Modellering af afhængigheder imellem variable ved brug af grafiske modeller

Introduktion til bayesianske netværk og t-kirsebærtræer

Katrine Engilbertsdóttir Kirkeby, Maria Knudsen og Ninna Vihrs

Aalborg Universitet
Kandidatuddannelsen i
Matematik - Speciale
September 2018 til juni 2019

AALBORG UNIVERSITET
STUDENTERRAPPORT

Kandidatuddannelsen i
Matematik - Speciale
Skjernvej 4A
9220 Aalborg Øst
http://www.math.aau.dk

Titel: Modellering af afhængigheder imellem variable ved brug af grafiske modeller

Tema: Statistik modellering med t-kirsebærtræer

Projektperiode: September 2018 til juni 2019

Projektgruppe: 5.217b

Deltagere:
Katrine Engilbertsdóttir Kirkeby
Maria Knudsen
Ninna Vihrs

Vejleder: Mikkel Meyer Andersen

Oplagstal: 4

Sidetal: 134

Afleveringsdato: 20. maj 2019

Synopsis:

Dette speciale giver en kort introduktion til bayesianske netværk og Markovnetværk. Det primære fokus er på en type af Markovnetværk kaldet k'te ordens t-kirsebærtræer. Der præsenteres både teoretiske, praktiske og numeriske resultater samt en del algoritmer til at lære strukturen af et t-kirsebærtræ ud fra data. Disse forsøger at finde en struktur af maksimal likelihood ved at maksimere strukturens såkaldte vægt. Der udledes både algoritmer, som undersøger alle mulige strukturer, og grådige algoritmer, som stepvis inkluderer kliker i grafen. De grådige algoritmer er hurtigere, og numeriske undersøgelser tyder på, at de ofte finder en struktur med maksimal vægt, samt at de ofte er i stand til at genfinde den sande struktur, såfremt denne er et t-kirsebærtræ. Analyser på forskellige datasæt viser også, at t-kirsebærtræer kan give gode klassificeringer, såfremt der er tilstrækkeligt data, og der vælges en tilpas lav orden. Grundet et stort antal kanter i højere ordens t-kirsebærtræer risikerer disse hurtigt at overtilpasse, hvorfor der også udledes en algoritme til at udtynde kanterne i en tilpasset struktur baseret på en likelihood ratio test. Denne algoritme benyttes med succes på udvalgte strukturer, hvilket giver simplere modeller enten med samme eller bedre præstationer.

Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Abstract

This Master's Thesis explores the possibility to describe dependencies between categorical variables with graphical models both with or without orientation. The main focus is however on a special type of undirected graphical models called k^{th} order t-cherry trees and how to learn such a structure from data.

The first chapter gives a very short introduction to some graph theoretic concepts and notation followed by a chapter introducing the type of graphical models with orientation called Bayesian networks plus the methods called hill-climbing and Chow-Liu trees for learning the structure. Except presenting the concept of Bayesian networks and the corresponding probability distribution, this chapter also presents general results about estimating probability distributions from data and some results for triangulated graphs. This information is also relevant for undirected graphical models. The rest of the thesis is about k^{th} order t-cherry trees exploring both some theoretic properties of the structure and related probability distribution and the possibility to construct algorithms for learning the structure from data.

Expressions for the likelihood function and Kullback-Leibler divergence to the true distribution for k^{th} order t-cherry trees are derived. Both these expressions involves what is called the weight of the structure which should be maximized in order to get the best structure based on the Kullback-Leibler divergence and likelihood. A variety of different algorithms for learning the structure by maximizing this weight is derived. These algorithms either increase the order of a lower order t-cherry tree or construct a kth order t-cherry tree directly from data. The algorithms attempt to maximize the weight either by a greedy search where cliques are added stepwise to the graph or by a complete search between all possible structures. Numeric results show that increasing the order is faster than learning the structure directly from data. When it comes to identifying the true structure of the graph and get the best approximation to the true probability distributions measured by Kullback-Leibler divergence, it is however better to construct the t-cherry tree directly from data. Numerical results also suggest that the greedy approaches often find structures whose weight are either maximal or close to. This can be improved further by adding more cliques in each step, but at the cost of longer execution time.

Analyses is made on selected data sets to assess the usefulness of kth order t-cherry trees for classification. They can give very accurate classifications compared to other graphical models if the data set is sufficient and a proper order is chosen. If this is not the case, high order t-cherry trees seem to suffer from overfitting. Therefore an approach to delete edges from a fitted structure is derived. This method deletes edges successively by applying a likelihood ratio test, and has with success been used on high order t-cherry trees to prevent overfitting and simplify models.

Forord

Dette speciale er udarbejdet af tre studerende på 3. og 4. semester af kandidatuddannelsen i Matematik ved Aalborg Universitet i 2018-19. Det forventes, at læseren har basalt kendskab til sandsynligheds- og grafteori.

Specialet starter med et kort afsnit med genopfriskning af grafteoretiske begreber og notation, herefter følger en introduktion til bayesianske netværk efterfulgt af afsnit omhandlende begrebet t-kirsebærtræer, som størstedelen af dette speciale vedrører. Hovedformålet med specialet er at udforske t-kirsebærtræer både teoretisk og praktisk. Hvis et afsnit primært er baseret på enkelte kilder, er disse angivet i afsnittets begyndelse. Kilder angives generelt med forfatter og årstal.

Alle statistiske analyser er lavet med softwareprogrammet **R** (R Core Team 2018), og tilhørende plot er enten lavet med pakken `ggplot2` (Wickham 2016) eller `GGally` (Schloerke et al. 2018). I forbindelse med udarbejdelsen af specialet har vi udviklet en **R**-pakke, som kan findes på `https://github.com/nvihrs14/tcherry`. En komplet liste over funktionerne i pakken kan findes i Bilag F.

Alle definitioner, sætninger, propositioner, lemmaer, algoritmer og eksempler er nummereret fortløbende og efter afsnit samt underafsnit. Figurer, tabeller og ligninger er ligeledes nummereret efter afsnit og underafsnit. Beviser afsluttes med ■. Sidst i specialet findes en liste over anvendt notation.

Vi vil gerne takke vores vejleder Mikkel Meyer Andersen for god og konstruktiv vejledning samt for at stille data til rådighed. Desuden ønsker vi at takke Poul Svante Eriksen og Mads Lindskou for et konstruktivt statusseminar og gode forslag.

Aalborg Universitet, 20. maj 2019

Katrine Engilbertsdóttir Kirkeby
kkirke13@student.aau.dk

Maria Knudsen
mknud14@student.aau.dk

Ninna Vihrs
nvihrs14@student.aau.dk

Indholdsfortegnelse

Abstract **i**

Forord **ii**

1 Indledning **1**

1.1 Problemformulering 1

1.2 Projektafgrænsning 1

2 Introduktion til grafteori **2**

3 Bayesianske netværk **4**

3.1 Betingede uafhængigheder og d-separation 5

3.2 Simultan sandsynlighedsfordeling 6

3.3 Marginale fordelinger i bayesianske netværk 10

3.3.1 Domænegrafer 13

3.4 Triangulerede grafer 16

3.4.1 Tilslutnings- og junktionstræer 20

3.5 Parameterestimation 28

3.5.1 Maksimum likelihood estimation 28

3.5.2 Bayesiansk estimation 30

3.6 Bestemmelse af strukturen i bayesianske netværk 31

3.6.1 Hill-climbing 32

3.6.2 Chow-Liu-træer 33

4 Tredje ordens t-kirsebærtræer **38**

4.1 Sandsynlighedsfordelinger baseret på t-kirsebærtræer 39

4.2 Implementering af konstruktionsalgoritmer 47

4.3 Simulationsstudie til vurdering af konstruktionsalgoritmer 49

5 Anvendelse af t-kirsebærtræer, Chow-Liu-træer og hill-climbing på dansk Y-STR data **56**

6 Højere ordens t-kirsebærtræer **62**

6.1 Klassen af k'te ordens tilslutningstræer 64

6.2 Udvidelse af k'te ordens t-kirsebærtræer til *(k+1)*'te ordens t-kirsebærtræer 77

6.3 Udvidelse og optimering af konstruktionsalgoritmer 81

6.3.1 k'te ordens t-kirsebærtræ konstrueret direkte fra data 81

6.3.2 *(k+1)*'te ordens t-kirsebærtræ konstrueret fra et k'te ordens 85
6.3.3 Grådig algoritme, der tilføjer p kliker ad gangen 91
6.3.4 Udtømmende afsøgning 93
6.4 Simulationsstudie til vurdering af grådige konstruktionsalgoritmers kvalitet . 95

7 Anvendelse af k'te ordens t-kirsebærtræer og hill-climbing til klassificering 99

8 Udtynding af kanter 102
8.1 Algoritme til udtynding af kanter 108
8.2 Anvendelse af algoritme til udtynding af kanter 110

9 Konklusion 112

Litteraturliste 114

A Bevis for at (3.2.2) er en sandsynlighedsfordeling 117

B Informationsteori 118

C Supplerende simulationer til Afsnit 4.3 121

D Information om STR loci 129

E Supplerende analyser til Afsnit 5 130

F Oversigt over funktioner i pakken `tcherry` 131

Liste over notation 133

1 Indledning

Grafiske modeller kombinerer grafteori og sandsynlighedsregning. Den grundlæggende idé er at visualisere variables afhængigheder via en graf og udnytte disse til at beskrive sandsynlighedsfordelingen. For at lære den grafiske struktur, der beskriver sådanne afhængigheder, er det enten nødvendigt at benytte ekspertudsagn eller at drage de nødvendige konklusioner fra observeret data. Hvis konklusionerne baseres på data, er det typisk nyttigt at begrænse antallet af parametre i modellerne ved at underlægge strukturen bestemte begrænsninger. En sådan begrænsning, som giver en meget simpel struktur, foreslog Chow & Liu (1968), hvor den grafiske struktur er det, der i grafteori kaldes et træ. Inden for grafiske modeller er dette kendt som et Chow-Liu-træ. Denne struktur medfører blandt andet, at det ikke er nødvendigt at approksimere simultane sandsynlighedsfordelinger for mere end 2 variable, grundet den meget simple afhængighedsstruktur. I nyere tid er det forsøgt at generalisere Chow-Liu-træer, og Kovács & Szántai (2012) foreslår en grafisk struktur, kaldet et k'te ordens t-kirsebærtræ, hvor det ikke er nødvendigt at bestemme simultane sandsynlighedsfordelinger for mere end k variable. Specialtilfældet $k = 2$ reducerer til Chow-Liu-træet. Det er muligt at finde det optimale Chow-Liu-træ ud fra data med en effektiv grådig algoritme (Chow & Liu 1968), men for et generelt k'te ordens t-kirsebærtræ er dette ikke længere tilfældet. Dermed opstår behovet for algoritmer, der effektivt er i stand til at finde en optimal eller tæt på optimal struktur, som er et k'te ordens t-kirsebærtræ. Derfor omhandler dette speciale især udviklingen af sådanne algoritmer, implementeringen af dem og kvaliteten af resultaterne. For at bestemme, hvad der menes med en optimal struktur, udledes også teoretiske formler for likelihood og Kullback-Leibler-afstand.

1.1 Problemformulering

Hvordan kan afhængigheder mellem variable modelleres med grafiske modeller, specielt den type af modeller, der kaldes t-kirsebærtræer? Hvilke algoritmer kan opstilles for at lære grafens struktur, hvor gode er disse i forhold til den approksimerede sandsynlighedsfordeling, korrektheden af den fundne struktur, eksekveringshastigheden og de prædiktive evner?

1.2 Projektafgrænsning

Dette speciale afgrænser sig til kun at behandle kategoriske variable og strukturlæring via observeret data. Desuden afgrænser det sig fra at behandle triangulering af ikke-trianguleredе grafer.

2 Introduktion til grafteori

Dette speciale omhandler forskellige typer af grafiske modeller. Udgangspunktet for alle disse modeller er den matematiske struktur, der kaldes en graf. Derfor vil dette afsnit introducere basal grafteori, og yderligere information kan findes i Rosen (2012), som dette afsnit også er baseret på. Definition 2.0.1 definerer tre typer af grafer.

Definition 2.0.1 (Orienteret, ikke-orienteret og vægtet graf)
En graf $G = (\mathcal{V}, \mathcal{E})$ består af en mængde af knuder $\mathcal{V} = \{A_1, \ldots, A_n\}$ og en mængde af kanter $\mathcal{E}$. Mængden af kanter kan enten bestå af ordnede par på formen (A_i, A_j) eller uordnede par på formen $\{A_i, A_j\}$. Hvis den består af ordnede par kaldes grafen orienteret, hvor kanten (A_i, A_j) er orienteret fra A_i til A_j. Knuden A_i refereres til som A_j's forælder, og A_j kaldes A_i's barn. Hvis parrene er uordnede kaldes grafen ikke-orienteret. Hvis alle kanter har en tilknyttet vægt, kaldes grafen en vægtet graf.

To knuder forbundet af en kant kaldes naboer. Mængden af alle naboer til knuden A kaldes nabomængden og benævnes $na(A)$. Mængden bestående af knuden selv og dens naboer benævnes i dette projekt $fa(A) = \{A\} \cup na(A)$. En mængde af knuder, hvor alle elementer er naboer, kaldes en komplet punktmængde. Hvis $na(A)$ er komplet, kaldes A en simpel knude. Hvis det for en graf $G = (\mathcal{V}, \mathcal{E})$ gælder, at $\mathcal{V}$ er komplet, kaldes grafen en komplet graf. Definition 2.0.2 definerer en særlig type af komplette punktmængder i en graf.

Definition 2.0.2 (Klike)
Lad $G = (\mathcal{V}, \mathcal{E})$ være en graf. Mængden $K \subseteq \mathcal{V}$ kaldes en klike, såfremt mængden er komplet, og for alle $v \in \mathcal{V} \setminus K$ er mængden $K \cup \{v\}$ ikke komplet.

Definition 2.0.3 og 2.0.4 definerer henholdsvis en sti og en kreds i en graf.

Definition 2.0.3 (Sti i en graf)
Hvis der er en sti fra A_i til A_j, betyder det, at det er muligt at gå fra A_i til A_j ved at følge grafens kanter. Hvis grafen er orienteret, skal orienteringerne på kanterne overholdes. De kanter, der benyttes, udgør selve stien, og de knuder, der passeres under vejs, kaldes knuder på stien. Hvis hver kant kun benyttes en gang, kaldes stien simpel.

Definition 2.0.4 (Kreds i en graf)
En kreds i en graf er en sti, der starter og ender i samme knude. Hvis der eksisterer en eller flere kredse i en graf, kaldes den cyklisk, ellers kaldes den acyklisk.

En graf kan blandt andet repræsenteres ved en såkaldt nabomatrix.

Definition 2.0.5 (Nabomatrix)
Lad $G = (\mathcal{V}, \mathcal{E})$ være en ikke-orienteret graf, hvor $\mathcal{V} = \{A_1, \ldots, A_n\}$. Nabomatricen E for grafen er matricen med indgangene

$$E_{ij} = \begin{cases} 1 & \{A_i, A_j\} \in \mathcal{E} \\ 0 & \text{ellers} \end{cases}.$$

Hvis G er orienteret har nabomatricen indgangene

$$E_{ij} = \begin{cases} 1 & (A_i, A_j) \in \mathcal{E} \\ 0 & \text{ellers} \end{cases}.$$

Det bemærkes, at nabomatricen for en ikke-orienteret graf er symmetrisk, hvorimod det typisk ikke er tilfældet for en orienteret graf.

En ikke-orienteret graf kaldes sammenhængende, hvis det er muligt at finde en sti mellem alle par af knuder. Definition 2.0.6 beskriver en særlig type sammenhængende graf, kaldet et træ.

Definition 2.0.6 (Træ)
En ikke-orienteret sammenhængende acyklisk graf kaldes et træ.

Definition 2.0.7 (Maksimalt udspændende træ)
Lad $G = (\mathcal{V}, \mathcal{E})$ være en vægtet sammenhængende ikke-orienteret graf. Et træ $T = (\mathcal{V}, \mathcal{E}')$ kaldes et udspændende træ for G, hvis $\mathcal{E}' \subseteq \mathcal{E}$. Dette træ er ydermere et maksimalt udspændende træ, hvis summen af kantvægtene i $\mathcal{E}'$ er maksimal.

Det er muligt at danne en orienteret graf ud fra et træ ved at vælge en rod og orientere alle kanter i retningen væk fra roden.

De næste afsnit omhandler, hvordan grafer kan anvendes i forbindelse med modellering af afhængigheder mellem variable.

3 Bayesianske netværk

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 2.3).

Formålet med dette afsnit er at give en grundlæggende introduktion til bayesianske netværk.

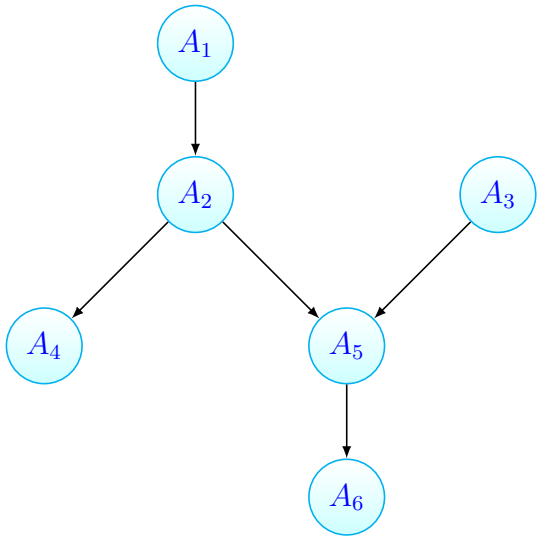
Definition 3.0.1 (Bayesiansk netværk)

Et bayesiansk netværk er en orienteret graf, der opfylder følgende:

- Hver knude repræsenterer en diskret stokastisk variabel, som har et endeligt udfaldsrum af gensidigt udelukkende tilstande.[a] Hvis der i forbindelse med bayesianske netværk nævnes et variabelnavn, kan det både hentyde til variablen og den knude, som repræsenterer den.
- Grafen er acyklisk (forkortes DAG for *Directed Acyclic Graph*).
- For hver variabel A er sandsynligheden $P(A|pa(A))$ specificeret, hvor $pa(A)$ betegner mængden af forældre til A.

[a] Det er muligt at konstruere bayesianske netværk med kontinuerte variable, men det bliver ikke behandlet i dette speciale.

Et eksempel på et bayesiansk netværk kan ses i Figur 3.0.1.



Figur 3.0.1: Et eksempel på et bayesiansk netværk med variablene $A_1, \ldots, A_6$ med tilhørende sandsynligheder $P(A_1)$, $P(A_2|A_1)$, $P(A_3)$, $P(A_4|A_2)$, $P(A_5|A_2, A_3)$, $P(A_6|A_5)$.

3.1 Betingede uafhængigheder og d-separation

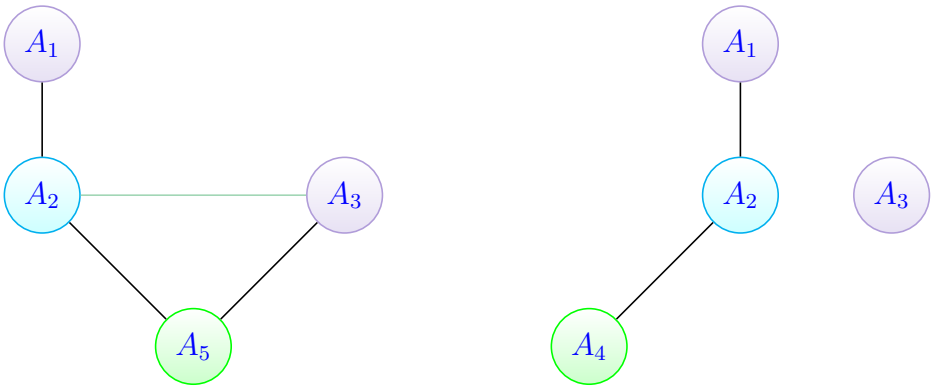
Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 2.2).
Et bayesiansk netværk illustrerer blandt andet betingede uafhængigheder mellem variable. Dette er tæt knyttet til det begreb, der hedder d-separation. Det bayesianske netværk bør være konstrueret på en sådan måde, at hvis A_i og A_j er d-separeret givet $\mathcal{G}$, er A_i betinget uafhængig af A_j givet $\mathcal{G}$. Den letteste måde at bestemme, om to variable er d-separeret givet en mængde af variable $\mathcal{G}$, er ved at betragte den såkaldte moralske graf, som konstrueres ud fra den såkaldte forfædrende graf.

Den forfædrende graf konstrueres ved at fjerne alle knuder, som ikke ligger på en orienteret sti mod enten A_i, A_j eller en variabel i $\mathcal{G}$. Den forfædrende graf indeholder dermed kun variablene og deres forfædre. Denne graf udvides til den moralske graf ved at forbinde alle knuder, som har et fælles barn, og herefter fjerne alle orienteringer. Det at forbinde knuder med fælles barn kaldes at gifte knuderne.

Variablene A_i og A_j er d-separeret givet $\mathcal{G}$, såfremt alle stier mellem A_i og A_j i den moralske graf går igennem $\mathcal{G}$. Hvis der ikke eksisterer nogen stier mellem A_i og A_j, er de også d-separeret. Hvis A_i og A_j ikke er d-separeret, er de d-sammenhængende givet $\mathcal{G}$. Et eksempel på hvordan dette kan benyttes, ses i Eksempel 3.1.1.

Eksempel 3.1.1

Dette eksempel tager udgangspunkt i det bayesianske netværk i Figur 3.0.1. For at afgøre, om A_1 og A_3 er d-separeret givet henholdsvis A_5 og A_4 konstrueres de tilhørende moralske grafer. Disse ses i Figur 3.1.1.



(a) En moralsk graf til undersøgelse af om A_1 og A_3 er d-separeret givet A_5.

(b) En moralsk graf til undersøgelse af om A_1 og A_3 er d-separeret givet A_4.

Figur 3.1.1: Eksempler på moralske grafer for det bayesianske netværk i Figur 3.0.1.

I Figur 3.1.1a går stien A_1, A_2, A_3 ikke gennem A_5. Derfor er A_1 og A_3 d-sammenhængende givet A_5. I Figur 3.1.1b er der ingen stier mellem A_1 og A_3, hvorfor disse er d-separeret givet A_4. Det bemærkes desuden, at A_1 og A_3 altid vil være d-separeret medmindre A_5 eller A_6 er givet.

I et bayesiansk netværk gælder der altid en særlig d-separationsegenskab, som beskrives i Proposition 3.1.2.

Proposition 3.1.2
Betragt et bayesiansk netværk over mængden af variable $\{A_1, \ldots, A_n\}$. Det gælder, at A_i og enhver variabel, som ikke er efterkommer af A_i, er d-separeret givet $pa(A_i)$.

Bevis. Lad A_j, $j \neq i$, være en variabel, som opfylder, at $A_j \notin pa(A_i)$, og at A_j ikke er efterkommer til A_i. Den forfædrende graf til at undersøge, om A_j er d-separeret fra A_i givet $pa(A_i)$, indeholder kun forfædre til variablene $A_i, pa(A_i), A_j$. Dette betyder, at ingen af A_i's børn kan indgå i grafen, da

- ingen børn til A_i kan være forfader til A_i eller dennes forældre, da dette ville skabe en orienteret kreds,
- ingen børn til A_i kan være forfader til A_j, da A_j i så fald ville være efterkommer af A_i.

Dette betyder, at i den moralske graf kan A_i aldrig giftes med en anden variabel.

I den moralske graf vil enhver sti til A_i gå enten via børn, forældre eller ægteskab. Da hverken børn eller ægteskab er muligt, må enhver sti mellem A_j og A_i gå gennem $pa(A_i)$, hvilket giver, at A_j og A_i er d-separeret givet $pa(A_i)$.

■

3.2 Simultan sandsynlighedsfordeling

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 2.3.2).
Ud fra et bayesiansk netværk er det muligt at bestemme en simultan sandsynlighedsfordeling for dets variable. Denne sandsynlighedsfordeling skal overholde de samme egenskaber, som er tilknyttet det bayesianske netværk. Specielt skal de betingede uafhængigheder, bestemt ved grafens d-separationsegenskaber, være overholdt. Hvis dette kriterium skal være opfyldt er det muligt at bestemme et udtryk for, hvordan den simultane sandsynlighedsfordeling nødvendigvis må se ud.

Lad $A_1, \ldots, A_n$ være en topologisk ordning ud fra DAG'en for det bayesianske netværk. At det er en topologisk ordning betyder, at hvis der er en orienteret kant fra A_i til A_j, er $i < j$.

Dette betyder specielt, at alle efterkommere af en variabel har højere indeks end denne, og alle forfædre har lavere. Det er muligt at lave en sådan ordning, da grafen ingen orienterede kredse har, hvorfor en knude ikke både kan være forfader og efterkommer til en variabel.

Per almindelige regneregler for sandsynligheder er

$$P(A_1, \ldots, A_n) = P(A_1) \prod_{i=2}^{n} P(A_i | A_{i-1}, \ldots, A_1). \tag{3.2.1}$$

Det udnyttes nu, at variablene skal opfylde de betingede uafhængigheder bestemt ved grafens d-separationer. Betragt en faktor $P(A_i|A_{i-1}, \ldots, A_1)$. Det bemærkes, at på grund af den topologiske ordning indeholder mængden $\{A_{i-1}, \ldots, A_1\}$ nødvendigvis alle A_i's forældre og ingen af A_i's efterkommere. Ved at udnytte d-separationsegenskaberne i Proposition 3.1.2 forsimples (3.2.1) derfor til

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P(A_i | pa(A_i)). \tag{3.2.2}$$

Per konvention sættes $P(A_i, \emptyset) = P(A_i)$ og $P(\emptyset) = 1$, hvilket også medfører, at $P(A_i|\emptyset) = P(A_i)$. I dette udtryk ses det, at sandsynlighedsfordelingen bliver produktet af de sandsynligheder, der er tilknyttet det bayesianske netværk. Det er nu stadfæstet, at hvis der skal findes en sandsynlighedsfordeling, som opfylder kravet, må den nødvendigvis være på formen (3.2.2). Proposition A.0.1 i Bilag A giver først og fremmest, at (3.2.2) er en sandsynlighedsfordeling. Proposition 3.2.1 giver, at (3.2.2) også overholder de betingede uafhængigheder bestemt ved grafens d-separationsegenskaber som ønsket.

Proposition 3.2.1
Sandsynlighedsfordelingen P i (3.2.2) opfylder, at hvis A_i og A_j er d-separeret givet en mængde $\mathcal{G}$, er $P(A_i|A_j, \mathcal{G}) = P(A_i|\mathcal{G})$.

Bevis. Resultatet bevises ved induktion. For et bayesiansk netværk med kun én variabel er det trivielt, at de betingede uafhængigheder er opfyldt.

Antag, at det er sandt for et bayesiansk netværk med $n - 1$ knuder. Betragt nu et bayesiansk netværk BN_n med n topologisk ordnede knuder $A_1, \ldots, A_n$. Lad BN_{n-1} være det samme bayesianske netværk, men med A_n fjernet. Grundet den topologiske ordning har A_n ingen børn. Derfor er alle betingede sandsynligheder som er tilknyttet BN_{n-1} også tilknyttet BN_n, der herudover kun har $P(A_n|pa(A_n))$ tilknyttet.

Da A_n ingen børn har, kan den kun indgå i en forfædrende graf, hvis den er en af de relevante variable. Hvis der kun arbejdes med variablene $A_1, \ldots, A_{n-1}$, vil den forfædrende

graf derfor få samme struktur for begge bayesianske netværk. Dermed har de to bayesianske netværk de samme d-separationsegenskaber for variablene $A_1, \ldots, A_{n-1}$.

Desuden ses det, at

$$\begin{aligned} P(A_1, \ldots, A_{n-1}) &= \sum_{A_n} \prod_{i=1}^{n} P(A_i|pa(A_i)) \\ &= \prod_{i=1}^{n-1} P(A_i|pa(A_i)) \sum_{A_n} P(A_n|pa(A_n)) \\ &= \prod_{i=1}^{n-1} P(A_i|pa(A_i)), \end{aligned}$$

og per induktionsantagelsen opfylder denne de ønskede betingede uafhængigheder for BN_{n-1}. Hvis A_i og A_j er d-separeret givet en mængde $\mathcal{G}$ og ingen af de involverede variable er A_n er det tilstrækkeligt at arbejde med denne sandsynlighedsfordeling, hvilket giver $P(A_i|A_j, \mathcal{G}) = P(A_i|\mathcal{G})$ per induktionsantagelsen. De eneste resterende tilfælde er $A_n \in \mathcal{G}$ eller $i = n$.

