram, som kan forårsage, at wolfram ikke er i en brugbar tilstand. Det er problemområdet "Kan ikke udskrive", vi ønsker at modellere, og ikke et problemområde, der dækker fejl på wolfram. Der ses dog på forkert brug af software på wolfram, der benyttes til at udskrive. Derfor er wolfram en årsag til kan ikke udskrive.

Et kausal netværk der dækker hele problemområdet vil det se ud som på figur A.7.



Figur A.7.: Kausalt netværk for problemområdet, kan ikke udskrive.

Litteratur

[Andersen et al., 1999] Andersen, J., Andersen, L. T., Hahn, F. G., Hansen, M. B., Jensen, R., Olesen, H., and Vinther, P. M. (1999). System til fejlfinding på cs-nettet. Technical report, Dept. of Computer Science, Aalborg University.

[Breese and Heckerman, 1995] Breese, J. S. and Heckerman, D. (1995). *Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment.* Microsoft Research, One Microsoft Way, Redmond.

[Hahn and Andersen, 2001] Hahn, F. and Andersen, L. T. (2001). Topologisk objektorienteret model generator. Technical report, Aalborg Universitet.

[Heckerman et al., 1995] Heckerman, D., Breese, J. S., and Rommelse, K. (1995). Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–56. Special issue on real-world applications on Bayesian networks.

[Hugin, 2000] Hugin (2000). *Hugin API Reference Manual, Version 4.2.* Hugin Expert A/S.

[Jensen, 2000] Jensen, F. V. (2000). *Kompendium Beslutningsstøttesystemer. Uddrag af bogen 'Bayesian Networks and Decision Graphs'.* Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7, DK-9220 Aalborg Ø, international edition.

[Jensen et al., 2001] Jensen, F. V., Kjærulff, U., Kristiansen, B., Langseth, H., Skaanning, C., Vomlel, J., and Vomlelová, M. (2001). The sacso methodology for troubleshooting complex systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM) - To Appear in a Special Issue on AI in Equipment Service.* To appear in a Special Issue on AI in Equipment Service.

[Kalagnanam and Henrion, 1990] Kalagnanam, J. and Henrion, M. (1990). A comparison of decision analysis and expert rules for sequential diagnosis. *Uncertainty in Artificial Intelligence*, pages 271–281.

[Langseth and Jensen, 2001] Langseth, H. and Jensen, F. V. (2001). Heuristics for two extensions of basic troubleshooting. *Seventh Scandinavian Conference on Artificial Intelligence (SCAI '01).*

[Pressman, 1997] Pressman, R. S. (1997). *Software Engineering - A Practitioner's Approach.* McGraw-Hill.

[Skaanning et al., 2000] Skaanning, C., Jensen, F. V., and Kjærulff, U. (2000). Printer troubleshooting using bayesian networks. In *Proceedings of the Thirteenth International Conference on Industrial & Enginnering Applications of AI & Expert Systems (IEA/AIE-2000).*

[Sochorova and Vomlel, 2000] Sochorova, M. and Vomlel, J. (2000). Np-hardness and solution methods. *The Proceedings of the Fifth Workshop on Uncertainty Processing, WUPES'2000 , Jindrichuv Hradec, the Czech Republic.*

[Srinivas, 1995] Srinivas, S. (1995). *A polynomial algorithm for computing the optimal repair strategy in a system with independent component failures.* UAI-95, Computer Science Department, Stanford University, Stanford, CA 94305, international edition.