Lad $A_n \in \mathcal{G}$. Da giver d-separationen, at alle stier mellem A_i og A_j i den moralske graf for problemstillingen går gennem enten A_n eller $\mathcal{G} \setminus \{A_n\}$. Hvis A_n udelades fra $\mathcal{G}$, vil A_n aldrig blive en del af den moralske graf, da den ingen børn har. Desuden er det muligt, at nogle af A_n's forfædre ikke bliver medtaget, eller nogle af forældrene ikke bliver gift. Derfor involverer denne ændring kun at slette kanter, så der kan ikke opstå nye stier i grafen. Da A_n ikke længere indgår, må alle resterende stier mellem A_i og A_j gå igennem $\mathcal{G} \setminus A_n$. Derfor er A_i og A_j d-separeret givet $\mathcal{G} \setminus A_n$, og tilfældet svarer til det forrige.

Betragt nu tilfældet $i = n$, og antag uden tab af generalitet, at $pa(A_n) \cap \mathcal{G} = \emptyset$. Hvis det sidste ikke er tilfældet, erstattes $pa(A_n)$ i det efterfølgende blot med $pa(A_n) \setminus \mathcal{G}$. For at vise, at $P(A_n|A_j, \mathcal{G}) = P(A_n|\mathcal{G})$ bemærkes det, at

$$\begin{aligned} P(A_n|A_j, \mathcal{G}) &= \sum_{pa(A_n)} P(A_n, pa(A_n)|A_j, \mathcal{G}) \\ &= \sum_{pa(A_n)} P(A_n|pa(A_n), A_j, \mathcal{G}) P(pa(A_n)|A_j, \mathcal{G}). \end{aligned} \tag{3.2.3}$$

Alle stier i den moralske graf mellem A_n og A_j går gennem $\mathcal{G}$. Da A_n ingen børn har, går alle disse stier også gennem A_n's forældre. Dermed vil enhver sti mellem A_n's forældre og A_j også gå gennem $\mathcal{G}$. Derfor er $pa(A_n)$ og A_j også d-separeret givet $\mathcal{G}$, og forrige tilfælde giver $P(pa(A_n)|A_j, \mathcal{G}) = P(pa(A_n)|\mathcal{G})$.

Lad $\mathcal{A} = \{A_1, \ldots, A_n\} \setminus \big(\{A_j\} \cup pa(A_n) \cup \mathcal{G}\big)$ og $\mathcal{A}_{-n} = \mathcal{A} \setminus \{A_n\}$. Da gælder det også, at

$$
\begin{aligned}
P(A_n|pa(A_n), A_j, \mathcal{G}) &= \frac{\sum\limits_{\mathcal{A}_{-n}} \prod\limits_{i=1}^{n} P(A_i|pa(A_i))}{\sum\limits_{\mathcal{A}} \prod\limits_{i=1}^{n} P(A_i|pa(A_i))} \\
&= \frac{P(A_n|pa(A_n)) \sum\limits_{\mathcal{A}_{-n}} \prod\limits_{i=1}^{n-1} P(A_i|pa(A_i))}{\sum\limits_{\mathcal{A}_{-n}} \prod\limits_{i=1}^{n-1} P(A_i|pa(A_i)) \sum\limits_{A_n} P(A_n|pa(A_n))} \\
&= P(A_n|pa(A_n)),
\end{aligned}
$$

hvilket viser, at A_n er betinget uafhængig af både A_j og $\mathcal{G}$ givet $pa(A_n)$. Derfor er $P(A_n|pa(A_n), A_j, \mathcal{G}) = P(A_n|pa(A_n), \mathcal{G})$, og (3.2.3) giver

$$
\begin{aligned}
P(A_n|A_j, \mathcal{G}) &= \sum_{pa(A_n)} P(A_n|pa(A_n), A_j, \mathcal{G}) P(pa(A_n)|A_j, \mathcal{G}) \\
&= \sum_{pa(A_n)} P(A_n|pa(A_n), \mathcal{G}) P(pa(A_n)|\mathcal{G}) \\
&= P(A_n|\mathcal{G}),
\end{aligned}
$$

hvilket viser det ønskede.

■

Slutteligt giver Proposition 3.2.2, at de betingede sandsynligheder tilknyttet det bayesianske netværk og de tilsvarende betingede fordelinger fundet via sandsynlighedsfordelingen i (3.2.2) stemmer overens.

Proposition 3.2.2
Benævn midlertidigt sandsynlighedsfordelingen i (3.2.2) med P^*. Da er $P^*(A_i|pa(A_i)) = P(A_i|pa(A_i))$, hvor sidste sandsynlighed stammer fra det bayesianske netværk.

Bevis. Ved at benytte d-separationsegenskaberne fra Proposition 3.1.2 giver Proposition 3.2.1, at $P^*(A_n|pa(A_n)) = P^*(A_n|A_{n-1}, \ldots, A_1)$. Dermed er

$$
\begin{aligned}
P^*(A_n|pa(A_n)) &= P^*(A_n|A_{n-1}, \ldots, A_1) \\
&= \frac{P^*(A_n, \ldots, A_1)}{P^*(A_{n-1}, \ldots, A_1)} \\
&= \frac{\prod\limits_{i=1}^{n} P(A_i|pa(A_i))}{\sum\limits_{A_n} \prod\limits_{i=1}^{n} P(A_i|pa(A_i))}
\end{aligned}
$$

$$= \frac{\prod_{i=1}^{n} P(A_i|pa(A_i))}{\left(\prod_{i=1}^{n-1} P(A_i|pa(A_i))\right) \sum_{A_n} P(A_n|pa(A_n))}$$
$$= P(A_n|pa(A_n)),$$

hvor nævneren i anden sidste lighed følger af, at A_n ingen efterkommere har. Der kan fortsættes iterativt for $A_{n-1}, \ldots, A_1$ via nøjagtig samme fremgangsmåde.

Hermed er det vist, at de betingede sandsynlighedsfordelinger stemmer overens.

■

Alle disse udledninger og egenskaber giver samlet set Sætning 3.2.3.

Sætning 3.2.3 (Kædereglen for bayesianske netværk)
Et bayesiansk netværk med variablene $A_1, \ldots, A_n$ specificerer entydigt en simultan sandsynlighedsfordeling givet ved

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P(A_i|pa(A_i)),$$

hvor de betingede sandsynligheder er dem, der er tilknyttet det bayesianske netværk. Per konvention sættes $P(A_i, \emptyset) = P(A_i)$ og $P(\emptyset) = 1$, hvilket også medfører, at $P(A_i|\emptyset) = P(A_i)$.

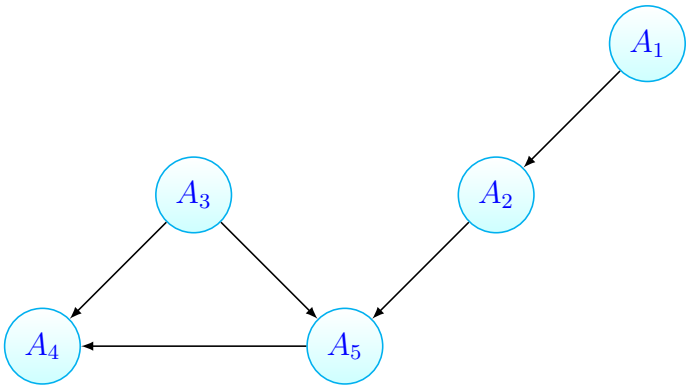
Bevis. Med entydigt menes der, at dette er den eneste sandsynlighedsfordeling, som opfylder, at d-separation repræsenterer betingede uafhængigheder, og at den er i overensstemmelse med de betingede fordelinger fra det bayesianske netværk.

Beregningerne i starten af afsnittet giver, at sandsynlighedsfordelingen kun kan have den angivne form, og Proposition A.0.1, 3.2.1 og 3.2.2 giver, at udtrykket opfylder de nødvendige kriterier.

■

3.3 Marginale fordelinger i bayesianske netværk

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 4.1).
Bayesianske netværk kan benyttes til at bestemme marginale fordelinger ud fra netværket både med og uden givet information. Det er altid muligt at bestemme marginale fordelinger ved at marginalisere den simultane sandsynlighedsfordeling. Betragt for eksempel det bayesianske netværk i Figur 3.3.1.



Figur 3.3.1: Et bayesiansk netværk med variablene A_1, A_2, A_3, A_4 og A_5 med tilhørende sandsynligheder $P(A_1)$, $P(A_2|A_1)$, $P(A_3)$, $P(A_4|A_3, A_5)$ og $P(A_5|A_2, A_3)$.

Ifølge Sætning 3.2.3 er

$$P(A_1, A_2, A_3, A_4, A_5) = P(A_1)P(A_2|A_1)P(A_3)P(A_4|A_3, A_5)P(A_5|A_2, A_3).$$

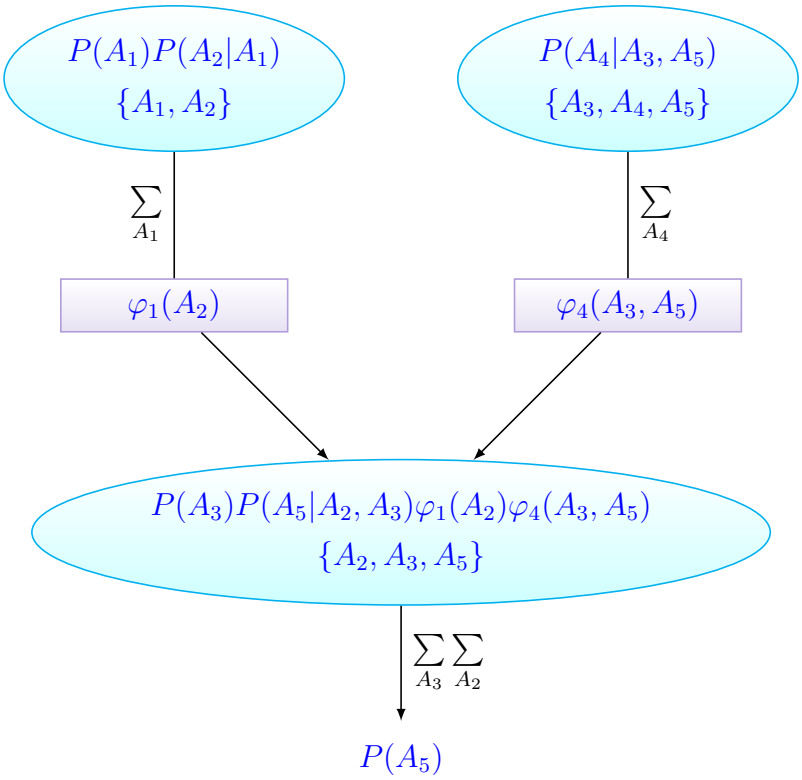
Sandsynligheden $P(A_5)$ kan nu findes ved beregningen

$$P(A_5) = \sum_{A_2}\sum_{A_3} P(A_3)P(A_5|A_2,A_3)\sum_{A_4} P(A_4|A_3,A_5)\sum_{A_1} P(A_1)P(A_2|A_1). \tag{3.3.1}$$

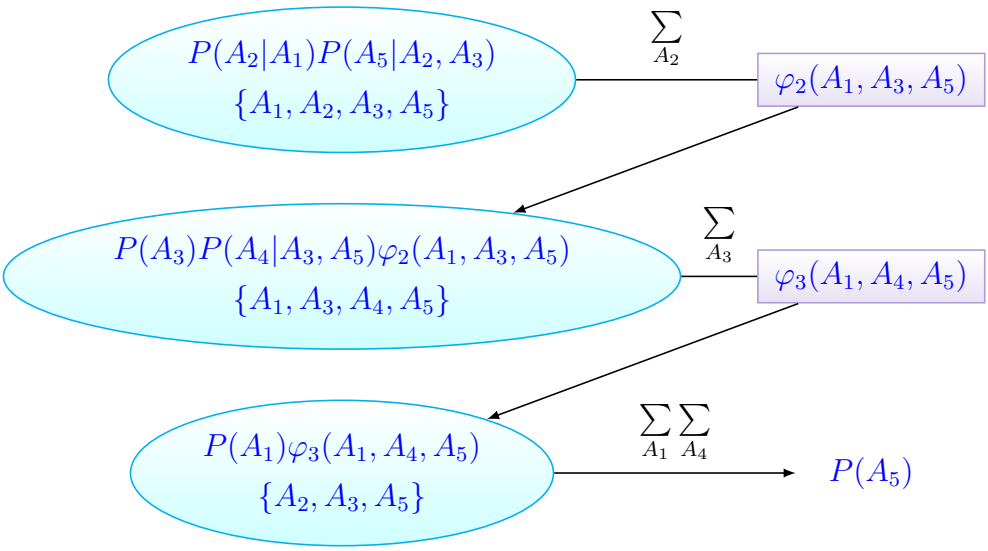
Bemærk dog, at ligheden også ville gælde for enhver anden summationsrækkefølge. Det ville for eksempel også være sandt, at

$$P(A_5) = \sum_{A_4}\sum_{A_1} P(A_1)\sum_{A_3} P(A_3)P(A_4|A_3,A_5)\sum_{A_2} P(A_2|A_1)P(A_5|A_2,A_3). \tag{3.3.2}$$

Der er dog en væsentlig forskel på de to udregninger, som bedst illustreres ved at betragte en grafisk repræsentation af beregningerne som i Figur 3.3.2.



(a) En grafisk illustration af beregningen i (3.3.1), hvor $\varphi_1(A_2) = \sum_{A_1} P(A_1)P(A_2|A_1)$ og $\varphi_4(A_3, A_5) = \sum_{A_4} P(A_4|A_3, A_5)$. Mængderne angiver de aktuelle variable i udtrykket.



(b) En grafisk illustration af beregningen i (3.3.2), hvor $\varphi_2(A_1, A_3, A_5) = \sum_{A_2} P(A_2|A_1)P(A_5|A_2, A_3)$ og $\varphi_3(A_1, A_4, A_5) = \sum_{A_3} P(A_3)P(A_4|A_3, A_5)\varphi_2(A_1, A_3, A_5)$. Mængderne angiver de aktuelle variable i udtrykket.

Figur 3.3.2: Grafer der illustrerer bestemmelsen af $P(A_5)$ ved to forskellige summationsrækkefølger.

Heraf ses det, at der i det første udtryk ikke marginaliseres over sandsynlighedsfordelinger med mere end tre involverede variable, mens der i den anden udregning også marginaliseres over sandsynlighedsfordelinger med fire variable. Generelt forsimpler det tingene at arbejde med så få variable som muligt, så den første udregning er derfor den mest effektive af de to.

Betragt nu situationen, hvor der er tilgængelig information, og det ønskes at bestemme, hvad dette betyder for sandsynlighedsfordelingen for en variabel. Det kunne for eksempel være, at sandsynligheden af interesse var $P(A_5|A_1 = a_1, A_2 = a_2)$. Til at bestemme denne sandsynlighed kræves $P(A_5, A_1 = a_1, A_2 = a_2)$. Denne bestemmes ved at indsætte de givne værdier for A_1 og A_2 i (3.3.1). Bemærk, at når variablenes tilstande er givet, marginaliseres der ikke over disse. Resultatet af dette bliver

$$\begin{aligned} &P(A_5, A_1 = a_1, A_2 = a_2) \\ &\quad = P(A_1 = a_1)P(A_2 = a_2|A_1 = a_1) \sum_{A_3} P(A_3)P(A_5|A_2 = a_2, A_3) \sum_{A_4} P(A_4|A_3, A_5). \end{aligned}$$

Den ønskede sandsynlighed bestemmes nu ved

$$P(A_5|A_1 = a_1, A_2 = a_2) = \frac{P(A_5, A_1 = a_1, A_2 = a_2)}{P(A_1 = a_1, A_2 = a_2)} = \frac{P(A_5, A_1 = a_1, A_2 = a_2)}{\sum_{A_5} P(A_5, A_1 = a_1, A_2 = a_2)}.$$

De næste afsnit handler om, hvordan det på en mere struktureret måde er muligt at bestemme en optimal summationsrækkefølge.

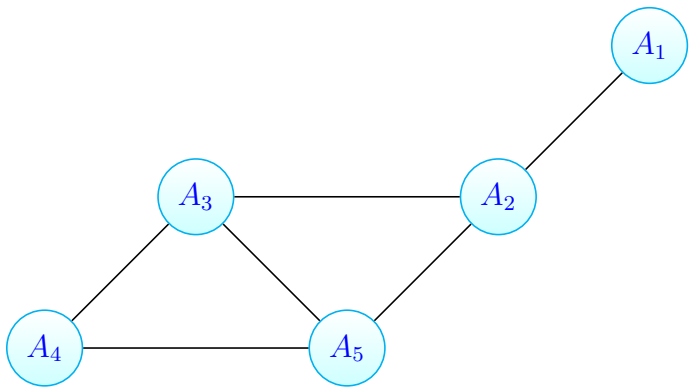
3.3.1 Domænegrafer

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 4.2).
I forrige afsnit fremgik det, at summationsrækkefølgen er væsentlig for beregningskompleksiteten, når marginale fordelinger bestemmes. For at få et indblik i hvilken summationsrækkefølge, der bør benyttes, betragtes det, som kaldes domæner. For at forstå domæner, kald da alle betingede sandsynligheder tilknyttet det bayesianske netværk potentialer. Et potentiale er generelt en funktion af variable, der til hver konfiguration knytter et ikke-negativt tal. Ved at gange potentialer sammen eller marginalisere over variable opstår nye potentialer. Et potentiales domæne er mængden af alle involverede variable. I Figur 3.3.2 angiver mængderne dermed de pågældende potentialers domæner. I forbindelse med at bestemme den bedste summationsrækkefølge, ønskes det at arbejde med domæner, der har så få variable som muligt.

For at få et overblik over hvilke domæner en bestemt summationsrækkefølge medfører, er det bekvemt at betragte en såkaldt domænegraf.

Definition 3.3.1 (Domænegraf)
Lad $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ være en mængde af potentialer over variablene $A_1, \ldots, A_n$. Da er domænegrafen for Φ en ikke-orienteret graf, hvis knuder repræsenterer variablene. Der er en kant mellem to knuder, såfremt de to variable indgår i et domæne sammen. For et bayesiansk netværk svarer domænegrafen til at moralisere hele netværket.

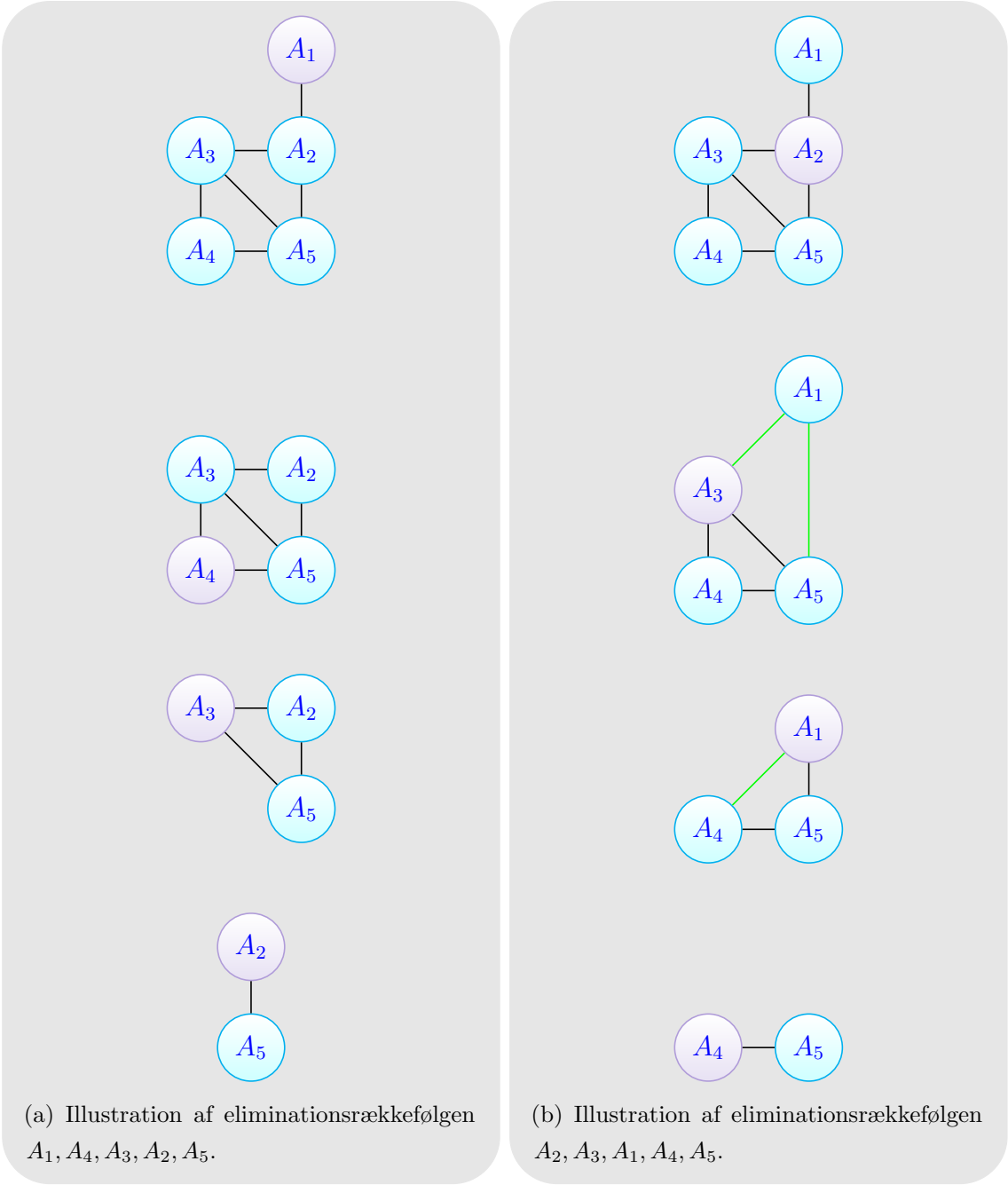
Bemærk, at ved at skifte fokus til domænegrafer benyttes en ikke-orienteret graf i stedet for en orienteret til at beskrive netværket. Det bemærkes dog, at selvom et bayesiansk netværk har en entydig domænegraf, kan den samme graf godt være domænegraf for flere forskellige bayesianske netværk. I dette speciale arbejdes der kun med sammenhængende domænegrafer.[1] Betragt for eksempel det bayesianske netværk i Figur 3.3.1. Den tilhørende domænegraf for dette ses i Figur 3.3.3.



Figur 3.3.3: Domænegraf for det bayesianske netværk i Figur 3.3.1.

Domænegrafen kan bruges til at få et overblik over, hvilke konsekvenser det har at marginalisere over en variabel. Hvis det ønskes at marginalisere over A_i i den simultane sandsynlighedsfordeling, er det nødvendigt at bestemme et potentiale ϕ, der udgør produktet af alle betingede sandsynligheder, som indeholder A_i. Domænet for ϕ er A_i og $na(A_i)$ i domænegrafen. Efter marginaliseringen opnås et potentiale med $na(A_i)$ som domæne. Processen kan illustreres i domænegrafen ved at fjerne knuden A_i og forbinde alle dennes naboer om nødvendigt. Hvis det ønskes at foretage flere marginaliseringer, kan processen gentages herfra. Rækkefølgen, som variablene fjernes i, kaldes en eliminationsrækkefølge. For beregningerne i (3.3.1) og (3.3.2) illustreres denne proces i henholdsvis Figur 3.3.4a og 3.3.4b.

[1] Hvilket reelt set svarer til, at det bayesianske netværk er sammenhængende.



(a) Illustration af eliminationsrækkefølgen A_1, A_4, A_3, A_2, A_5.

(b) Illustration af eliminationsrækkefølgen A_2, A_3, A_1, A_4, A_5.

Figur 3.3.4: Domænegrafer til illustration af forskellige eliminationsrækkefølger for det bayesianske netværk i Figur 3.3.1. Den lilla knude indikerer hvilken knude, der elimineres næst. Nye kanter i grafen markeres med grøn.

Bemærk, at i Figur 3.3.4a introduceres aldrig nye kanter, hvorimod eliminationsrækkefølgen i Figur 3.3.4b giver anledning til flere nye kanter undervejs. Dette betyder, at undervejs i processen vil nogle variable komme til at indgå i et domæne med variable, som de ikke tidligere var i domæne med. Dette giver anledning til, at summationsrækkefølgen i (3.3.2) omfatter større domæner end den i (3.3.1). Hvis der på intet tidspunkt introduceres nye kanter, kaldes det en perfekt eliminationsrækkefølge. For det bayesianske netværk i Figur 3.3.1

med domænegrafen i Figur 3.3.3 er A_1, A_4, A_3, A_2, A_5 altså en perfekt eliminationsrækkefølge resulterende i A_5. Bemærk, at der kan eksistere flere perfekte eliminationsrækkefølger i en domænegraf.

3.4 Triangulerede grafer

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 4.3).
Hvorvidt en graf har en perfekt eliminationsrækkefølge er tæt knyttet til begrebet en trianguleret graf. Triangulerede grafer kommer også til at spille en stor rolle, når ikke-orienterede grafiske modeller beskrives senere. I Definition 3.4.1 og 3.4.2 gives to måder, hvorpå en trianguleret graf kan defineres.

Definition 3.4.1 (Trianguleret graf)
Hvis der i en ikke-orienteret graf findes en perfekt eliminationsrækkefølge, kaldes grafen triangulær.

Definition 3.4.2 (Trianguleret graf)
En graf er triangulær, hvis der for enhver kreds med mindst fire knuder eksisterer en kortere kreds i denne. Altså skal der for enhver kreds med mere end tre knuder være en mulig genvej.

Definition 3.4.1 og 3.4.2 er ækvivalente jævnfør Rose (1970). Det vil typisk være Definition 3.4.1, der benyttes i dette afsnit, men afhængig af situationen vil det i nogle tilfælde være mere hensigtsmæssigt at benytte Definition 3.4.2.

Proposition 3.4.5 giver, at hvis der findes bare én perfekt eliminationsrækkefølge i grafen, findes der en perfekt eliminationsrækkefølge, som ender på en hvilken som helst variabel. For at bevise Proposition 3.4.5 benyttes Lemma 3.4.3 og 3.4.4.

Lemma 3.4.3
Hvis en simpel knude elimineres fra en trianguleret graf, resulterer dette i en ny trianguleret graf.

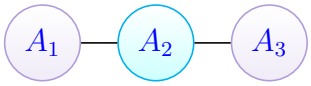
Bevis. Lad T være en trianguleret graf med perfekt eliminationsrækkefølge $A_1, \ldots, A_n$, og lad A_j være en simpel knude. Hvis det kan vises, at $A_j, A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_n$ også er en perfekt eliminationsrækkefølge, vil $A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_n$ være en perfekt eliminationsrækkefølge i grafen uden A_j, som dermed er triangulær, og resultatet er vist.

At $A_j, A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_n$ er en perfekt eliminationsrækkefølge, følger af det faktum, at A_j er simpel. Derfor introducerer det ingen nye kanter at eliminere A_j først, da nabomængden allerede er komplet. Det giver derfor heller ingen andre knuder nye naboer, så resten af eliminationsrækkefølgen kan fortsættes uden komplikationer.

■

Lemma 3.4.4
Er der mindst to knuder i en trianguleret graf, har grafen mindst to simple knuder. Hvis grafen ikke er komplet, kan knuderne vælges, så de ikke er naboer.

Bevis. Hvis grafen er komplet, er alle knuder simple knuder, og resultatet er trivielt opfyldt. Antag derfor, at grafen ikke er komplet. Resultatet bevises ved induktion. En ikke-komplet sammenhængende graf må have mindst tre knuder, og har i så fald formen i Figur 3.4.1. Bemærk, at grafen er trianguleret.



Figur 3.4.1: Trianguleret graf med 3 knuder.

Denne graf har to simple knuder, som ikke er naboer.

Antag for induktion, at enhver trianguleret graf med $n-1$ knuder, $n \geq 4$, har mindst to simple knuder, som ikke er naboer. Betragt en ikke-komplet, trianguleret graf T_n med n knuder med en perfekt eliminationsrækkefølge startende med A. Da eliminationsrækkefølgen er perfekt, må A være simpel, og det mangles derfor bare at bestemme én simpel knude mere.

Lad T_{n-1} være grafen T_n uden knuden A. Bemærk, at T_{n-1} er triangulær per Lemma 3.4.3. Da T_n ikke er komplet, og A er en simpel knude, må der eksistere mindst én knude i T_n, som ikke er nabo til A, da grafen ellers ville være komplet. Hvis T_{n-1} er komplet, vil enhver sådan knude være simpel i T_n, og resultatet er opfyldt.

Hvis T_{n-1} ikke er komplet, giver induktionsantagelsen, at der findes mindst to simple knuder i denne, som ikke er naboer. Hvis bare en af disse ikke er nabo til A, er denne også simpel i T_n. Antag derfor for modstrid, at disse to begge er naboer til A. Da A er simpel, betyder dette, at alle dens naboer er forbundet, og de to knuder er dermed naboer, hvilket er en modstrid. Mindst en af dem må altså også være simpel i T_n, og resultatet er vist.

■

Proposition 3.4.5
For enhver variabel A i en trianguleret graf eksisterer en perfekt eliminationsrækkefølge resulterende i A.

Bevis. Start eliminationsrækkefølgen ved at eliminere en simpel knude, som ikke er A. Lemma 3.4.4 giver, at der eksisterer mindst to simple knuder i grafen, så der må findes én, som ikke er A. Dette introducerer ingen nye kanter, da knuden er simpel. Lemma 3.4.3 giver, at den resulterende graf også er triangulær, og Lemma 3.4.4 giver derfor igen, at der kan elimineres en simpel knude, som ikke er A. Processen kan fortsættes iterativt, så længe der er mindst to knuder i den resulterende graf, og når dette ikke længere er tilfældet, indeholder grafen kun A, og en perfekt eliminationsrækkefølge resulterende i A er bestemt.

■

Sætning 3.4.6 giver en nødvendig og tilstrækkelig betingelse for, at en graf er triangulær.

Sætning 3.4.6
En graf er triangulær, hvis og kun hvis alle dens knuder kan elimineres ved successivt at fjerne simple knuder i den aktuelle graf.

Bevis. Når en simpel knude elimineres, introduceres ingen nye kanter, så hvis alle knuder kan elimineres ved successivt at eliminere simple knuder, giver dette en perfekt eliminationsrækkefølge, og grafen er triangulær.

Hvis grafen er triangulær, må den første knude i en perfekt eliminationsrækkefølge være simpel. Lemma 3.4.3 giver, at den resulterende graf også er triangulær, og den første knude i en perfekt eliminationsrækkefølge for denne må være simpel i denne graf. Ved at fortsætte iterativt opnås det, at alle knuder kan elimineres ved successivt at eliminere simple knuder i de resulterende grafer.

■

Når en variabel A elimineres fra en trianguleret graf, benævn da $fa(A)$ som et domæne. For eliminationsrækkefølgen A_1, A_4, A_3, A_2, A_5 for domænegrafen i Figur 3.3.3 opstår domænerne $D_1 = \{A_1, A_2\}, D_2 = \{A_3, A_4, A_5\}, D_3 = \{A_2, A_3, A_5\}$ og $D_4 = \{A_2, A_5\}$, hvilket ses af Figur 3.3.4a. Hvis grafen er en domænegraf, bliver disse mængder domænerne i de potentialer, der marginaliseres over. Bemærk, at $D_4 \subseteq D_3$. Hvis der generelt findes domæner D_i og D_j, hvor $D_i \subseteq D_j$, udelades D_i. De resterende mængder udgør da eliminationsrækkefølgens domænemængde. Domænemængden for eliminationsrækkefølgen A_1, A_4, A_3, A_2, A_5 er således $\{D_1, D_2, D_3\}$.

Proposition 3.4.7 giver, at alle perfekte eliminationsrækkefølger resulterer i samme domænemængde nemlig mængden af kliker i grafen. For en trianguleret domænegraf betyder det specielt, at enhver perfekt eliminationsrækkefølge giver lige effektive marginaliseringer.

Proposition 3.4.7
Lad $A_1, \ldots, A_n$ være en perfekt eliminationsrækkefølge med domænemængde $\mathcal{D}$. Lad ydermere $\mathcal{K}$ betegne mængden af kliker i den triangulerede graf. Da er $\mathcal{D} = \mathcal{K}$.

Bevis. Først vises det, at $\mathcal{K} \subseteq \mathcal{D}$. Lad derfor $K \in \mathcal{K}$ og A_i være den variabel i K, som først elimineres i en perfekt eliminationsrækkefølge. Lad $D \in \mathcal{D}$ være det domæne, som involverer eliminationen af A_i. Derfor må D indeholde A_i og alle dennes naboer, som endnu ikke er elimineret.

Da eliminationsrækkefølgen er perfekt, er punktmængden i et domæne komplet. Grundet K er en klike, er den komplet, hvilket medfører, at alle knuder i K er nabo til A_i. Ingen af disse er elimineret, når A_i elimineres, da A_i var antaget at være den første eliminerede variabel i K. Dette giver, at $K \subseteq D$.

Lad nu $A_j \in D$. Fordi D er komplet, er A_j nabo til alle knuder i D og dermed også til alle knuder i K som jo er en delmængde af D. Dette betyder, at $\{A_j\} \cup K$ danner en komplet mængde af knuder. Dette ville være i modstrid med, at K er en klike, hvis ikke $A_j \in K$. Dermed er $D \subseteq K$. Dette betyder, at $K = D \in \mathcal{D}$, og det kan konkluderes, at $\mathcal{K} \subseteq \mathcal{D}$.

Herefter vises det, at $\mathcal{D} \subseteq \mathcal{K}$. Lad derfor $D \in \mathcal{D}$, og det erindres, at D er komplet. Antag nu, at D ikke er en klike. Da vides det, at der eksisterer en klike $K \in \mathcal{K}$ således, at $D \subseteq K \in \mathcal{D}$, hvor det sidste følger af, at $\mathcal{K} \subseteq \mathcal{D}$. Dette betyder, at $D \notin \mathcal{D}$, da mængderne i denne per definition ikke kan være delmængder af hinanden, men dette er en modstrid grundet antagelsen, $D \in \mathcal{D}$. Derfor må D være en klike, hvilket betyder, at $D \in \mathcal{K}$, som giver, at $\mathcal{D} \subseteq \mathcal{K}$.

Hermed er det vist, at $\mathcal{D} = \mathcal{K}$.

■

For at bestemme mængden af kliker i en trianguleret graf er det altså tilstrækkeligt at bestemme domænemængden for en perfekt eliminationsrækkefølge. Sætning 3.4.6 giver, at denne domænemængde kan bestemmes iterativt ud fra simple knuder ved følgende fremgangsmåde:

- Fjern en simpel knude A fra grafen, og gem $fa(A)$ midlertidigt i domænemængden.
- Så længe grafen stadig har knuder, gentages forrige. Sætning 3.4.6 giver, at proceduren altid kan gennemføres.

- Domænemængden færdiggøres ved at fjerne alle mængder, som er delmængder af andre mængder. Dette giver hermed også mængden af kliker i grafen per Proposition 3.4.7.

Bemærk at denne procedure giver en effektiv måde til at bestemme grafens kliker, hvilket ikke generelt er muligt for en vilkårlig graf.

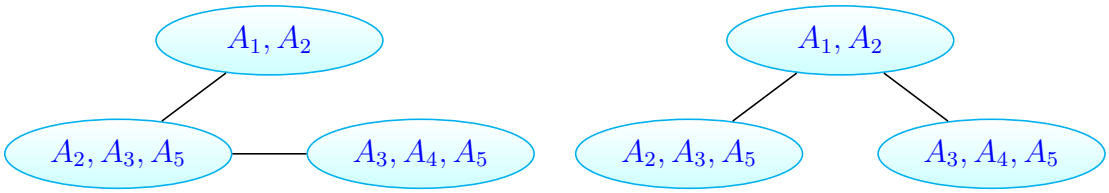
Når mængden af kliker er bestemt, er det muligt at danne et såkaldt tilslutningstræ og herfra et såkaldt junktionstræ, hvilket næste afsnit omhandler.

3.4.1 Tilslutnings- og junktionstræer

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 4.3 og 4.4).
Tilslutnings- og junktionstræer er to tæt knyttede begreber, der gør det muligt på en struktureret måde at bestemme marginale fordelinger i bayesianske netværk eventuelt med givet information. Tilslutningstræer er også et begreb, der senere kommer til at spille en meget stor rolle i forbindelse med at definere en sandsynlighedsfordeling ud fra et Markovnetværk. Et tilslutningstræ knytter sig til en grafs kliker og er defineret i Definition 3.4.8.

Definition 3.4.8 (Tilslutningstræ)
Betragt mængden af kliker $\mathcal{K}$ i en ikke-orienteret graf, og lad klikerne være knuder i et træ G. Betragt to knuder K_i og K_j i G. Træet G kaldes for et tilslutningstræ, hvis alle knuder på en den simple sti mellem K_i og K_j i G indeholder $K_i \cap K_j$. Dette kaldes, at træet overholder RIP-egenskaben, hvor RIP er en forkortelse af *Running Intersection Property*. Til en kant i G mellem to knuder K_i og K_j kan mængden $S = K_i \cap K_j$ tilknyttes. En sådan mængde S kaldes en separator.

Eksempel 3.4.9
I domænegrafen i Figur 3.3.3 er A_1, A_4, A_3, A_2, A_5 en perfekt eliminationsrækkefølge, hvilket gav anledning til domænemængden $\{\{A_1, A_2\}, \{A_3, A_4, A_5\}, \{A_2, A_3, A_5\}\}$, som også er grafens kliker per Proposition 3.4.7. I Figur 3.4.2 ses to måder at organisere disse kliker i et træ, hvoraf kun det ene er et tilslutningstræ.



(a) Et tilslutningstræ. (b) Ikke et tilslutningstræ.

Figur 3.4.2: Klikerne $\{A_1, A_2\}$,$\{A_3, A_4, A_5\}$,$\{A_2, A_3, A_5\}$ fra grafen i Figur 3.3.3 organiseret i to træer.

Algoritme 3.4.10 giver en måde, hvorpå klikerne for en trianguleret graf kan organiseres i et tilslutningstræ. Algoritmen tager udgangspunkt i Sætning 3.4.6, som giver at alle knuder kan elimineres ved successivt at fjerne simple knuder.

Algoritme 3.4.10 `Tilslutningstræ for triangulerede grafer`

`Input:` Trianguleret graf med n knuder.

1. Sæt $i = 0$
2. Vælg en simpel knude A_1.
3. Eliminer fra grafen variable $A_1, \ldots, A_k$, som kun har naboer i $fa(A_1)$.
   (Bemærk, at alle disse knuder er simple.)
4. Sæt $i = i + k$. Sæt herefter $K_i = fa(A_1)$ og $S_i = K_i \setminus \{A_1, \ldots, A_k\}$.
   (Bemærk, at K_i er en klike, og at S_i per konstruktion kommer til at indeholde de variable fra K_i, som også indgår i andre kliker.)
5. Så længe $i < n$ gentages punkt 2. – 5.
6. Forbind hver klike K_p til en klike K_q, som opfylder, at $q > p$, og $S_p \subset K_q$.
   (S_p bliver separatoren for kanten mellem de to kliker.)

`Output:` Et tilslutningstræ.

I punkt 7. er det muligt at finde en passende klike K_q, fordi S_p som en delmængde af en klike er komplet, og på et eller andet tidspunkt må en af variablene i S_p blive elimineret. Dette involverer en klike, der indeholder denne variabel og derfor også alle dens ikke eliminerede naboer, som specielt omfatter de andre knuder i S_p. Denne klike må have højere indeks, da denne eliminering ikke er foretaget endnu. K_q kaldes K_p's forælder. Proposition 3.4.11 giver, at algoritmen faktisk konstruerer et tilslutningstræ.

Proposition 3.4.11
Algoritme 3.4.10 organiserer klikerne for en trianguleret graf i et tilslutningstræ.

Bevis. Først bevises det, at den konstruerede graf er et træ. Punkt 7. i Algoritme 3.4.10 kan betragtes som gentagne gange at forbinde forskellige træer via kun én kant.[2] Dette kan aldrig introducere en kreds, så den konstruerede graf er et træ.

Det mangles nu kun at vise, at træet opfylder RIP-egenskaben. Betragt til dette formål en simpel sti mellem to kliker K_i og K_j, hvor $i < j$, og lad $A \in K_i \cap K_j$. Da $A \in K_j$ er denne ikke elimineret i forbindelse med K_i, hvilket giver, at $A \in S_i$. Per konstruktion af træet gælder det derfor også, at $A \in K_l$, hvor K_l er K_i's forælder. Hvis $l = j$ er argumentet fuldbragt. Hvis ikke gentages argumentet for den klike af K_l og K_j, som har lavest indeks, og dette giver, at alle kliker på stien indeholder A. Dermed er det konstruerede træ et tilslutningstræ.

■

Sætning 3.4.12 giver, at tilslutningstræer kun kan konstrueres for triangulerede grafer.

Sætning 3.4.12
En graf er triangulær, hvis og kun hvis klikerne i grafen kan organiseres i et tilslutningstræ.

Bevis. Antag først, at klikerne kan organiseres i et tilslutningstræ, og lad B være et blad i træet med naboen K. I et træ er der altid præcis én simpel sti mellem to knuder. Hvis en variabel fra B også ligger i en anden klike, vil stien mellem disse to kliker gå igennem B's eneste nabo K. Da der er tale om et tilslutningstræ, ligger variablen derfor også i kliken K. Hvis alle variable i B også ligger i andre kliker, vil $B \subseteq K$, hvorved B ikke kan være en klike. Derfor må der i B findes variable, som ikke ligger i andre kliker. Vælg en sådan variabel A. Grundet definitionen af en klike vil to naboknuder altid indgå i mindst én klike sammen. Da A kun findes i B, må alle A's naboer også ligge i B. Da dette er en klike, er denne nabomængde komplet, så A er simpel og kan elimineres uden problemer. Der kan fortsættes med at eliminere eventuelle andre variable fra B, som ikke indgår i andre kliker. Herefter er alle resterende variable i B også i K, og B er ikke længere en klike i den resulterende graf. Den resulterende graf har dog stadig alle øvrige kliker, og ved at fjerne B fra det oprindelige tilslutningstræ opnås et tilslutningstræ for den nye graf. Processen fortsættes iterativt, og dermed opnås en perfekt eliminationsrækkkefølge, hvilket giver, at grafen er triangulær.

Lad nu T være en trianguleret graf. Da giver Proposition 3.4.11 at klikerne kan organisere i et tilslutningstræ med Algoritme 3.4.10, hvilket beviser resultatet.

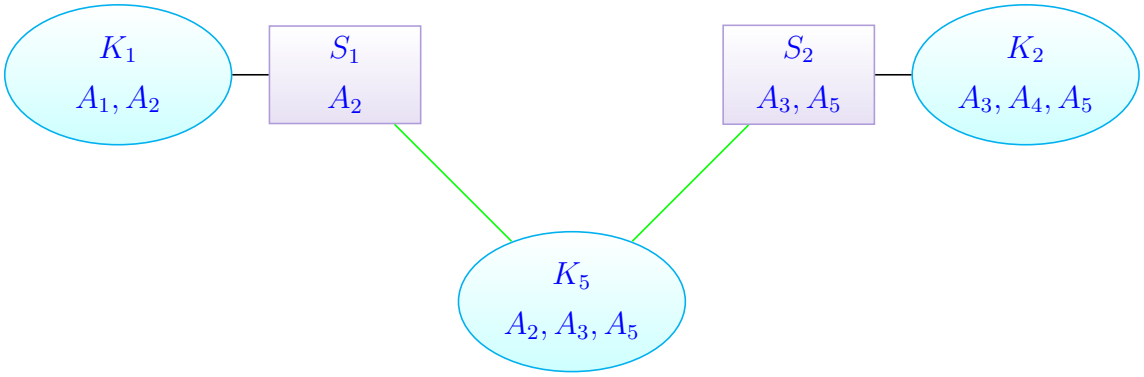
[2] Til at starte med er alle knuder et træ med kun én knude.

■

Bemærk, at beviset for Sætning 3.4.12 giver en metode til at bestemme en perfekt eliminationsrækkefølge ud fra et tilslutningstræ. Dette og Algoritme 3.4.10 illustreres i Eksempel 3.4.13.

Eksempel 3.4.13

Betragt domænegrafen i Figur 3.3.3. Ved at bruge Algoritme 3.4.10 fås tilslutningstræet i Figur 3.4.3.



Figur 3.4.3: Brug af Algoritme 3.4.10 på domænegrafen i Figur 3.3.3. Kliker og separatorer er bestemt i punkt 6., og de grønne kanter sættes i punkt 7.

Grafen i Figur 3.4.3 kan nu bruges til at bestemme en perfekt eliminationsrækkefølge. Først vælges et tilfældigt blad, som kunne være K_2. Heri er den eneste variabel, der ikke indgår i naboseparatoren, A_4, som derfor elimineres først, og K_2 udelades fra grafen. Herefter vælges K_5, som er et blad i den nye graf, hvorefter A_3 og A_5 elimineres herfra. Når K_5 udelades fra grafen, er der kun et enkelt blad tilbage, og de resterende variable A_1 og A_2 elimineres. Dette giver den perfekte eliminationsrækkkefølge A_4, A_3, A_5, A_1, A_2.

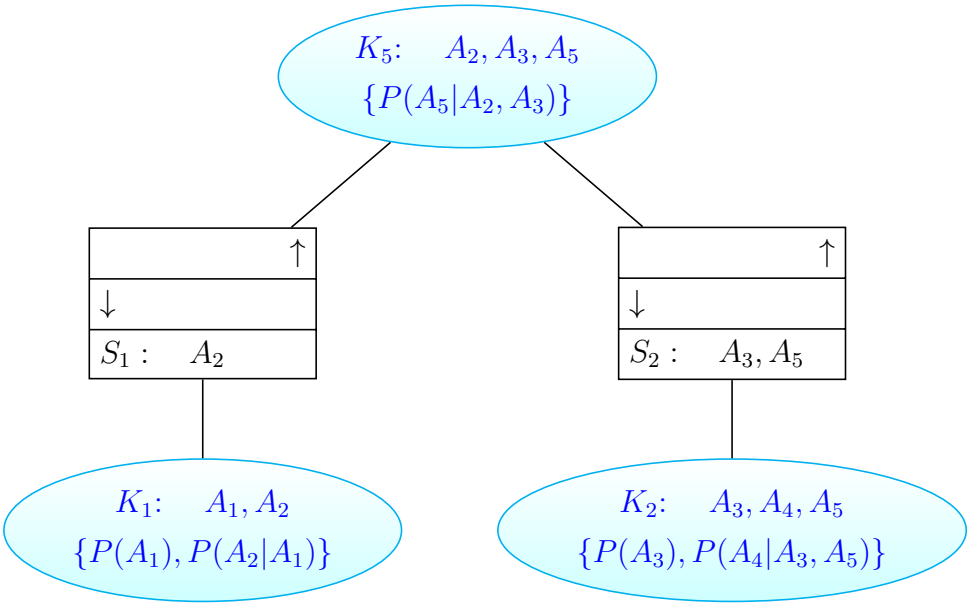
For effektivt at bestemme marginale fordelinger i bayesianske netværk, udnyttes det, at det er forholdsvis let at bestemme en perfekt eliminationsrækkefølge ud fra et tilslutningstræ. Tilslutningstræet kombineres derfor med al anden nødvendig information i et såkaldt junktionstræ.

Definition 3.4.14 (Junktionstræ)
Betragt et bayesiansk netværk med en trianguleret domænegraf og tilhørende tilslutningstræ G. Modificer G på følgende måde:

- Til hver knude i grafen tilføjes mængden af de potentialer i det bayesianske netværk, hvis domæne er indeholdt i den pågældende klike. Bemærk dog, at hvis et potentiales domæne er indeholdt i flere kliker, vælges kun én.
- Til hver separator tilknyttes to postkasser med det formål at opbevare information. En til information sendt hver vej i grafen.

Da kaldes den resulterende graf et junktionstræ. For hver knude kaldes postkasser indeholdende information sendt i knudens retning dens indbakker og postkasser indeholdende information sendt fra knuden til andre knuder for dens udbakker.

For det bayesianske netværk i Figur 3.3.1 er et junktionstræ vist i Figur 3.4.4.



Figur 3.4.4: Junktionstræ for det bayesianske netværk i Figur 3.3.1. Pilens retning indikerer, hvilken vej beskeden i denne postkasse sendes.

I det efterfølgende vil det ofte blive relevant at tage produktet af potentialerne i en mængde Φ for at opnå et nyt potentiale. For at forsimple notationen vil dette blive skrevet $\prod \Phi$.

De tommer postkasser i junktionstræet fyldes med passende mængder af potentialer. Idéen er, at en knude kan kombinere indkommende potentialer med sine egne potentialer, foretage en passende marginalisering og sende den resulterende mængde af potentialer til en anden knude. Betragt for eksempel en klike K_i i et junktionstræ med mængden af potentialer

Φ_i med naboseparatorer $S_1, \ldots, S_n$. Antag, at det ønskes at sende en besked fra K_i til S_1. Benævn de indkommende beskeder fra $S_2, \ldots, S_n$ som $\Psi_2, \ldots, \Psi_n$. Den aktuelle mængde af potentialer er altså, $\Phi = \Phi_i \cup \Psi_2 \cup \cdots \cup \Psi_n$. Inddel denne mængde i to dele: En mængde Φ_A med potentialer, der involverer variablene $K_i \backslash S_1$, og mængden af øvrige potentialer Φ_B. Lad $\varphi_A = \sum_{K_i \backslash S_1} \prod \Phi_A$. Da sendes mængden af potentialer $\{\varphi_A\} \cup \Phi_B$ til K_i's udbakke i S_1. Bemærk, at potentialer ikke ganges sammen, før det er strengt nødvendigt. Dette giver, at der hele tiden marginaliseres over potentialer med så små domæner som muligt.

Før en knude kan sende en besked, skal den pågældende retning være det, der kaldes aktiv, hvilket defineres i Definition 3.4.15.

Definition 3.4.15 (Aktiv retning)
Lad K være en klike i et junktionstræ med en naboseparator S. Da er retningen fra K mod S aktiv, såfremt alle K's indbakker i øvrige naboseparatorer allerede har modtaget beskeder.

Proposition 3.4.16 giver, at det altid er muligt at fylde alle postkasser i junktionstræet.

Proposition 3.4.16
Det er i et junktionstræ altid muligt at fylde alle postkasser kun ved iterativt at sende beskeder i aktive retninger.

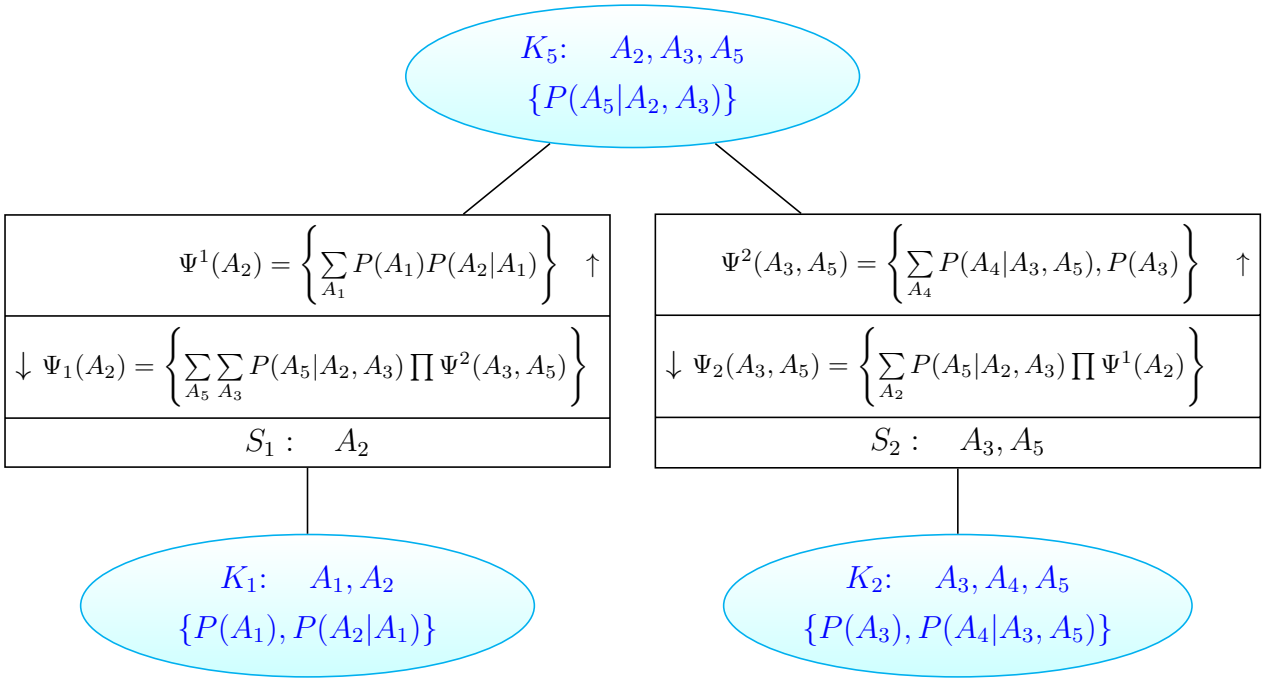
Bevis. Vælg først en af knuderne i træet som rod R, og betragt bladene i træet. Hvert blad har kun én naboseparator, og retningen mod denne er derfor allerede aktiv. Start derfor med at lade bladene sende beskeder til deres naboseparator. Betragt herefter træet uden disse blade. Hvert blad i det nye træ har nu kun én naboseparator, som endnu ikke har modtaget en besked. Alle øvrige naboseparatorer modtog en besked i retning mod roden i forrige skridt. Retningen mod disse nye blades naboseparator er derfor nu aktiv. Send derfor beskeder i denne retning, og der fortsættes iterativt. Dette fylder alle postkasser i retning mod roden, og proceduren kaldes `CollectEvidence`(R).

Betragt nu roden. Alle dennes indbakker er allerede fyldt, så retningerne væk fra denne er aktive. Send derfor beskeder i disse retninger. Lad N være en naboknude til roden. Denne har netop modtaget en besked i sin indbakke fra roden. Ud over roden har denne knude kun naboer længere nede i træet. N's indbakker fra disse naboer blev fyldt via proceduren `CollectEvidence`(R). Derfor er retningen fra N til naboerne længere nede i træet nu aktiv. Send derfor beskeder i denne retning. Der kan fortsættes iterativt, indtil bladene nås, og dette fylder alle postkasser i retning væk fra roden. Denne procedure kaldes `DistributeEvidence`(R).

Alle postkasser i begge retninger er derfor fyldte, og sætningen er hermed bevist.

■

Når alle postkasser i junktionstræet er fyldte, kaldes det et fuldt junktionstræ, og proceduren, der benyttes til at fylde dette, kaldes fuld udbredelse. Fuld udbredelse for junktionstræet i Figur 3.4.4 ses i Figur 3.4.5.



Figur 3.4.5: Illustration af fuld udbredelse for junktionstræet i Figur 3.4.4. Beskedernes indeks følger separatorens indeks, og indeksets placering følger pilens retning.

I Figur 3.4.5 indgår udtrykket $\sum_{A_4} P(A_4|A_3, A_5)$, hvilket giver 1, da der er tale om en sandsynlighedsfordeling. Hvis den slags udtryk indgår, sættes de derfor lig 1 uden nogen udregninger.

Bemærk ligheden mellem proceduren `CollectEvidence`(R) og det at bestemme en perfekt eliminationsrækkefølge ud fra et tilslutningstræ resulterende i variablene i R. Det første skridt i denne procedure er at vælge et blad og eliminere alle variable, som ikke indgår i naboseparatoren. I proceduren `CollectEvidence`(R) foretages der yderligere den handling, at produktet af bladets potentialer marginaliseres over de eliminerede variable. Dette produkt må nødvendigvis omfatte alle potentialer, der har de eliminerede variable i domænet. Ellers ville variablene indgå i separatoren, og dermed ikke blive elimineret. Proceduren gentages nu ved at slette det pågældende blad og vælge et nyt blad. Dette giver en fremgangmåde til at bruge informationen fra et fuldt junktionstræ til at bestemme marginale fordelinger, hvilket Proposition 3.4.17 omhandler.

Proposition 3.4.17
Betragt et fuldt junktionstræ for et bayesiansk netværk. Lad K være en klike med mængden af potentialer Φ, hvis indbakker har modtaget beskederne $\Psi_1, \ldots, \Psi_n$. Da er

$$P(K) = \prod \Phi \prod \Psi_1 \cdots \prod \Psi_n.$$

Lad ydermere S være en separator, hvis postkasser har modtaget beskederne Ψ_S og Ψ^S. Da er

$$P(S) = \prod \Psi_S \prod \Psi^S.$$

Bevis. De indkommende beskeder til K betragtes som resultatet af en `CollectEvidence`(K) procedure. Som argumenteret for før propositionen svarer de indkommende beskeder til de marginaliseringer, det involverer at bestemme $P(K)$ ved brug af en perfekt eliminationsrækkefølge. Derfor er

$$P(K) = \prod \Phi \prod \Psi_1 \cdots \prod \Psi_n.$$

Antag nu, at S er naboseparator til K, og at S har modtaget beskeden $\Psi_n = \Psi_S$ til K. Beskeden Ψ^S, som K sender til S, opfylder per definition, at

$$\prod \Psi^S = \sum_{K \setminus S} \prod \Phi \prod \Psi_1 \cdots \prod \Psi_{n-1}.$$

Da S er naboseparator til K, er $S \subseteq K$, så

$$P(S) = \sum_{K \setminus S} P(K) = \sum_{K \setminus S} \prod \Phi \prod \Psi_1 \cdots \prod \Psi_S.$$

Da Ψ_S er en besked i en af S's postkasser, afhænger denne kun af variablene i S, så

$$P(S) = \prod \Psi_S \sum_{K \setminus S} \prod \Phi \prod \Psi_1 \cdots \Psi_{n-1} = \prod \Psi_S \prod \Psi^S.$$

■

Hvis nogle af variablenes tilstande er givet, noteres dette i de relevante potentialer i det tomme junktionstræ. Der foretages herefter fuld udbredelse, hvorefter der regnes som før. Bemærk som tidligere beskrevet, at når variables tilstand er kendt på forhånd, marginaliseres der ikke over disse.

Junktionstræer giver altså mulighed for at bestemme marginale fordelinger effektivt, både fordi der altid benyttes en perfekt eliminationsrækkefølge, og fordi beregningerne kan genbruges til at bestemme flere marginale fordelinger.

Hvis det skulle ske, at domænegrafen for et bayesiansk netværk ikke er triangulær, findes der algoritmer til at triangulere denne, men dette speciale afgrænser sig fra at behandle dette emne.

Proposition 3.4.17 giver mulighed for at betragte sandsynlighedsfordelingen specificeret af et bayesiansk netværk på en alternativ form. Lad netværket være over variablene $A_1, \ldots, A_n$. Lad $\mathcal{K}$ være mængden af indeks på klikerne og Φ_i mængden af potentialer tilhørende kliken K_i. Lad desuden $\mathcal{S}$ være mængden af indeks på separatorerne, og lad Ψ_j og Ψ^j være beskederne i postkassen til separator S_j. Definer potentialet $\varphi_i = \prod \Phi_i$ samt potentialerne $\psi_j = \prod \Psi_j$ og $\psi^j = \prod \Psi^j$. Da er

$$P(A_1, \ldots, A_n) = \prod_{i \in \mathcal{K}} \varphi_i = \frac{\prod\limits_{i \in \mathcal{K}} \varphi_i \prod\limits_{j \in \mathcal{S}} \psi_j \psi^j}{\prod\limits_{j \in \mathcal{S}} \psi_j \psi^j} = \frac{\prod\limits_{i \in \mathcal{K}} P(K_i)}{\prod\limits_{j \in \mathcal{S}} P(S_j)}, \tag{3.4.1}$$

hvor første lighed følger af Sætning 3.2.3 og sidste af Proposition 3.4.17. Dette giver, at sandsynlighedsfordelingen kan udtrykkes ud fra kliker og separatorer fra et tilslutningstræ for domænegrafen.

Da en del af resultaterne i dette afsnit omhandler ikke-orienterede grafer, er de også aktuelle for Markovnetværk, der bliver behandlet i Afsnit 4.

3.5 Parameterestimation

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 6) og Cohen (2016: Kap. 2.2, 3.1 og 3.2.1).

Indtil nu har det for bayesianske netværk været antaget, at både grafens struktur og de tilhørende betingede sandsynligheder er kendt. Dette afsnit omhandler, hvordan sandsynlighedsfordelingerne for et bayesiansk netværk med kendt struktur kan estimeres fra givet data. Disse sandsynligheder er modellens parametre, som angives med θ. Afsnit 3.6 vil herefter give eksempler på, hvordan strukturen kan bestemmes, hvis denne også er ukendt.

3.5.1 Maksimum likelihood estimation

Betragt et bayesiansk netværk BN for variablene $A_1, \ldots, A_n$, hvor A_i har de mulige tilstande $(a_{i1}, \ldots, a_{in_i})$ og forældre $A_{i_{f1}}, \ldots, A_{i_{fm}}$. Lad $\mathcal{D}$ være en mængde af observeret data uden manglende værdier med elementer på formen $d = (d_1, \ldots, d_n)$, og antag, at alle observationer er uafhængige. Da er likelihoodfunktionen

$$
\begin{aligned}
L(BN(\theta);\mathcal{D}) &= \prod_{d\in\mathcal{D}} P(A_1 = d_1, \ldots, A_n = d_n | BN(\theta)) \\
&= \prod_{d\in\mathcal{D}} \prod_{i=1}^{n} P(A_i = d_i | A_{i_{f1}} = d_{i_{f1}}, \ldots, A_{i_{fm}} = d_{i_{fm}}),
\end{aligned}
$$

hvor sidste lighed følger af Sætning 3.2.3.

Antag, at de betingede sandsynligheder i det bayesianske netværk er uafhængige, og betragt først en variabel A_i uden forældre. Lad dennes sandsynlighedsfordeling være $P(A_i) = (p_{i1}, \ldots, p_{in_i})$, og bemærk, at dette er en kategorisk fordeling. For at estimere denne sandsynlighedsfordeling er det, grundet uafhængigheden, nok at betragte

$$
L_i(BN(\theta);\mathcal{D}) = \prod_{d\in\mathcal{D}} P(A_i = d_i) = \prod_{j=1}^{n_i} p_{ij}^{N(a_{ij})}.
$$

$N(a_{ij})$ angiver antal tilfælde i datamængden, hvor $d_i = a_{ij}$. For at maksimere dette udtryk tages logaritmen, hvilket giver

$$
l_i(BN(\theta);\mathcal{D}) = \sum_{j=1}^{n_i} N(a_{ij}) \log(p_{ij}).
$$

Det er ikke nok bare at maksimere dette med hensyn til p_{ij}, da det skal gælde, at $\sum_{j=1}^{n_i} p_{ij} = 1$. Denne betingelse tilføjes til udtrykket via en *Lagrange multiplier*, hvilket giver

$$
l_i(BN(\theta);\mathcal{D},\lambda) = \sum_{j=1}^{n_i} N(a_{ij}) \log(p_{ij}) + \lambda \left(1 - \sum_{j=1}^{n_i} p_{ij}\right).
$$

Likelihoodligningen bliver dermed

$$
\frac{\mathrm{d}l_i}{\mathrm{d}p_{ij}} = \frac{N(a_{ij})}{p_{ij}} - \lambda = 0.
$$

Maksimum likelihood estimatet af p_{ij} bliver derfor $\hat{p}_{ij}^L = \frac{N(a_{ij})}{\lambda}$. For at bestemme λ udnyttes det, at

$$
1 = \sum_{j=1}^{n_i} \hat{p}_{ij}^L = \sum_{j=1}^{n_i} \frac{N(a_{ij})}{\lambda},
$$

hvilket giver, at $\lambda = \sum_{j=1}^{n_i} N(a_{ij}) = |\mathcal{D}|$. Det opnås derfor, at

$$
\hat{p}_{ij}^L = \frac{N(a_{ij})}{|\mathcal{D}|}.
$$

Estimatet bliver derfor frekvensen af a_{ij} i data.

Lad nu A_i have forældre. For at foretage estimation af $P(A_i|pa(A_i))$ er det, grundet uafhængigheden, igen nok at betragte

$$L_i(BN(\theta);\mathcal{D}) = \prod_{d\in\mathcal{D}} P(A_i = d_i|A_{i_{f1}} = d_{i_{f1}}, \ldots, A_{i_{fm}} = d_{i_{fm}}).$$

Lad ydermere $P(A_i|A_{i_{f1}} = a_{i_{f1}}, \ldots, A_{i_{fm}} = a_{i_{fm}})$ og $P(A_i|A_{i_{f1}} = a'_{i_{f1}}, \ldots, A_{i_{fm}} = a'_{i_{fm}})$ være uafhængige for forskellige forælderkonfigurationer. Hvis det ønskes at estimere udtrykket $P(A_i|A_{i_{f1}} = a_{i_{f1}}, \ldots, A_{i_{fm}} = a_{i_{fm}}) = (p_{i1}, \ldots, p_{in_i})$, er det derfor nok at betragte

$$L_i(BN(\theta);\mathcal{D}) = \prod_{d\in\mathcal{D}_{if}} P(A_i = d_i|A_{i_{f1}} = a_{i_{f1}}, \ldots, A_{i_{fm}} = a_{i_{fm}}).$$

$\mathcal{D}_{if}$ er de tilfælde i $\mathcal{D}$, hvor $d_{i_{f1}} = a_{i_{f1}}, \ldots, d_{i_{fm}} = a_{i_{fm}}$. Altså er situationen den samme som før blot restringeret til en muligvis mindre delmængde af datamængden. Maksimum likelihood estimatet af p_{ij} bliver derfor mere generelt

$$\hat{p}^L_{ij} = \frac{N(a_{ij}, a_{i_{f1}}, \ldots, a_{i_{fm}})}{N(a_{i_{f1}}, \ldots, a_{i_{fm}})}. \tag{3.5.1}$$

Maksimum likelihood estimation giver ganske intuitive estimater, men de kan have en uhensigtsmæssig effekt. Hvis en bestemt konfiguration af variable aldrig forekommer i databasen, vil dennes sandsynlighed blive estimeret til nul, hvilket ofte er uhensigtsmæssigt.

3.5.2 Bayesiansk estimation

Et alternativ til maksimum likelihood estimation er bayesiansk estimation. Betragt på ny en variabel A_i, der følger en kategorisk fordeling med udfaldsrum $\{a_{i1}, \ldots, a_{in}\}$ og tilhørende sandsynligheder $p_i = (p_{i1}, \ldots, p_{in})$. Udgangspunktet i bayesiansk estimation er at bestemme en fordeling for den ukendte parameter p_i. Til dette vælges først en prior fordeling for p_i. Dennes tæthedsfunktion vælges til

$$f_{prior}(p_i|\alpha_1, \ldots, \alpha_n) = \frac{\Gamma\left(\sum_{j=1}^{n} \alpha_j\right)}{\prod_{j=1}^{n} \Gamma(\alpha_j)} \prod_{j=1}^{n} p_{ij}^{\alpha_j - 1},$$

hvilket giver en dirichletfordeling med parameter $\alpha_{prior} = (\alpha_1, \ldots, \alpha_n)$. Denne fordeling er velegnet for p_i, da fordelingen kræver, at $p_{ij} \geq 0$, og $\sum_{j=1}^{n} p_{ij} = 1$, hvilket sandsynlighederne naturligt skal opfylde. Som det ses, afhænger fordelingen af α_{prior}, som skal have positive indgange.

Betragt igen en datamængde $\mathcal{D}$ med uafhængige observationer og lad alle realisationer af A_i være samlet i vektoren d_i. Det ønskes nu at bestemme posteriorfordelingen af p_i givet dette data, og ved brug af Bayes' formel fås,

$$f_{posterior}(p_i|d_i) = \frac{P(d_i|p_i)f_{prior}(p_i)}{P(d_i)} = \frac{\prod_{j=1}^{n}\left(p_{ij}^{N(a_{ij})}\right)\frac{\Gamma\left(\sum_{j=1}^{n}\alpha_j\right)}{\prod_{j=1}^{n}\Gamma(\alpha_j)}\prod_{j=1}^{n}p_{ij}^{\alpha_j-1}}{P(d_i)} \propto \prod_{j=1}^{n}p_{ij}^{(N(a_{ij})+\alpha_j)-1}.$$

Den udeladte konstant i det sidste udtryk er blot en normaliseringskonstant, som ikke afhænger af p_i. Dermed ses det, at der igen opnås en dirichletfordeling med parametrene $\alpha_{posterior} = (N(a_{i1}) + \alpha_1, \ldots, N(a_{in}) + \alpha_n)$. Det faktum, at posteriorfordelingen er samme type fordeling som priorfordelingen, betyder, at dirichletfordelingen er en såkaldt konjugeret prior for den kategoriske fordeling.

Hvis der ønskes et enkelt estimat for p_i, kan den forventede værdi ud fra posteriorfordelingen for eksempel benyttes. Hvis $p_i = (p_{i1}, \ldots, p_{in})$ følger en dirichletfordeling med parameter $\alpha = (\alpha_1, \ldots, \alpha_n)$, gælder det jævnfør Cohen (2016: s. 35), at

$$E[p_{ij}] = \frac{\alpha_j}{\sum_{k=1}^{n}\alpha_k}.$$

Da posteriorfordelingen er en dirichletfordeling, bliver et sådant estimat for p_{ij} derfor

$$\hat{p}_{ij}^{B} = E_{posterior}[p_{ij}] = \frac{N(a_{ij}) + \alpha_j}{\sum_{j=1}^{n}(N(a_{ij}) + \alpha_j)} = \frac{N(a_{ij}) + \alpha_j}{|\mathcal{D}| + \sum_{j=1}^{n}\alpha_j}.$$

Dette estimat kommer altså til at svare til frekvensen, hvis der tilføjes ekstra fiktive observation af hvert udfald. Det har derfor ikke den svaghed, at ikke-observerede udfald får sandsynlighed nul.

Ovenstående tilfælde svarer i det bayesianske netværk til at estimere sandsynligheder for en knude uden forældre. Hvis knuden har forældre, bestemmes den betingede sandsynlighedsfordeling ved at restringere data til en specifik forælderkonfiguration som ved maksimum likelihood estimation.

3.6 Bestemmelse af strukturen i bayesianske netværk

Dette afsnit er baseret på Jensen & Nielsen (2007: Kap. 7).

Dette afsnit omhandler, hvordan strukturen i et bayesiansk netværk kan bestemmes ud fra data. Afsnittet beskæftiger sig med to metoder, *hill-climbing* og Chow-Liu-træer.

3.6.1 Hill-climbing

Idéen med *hill-climbing* er at udregne en score for forskellige bayesianske netværk og vælge det med den højeste score. Denne score skal ideelt set vurdere, hvor godt modellen passer til data sammenlignet med, hvor kompleks modellen er. Et bayesiansk netværks kompleksitet kan måles i, hvor mange uafhængige parametre modellen har. Det antages, at de betingede sandsynlighedsfordelinger og det givne data er uafhængige, som beskrevet i Afsnit 3.5. Til enhver variabel med m udfald er der tilknyttet m sandsynligheder. Da disse summer til én, er det kun $m-1$ af disse, som er uafhængige. Hvis variablen har forældre, er der derfor $m-1$ uafhængige sandsynligheder per forælderkonfiguration. Dermed bliver antal uafhængige parametre i et bayesiansk netværk BN over variablene $A_1, \ldots, A_n$

$$\text{size}(BN) = \sum_{i=1}^{n} \texttt{\#konfig}(pa(A_i)) \cdot (\texttt{\#konfig}(A_i) - 1),$$

hvor `#konfig`() angiver antal mulige konfigurationer af de involverede variable. Per konvention er $\texttt{\#konfig}(\emptyset) = 1$.

Et eksempel på en score, der både involverer, hvor godt modellen passer til data $\mathcal{D}$ og modellens kompleksitet, er BIC-kriteriet, hvor BIC er en forkortelse for *Bayesian Information Criterion*, som er defineret ved

$$BIC(BN|\mathcal{D}) = \log_2\left(L\left(BN\left(\hat{\theta}^L\right);\mathcal{D}\right)\right) - \frac{\text{size}(BN)}{2}\log_2(|\mathcal{D}|).$$

Første led er højt, hvis modellen passer godt til data, hvorimod andet led straffer i henhold til modellens kompleksitet. Bemærk, at med denne definition ønskes en høj BIC score.

Det er ikke muligt at undersøge alle mulige strukturer af det bayesianske netværk, da der er for mange muligheder, med mindre netværket er over meget få variable. Derfor er det nødvendigt at fastlægge en procedure til at udvælge de strukturer, der undersøges. Dette kunne for eksempel være en grådig søgning, hvilket proceduren *hill-climbing* benytter. Denne søgning starter med en DAG, og BIC udregnes for det tilhørende bayesianske netværk. Herefter konstrueres én ny DAG ud fra den oprindelige ved at foretage én af følgende tre operationer:

1. Tilføje en ny kant.
2. Fjerne en eksisterende kant.
3. Ændre en kants orientering.

Bemærk, at en sådan operation kun er lovlig, hvis den resulterende graf stadig er en DAG. Dette gentages for alle lovlige operationer, og BIC udregnes for alle de tilhørende nye

bayesianske netværk. Herefter vælges DAG'en for det netværk, som har højest BIC, og hvis dette ikke er den oprindelige gentages hele proceduren for denne DAG. Denne algoritme er opsummeret i Algoritme 3.6.1.

Algoritme 3.6.1 `Hill-climbing`

`Input`: Data med realisationer af variablene.

1. Start med en tilfældig DAG D, regn netværkets BIC, og gem D og BIC i et datasæt L.
2. Konstruer en ny DAG $D1$ ud fra D ved at udføre én af følgende operationer:
   (a) Tilføj en ny kant.
   (b) Fjern en eksisterende kant.
   (c) Ændr orienteringen af en kant.
3. Regn BIC for netværket med $D1$ som DAG, og tilføj $D1$ og BIC til L.
4. Gentag punkt 2. – 3. for alle lovlige operationer på D.
5. Vælg DAG'en D_{ny} for det netværk med højest BIC fra L.
6. Hvis $D_{ny} \neq D$, sættes $D = D_{ny}$, og alle øvrige netværk i L fjernes.
7. Gentag punkt 2. – 6., indtil D ikke ændres.

`Output`: DAG'en for det valgte bayesianske netværk.

3.6.2 Chow-Liu-træer

Dette afsnit er yderligere baseret på Chow & Liu (1968).
I stedet for at inkorporere kompleksiteten i en score er det muligt at begrænse DAG'ens struktur for at undgå komplekse modeller. For eksempel er det muligt at begrænse sig til kun at betragte DAG's, der har en orienteret træ-struktur. Dette medfører, at hver knude højst har én forælder, hvilket betyder, at estimation af parametre i dette tilfælde aldrig involverer mere end to variable ad gangen. Dermed opnås en forholdsvis simpel model.

Det giver intuitivt god mening, at når hver knude højst kan have én forælder, bør knuder, der er meget afhængige, forbindes. Til at vurdere, hvor afhængige variable er, benyttes *mutual information* givet i (B.0.6).

Baseret på *mutual information* kan Algoritme 3.6.2 benyttes til at bestemme et bayesiansk netværk, hvor DAG'en har en træ-struktur.

Algoritme 3.6.2 `Chow-Liu-træ`

`Input`: Data med realisationer af variablene.

1. Beregn *mutual information* for alle par af variable. Sorter parrene efter aftagende *mutual information*.
2. Tilføjelse af kanter:
   (a) Tilføj en kant mellem første par.
   (b) Tilføj en kant mellem næste par, hvis dette ikke introducerer en kreds.
   (c) Gentag punkt (b) indtil der er tilføjet $n-1$ kanter.
3. Vælg en rod tilfældigt, og orienter alle kanter i retningen væk fra roden.

`Output`: Nabomatricen for Chow-Liu-træet.

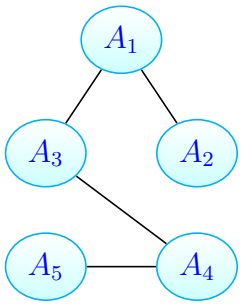
Et bayesiansk netværk konstrueret ved denne fremgangsmåde kaldes et Chow-Liu-træ. Bemærk, at konstruktionen af den ikke-orienterede graf svarer til at bestemme et maksimalt udspændende træ for den komplette graf med *mutual information* som vægte. Den her beskrevne algoritme kaldes Kruskals algoritme, og det kan bevises, at den resulterende graf er et maksimalt udspændende træ, se for eksempel Kruskal (1956).[3] I Eksempel 3.6.3 ses et eksempel på anvendelsen af Algoritme 3.6.2.

Eksempel 3.6.3
I tabellen i Figur 3.6.1a ses et udsnit af fiktive *mutual information* værdier for kombinationer af variablene $A_1, \ldots, A_5$ sorteret efter aftagende *mutual information*. Den første kant, der sættes, er $\{A_1, A_2\}$. Herefter sættes kanterne $\{A_3, A_5\}$ og $\{A_1, A_3\}$, som begge kan placeres uden at introducere kredse. Den næste kant, der foreslås, er $\{A_2, A_3\}$, men dette ville danne kredsen A_1, A_2, A_3, A_1, så der fortsættes til næste par. Dette er $\{A_1, A_5\}$, som dog også ville introducere en kreds. Derfor fortsættes til $\{A_4, A_5\}$, som accepteres. Grafen indeholder nu det korrekte antal kanter, og algoritmen stopper. Det resulterende skelet for Chow-Liu-træet ses i Figur 3.6.1b.

[3] Der findes også andre algoritmer til at bestemme maksimalt udspændende træer, men i dette speciale er det ikke så vigtigt, præcis hvordan det findes, så længe resultatet er et maksimalt udspændende træ.

Variable	Mutual information
A_1, A_2	**2,7**
A_3, A_5	**2,2**
A_1, A_3	**1,9**
A_2, A_3	1,8
A_1, A_5	1,4
A_4, A_5	**1,1**

(a) Tabel over *mutual information.*



(b) Skelet for Chow-Liu-træ.

Figur 3.6.1: Eksempel på anvendelsen af Algoritme 3.6.2.

Sætning 3.6.4 giver, at ud af alle bayesianske netværk med den angivne træ-struktur, giver Chow-Liu-træet den bedste approksimation til den sande sandsynlighedsfordeling.

Sætning 3.6.4
Lad P være den sande sandsynlighedsfordeling for variablene $A = (A_1, \ldots, A_n)$ og P^{CL} være den sandsynlighedsfordeling, som Chow-Liu-træet over variablene giver. Lad desuden $KL()$ være Kullback-Leibler-afstanden givet i (B.0.4). Da er

$$KL(P, P^{CL}) \leq KL(P, P^T)$$

for enhver anden sandsynlighedsfordeling P^T baseret på et bayesiansk netværk med en træ-struktur.

Bevis. Det vides fra Sætning 3.2.3, at

$$P^T(A) = \prod_{i=1}^{n} P(A_i | A_{i_f}),$$

hvor A_{i_f} er forælderen til A_i. Hvis A_i ingen forælder har sættes $A_{i_f} = \emptyset$. Da er

$$\begin{aligned}
KL(P, P^T) &= \sum_A P(A) \log_2 \left(\frac{P(A)}{P^T(A)} \right) \\
&= \sum_A P(A) \log_2 \left(\frac{P(A)}{\prod_{i=1}^{n} P(A_i | A_{i_f})} \right) \\
&= \sum_A P(A) \log_2 (P(A)) - \sum_A P(A) \sum_{i=1}^{n} \log_2 (P(A_i | A_{i_f})) \\
&= -H(A) - \sum_A P(A) \sum_{i=1}^{n} \left(\log_2 \left(\frac{P(A_i, A_{i_f})}{P(A_{i_f})} \right) - \log_2 (P(A_i)) + \log_2 (P(A_i)) \right)
\end{aligned}$$

$$
\begin{aligned}
&= -H(A) - \sum_{A} P(A) \sum_{i=1}^{n} \log_2 \left(\frac{P\left(A_i, A_{i_f}\right)}{P(A_i)P(A_{i_f})} \right) - \sum_{A} P(A) \sum_{i=1}^{n} \log_2(P(A_i)) \\
&= -H(A) - \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} \log_2 \left(\frac{P(A_i, A_{i_f})}{P(A_i)P(A_{i_f})} \right) \sum_{\substack{A_j \\ j \neq i, i_f}} P(A) - \sum_{i=1}^{n} \sum_{A_i} \log_2(P(A_i)) \sum_{\substack{A_j \\ j \neq i}} P(A) \\
&= -H(A) - \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} \log_2 \left(\frac{P(A_i, A_{i_f})}{P(A_i)P(A_{i_f})} \right) P(A_i, A_{i_f}) - \sum_{i=1}^{n} \sum_{A_i} \log_2(P(A_i)) P(A_i) \\
&= -H(A) - \sum_{i=1}^{n} MI(A_i, A_{i_f}) + \sum_{i=1}^{n} H(A_i), \qquad (3.6.1)
\end{aligned}
$$

hvor $H(A)$ er entropien givet ved (B.0.1). Det ses, at det eneste i 3.6.1, der afhænger af strukturen, er $\sum_{i=1}^{n} MI(A_i, A_{i_f})$, hvorfor Kullback-Leibler-afstanden minimeres, når dette maksimeres. At vælge forældrene, så $\sum_{i=1}^{n} MI(A_i, A_{i_f})$ maksimeres, er netop det, konstruktionen af et Chow-Liu-træ gør, så dette giver en sandsynlighedsfordeling, som har den mindste Kullback-Leibler-afstand til den sande sandsynlighedsfordeling.

■

I praksis kan *mutual information* ikke bestemmes, fordi fordelingen er ukendt. Den kan derimod estimeres fra data ved

$$
\widehat{MI}(A_i, A_j) = \sum_{A_i, A_j} P^L(A_i, A_j) \log_2 \left(\frac{P^L(A_i, A_j)}{P^L(A_i)P^L(A_j)} \right),
$$

hvor P^L angiver sandsynlighedsfordelingen estimeret ved maksimum likelihood estimation, som i Afsnit 3.5. Estimatet kan også findes med bayesiansk estimation. Sætning 3.6.5 giver, at hvis et Chow-Liu-træ konstrueres med estimaterne for MI, opnås et bayesiansk netværk med en træ-struktur af maksimal likelihood.

Sætning 3.6.5
Hvis det bayesianske netværk restringeres til at have en træ-struktur, giver Chow-Liu-træet et bayesiansk netværk af maksimal likelihood, såfremt maksimum likelihood estimation bruges for parametrene.

Bevis. Lad BN være et bayesiansk netværk med en træ-struktur. Genkald, at likelihood-funktionen for BN givet et datasæt $\mathcal{D}$ med uafhængige observationer er givet ved

$$
L(BN(\theta); \mathcal{D}) = \prod_{d \in \mathcal{D}} \prod_{i=1}^{n} P(A_i = d_i | A_{i_f} = d_{i_f}),
$$

hvor A_{i_f} er forælderknuden til A_i. Hvis A_i ingen forælder har sættes $A_{i_f} = \emptyset$. Derfor bliver log-likelihoodfunktionen

$$l(BN(\theta); \mathcal{D}) = \sum_{d \in \mathcal{D}} \sum_{i=1}^{n} \log_2(P(A_i = d_i | A_{i_f} = d_{i_f})). \tag{3.6.2}$$

Bemærk, at den samme konfiguration af variable kan gå igen i flere datapunkter. Derfor er det ikke nødvendigt at beregne sandsynligheden hver gang, men blot en enkelt gang per konfiguration. Lad $N()$ være antal gange, en konfiguration er set i data. Dermed fås det, at $P^L(A_i = a_i, A_{i_f} = a_{i_f}) = \frac{N(A_i = a_i, A_{i_f} = a_{i_f})}{|\mathcal{D}|}$, så

$$N(A_i = a_i, A_{i_f} = a_{i_f}) = |\mathcal{D}| P^L(A_i = a_i, A_{i_f} = a_{i_f}).$$

Dermed kan (3.6.2) omskrives til

$$l(BN(\theta); \mathcal{D}) = |\mathcal{D}| \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} P^L(A_i, A_{i_f}) \log_2(P(A_i | A_{i_f})),$$

hvor summen over variable som altid er over deres mulige tilstande. Da der ønskes et bayesiansk netværk af maksimal likelihood, benyttes maksimum likelihood estimatet for $P(A_i | A_{i_f})$, og dette giver

$$\begin{aligned}
l(BN(\hat{\theta}^L); \mathcal{D}) &= |\mathcal{D}| \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} P^L(A_i, A_{i_f}) \log_2(P^L(A_i | A_{i_f})) \\
&= |\mathcal{D}| \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} P^L(A_i, A_{i_f}) \left(\log_2 \left(\frac{P^L(A_i, A_{i_f})}{P^L(A_{i_f})} \right) - \log_2(P^L(A_i)) + \log_2(P^L(A_i)) \right) \\
&= |\mathcal{D}| \left(\sum_{i=1}^{n} \sum_{A_i, A_{i_f}} P^L(A_i, A_{i_f}) \log_2 \left(\frac{P^L(A_i, A_{i_f})}{P^L(A_i) P^L(A_{i_f})} \right) + \sum_{i=1}^{n} \sum_{A_i, A_{i_f}} P^L(A_i, A_{i_f}) \log_2(P^L(A_i)) \right) \\
&= |\mathcal{D}| \left(\sum_{i=1}^{n} \widehat{MI}(A_i, A_{i_f}) + \sum_{i=1}^{n} \sum_{A_i} P^L(A_i) \log_2(P^L(A_i)) \right).
\end{aligned}$$

Det ses, at det eneste som afhænger af strukturen er $\sum_{i=1}^{n} \widehat{MI}(A_i, A_{i_f})$. En struktur af maksimal likelihood opnås derfor ved at maksimere dette, hvilket konstruktionen af Chow-Liu-træet netop gør.

■

4 Tredje ordens t-kirsebærtræer

Dette afsnit er baseret på Jordan (1999) og Kovács & Szántai (2008).
Afsnit 3 har behandlet, hvordan bayesianske netværk kan bruges til at modellere afhængigheder mellem variable. Dette afsnit benytter en anden form for modellering, som tager udgangspunkt i en type af grafiske modeller kaldet Markovnetværk.

Definition 4.0.1 (Markovnetværk)
Et Markovnetværk er en ikke-orienteret graf, hvor hver knude repræsenterer en diskret stokastisk variabel med endeligt udfaldsrum.

I et Markovnetværk betragtes to variable som uafhængige givet en mængde $\mathcal{G}$, hvis alle stier i grafen mellem de to variable går gennem $\mathcal{G}$.

For et Markovnetværk approksimeres variablenes sandsynlighedsfordeling ved et produkt af potentialer over grafens kliker. I dette speciale arbejdes der kun med Markovnetværk, hvor grafen er trianguleret, hvorved klikerne kan organiseres i et tilslutningstræ jævnfør Sætning 3.4.12. Hvis $\mathcal{K}$ er mængden af grafens kliker, og $\mathcal{S}$ er mængden af separatorer, benyttes sandsynlighedsfordelingen

$$P(A_1, \ldots, A_n) = \frac{\prod_{K \in \mathcal{K}} P(K)}{\prod_{S \in \mathcal{S}} P(S)^{\#_S}}, \tag{4.0.1}$$

hvor $\#_S$ angiver antal gange, som separatoren S indgår i tilslutningstræet. Bemærk, at dette udtryk svarer til (3.4.1) for bayesianske netværk. Den eneste forskel er, at tilslutningstræet i dette udtryk var for domænegrafen i stedet. Et Markovnetværk kan godt repræsentere domænegrafen for et bayesiansk netværk, men da forskellige bayesianske netværk kan have samme domænegraf, kan der ikke entydigt tilknyttes et bayesiansk netværk til Markovnetværket.

Benyt indekseringen af kliker og separatorer i Algoritme 3.4.10. Lad $\mathcal{I}_K$ og $\mathcal{I}_S$ være mængden af indeks for henholdsvis klikerne og separatorerne. Definer desuden $R_i = K_i \setminus S_i$ for $i < n$. Bemærk, at da kan (4.0.1) også skrives som

$$P(A_1, \ldots, A_n) = P(K_n) \prod_{i \in \mathcal{I}_S} P(R_i | S_i).$$

Ud fra denne formulering er det muligt at bestemme antal frie parametre i modellen. Antag, at sandsynlighedsfordelingerne i produktet er uafhængige. For sandsynlighedsfordelingen $P(K_n)$ er der $\texttt{\#konfig}(K_n) - 1$ frie parametre. For fordelingerne på formen $P(R_i|S_i)$ er der $(\texttt{\#konfig}(R_i) - 1) \cdot \texttt{\#konfig}(S_i)$ frie parametre. For et Markovnetværk M er der altså

$$\begin{aligned}\text{size}(M) &= \texttt{\#konfig}(K_n) - 1 + \sum_{i \in \mathcal{I}_S} \Big(\big(\texttt{\#konfig}(R_i) - 1 \big) \cdot \texttt{\#konfig}(S_i) \Big) \\ &= \sum_{i \in \mathcal{I}_K} \texttt{\#konfig}(K_i) - \sum_{i \in \mathcal{I}_S} \texttt{\#konfig}(S_i) - 1\end{aligned}$$

frie parametre.

I resten af specialet behandles Markovnetværk, hvor grafen har en særlig struktur, kaldet t-kirsebærtræer. I dette afsnit introduceres konceptet for 3. ordens t-kirsebærtræer, og efterfølgende generaliseres konceptet til vilkårlig orden k i Afsnit 6.

4.1 Sandsynlighedsfordelinger baseret på t-kirsebærtræer

Dette afsnit er baseret på Kovács & Szántai (2010).
Den grafiske struktur kaldet et t-kirsebærtræ defineres i Definition 4.1.1

Definition 4.1.1 (t-kirsebærtræ)
Et t-kirsebærtræ er en graf, der kan konstrueres rekursivt ved følgende fremgangsmåde:

1. Start med to knuder forbundet af en kant. Dette er også det mindst mulige t-kirsebærtræ.
2. t-kirsebærtræet udvides med en ny knude, ved at forbinde denne via to nye kanter til to knuder, som er naboer i det eksisterende t-kirsebærtræ. De tre involverede knuder danner et kirsebær. Hvis den nye knude er A_k, og de to knuder, den forbindes til, er A_i og A_j, benævnes kirsebærret $(\{A_i, A_j\}, A_k)$.
3. Punkt 2. gentages, til grafen har det ønskede antal knuder.

Bemærk, at et t-kirsebærtræ ikke er et træ i grafisk forstand, da det indeholder kredse.

Når et Markovnetværk benyttes til at approksimere en simultan sandsynlighedsfordeling, arbejdes der med grafens kliker. Genkald fra Afsnit 3.4, at det er lettere at bestemme en grafs kliker, hvis den er triangulær.

Proposition 4.1.2

Et t-kirsebærtræ er en trianguleret graf.[a]

[a] Dette er ikke et resultat fra Kovács & Szántai (2010), men vores eget resultat med bevis.

Bevis. Lad t-kirsebærtræet være over variablene $A_1, \ldots, A_n$, og lad deres indeks beskrive, hvornår de tilføjes til træet. For eksempel er $(\{A_1, A_2\}, A_3)$ det første kirsebær. Det ønskes at bestemme en perfekt eliminationsrækkefølge i grafen.

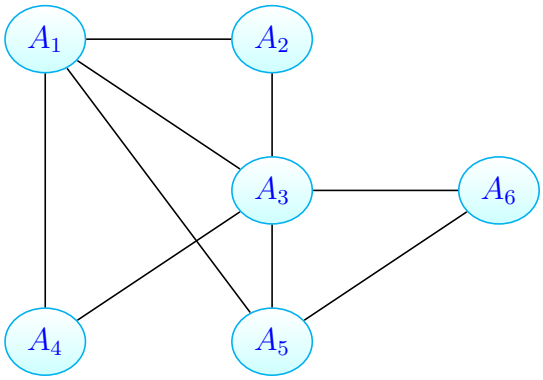
Betragt den sidst tilføjede knude A_n, som kun indgår i ét kirsebær. Dets to naboer er forbundet jævnfør definitionen af et t-kirsebærtræ. Derfor er A_n en simpel knude og elimineres først. Proceduren gentages rekursivt, hvilket giver, at $A_n, A_{n-1}, \ldots, A_1$ er en perfekt eliminationsrækkefølge. Hermed er det bevist, at grafen er triangulær.

■

Da et t-kirsebærtræ er triangulært, giver Sætning 3.4.12, at klikerne kan organiseres i et tilslutningstræ, som defineret i Definition 3.4.8. Til at konstruere tilslutningstræet kan Algoritme 3.4.10 benyttes. Dette illustreres for et t-kirsebærtræ i Eksempel 4.1.3.

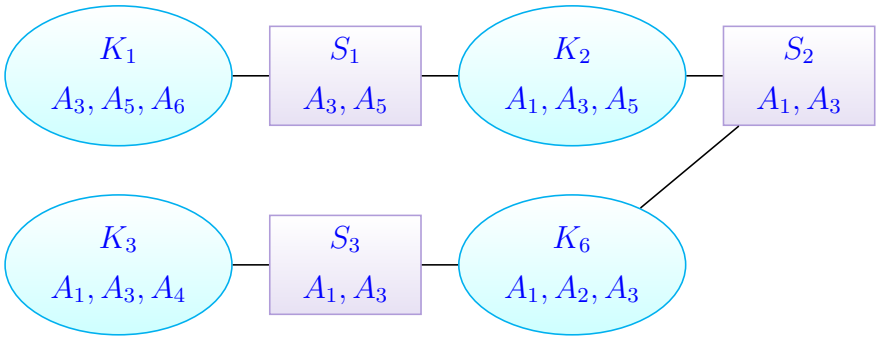
Eksempel 4.1.3

Betragt t-kirsebærtræet i Figur 4.1.1.



Figur 4.1.1: Et t-kirsebærtræ med kirsebærene $(\{A_1, A_2\}, A_3), (\{A_1, A_3\}, A_4), (\{A_1, A_3\}, A_5)$ og $(\{A_3, A_5\}, A_6)$.

Ved at benytte eliminationsrækkefølgen $A_6, A_5, A_4, A_3, A_2, A_1$ giver Algoritme 3.4.10 tilslutningstræet i Figur 4.1.2.



Figur 4.1.2: Konstruktion af et tilslutningstræ for t-kirsebærtræet i Figur 4.1.1.

Bemærk, at når Algoritme 3.4.10 benyttes, ses det, at grafens kliker bliver kirsebærene. For kliken K_h, der svarer til kirsebærret $(\{A_i, A_j\}, A_k)$, bemærkes det yderligere, at separatoren S_h bliver $\{A_i, A_j\}$. Med denne viden om kliker og separatorer specificerer et t-kirsebærtræ sandsynlighedsfordelingen

$$P(A_1, \ldots, A_n) = \frac{\prod\limits_{(\{A_i,A_j\},A_k)\in\mathcal{K}} P(A_i, A_j, A_k)}{\prod\limits_{\{A_i,A_j\}=S\in\mathcal{S}} P(A_i, A_j)^{\#S}},$$

hvor $\mathcal{K}$ er mængden af kliker, og dermed også mængden af kirsebær, og $\mathcal{S}$ er mængden af separatorer. Bemærk, at denne approksimation for t-kirsebærtræer kun benytter simultane sandsynlighedsfordelinger for to og tre variable. Sætning 4.1.5 giver Kulback-Leibler afstanden mellem den sande fordeling og denne approksimation. Sætningen benytter Lemma 4.1.4, som giver, at hver variabel optræder én gang mere i kliker end i separatorer.

Lemma 4.1.4

Betragt et tilslutningtræ T over variablene $A_1, \ldots, A_n$. Lad $\mathcal{K}$ og $\mathcal{S}$ betegne henholdsvis mængden af kliker og separatorer. Da gælder det for en variabel A_i, at

$$\sum_{S\in\mathcal{S}} 1_{[A_i\in S]} \cdot \#S = \left(\sum_{K\in\mathcal{K}} 1_{[A_i\in K]}\right) - 1. \tag{4.1.1}$$

Bevis. Lad først $\sum\limits_{S\in\mathcal{S}} 1_{[A_i\in S]} \cdot \#S = 0$. Det må gælde, at A_i indgår i mindst én klike. Hvis A_i indgår i to kliker, vil A_i indgå i alle kliker på stien mellem disse, og må derfor indgå i mindst én separator. Dette er i strid med antagelsen, så $\sum\limits_{K\in\mathcal{K}} 1_{[A_i\in K]} = 1$. Dermed er (4.1.1) opfyldt.

Lad nu $\sum\limits_{S\in\mathcal{S}} 1_{[A_i\in S]} \cdot \#S > 0$. Da må det gælde, at A_i ligger i mindst to kliker, da en separator indeholder fællesmængden af de to kliker, som kanten forbinder. For hvert par af kliker, findes der en sti mellem disse, da T er et træ. Fordi T er et tilslutningstræ ligger A_i

også i alle kliker på denne sti. Delgrafen T_{A_i} kun med kliker indeholdende A_i er derfor også et træ, med $\sum_{K \in \mathcal{K}} 1_{[A_i \in K]}$ knuder. Da alle knuder i dette træ indeholder A_i, må alle separatorene også indeholde A_i. Desuden kan der ikke være andre separatorer i T, som indeholder A_i, da ingen øvrige kliker indeholder A_i. T_{A_i} har præcis $\left(\sum_{K \in \mathcal{K}} 1_{[A_i \in K]}\right) - 1$ kanter, hvorved ligning (4.1.1) gælder.

Hermed er det bevist, at $\sum_{S \in \mathcal{S}} 1_{[A_i \in S]} \cdot \#_S = \left(\sum_{K \in \mathcal{K}} 1_{[A_i \in K]}\right) - 1$.

■

Sætning 4.1.5
Lad T være et tilslutningstræ for et t-kirsebærtræ over variablene $A = (A_1, \ldots, A_n)$. Betegn mængden af kliker med $\mathcal{K}$ og mængden af separatorer med $\mathcal{S}$. Lad $MI()$ være givet ved (B.0.7). Da er Kullback-Leibler-afstanden mellem den sande og den approksimerede simultane sandsynlighedsfordeling for A givet ved

$$KL(P, P^{tKT}) = -H(A) - \left(\sum_{\{A_i, A_j, A_k\} \in \mathcal{K}} MI(A_i, A_j, A_k) - \sum_{\{A_i, A_j\} = S \in \mathcal{S}} \#_S \cdot MI(A_i, A_j) \right) + \sum_{q=1}^{n} H(A_q), \tag{4.1.2}$$

hvor P betegner den sande sandsynlighedsfordeling, og P^{tKT} betegner den approksimerede simultane sandsynlighedsfordeling for t-kirsebærtræet.

Bevis. Ved brug af definitionen på Kullback-Leibler-afstanden og logaritmeregneregler fås,

$$\begin{aligned}
KL(P, P^{tKT}) &= \sum_A P(A) \log_2 \left(\frac{P(A)}{P^{tKT}(A)} \right) \\
&= \sum_A P(A) \log_2(P(A)) - \sum_A P(A) \log_2(P^{tKT}(A)) \\
&= -H(A) - \sum_A P(A) \log_2 \left(\frac{\prod_{\{A_i, A_j, A_k\} \in \mathcal{K}} P(A_i, A_j, A_k)}{\prod_{\{A_i, A_j\} = S \in \mathcal{S}} P(A_i, A_j)^{\#_S}} \right) \\
&= -H(A) - \sum_A P(A) \log_2 \left(\prod_{\{A_i, A_j, A_k\} \in \mathcal{K}} P(A_i, A_j, A_k) \right) + \sum_A P(A) \log_2 \left(\prod_{\{A_i, A_j\} = S \in \mathcal{S}} P(A_i, A_j)^{\#_S} \right)
\end{aligned}$$

Det ønskes at lægge $\sum_A P(A) \log_2 \left(\prod_{\{A_i, A_j, A_k\} \in \mathcal{K}} P(A_i) P(A_j) P(A_k) \right)$ til og trække den fra igen. Bemærk til dette formål, at jævnfør Lemma 4.1.4 er

$$
\begin{aligned}
\prod_{\{A_i,A_j,A_k\}\in\mathcal{K}} P(A_i)P(A_j)P(A_k) &= \prod_{q=1}^{n} P(A_q)^{\sum_{K\in\mathcal{K}} 1_{[A_q\in K]}} \\
&= \prod_{q=1}^{n} P(A_q)^{\left(\sum_{S\in\mathcal{S}} 1_{[A_q\in S]}\cdot \#_S\right)+1} \\
&= \prod_{q=1}^{n} P(A_q)^{\left(\sum_{S\in\mathcal{S}} 1_{[A_q\in S]}\cdot \#_S\right)} \prod_{q=1}^{n} P(A_q) \\
&= \prod_{\{A_i,A_j\}=S\in\mathcal{S}} (P(A_i)P(A_j))^{\#_S} \prod_{q=1}^{n} P(A_q).
\end{aligned}
$$

Dermed fås det, at

$$
\begin{aligned}
KL(P,P^{tKT}) &= -H(A) - \sum_A P(A)\log_2\left(\frac{\prod_{\{A_i,A_j,A_k\}\in\mathcal{K}} P(A_i,A_j,A_k)}{\prod_{\{A_i,A_j,A_k\}\in\mathcal{K}} P(A_i)P(A_j)P(A_k)}\right) \\
&\quad + \sum_A P(A)\log_2\left(\frac{\prod_{\{A_i,A_j\}=S\in\mathcal{S}} P(A_i,A_j)^{\#_S}}{\prod_{\{A_i,A_j\}=S\in\mathcal{S}} \big(P(A_i)P(A_j)\big)^{\#_S}}\right) - \sum_A P(A)\log_2\left(\prod_{q=1}^{n} P(A_q)\right) \\
&= -H(A) - \sum_A P(A) \sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} \log_2\left(\frac{P(A_i,A_j,A_k)}{P(A_i)P(A_j)P(A_k)}\right) \\
&\quad + \sum_A P(A) \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \log_2\left(\frac{P(A_i,A_j)}{P(A_i)P(A_j)}\right) + \sum_{q=1}^{n} H(A_q) \\
&= -H(A) - \sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} \sum_{A_i,A_j,A_k} P(A_i,A_j,A_k)\log_2\left(\frac{P(A_i,A_j,A_k)}{P(A_i)P(A_j)P(A_k)}\right) \\
&\quad + \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \sum_{A_i,A_j} \#_S P(A_i,A_j)\log_2\left(\frac{P(A_i,A_j)}{P(A_i)P(A_j)}\right) + \sum_{q=1}^{n} H(A_q) \\
&= -H(A) - \left(\sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i,A_j,A_k) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S\cdot MI(A_i,A_j)\right) + \sum_{q=1}^{n} H(A_q),
\end{aligned}
$$

hvor anden og tredje lighed følger af marginalisering af en sandsynlighedsfordeling. Dette beviser resultatet.

■

Bemærk, at det eneste i Kullback-Leibler-afstanden, som afhænger af grafens struktur, er

$$
\sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i,A_j,A_k) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S\cdot MI(A_i,A_j),
$$

så det er tilstrækkeligt at maksimere dette for at minimere Kullback-Leibler-afstanden. Dette udtryk kaldes tilslutningstræets eller strukturens vægt.

Det er muligt at danne et t-kirsebærtræ ud fra et træ. Dette gøres ved følgende algoritme:

Algoritme 4.1.6 `t-kirsebærtræ konstrueret ud fra et træ`

`Input`: Et træ G.

1. Vælg tre knuder i G, som er forbundet af præcis to kanter. Tilføj den manglende kant for at gøre denne punktmængde komplet. Dette er det første kirsebær, og lad dette være tKT.

2. Vælg en knude A_i i G, som er nabo til knuden A_j, der allerede er i tKT. Knuden A_i forbindes til tKT via nabokanten og tilføjelsen af én ny kant fra A_i til en af A_j's naboer i tKT. Tilføj det nye kirsebær til tKT.

3. Punkt 2. gentages, indtil alle knuderne fra G indgår i tKT.

`Output`: t-kirsebærtræet tKT.

Hvis t-kirsebærtræet konstrueres ud fra et Chow-Liu-træ, vides det, at dettes approksimation af sandsynlighedsfordelingen er mindst lige så god, som den opnået ved Chow-Liu-træet, hvilket Sætning 4.1.7 giver.

Sætning 4.1.7
Lad P, P^{CL} og P^{tKT} være henholdsvis den sande sandsynlighedsfordeling for variablene $A_1, \ldots, A_n$ og approksimationen opnået ved med henholdsvis et Chow-Liu-træ og et t-kirsebærtræ konstrueret ud fra Chow-Liu-træet. Da gælder det, at

$$KL(P, P^{CL}) \geq KL(P, P^{tKT}).$$

Bevis. Genkald fra (3.6.1), at

$$KL(P, P^{CL}) = -H(A) - \sum_{q=1}^{n} MI(A_q, A_{q_f}) + \sum_{q=1}^{n} H(A_q),$$

hvor A_{qf} er A_q's forælder, og genkald fra (4.1.2), at

$$KL(P, P^{tKT}) = -H(A) - \left(\sum_{\{A_i, A_j, A_k\} \in \mathcal{K}} MI(A_i, A_j, A_k) - \sum_{\{A_i, A_j\} = S \in \mathcal{S}} \#_S \cdot MI(A_i, A_j) \right) + \sum_{q=1}^{n} H(A_q).$$

Bemærk, at første og sidste led både optræder i $KL(P, P^{CL})$ og $KL(P, P^{tKT})$.

Det er derfor tilstrækkeligt at vise, at

$$\sum_{q=1}^{n} MI(A_q, A_{q_f}) \leq \sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i, A_j, A_k) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \cdot MI(A_i, A_j). \tag{4.1.3}$$

Jævnfør Sætning B.0.7 og (B.0.8) gælder følgende:

$$\begin{aligned} MI(A_i, A_j) &= H(A_i) - H(A_i|A_j), \\ MI(A_i, A_j) &= H(A_i) + H(A_j) - H(A_i, A_j), \\ MI(A_i, A_j, A_k) &= H(A_k) + MI(A_i, A_j) - H(A_k|A_i, A_j). \end{aligned}$$

[4]Lad gennem resten af beviset A_1 være roden i Chow-Liu-træet, og lad denne være A_2's forælder. Da er A_1 den eneste knude i træet uden forælder. Ved at benytte konventionen $MI(A_1, \emptyset) = MI(A_1) = 0$ bliver venstresiden i (4.1.3)

$$\begin{aligned} \sum_{q=2}^{n} MI(A_q, A_{q_f}) &= \sum_{q=3}^{n} (H(A_q) - H(A_q|A_{qf})) + MI(A_2, A_1) \\ &= \sum_{q=3}^{n} (H(A_q) - H(A_q|A_{qf})) + H(A_1) + H(A_2) - H(A_1, A_2) \\ &= \sum_{q=1}^{n} H(A_q) - \sum_{q=3}^{n} H(A_q|A_{qf}) - H(A_1, A_2). \end{aligned} \tag{4.1.4}$$

Bemærk, at i summen over klikerne i (4.1.3) er valget af i, j og k arbitrært, og genkald, at klikerne også er grafens kirsebær. Derfor er

$$\begin{aligned} &\sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i, A_j, A_k) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \cdot MI(A_i, A_j) \\ &\quad = \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} (H(A_k) + MI(A_i,A_j) - H(A_k|A_i, A_j)) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \cdot MI(A_i, A_j) \end{aligned}$$

Lad variablenes indeks angive, i hvilken rækkefølge knuderne er tilføjet til kirsebærtræet, hvor $(\{A_1, A_2\}, A_3)$ er det første kirsebær, og A_1 og A_2 stadig har den antagede betydning i Chow-Liu-træet. For alle andre kirsebær $(\{A_i, A_j\}, A_k)$ er $\{A_i, A_j\}$ en separator, og dette giver alle separatorerne. Derfor er

[4]Beviset afviger herefter en smule fra beviset i Kovács & Szántai (2010), da vi under udarbejdelsen af specialet opdagede en mindre fejl i et af udtrykkene, som artiklens forfattere efterfølgende har anerkendt.

$$
\begin{aligned}
&\sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}}(H(A_k)+MI(A_i,A_j)-H(A_k|A_i,A_j)) \quad - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \cdot MI(A_i,A_j) \\
&= \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k) \quad - \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k|A_i,A_j) + MI(A_1,A_2) \\
&= \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k) \quad - \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k|A_i,A_j) + H(A_1)+H(A_2)-H(A_1,A_2) \\
&= \sum_{q=1}^{n} H(A_q) \quad - \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k|A_i,A_j) - H(A_1,A_2). \qquad (4.1.5)
\end{aligned}
$$

Bemærk, at $\sum_{q=1}^{n} H(A_q)$ og $-H(A_1,A_2)$ indgår i både (4.1.4) og (4.1.5), så det er altså nu tilstrækkeligt at vise, at

$$\sum_{q=3}^{n} H(A_q|A_{qf}) \geq \sum_{(\{A_i,A_j\},A_k)\in\mathcal{K}} H(A_k|A_i,A_j). \qquad (4.1.6)$$

Jævnfør (B.0.3) gælder det, at

$$H(A_i|A_j) \geq H(A_i|A_j,A_k). \qquad (4.1.7)$$

Det ville derfor være belejligt, hvis alle led på højresiden af (4.1.6) kunne matches med led på venstresiden, så (4.1.7) kan benyttes.

Bemærk først, at på højresiden i (4.1.6) gennemløber indekset k værdierne $3,\ldots,n$. Bemærk næst, at når en variabel tilføjes til t-kirsebærtræet T, indgår mindst én kant fra Chow-Liu-træet[5] C i den nye klike i T. Da alle kanter i C er orienteret væk fra roden, som indgår i den først tilføjede klike i T, må denne kant i C være orienteret mod den nye knude, som derfor er barn af en knude i samme klike i T. For alle udtryk på formen $H(A_k|A_i,A_j)$ gælder det derfor enten, at $A_i = A_{k_f}$ eller $A_j = A_{k_f}$. Alle led på højresiden i (4.1.6) kan altså passende matches med led på venstresiden, så (4.1.7) giver, at

$$\sum_{q=1}^{n} H(A_q|A_{qf}) \geq \sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} H(A_k|A_i,A_j),$$

hvilket var det ønskede.

■

Sætning 4.1.7 giver, at sandsynlighedsfordelingen fra ethvert t-kirsebærtræ konstrueret fra et Chow-Liu-træ approksimerer den sande sandsynlighedsfordeling mindst lige så godt, men det vil typisk være muligt at konstruere forskellige t-kirsebærtræer fra Chow-Liu-træet. Det

[5]Bemærk, at kantens orientering ikke er med i T.

ønskes derfor at finde en måde til at konstruere det t-kirsebærtræ, der giver den bedste approksimation. Dette behandles nærmere i næste afsnit, der omhandler, hvordan et t-kirsebærtræ kan konstrueres fra observeret data.

4.2 Implementering af konstruktionsalgoritmer

Algoritme 4.1.6 kan benyttes til at bestemme et t-kirsebærtræ ud fra et Chow-Liu-træ. I dette afsnit forsøges det at udvide denne algoritme til at opnå en god approksimation af den ønskede sandsynlighedsfordeling i den forstand at Kullback-Leibler-afstanden minimeres. Jævnfør Sætning 4.1.5 svarer dette til at maksimere vægten givet ved

$$\sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i,A_j,A_k) - \sum_{\{A_i,A_j\}=S\in\mathcal{S}} \#_S \cdot MI(A_i,A_j). \tag{4.2.1}$$

I praksis vil det typisk være nødvendigt at estimere de to *mutual information* størrelser. Dette gøres ved at estimere de nødvendige sandsynligheder via metoderne i Afsnit 3.5.
En metode til at opnå den bedste udvidelse af et Chow-Liu-træ ville være at regne (4.2.1) ud for alle muligheder og vælge det med den højeste vægt. Dette er dog ikke særlig effektivt, så i stedet benyttes en grådig fremgangsmåde.

Det vælges i dette afsnit at benytte samme fremgangsmåde som i Kovács & Szántai (2010) nemlig at bruge en grådig algoritme til at maksimere

$$\sum_{\{A_i,A_j,A_k\}\in\mathcal{K}} MI(A_i,A_j,A_k).$$

Bemærk, at dette ikke er det samme som (4.2.1), men det giver færre kombinationer kun at betragte mulige kliker i stedet for mulige klike-separator par. Det er ikke nødvendigvis en god idé kun at betragte *mutual information* for klikerne, og denne problematik undersøges yderligere i Afsnit 6.3 og 6.4.

Denne fremgangsmåde er beskrevet i Algoritme 4.2.1, og vi har selv implementeret denne i funktionen `tcherry_CL`, som findes i vores egen **R**-pakke `tcherry` (Kirkeby et al. 2019). Denne **R**-pakke er udviklet som en del af specialet og kan findes på `https://github.com/nvihrs14/tcherry`. Algoritmen kalder funktionen `ChowLiu` fra samme pakke, som finder et Chow-Liu-træ med metoden beskrevet i Algoritme 3.6.2.

Algoritme 4.2.1 *t*-`kirsebærtræ konstrueret ud fra Chow-Liu-træet`

`tcherry_CL(data, ...)`

(`data`: Et datasæt med realisationer af variablene, ...: Øvrige argumenter til beregning af mutual information.)

1. Konstruer Chow-Liu-træet C fra data.
2. Regn den estimerede *mutual information* for alle kombinationer af tre variable (tripler). Sorter triplerne efter aftagende *mutual information*.
3. Bestemmelse af første kirsebær:
   (a) Betragt det første trippel t og tilhørende kanter fra C.
   (b) Gør t komplet ved at tilføje kanter.
   (c) Hvis der kun tilføjes én kant, lad da denne graf være det første kirsebær i et t-kirsebærtræ T.
   (d) Ellers, lad t være næste trippel.
   (e) Gentag punkt (b) – (d), indtil et kirsebær accepteres.
   (f) Fjern t fra listen over tripler.
4. Gentag følgende til alle resterende knuder er tilføjet til T:
   (a) Betragt det første trippel t på listen og de tilhørende kanter fra C og T.
   (b) Hvis to af knuderne i t er i T:
      i. Gør t komplet.
      ii. Hvis kun én kant tilføjes:
         – Accepter det nye kirsebær ved at tilføje det til T.
         – Fjern alle tripler, som allerede er i t-kirsebærtræet.
   (c) Hvis kirsebærret ikke accepteres, sæt t til at være det næste trippel på listen.
   (d) Gentag punkt (b) – (c), indtil et kirsebær accepteres.

`Output`: Nabomatricen, klikerne, separatorene.

I stedet for at bestemme et t-kirsebærtræ ud fra Chow-Liu-træet er det også muligt at konstruere det direkte fra data. En mulig fremgangsmåde ville være at bestemme alle mulige t-kirsebærtræer og estimere (4.2.1) for dem alle, men igen ville det ikke være særlig effektivt. Derfor vælges i stedet en grådig fremgangsmåde. Vi har implementeret et bud på en sådan

algoritme i funktionen `tcherry_step`, og fremgangsmåden er beskrevet i Algoritme 4.2.2.

Algoritme 4.2.2 t-`kirsebærtræ konstrueret direkte fra data`

`tcherry_step(data, ...)`

(`data`: Et datasæt med realisationer af variablene, `...`: Øvrige argumenter til beregning af mutual information.)

1. Regn den estimerede *mutual information* for alle kombinationer af tre variable (tripler).
2. Lad triplet med højest *mutual information* være det første kirsebær.
3. Gentag følgende indtil alle knuder er i t-kirsebærtræet:
   (a) Gentag for alle knuder A_k, som endnu ikke er i t-kirsebærtræet:
      – Gentag for alle kanter $\{A_i, A_j\}$ i t-kirsebærtræet:
         i. Regn $MI(A_i, A_j, A_k) - MI(A_i, A_j)$ for kirsebærret $(\{A_i, A_j\}, A_k)$, og gem værdien.
   (b) Tilføj kirsebærret med den højeste værdi fra ii. til t-kirsebærtræet.

`Output`: Nabomatricen, vægt, klikerne, separatorene.

Næste afsnit indeholder simulationer til at sammenligne præstationen af disse algoritmer.

4.3 Simulationsstudie til vurdering af konstruktionsalgoritmer

Udgangspunktet i dette afsnit er at simulere data ud fra et t-kirsebærtræ. Til dette formål har vi implementeret funktionen `random_tcherry`, der kan generere et tilfældigt t-kirsebærtræ med n knuder. Dette gøres ved først at vælge et par af knuder uniformt blandt alle mulige og forbinde disse med en kant. Herefter vælges en ny knude uniformt blandt de resterende knuder, og denne sættes på en uniformt valgt kant i det foreløbige t-kirsebærtræ. Der fortsættes iterativt til det ønskede antal knuder er tilføjet.

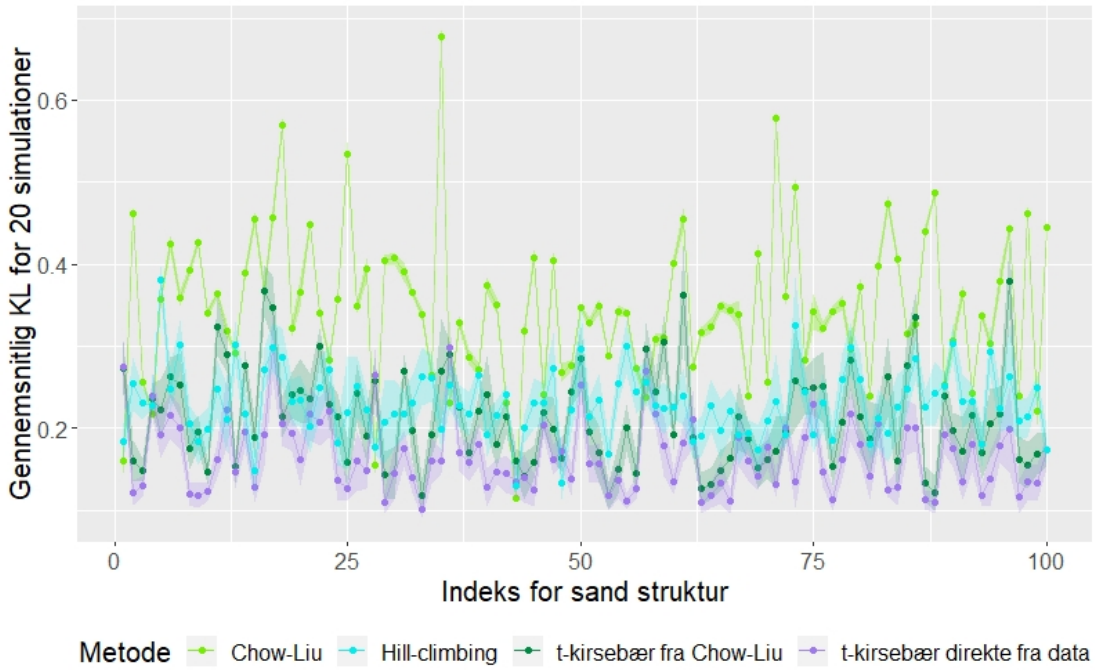
Denne funktion genererer også en sandsynlighedsfordeling tilhørende strukturen. Dette gøres ved at tage udgangspunkt i et bayesiansk netværk, som har t-kirsebærtræet som domænegraf. Den valgte struktur af dette har to knuder uden forældre, som er de først tilføjede knuder. Alle andre knuder har præcis to forældre, nemlig de to knuder, som de forbindes til, når de introduceres til t-kirsebærtræet. Sandsynlighedsfordelingen genereres herefter ved at specificere sandsynlighedsfordelingerne i Sætning 3.2.3. For forældreløse knuder

bestemmes fordelingen altid tilfældigt. For en knude A_i med forældre A_j og A_k bestemmes de enten tilfældigt eller, hvis argumentet `noise` er givet, ved følgende fremgangsmåde:

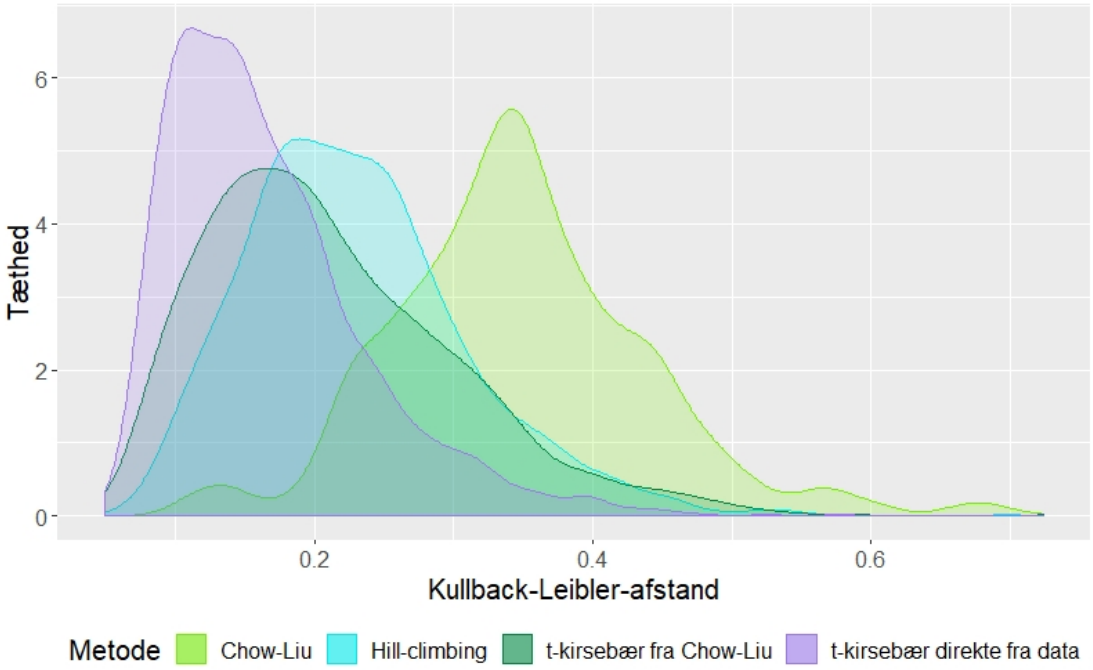
- Lad A_k have de mulige udfald $u_1, \ldots, u_m$, og bestem $P(A_i|A_j, A_k = u_1)$ tilfældigt.
- For $p = 2, \ldots, m$ bestem $P(A_i|A_j, A_k = u_p)$ ved på passende vis at normalisere $P(A_i|A_j, A_k = u_1) + \epsilon$, hvor $\epsilon \sim \text{Unif}(0, \texttt{noise})$.

For små værdier af `noise` gælder det, at $P(A_i|A_j, A_k) \approx P(A_i|A_j)$, hvilket muligvis giver Chow-Liu-træet en fordel i forhold til estimation. Det er for at undersøge dette nærmere, at denne mulighed er inkluderet.

I Figur 4.3.1 og 4.3.2 ses resultaterne af et simulationsstudie, hvor 100 tilfældige t-kirsebærtræer er genereret med tilfældige sandsynlighedsfordelinger, og for hvert af disse er der simuleret 20 datasæt med 500 observationer i hvert. Funktionerne `ChowLiu, tcherry_CL` og `tcherry_step` fra pakken `tcherry` samt `hc` fra pakken `bnlearn` (Scutari 2010) er herefter benyttet på disse datasæt for at finde strukturen. Disse strukturer er herefter benyttet til at give estimerede sandsynlighedsfordelinger, og kvaliteten af disse bestemmes ved Kullback-Leibler-afstanden til den sande fordeling. Dette er gjort med henholdsvis 5, 7 og 10 variable. Kun analysen for 5 variable er inkluderet i dette afsnit. For 7 og 10 variable se Bilag C.



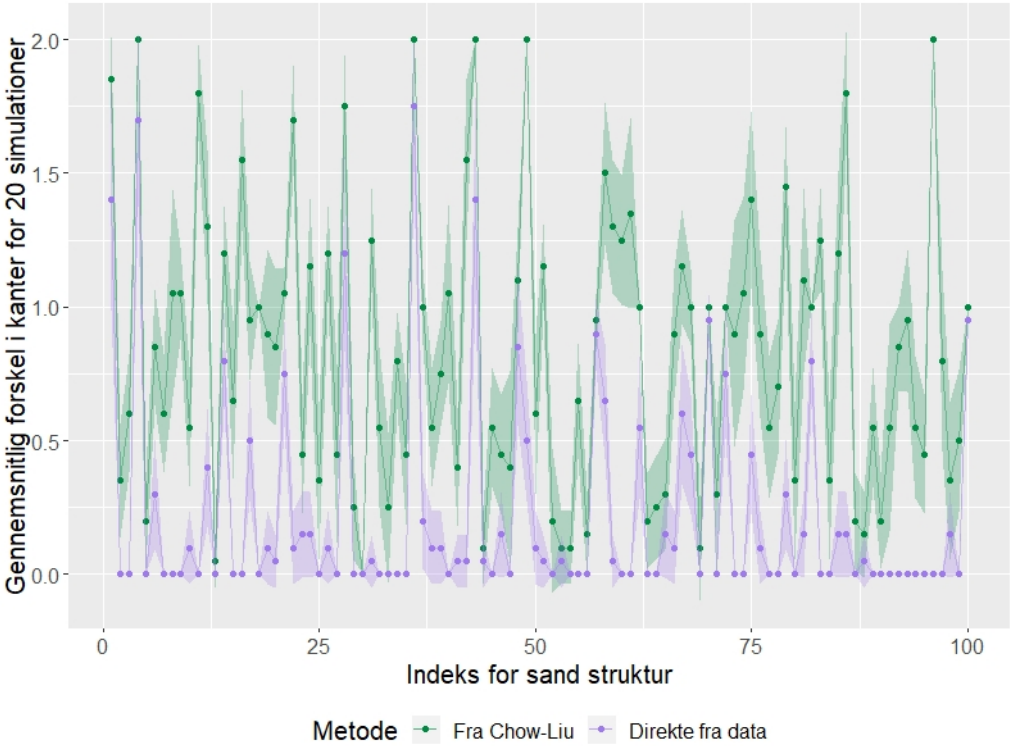
Figur 4.3.1: Gennemsnitlig Kullback-Leibler-afstand mellem estimeret og sand sandsynlighedsfordeling hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med 5 variable. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.



Figur 4.3.2: Estimeret tæthed for alle Kullback-Leibler-afstande mellem estimerede og sande sandsynlighedsfordelinger for 100 tilfældige t-kirsebærtræer med 5 variable. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Disse algoritmer er for hver sand struktur benyttet på 20 simulerede datasæt.

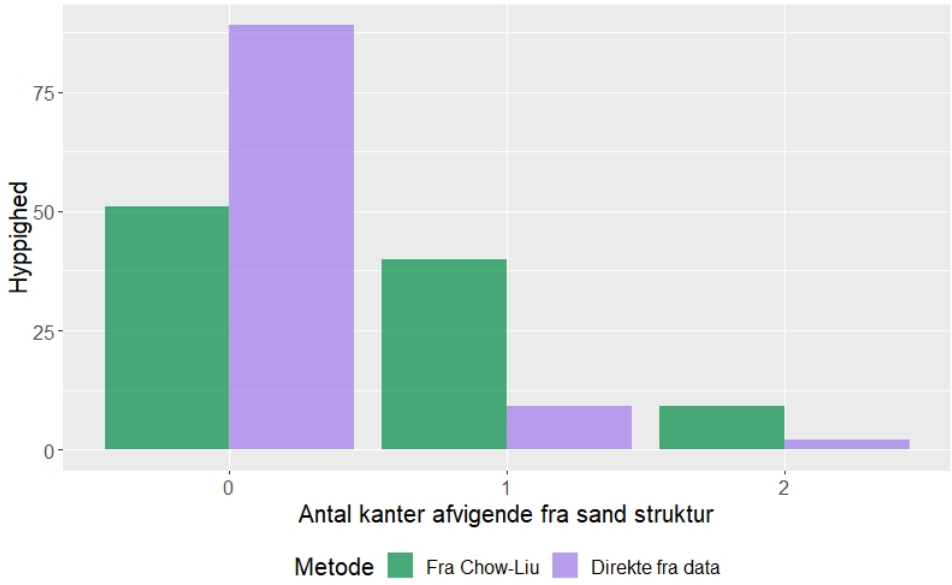
Det ses generelt, at Chow-Liu-træet giver den dårligste approksimation til den sande sandsynlighedsfordeling, men til gengæld varierer resultatet meget lidt for forskellige datasæt simuleret fra samme fordeling. De t-kirsebærtræer, der konstrueres direkte fra data, lader til at klare sig bedre end dem, der konstrueres fra Chow-Liu-træet. *Hill-climbing* ligger generelt på samme niveau som de t-kirsebærtræer, der konstrueres fra Chow-Liu-træet.

For de to t-kirsebærtræsimplementeringer kontrolleres yderligere, hvor mange kanter de har til forskel fra den sande struktur med funktionen `diff_edges_tch` også fra pakken `tcherry`. Dette antal kanter er valgt som det antal kanter, der ikke indgår i begge grafer, delt med to. Dette giver altså antal kanter, som er placeret forskelligt i de to t-kirsebærtræer. Resultatet af dette ses i Figur 4.3.3.



Figur 4.3.3: Gennemsnitlig antal fejlkanter mellem estimeret og sand struktur hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med 5 variable. Yderligere er der angivelse af et 95% konfidensinterval.

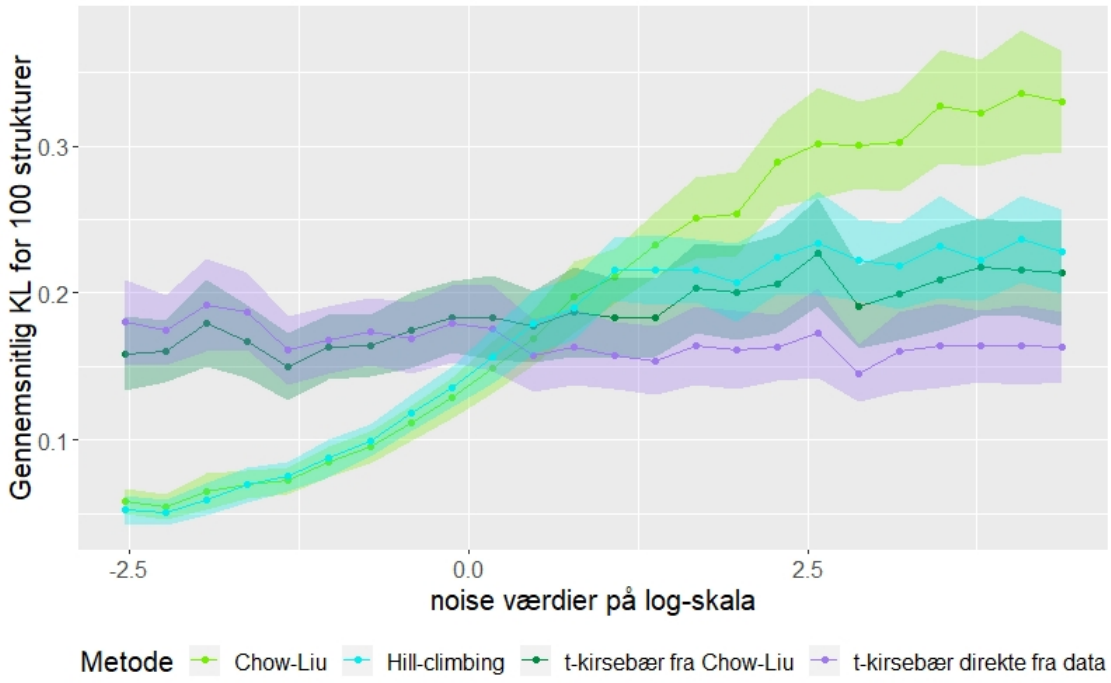
Det ses, at de t-kirsebærtræer, der konstrueres direkte fra data, generelt er bedre til at genfinde den sande struktur. Dette ses endnu tydeligere i Figur 4.3.4, hvor et barplot over det mest typiske antal fejlkanter for hver sand struktur ses.



Figur 4.3.4: Mest typiske antal fejlkanter for fundne strukturer hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med 5 variable.

For at undersøge hvordan metoderne klarer sig for forskellige værdier af `noise`, er et tilsvarende simulationsstudie lavet, hvor der dog for hver sand struktur kun simuleres ét datasæt, men til gengæld benyttes flere forskellige værdier af `noise` argumentet. Dette er kun gjort for 5 variable.

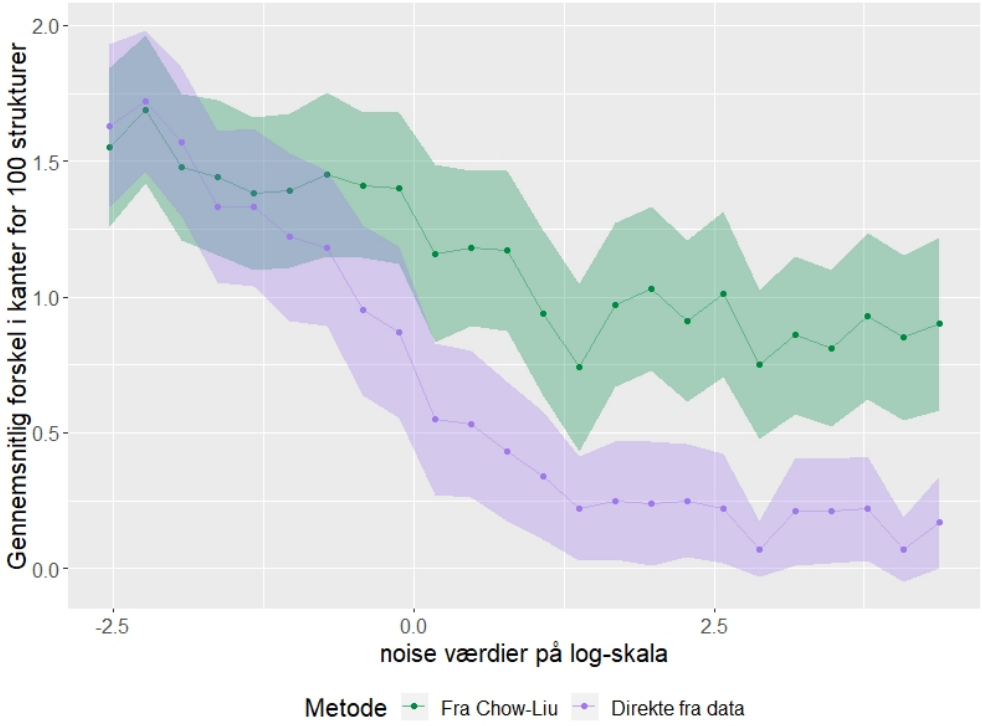
I Figur 4.3.5 ses gennemsnitlig Kullback-Leibler-afstand mellem den estimerede og sande fordeling hen over de 100 forskellige sande strukturer for forskellige værdier af `noise`.



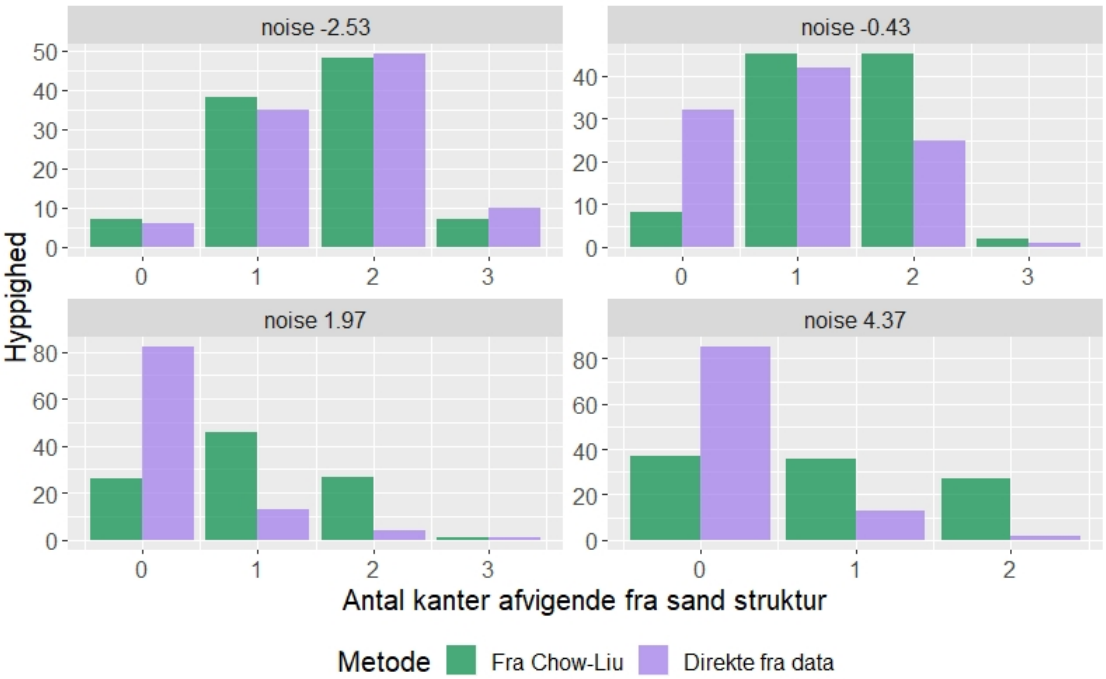
Figur 4.3.5: Gennemsnitlig Kullback-Leibler-afstand mellem estimeret og sand sandsynlighedsfordeling baseret på ét simuleret datasæt hen over 100 tilfældige t-kirsebærtræer med 5 variable og med forskellige `noise` værdier. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.

Det ses, at for små værdier af `noise` klarer Chow-Liu-træet og *hill-climbing* sig bedre, men i takt med at værdierne stiger, bliver approksimationen med Chow-Liu-træet gradvist dårlige, hvorimod *hill-climbing* stabiliserer sig omkring samme niveau som t-kirsebærtræet konstrueret ud fra Chow-Liu-træet. Til at starte med klarer de to t-kirsebærtræsimplementeringer sig lige godt, men senere begynder t-kirsebærtræet konstrueret direkte fra data at give en bedre approksimation.

I Figur 4.3.6 ses en sammenligning af, hvor godt de to t-kirsebærtræsimplementeringer genfinder den sande struktur. I Figur 4.3.7 ses barplot over antal fejlkanter i forhold til den sande struktur for de udvalgte værdier af `noise`: -2,53, -0,43, 1,97 og 4,37.



Figur 4.3.6: Gennemsnitlig forskel i kanter mellem estimeret og sand struktur baseret på ét simuleret datasæt hen over 100 tilfældige t-kirsebærtræer med 5 variable og forskellige `noise` værdier. Yderligere er der angivelse af et 95% konfidensinterval.



Figur 4.3.7: Antal fejlkanter for 100 forskellige estimerede strukturer med 5 variable baseret på ét simuleret datasæt med `noise` værdierne: -2,53, -0,43, 1,97 og 4,37.

Det ses, at t-kirsebærtræet konstrueret direkte fra data hurtigt bliver i stand til at finde den sande struktur i langt de fleste tilfælde, hvorimod det fra Chow-Liu-træet har langt sværere ved det.

Konklusionen på dette afsnit er, at Chow-Liu-træet bør benyttes frem for t-kirsebærtræerne, hvis der ikke er stærk tro på, at denne afhængighedsstruktur er utilstrækkelig. Når det kommer til at vælge mellem de to t-kirsebærtræsimplementeringer, klarer t-kirsebærtræet konstrueret direkte fra data sig generelt bedre både med hensyn til Kullback-Leibler-afstanden og med hensyn til at finde den sande struktur. Dog bør det bemærkes, at der kun er undersøgt det scenario, hvor den sande struktur er et t-kirsebærtræ.

Det er også værd at bemærke, at de forskellige strukturer sammenlignet i dette afsnit har forskellige antal parametre i form af estimerede sandsynligheder. Som for ethvert andet felt inden for statistik bør dette også tages i betragtning, når der tilpasses sådanne modeller til data i praksis. Det ønskes generelt, at kunne klare sig med en så simpel struktur som muligt, svarende til få parametre, for at undgå overtilpasning af modellerne. Problematikken om overtilpasning behandles nærmere i Afsnit 8.

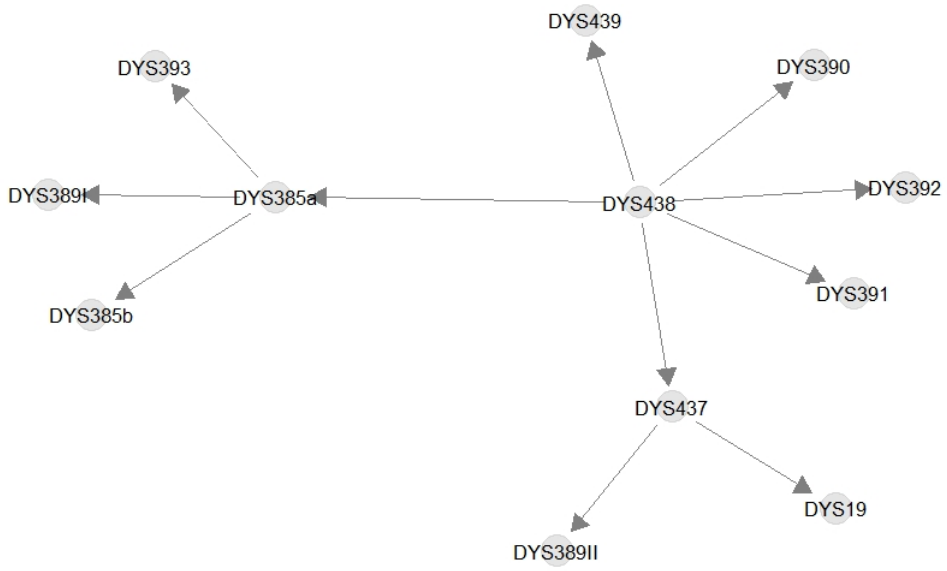
5 Anvendelse af t-kirsebærtræer, Chow-Liu-træer og hill-climbing på dansk Y-STR data

I Hallenberg et al. (2005) findes data med *Short Tandem Repeat* (STR) fra 11 loci[6] på Y-kromosomet fra 185 danske mænd, som ikke er beslægtet. Betegnelsen STR bruges om gentagelser af locusspecifikke basesekvenser. Data indeholder alleler for de 11 loci, hvor en allel angiver antal gentagelser, der er set for dette locus. En allel har et endeligt antal udfald, og data kan derfor behandles som kategorisk. For mere information om de mulige udfald se Tabel D.0.1. En kombination af alleler for alle 11 loci kaldes en haplotype. Data indeholder et locus kaldet $DYS385a/b$. $DYS385$ indgår i to områder på kromosomet, hvilket giver anledning til to alleler, derved referencen a/b, hvor a typisk tildeles den mindste allel. Det vides altså ikke, hvilken allel, der hører til hvor. I dette speciale vælges det at behandle $DYS385a$ og $DYS385b$ som to loci, som i Andersen et al. (2018). Det vides også, jævnfør Butler (2011: s. 377), at gentagelserne i $DYS389I$ også tæller med i $DYS389II$. Det vælges derfor at modificere $DYS389II$ ved at fratrække bidraget fra $DYS389I$.

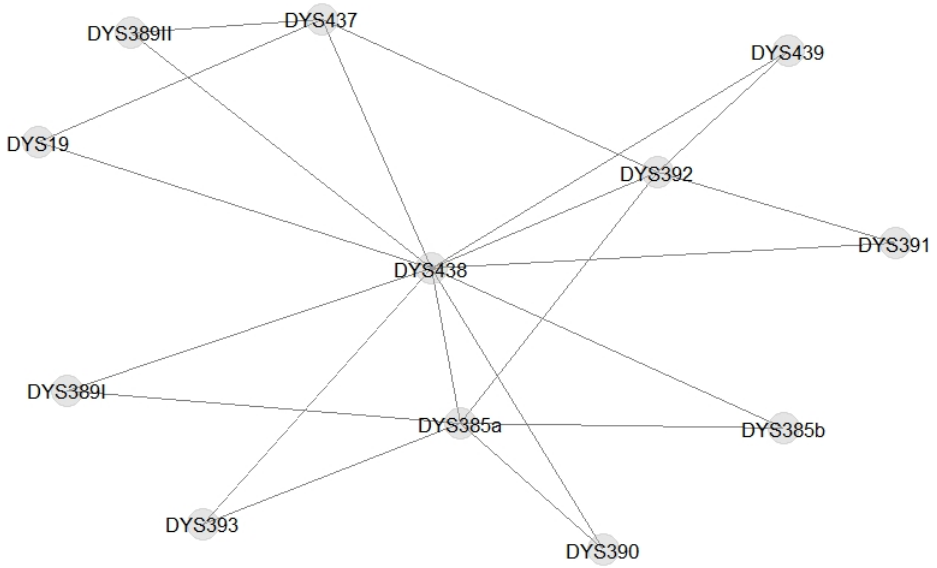
Det ønskes at tilpasse et Chow-Liu-træ og t-kirsebærtræer på dette data. Til dette formål benyttes funktionerne `ChowLiu`, `tcherry_CL` og `tcherry_step`, der findes i vores **R**-pakke `tcherry`. I forbindelse med udregningen af sandsynligheder estimeres alle sandsynligheder i dette afsnit med bayesiansk estimation, hvor $\alpha = 0{,}001$. Dette er nødvendigt for at undgå nul-sandsynligheder.

Funktionen `ChowLiu` giver mulighed for at vælge en rod tilfældigt eller at benytte en brugerspecifik rod. I dette speciale er det valgt at benytte roden $DYS438$ som i Andersen et al. (2018), og det resulterende bayesianske netværk ses i Figur 5.0.1a. Funktionen `tcherry_CL` giver mulighed for at bestemme strukturen af et t-kirsebærtræ ud fra Chow-Liu-træet via Algoritme 4.2.1. Det resulterende t-kirsebærtræ ses i Figur 5.0.1b.

[6] En locus betegner en bestemt placering på et kromosom.



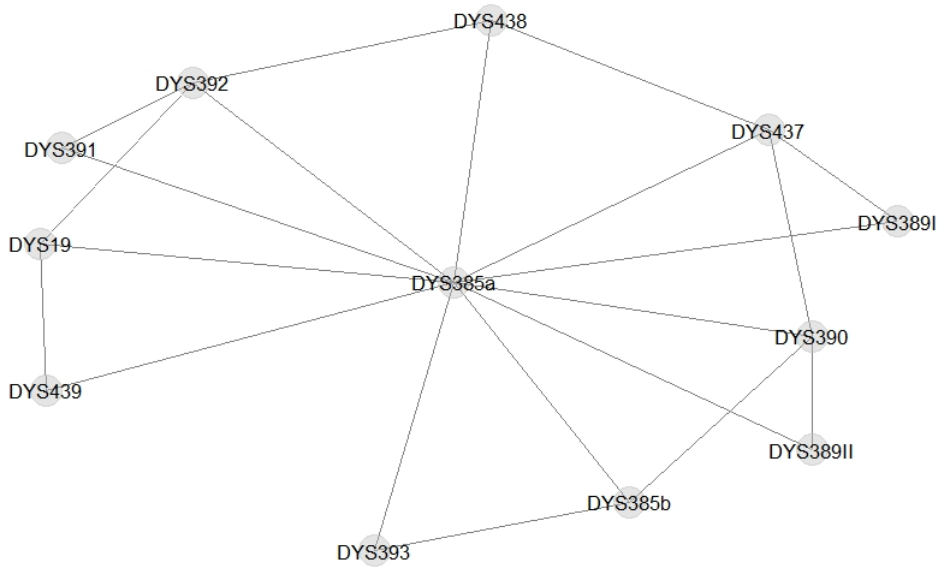
(a) Chow-Liu-træ for Y-STR datasættet.



(b) t-kirsebærtræ for Y-STR datasættet.

Figur 5.0.1: Chow-Liu-træ og t-kirsebærtræ konstrueret ud fra Chow-Liu-træet for Y-STR datasættet.

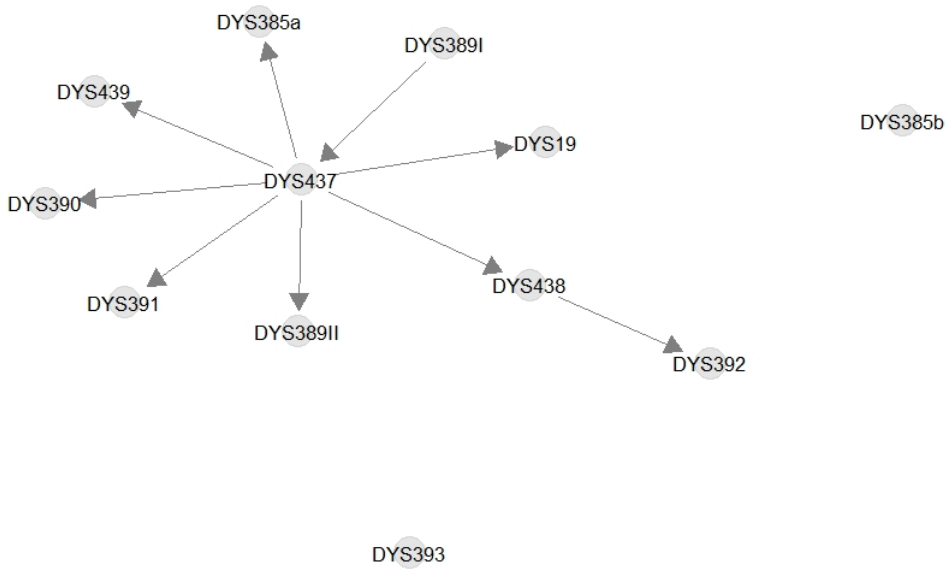
Hvis t-kirsebærtræet i stedet bestemmes direkte fra data med Algoritme 4.2.2 via funktionen `tcherry_step`, fås grafen i Figur 5.0.2.



Figur 5.0.2: t-kirsebærtræ konstrueret direkte fra Y-STR datasættet.

Dette giver en markant anderledes struktur, end når t-kirsebærtræet begrænses til at indeholde Chow-Liu-træet. Funktionen `diff_edges_tch`, fra pakken `tcherry` giver, at de to konstruerede t-kirsebærtræer har 12 kanter til forskel.

Udover disse metoder, der lægger en begrænsning på strukturen af netværket, er det også forsøgt at tilpasse et netværk med en fri struktur ved hjælp af Algoritme 3.6.1. Dette er gjort med funktionen `hc` fra pakken `bnlearn`. Dette giver netværket i Figur 5.0.3.

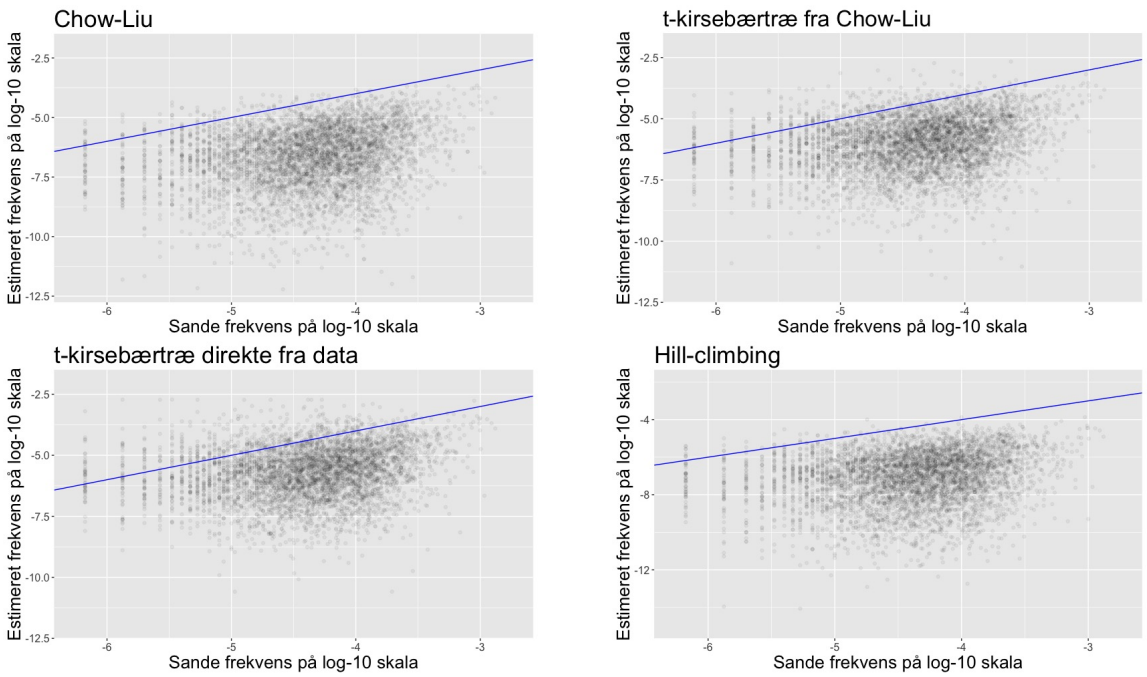


Figur 5.0.3: Bayesiansk netværk fundet med `hill-climbing` for Y-STR datasættet.

For at undersøge hvor anvendelige modellerne er til at approksimere frekvensen af haplotyper,

benyttes metoderne på simuleret data, hvor frekvensen af en bestemt haplotype er kendt. Andersen et al. (2018) har genereret simulationer af haplotyper, som kan bestemmes med udgangspunkt i de to kit Minimal og PowerPlexY (henholdsvis 9 og 12 loci). For disse to kits er der simuleret databaser af henholdsvis 500 og 1000 observationer. For hvert kit for hver størrelse er der genereret 5000 databaser. For en detaljeret beskrivelse af disse simulationer se Andersen et al. (2018).

For at bestemme den estimerede frekvens af den givne haplotype, med udgangspunkt i en grafisk model, indsættes haplotypen som information i netværket, hvorefter funktionen `pEvidence` fra pakken `gRain` (Højsgaard 2012) anvendes. Pakken `gRain` benytter metoderne beskrevet i Afsnit 3.3. I Figur 5.0.4 ses de estimerede frekvenser med henholdsvis Chow-Liu-træet, de to t-kirsebærtræer og *hill-climbing* mod de sande frekvenser for databaserne af størrelse 501 med kittet PowerPlexY. Det bemærkes, at den givne haplotype er inkluderet i denne database.



Figur 5.0.4: Sammenligning af sande frekvenser og forskellige metoders estimerede frekvenser af en given haplotype på simulerede databaser af størrelse 501 med kittet PowerPlexY. Den blå linje angiver identitetslinjen.

Det ses, at alle metoder generelt underestimerer den sande frekvens, men at t-kirsebærtræerne lader til at give en forbedring. Den generelle tendens til at underestimere kan skyldes, at estimatorerne er nødsaget til at tillægge sandsynlighedsmasse til haplotyper, som ikke kan observeres i virkeligheden (Andersen et al. 2018).

En grafisk sammenligning af estimaterne fra de forskellige modeller kan ses i Bilag E.

For at vurdere hvor godt modellerne estimerer den sande frekvens, udregnes fejlmålet

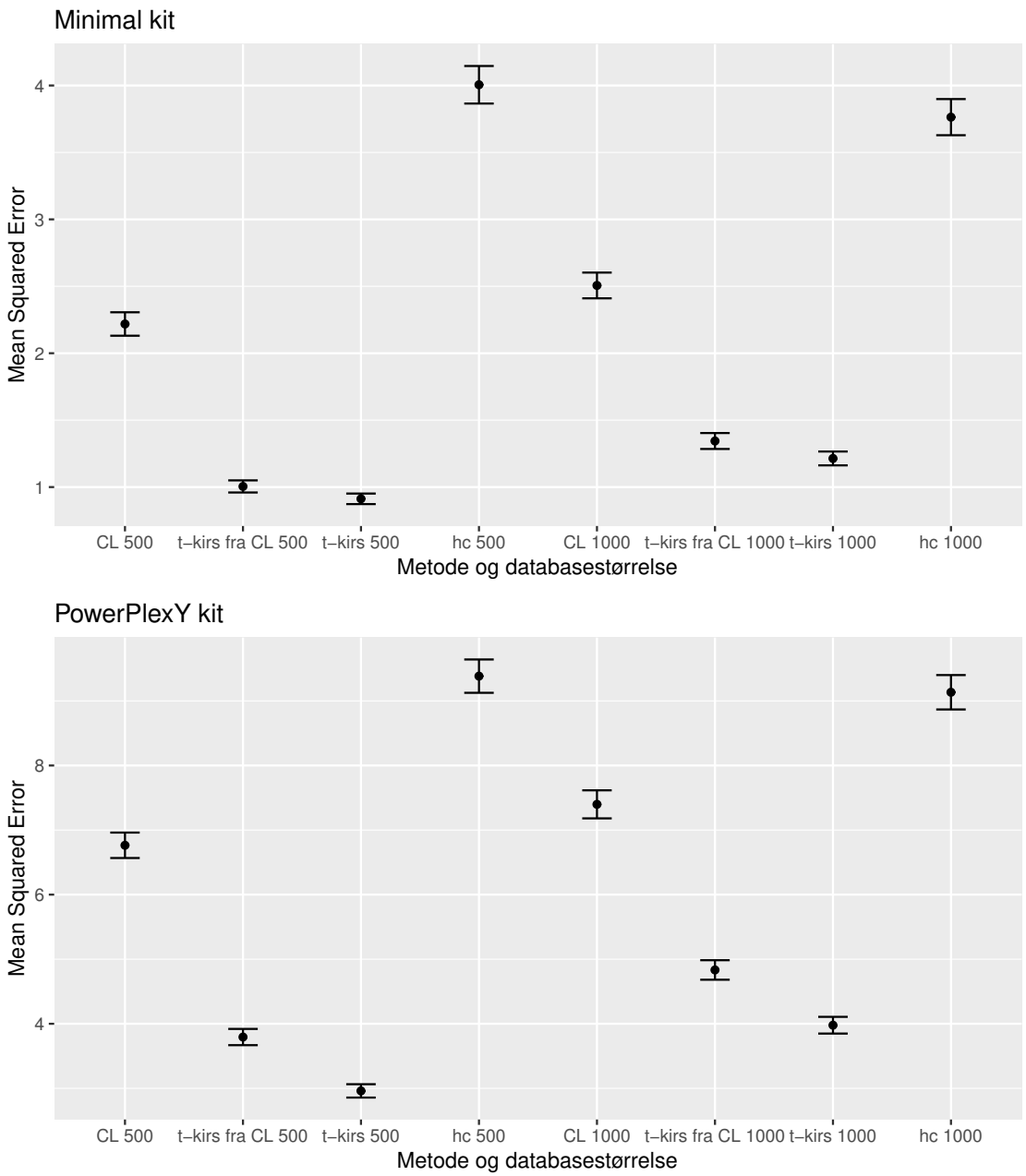
$$MSE(p, \hat{p}) = \frac{1}{n} \sum_{i=1}^{n} (\log_{10}(p_i) - \log_{10}(\hat{p}_i))^2,$$

hvor n er det samlede antal estimerede frekvenser, p_i er den sande frekvens med indeks i, og $\hat{p}_i$ er den estimerede frekvens. En sammenfatning af dette mål for de forskellige modeller og databasetyper ses i Tabel 5.0.1.

Model	Antal loci	Databasestørrelse	MSE
Chow-Liu	9	500	2,218
t-kirsebærtræ fra Chow-Liu	9	500	1,005
t-kirsebærtræ	9	500	0,912
Hill-climbing	9	500	4,006
Chow-Liu	9	1000	2,506
t-kirsebærtræ fra Chow-Liu	9	1000	1,344
t-kirsebærtræ	9	1000	1,214
Hill-climbing	9	1000	3,764
Chow-Liu	12	500	6,764
t-kirsebærtræ fra Chow-Liu	12	500	3,795
t-kirsebærtræ	12	500	2,961
Hill-climbing	12	500	9,384
Chow-Liu	12	1000	7,398
t-kirsebærtræ fra Chow-Liu	12	1000	4,833
t-kirsebærtræ	12	1000	3,978
Hill-climbing	12	1000	9,134

Tabel 5.0.1: MSE for estimerede frekvenser med Chow-Liu-træet, t-kirsebærtræer og *hill-climbing* for forskellige databasetyper.

Det ses, at t-kirsebærtræerne giver en forbedring i forhold til Chow-Liu-træet, der til gengæld klarer sig bedre end *hill-climbing*. Det ses også, at alle modeller klarer sig bedre, når der kun er målt på 9 loci end på 12. Dette hænger sammen med den tidligere bemærkning om, at urealistiske haplotyper tillægges sandsynlighedsmasse, men for 9 loci er der færre mulige haplotyper end med 12 loci. De udregnede MSE-værdier med tilhørende 95% konfidensintervaller er plottet i Figur 5.0.5.



Figur 5.0.5: Sammenligning af MSE-værdier for forskellige estimationsmetoder, databasestørrelser og kit.

Det ses, at de to metoder til at konstruere t-kirsebærtræer klarer sig næsten lige godt for 9 loci, hvorimod for 12 loci giver det en forbedring at konstruere t-kirsebærtræet direkte fra data i forhold til at konstruere det fra Chow-Liu-træet.

6 Højere ordens t-kirsebærtræer

Dette afsnit er baseret på Kovács & Szántai (2012).
Afsnit 4 introducerede konceptet t-kirsebærtræer, som gav anledning til at opnå en approksimation af en diskret sandsynlighedsfordeling kun ved brug af 2. og 3. ordens marginale fordelinger. Dette koncept generaliseres i dette afsnit til at opnå approksimationer med højere ordens fordelinger.

Først generaliseres definitionen af et t-kirsebærtræ til et k'te ordens t-kirsebærtræ.

Definition 6.0.1 (k'te ordens t-kirsebærtræ)
Et k'te ordens t-kirsebærtræ er en graf, der kan konstrueres på følgende måde:

1. Start med $k-1$ knuder, som udgør en komplet punktmængde. En sådan punktmængde kaldes en hyperkant i grafen. Dette er også det mindst mulige k'te ordens t-kirsebærtræ.

2. k'te ordens t-kirsebærtræet udvides med en ny knude ved at forbinde denne til alle knuder i en hyperkant i det eksisterende k'te ordens t-kirsebærtræ via $k-1$ nye kanter. De k involverede knuder danner et hyperkirsebær. Hvis den nye knude er A_k, og hyperkanten er $\{A_1, \ldots, A_{k-1}\}$, benævnes dette hyperkirsebær $(\{A_1, \ldots, A_{k-1}\}, A_k)$.

3. Punkt 2. gentages, til grafen har det ønskede antal knuder.

Bemærk, at for $k = 2$ reducerer Definition 6.0.1 til et træ, og for $k = 3$ svarer det til Definition 4.1.1 for et t-kirsebærtræ. Proposition 6.0.2 indeholder nogle nyttige egenskaber om k'te ordens t-kirsebærtræer.

Proposition 6.0.2
Lad G være et k'te ordens t-kirsebærtræ. Da gælder følgende:[a]

1. G har $n-(k-1)$ kliker, hvis G har mindst k knuder.

2. G har $1+(k-1)(n-(k-1))$ hyperkanter.

3. G har $(k-1)n - \frac{(k-1)k}{2}$ kanter.

[a] Dette er ikke et resultat fra Kovács & Szántai (2012), men vores eget resultat med bevis.

Bevis. For at vise punkt 1. bemærk, at der dannes en klike hver gang punkt 2. i Definition 6.0.1 benyttes. For at opnå det rigtige antal knuder sker dette $n-(k-1)$ gange.

For at vise punkt 2. bemærkes det, at når punkt 1. i Definition 6.0.1 benyttes, dannes én hyperkant. Det bestemmes nu, hvor mange nye hyperkanter punkt 2. i Definition 6.0.1 giver. Enhver delmængde af $\{A_1, \ldots, A_k\}$ med $k-1$ knuder og A_k i mængden udgør en ny hyperkant. Der er $k-1$ sådanne delmængder. Derfor er der i alt $1+(k-1)(n-(k-1))$ hyperkanter.

For at vise punkt 3. bemærk, at når punkt 1. i Definition 6.0.1 benyttes, dannes $\frac{(k-1)(k-2)}{2}$ kanter, og hver gang punkt 2. i Definition 6.0.1 benyttes, dannes $k-1$ kanter. Derfor bliver antal kanter i alt

$$\begin{aligned}\frac{(k-1)(k-2)}{2}+(k-1)(n-(k-1)) &= \frac{(k-1)(k-2)}{2}+(k-1)n-(k-1)(k-1)\\ &= \frac{(k-1)(k-2-2k+2)}{2}+(k-1)n\\ &= (k-1)n-\frac{(k-1)k}{2}.\end{aligned}$$

■

Ligesom for $k=3$ er k'te ordens t-kirsebærtræer triangulære.

Proposition 6.0.3

Et k'te ordens t-kirsebærtræ er en trianguleret graf.[a]

[a] Dette er ikke et resultat fra Kovács & Szántai (2012), men vores eget resultat med bevis.

Bevis. Den sidst tilføjede knude er grundet Definition 6.0.1 en simpel knude. Eliminer derfor denne. Den næstsidst tilføjede knude er nu simpel i den resulterende graf igen per konstruktion. Eliminer derfor denne. Fortsæt iterativt med at eliminere simple knuder svarende til den nye knude tilføjet i punkt 2. i Definition 6.0.1, indtil den første hyperkant nås. Denne punktmængde er per definition komplet, så alle knuder er simple og kan elimineres i vilkårlig rækkefølge. Dette giver en perfekt eliminationsrækkefølge, hvorfor grafen er triangulær.

■

I det efterfølgende ønskes det at benytte et k'te ordens t-kirsebærtræ som grafen i et Markovnetværk. For at bestemme den approksimerede sandsynlighedsfordeling i (4.0.1) betragtes et tilslutningstræ for et k'te ordens t-kirsebærtræ. Da k'te ordens t-kirsebærtræer er triangulære, kan Algoritme 3.4.10 benyttes til at organisere klikerne i et tilslutningstræ. Ved denne fremgangsmåde ses det (gense eventuelt Eksempel 4.1.3), at grafens kliker bliver hyperkirsebærrene.

Algoritmen giver også, at for en klike K_i, $i < n$, bestående af knuderne i hyperkirsebærret $(\{A_1, \ldots, A_{k-1}\}, A_k)$ er separatoren S_i hyperkanten $\{A_1, \ldots, A_{k-1}\}$. Dette betyder, at et k'te ordens t-kirsebærtræ over n variable giver anledning til sandsynlighedsfordelingen

$$P(A_1, \ldots, A_n) = \frac{\prod\limits_{(\{A_1,\ldots,A_{k-1}\},A_k)\in\mathcal{K}} P(A_1, \ldots, A_k)}{\prod\limits_{\{A_1,\ldots,A_{k-1}\}=S\in\mathcal{S}} P(A_1, \ldots, A_{k-1})^{\#_S}},$$

hvor $\#_S$ angiver antal gange separatoren indgår i tilslutningstræet. Bemærk, at alle kliker har størrelse k, og alle separatorer i tilslutningstræer har størrelse $k-1$. Proposition 6.0.4 giver med udgangspunkt i dette en ækvivalent betingelse til at være et k'te ordens t-kirsebærtræ.

Proposition 6.0.4
En graf er et k'te ordens t-kirsebærtræ, hvis og kun hvis klikerne kan organiseres i et tilslutningstræ, hvor alle kliker har størrelse k og alle separatorer størrelse $k-1$.[a]

[a] Dette er ikke et resultat fra Kovács & Szántai (2012), men vores eget resultat med bevis.

Bevis. Det er allerede vist, at hvis grafen er et k'te ordens t-kirsebærtræ, kan klikerne organiseres i et tilslutningstræ, og kliker samt separatorer har den angivne størrelse.

Det vises nu, at et tilslutningstræ T over variablene $A_1, \ldots, A_n$, hvor alle kliker har størrelse k, og alle separatorer har størrelse $k-1$, svarer til et tilslutningstræ for et k'te ordens t-kirsebærtræ. Bemærk, at grafen for tilslutningstræet kan gendannes ved at genskabe alle kliker i grafen. Start derfor med en graf G med knuderne $A_1, \ldots, A_n$ og ingen kanter. Vælg en klike R i T som rod, og gør R komplet i G. Dette konstruerer det første hyperkirsebær. Betragt en naboklike K til R i T og deres separator S. Da $|R| = |K| = k$ og $|S| = k-1$ er $|K \setminus S| = 1$. Lad $A_i = K \setminus S$. Gør K komplet i G, og bemærk, at alle variable i S allerede er forbundet i G, da $S \subseteq R$. S udgør derfor en eksisterende hyperkant i grafen, og de nye kanter svarer til at forbinde A_i til denne hyperkant. Det kan derfor betragtes som en anvendelse af punkt 2. i Definition 6.0.1. Der fortsættes iterativt med at behandle kliker i T, som er naboer til allerede genskabte kliker, hvilket kan betragtes som iterativt at anvende punkt 2. i Definition 6.0.1. Derfor er den resulterende graf et k'te ordens t-kirsebærtræ.

■

6.1 Klassen af k'te ordens tilslutningstræer

Dette afsnit er baseret på Kovács & Szántai (2012: Kap 3.1-3.3).
Et tilslutningstræ for et k'te ordens t-kirsebærtræ er et specialtilfælde af det, der kaldes et k'te ordens tilslutningstræ.

Definition 6.1.1 (k'te ordens tilslutningstræ)
Et k'te ordens tilslutningstræ er et tilslutningstræ, hvis største klike har k knuder.

Bemærk, at et k'te ordens tilslutningstræ giver anledning til en approksimation af sandsynlighedsfordelingen, der kun benytter marginale fordelinger af k eller færre variable. Senere bevises det, at det k'te ordens tilslutningstræ, som giver den bedste approksimation til den sande sandsynlighedsfordeling, er et tilslutningstræ for et k'te ordens t-kirsebærtræ. Kvaliteten af en approksimation måles som altid med Kullback-Leibler-afstanden, så denne bestemmes først for et k'te ordens tilslutningstræ i Sætning 6.1.2.

Sætning 6.1.2
Lad T være et k'te ordens tilslutningstræ over variablene $A = (A_1, \ldots, A_n)$. Betegn mængden af kliker med $\mathcal{K}$ og mængden af separatorer med $\mathcal{S}$. Lad $MI()$, være givet ved (B.0.7). Da er Kullback-Leibler-afstanden mellem den sande og den approksimerede simultane sandsynlighedsfordeling for A givet ved

$$KL(P, P^T) = -H(A) - \left(\sum_{K \in \mathcal{K}} MI(K) - \sum_{S \in \mathcal{S}} \#_S \cdot MI(S) \right) + \sum_{k=1}^{n} H(A_k), \tag{6.1.1}$$

hvor P betegner den sande sandsynlighedsfordeling, og P^T betegner sansynlighedsfordelingen approksimeret ud fra T.

Bevis. Beviset er fuldstændig i tråd med beviset for Sætning 4.1.5 bortset fra, at antallet af knuder i kliker og separatorer ikke eksplicit er angivet. Udregningerne gentages derfor ikke her.

■

Som i Afsnit 4 bemærkes det, at det eneste i (6.1.1), som afhænger af grafens struktur, er

$$\sum_{K \in \mathcal{K}} MI(K) - \sum_{S \in \mathcal{S}} \#_S \cdot MI(S), \tag{6.1.2}$$

hvilket kaldes tilslutningstræet eller strukturens vægt. For at minimere Kullback-Leibler-afstanden maksimeres tilslutningstræets vægt. I praksis kendes de involverede sandsynlighedsfordelinger dog ikke og må derfor estimeres. Hvis dette gøres med maksimum likelihood estimation, giver Sætning 6.1.3, at hvis den estimerede vægt maksimeres, opnås en struktur af maksimal likelihood.

Sætning 6.1.3
Lad T være et k'te ordens tilslutningstræ for variablene $A = (A_1, \ldots, A_n)$. Da er log-likelihoodfunktionen for det tilsvarende Markovnetværk M^T med de indsatte maksimum likelihood estimater for sandsynlighedsfordelingerne

$$l(M^T(\hat{\theta}^L); \mathcal{D}) = |\mathcal{D}| \left(\sum_{K \in \mathcal{K}} \widehat{MI}(K) - \sum_{S \in \mathcal{S}} \#_S \cdot \widehat{MI}(S) + \sum_{i=1}^{n} \widehat{H}(A_i) \right),$$

hvor $\mathcal{K}$ og $\mathcal{S}$ er henholdsvis mængden af kliker og separatorer, og $\mathcal{D}$ er en mængde af observeret data, hvor det antages, at observationerne er uafhængige.[a]

[a] Dette er ikke et resultat fra Kovács & Szántai (2012), men vores eget resultat med bevis.

Bevis. Likelihoodfunktionen for Markovnetværket M^T er givet ved

$$L(M^T(\hat{\theta}^L); \mathcal{D}) = \prod_{\mathcal{D}} \frac{\prod_{K \in \mathcal{K}} P^L(K)}{\prod_{S \in \mathcal{S}} P^L(S)^{\#_S}}.$$

Bemærk, at i brøken sættes variablene lig med den aktuelle observation fra datasættet. Dermed bliver log-likelihoodfunktionen

$$l(M^T(\hat{\theta}^L); \mathcal{D}) = \sum_{\mathcal{D}} \left(\sum_{K \in \mathcal{K}} \log\left(P^L(K)\right) - \sum_{S \in \mathcal{S}} \#_S \cdot \log\left(P^L(S)\right) \right).$$

I stedet for at summe over observationerne i datasættet er det tilstrækkeligt at summe over alle mulige konfigurationer af data og tælle antal gange, de indgår i datasættet. En konfiguration a af A indgår $|\mathcal{D}|P^L(a)$ gange. Da fås følgende, hvor en sum over en mængde af variable er summen over variablenes mulige tilstande som sædvanligt.

$$l(M^T(\hat{\theta}^L); \mathcal{D}) = |\mathcal{D}| \left(\sum_{A} P^L(A) \left(\sum_{K \in \mathcal{K}} \log\left(P^L(K)\right) - \sum_{S \in \mathcal{S}} \#_S \cdot \log\left(P^L(S)\right) \right) \right). \tag{6.1.3}$$

Som i beviset for Sætning 4.1.5 gælder det, at

$$\prod_{K \in \mathcal{K}} \prod_{B \in K} P^L(B) = \prod_{S \in \mathcal{S}} \left(\prod_{B \in S} P^L(B) \right)^{\#_S} \prod_{i=1}^{n} P^L(A_i).$$

Ved at tage logaritmen fås

$$\sum_{K \in \mathcal{K}} \log\left(\prod_{B \in K} P^L(B) \right) = \sum_{S \in \mathcal{S}} \#_S \cdot \log\left(\prod_{B \in S} P^L(B) \right) + \sum_{i=1}^{n} \log\left(P^L(A_i)\right).$$

Dette udtryk trækkes fra og lægges til på passende vis i (6.1.3), hvilket giver

$$
\begin{aligned}
&l(M^T(\hat{\theta}^L);\mathcal{D}) \\
&= |\mathcal{D}|\Bigg(\sum_A P^L(A)\Bigg(\sum_{K\in\mathcal{K}} \log\left(\frac{P^L(K)}{\prod_{B\in K} P^L(B)}\right) - \sum_{S\in\mathcal{S}} \#_S \cdot \log\left(\frac{P^L(S)}{\prod_{B\in S} P^L(B)}\right) \\
&\quad + \sum_{i=1}^n \log\Big(P^L(A_i)\Big)\Bigg)\Bigg) \\
&= |\mathcal{D}|\Bigg(\sum_A P^L(A) \sum_{K\in\mathcal{K}} \log\left(\frac{P^L(K)}{\prod_{B\in K} P^L(B)}\right) - \sum_A P^L(A) \sum_{S\in\mathcal{S}} \#_S \cdot \log\left(\frac{P^L(S)}{\prod_{B\in S} P^L(B)}\right) \\
&\quad + \sum_A P^L(A) \sum_{i=1}^n \log\Big(P^L(A_i)\Big)\Bigg) \\
&= |\mathcal{D}|\Bigg(\sum_{K\in\mathcal{K}} \sum_K \sum_{A\setminus K} P^L(A) \log\left(\frac{P^L(K)}{\prod_{B\in K} P^L(B)}\right) - \sum_{S\in\mathcal{S}} \#_S \cdot \sum_S \sum_{A\setminus S} P^L(A) \log\left(\frac{P^L(S)}{\prod_{B\in S} P^L(B)}\right) \\
&\quad + \sum_{i=1}^n \sum_{A_i} \sum_{\substack{A_j \\ j\neq i}} P^L(A) \log\big(P^L(A_i)\big)\Bigg) \\
&= |\mathcal{D}|\Bigg(\sum_{K\in\mathcal{K}} \sum_K P^L(K) \log\left(\frac{P^L(K)}{\prod_{B\in K} P^L(B)}\right) - \sum_{S\in\mathcal{S}} \#_S \cdot \sum_S P^L(S) \log\left(\frac{P^L(S)}{\prod_{B\in S} P^L(B)}\right) \\
&\quad + \sum_{i=1}^n \sum_{A_i} P^L(A_i) \log\big(P^L(A_i)\big)\Bigg) \\
&= |\mathcal{D}|\Bigg(\sum_{K\in\mathcal{K}} \widehat{MI}(K) - \sum_{S\in\mathcal{S}} \#_S \cdot \widehat{MI}(S) - \sum_{i=1}^n \widehat{H}(A_i)\Bigg),
\end{aligned}
$$

hvilket viser resultatet.

■

I Algoritme 6.1.4 beskrives en algoritme, der ud fra et k'te ordens tilslutningstræ konstruerer et tilslutningstræ for et k'te ordens t-kirsebærtræ, og dette har enten samme eller højere vægt end det oprindelige k'te ordens tilslutningstræ, hvilket vises i Sætning 6.1.7.

Algoritme 6.1.4 `Tilslutningstræ for et` k`'te ordens` t`-kirsebærtræ konstrueret ud fra et` k`'te ordens tilslutningstræ`

`Input`: k'te ordens tilslutningstræ T.

1. Vælg en klike i T med k knuder som rod. Kald denne klike R.

2. For alle stier på formen $R, S_1, K_1, \ldots, S_q, K_q$ fra roden R til et blad K_q med naboseparator S_q:

   (a) Sæt $i = 1$ og $K_{i_0} = R$.
   (Indekset i angiver hvilken klike på stien, der arbejdes med, og K_{i_0} opdateres løbende, så det altid er kliken før K_i på stien.)

   (b) Hvis $|K_i| < k$, eller $|S_i| < k - 1$:

   - Lad kliken $K_{i_0} = \{A_1, \ldots, A_p, \ldots, A_k\}$.
   - Lad separatoren $S_i = \{A_1, \ldots, A_p\}$.
   - Lad kliken $K_i = \{A_1, \ldots, A_p, A_{k+1}, \ldots, A_{k+m}\}$.
   - Fjern K_i og S_i fra T.
   - For $j = 1, \ldots, m$:
     - Dan kliken $K_{i_j} = \{A_1, \ldots, A_p, \ldots, A_{k-j}, A_{k+1}, \ldots, A_{k+j}\}$ og forbind den til $K_{i_{j-1}}$ med separatoren $S_{i_j} = K_{i_j} \setminus A_{k+j}$.
   - Forbind alle K_i's naboer længere nede i træet til K_{i_m} via deres separatorer til K_i.
     (Bemærk, at for en naboklike K med separator S til K_i gælder det at $S = K \cap K_{i_m}$. Dette følger af, at $K_{i_m} = \{A_1, \ldots, A_p, \ldots, A_{k-m}, A_{k+1}, \ldots, A_{k+m}\}$, så $K_i \subseteq K_{i_m}$. Dermed gælder det nødvendigvis, at $S \subseteq K \cap K_{i_m}$. Det eneste, der ville forhindre lighed, er hvis $(K_{i_m} \setminus K_i) \cap K \neq \emptyset$. Dette er dog ikke muligt, da $K_{i_m} \setminus K_i \subseteq K_{i_0}$, og hvis en knude både ligger i K og K_{i_0}, ligger den også i K_i grundet RIP-egenskaben, da K_i befinder sig på en sti mellem disse.)
   - Sæt $K_{(i+1)_0} = K_{i_m}$.

   (c) Ellers, sæt $K_{(i+1)_0} = K_i$.

   (d) Sæt $i = i + 1$.

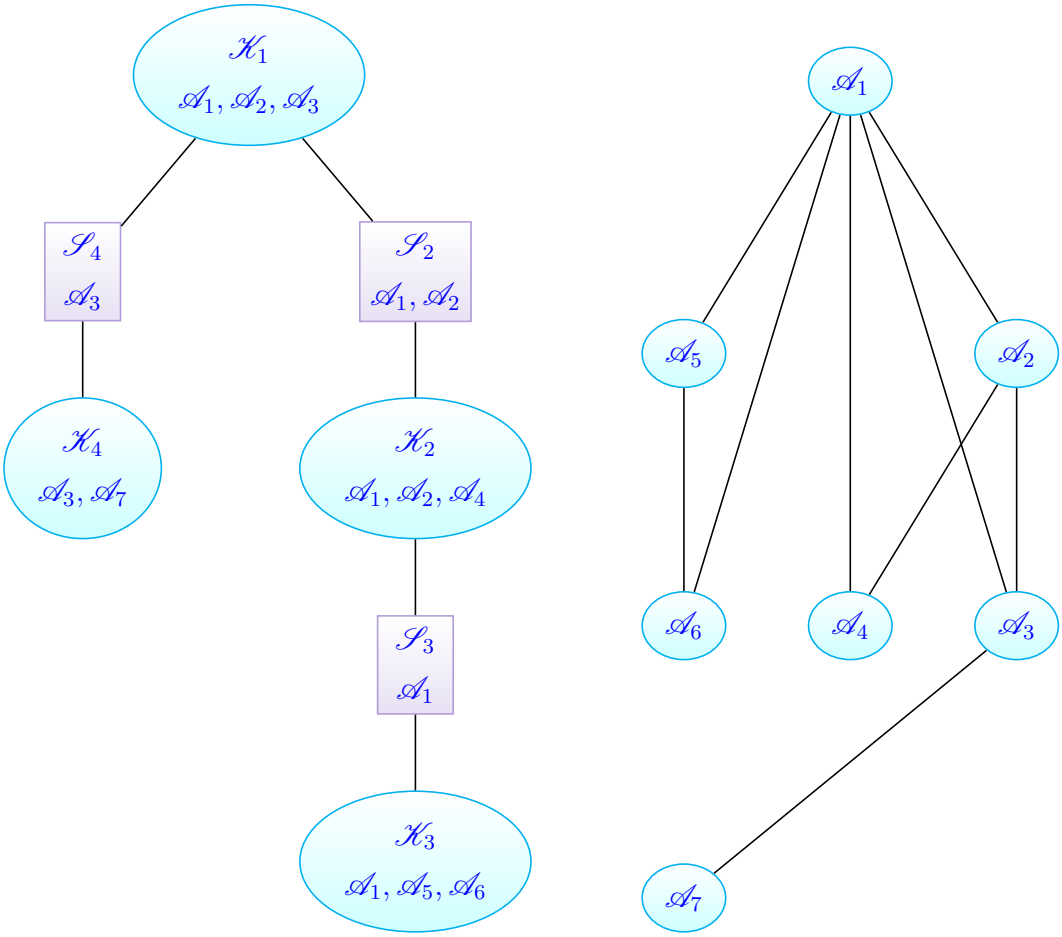
   (e) Gentag punkt (c) – (e) så længe $i \leq q$.

`Output`: Tilslutningstræ for et k'te ordens t-kirsebærtræ.

Et eksempel på anvendelsen af Algoritme 6.1.4 ses i Eksempel 6.1.5.

Eksempel 6.1.5

Betragt det 3. ordens tilslutningstræ med tilhørende graf i Figur 6.1.1.



(a) 3. ordens tilslutningstræ. (b) Tilhørende graf.

Figur 6.1.1: Et eksempel på et 3. ordens tilslutningstræ og den tilhørende graf.

Algoritme 6.1.4 benyttes nu for dette eksempel. Som rod vælges $R = \mathscr{K}_1$. Først betragtes stien

$$R, S_1 = \mathscr{S}_2, K_1 = \mathscr{K}_2, S_2 = \mathscr{S}_3, K_2 = \mathscr{K}_3.$$

Det ses, at $|K_1| = 3$ og $|S_1| = 2$, så punkt (c) igangsættes ikke. Sæt $K_{2_0} = K_1$. Det ses, at $|K_2| = 3$, men $|S_2| = 1$, så punkt (c) igangsættes. Her er $p = 1$ og $m = 2$. Sæt

$$A_1 = \mathscr{A}_1, \quad A_2 = \mathscr{A}_2, \quad A_3 = \mathscr{A}_4, \quad A_4 = \mathscr{A}_5 \text{ og } A_5 = \mathscr{A}_6.$$

Dan klikerne og separatorerne

$$\begin{aligned} K_{2_1} &= \{A_1, A_2, A_4\} = \{\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_5\} := \mathscr{K}_{3_1} \\ S_{2_1} &= \{A_1, A_2\} = \{\mathscr{A}_1, \mathscr{A}_2\} := \mathscr{S}_{3_1} \\ K_{2_2} &= \{A_1, A_4, A_5\} = \{\mathscr{A}_1, \mathscr{A}_5, \mathscr{A}_6\} := \mathscr{K}_{3_2} \\ S_{2_2} &= \{A_1, A_4\} = \{\mathscr{A}_1, \mathscr{A}_5\} := \mathscr{S}_{3_2}. \end{aligned}$$

Disse sættes på træet som vist i Figur 6.1.2a. Dette færdiggør denne sti, og der fokuseres nu på stien

$$R, S_1 = \mathscr{S}_4, K_1 = \mathscr{K}_4.$$

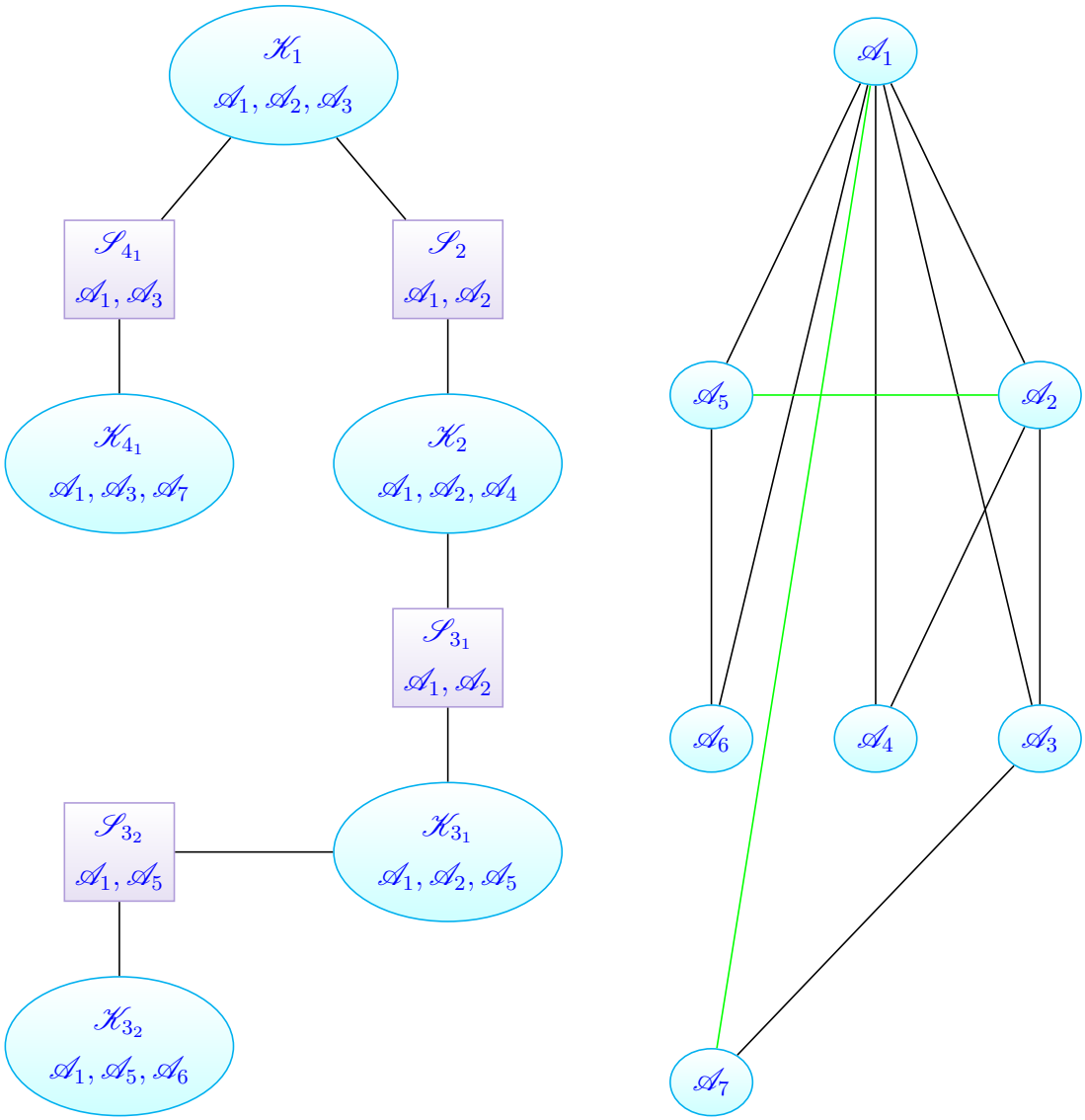
I dette tilfælde er der hverken tilstrækkeligt med knuder i kliken eller separatoren, så punkt (c) igangsættes. Her er $p = 1$ og $m = 1$. Sæt

$$A_1 = \mathscr{A}_3, \quad A_2 = \mathscr{A}_1, \quad A_3 = \mathscr{A}_2 \ \text{ og } \ A_4 = \mathscr{A}_7.$$

Dan kliken og separatoren

$$\begin{aligned} K_{1_1} &= \{A_1, A_2, A_4\} = \{\mathscr{A}_3, \mathscr{A}_1, \mathscr{A}_7\} := \mathscr{K}_{4_1} \\ S_{1_1} &= \{A_1, A_2\} = \{\mathscr{A}_3, \mathscr{A}_1\} := \mathscr{S}_{4_1}. \end{aligned}$$

Disse sættes på træet som vist i Figur 6.1.2a. Dette færdiggør proceduren. I Figur 6.1.2b ses den tilhørende graf for det resulterende tilslutningstræ.



(a) Tilslutningstræ for et 3. ordens t-kirsebærtræ konstrueret ud fra tilslutningstræet i Figur 6.1.1a via Algoritme 6.1.4.

(b) Tilhørende 3. ordens t-kirsebærtræ, hvor de grønne kanter angiver de nye kanter i forhold til grafen i Figur 6.1.1b.

Figur 6.1.2: Eksempel på anvendelsen af Algoritme 6.1.4 på tilslutningstræet i Figur 6.1.1a.

Det ses ved at sammenligne graferne i Figur 6.1.1b og 6.1.2b, at proceduren kun tilføjer kanter til den oprindelige graf.

Proposition 6.1.6 giver, at Algoritme 6.1.4 giver den ønskede struktur.

Proposition 6.1.6

Algoritme 6.1.4 returnerer et tilslutningstræ for et k'te ordens t-kirsebærtræ.[a]

[a] Dette er ikke et resultat fra Kovács & Szántai (2012), men vores eget resultat med bevis.

Bevis. Først vises det, at klikerne og separatorerne har den rigtige størrelse i forhold til et k'te ordens t-kirsebærtræ. I Algoritme 6.1.4 bliver alle kliker og separatorer af forkert størrelse erstattet med m nye i punkt (c). Bemærk, at i dette punkt er $|K_{i_j}| = k$ og $|S_{i_j}| = k - 1$, så alle kliker og separatorer har den rette størrelse i det resulterende træ.

Hvis det nu kan vises, at Algoritme 6.1.4 returnerer et tilslutningstræ, giver Proposition 6.0.4, at den tilhørende graf til tilslutningstræet er et k'te ordens t-kirsebærtræ, hvilket viser resultatet.

Det er tilstrækkeligt at vise, at den resulterende graf stadig er et tilslutningstræ efter én gennemgang af punkt (c). Det bevises derfor, at dette stadig opfylder RIP-egenskaben. Lad derfor T være et tilslutningstræ og $\widetilde{T}$ træet opnået efter én gennemgang af punkt (c). Det ønskes at vise, at RIP-egenskaben gælder for $\widetilde{T}$ ved at vise, at hvis en knude ligger i to kliker, ligger den også i alle kliker på en sti mellem klikerne. Dette bevises ved at betragte følgende tilfælde:

1. Antag først, at stien i $\widetilde{T}$ også er en sti i T. Da opfylder den RIP-egenskaben, da T er et tilslutningstræ.

2. Benyt notationen i punkt (c), og antag næst, at stien i $\widetilde{T}$ er på formen

$$K_{i_r}, K_{i_{r+1}}, \ldots, K_{i_{r+u}},$$

hvor $r, u \in \{0, \ldots, m\}$, og $r + u \leq m$.

Antag for modstrid, at en knude A ligger i K_{i_r} og $K_{i_{r+u}}$, men at der er en klike på stien, som A ikke ligger i. Lad K_{i_v} være den sidste klike på stien, som opfylder, at $A \notin K_{i_v}$. Bemærk, at $i_v \neq r + u$. Da gælder det, at $A \in K_{i_{v+1}}$, men $A \notin S_{i_{v+1}}$. Den eneste knude, som opfylder dette, er $A = A_{k+(v+1)}$, igen med notationen fra punkt (c). Per konstruktion gælder det dog, at $A_{k+(v+1)} \notin K_{i_s}$ for $s < v + 1$, så $A \notin K_{i_r}$, hvilket er en modstrid.

3. Betragt slutteligt en sti i $\widetilde{T}$, der både indeholder oprindelige og nye kliker. Til nye kliker benyttes altid notationen K_{i_j}, $j \in \{1, \ldots, m\}$, fra punkt (c). Bemærk, at K_{i_0} er en oprindelig klike. Dette deles yderligere op i fire tilfælde.

   (a) Betragt stien

$$K_1, \ldots, K_r, K_{i_m}, K_{r+1}, \ldots, K_{r+s}$$

   i $\widetilde{T}$. Dette tilfælde er illustreret i Figur 6.1.3a. Da er

$$K_1, \ldots, K_r, K_i, K_{r+1}, \ldots, K_{r+s}$$

   en sti i T. Lad $A \in K_1 \cap K_{r+s}$. Da er A i alle kliker på stien, fordi T er et tilslutningstræ. Det mangles kun, at $A \in K_{i_m}$, hvilket følger af, at $A \in K_i \subseteq K_{i_m}$.

(b) Betragt stien

$$K_1, \ldots, K_r, K_{i_0}, \ldots, K_{i_m}, K_{r+1}, \ldots, K_{r+s}$$

i $\widetilde{T}$, og lad $A \in K_1 \cap K_{r+s}$. Dette tilfælde er illustreret i Figur 6.1.3b. Da er

$$K_1, \ldots, K_r, K_{i_0}, K_i, K_{r+1}, \ldots, K_{r+s}$$

en sti i T, og A ligger i alle kliker på denne, fordi T er et tilslutningstræ. Det er derfor nu tilstrækkeligt at vise, at A ligger i alle kliker på stien

$$K_{i_0}, \ldots, K_{i_m}.$$

Per forrige gælder det specielt, at $A \in K_i \cap K_{i_0}$. Da $K_i \subseteq K_{i_m}$ giver dette, at $A \in K_{i_m} \cap K_{i_0}$ og tilfældet reducerer til punkt 2.

(c) Betragt stien

$$K_1, \ldots, K_r, K_{i_0}, \ldots, K_{i_s}$$

i $\widetilde{T}$. Dette tilfælde er illustreret i Figur 6.1.3c. Lad $A \in K_1 \cap K_{i_s}$. Da $A \in K_{i_s}$, gælder det, at $A \in K_i \cup K_{i_0}$. Hvis $A \in K_i$, gælder det ydermere, at $A \in K_{i_0}$ da

$$K_1, \ldots, K_r, K_{i_0}, K_i$$

er en sti i T. Hvis $A \in K_{i_0}$, giver punkt 2., at A ligger i alle kliker på stien

$$K_{i_0}, \ldots, K_{i_s},$$

og punkt 1. giver, at A også ligger i alle kliker på stien

$$K_1, \ldots, K_r, K_{i_0}.$$

(d) Betragt stien

$$K_{i_s}, \ldots, K_{i_m}, K_1, \ldots, K_r$$

i $\widetilde{T}$. Dette tilfælde er illustreret i Figur 6.1.3d. Lad $A \in K_r \cap K_{i_s}$. Da $A \in K_{i_s}$, gælder det, at $A \in K_i \cup K_{i_0}$. Hvis $A \in K_{i_0}$, gælder det også, at $A \in K_i \subseteq K_{i_m}$, da

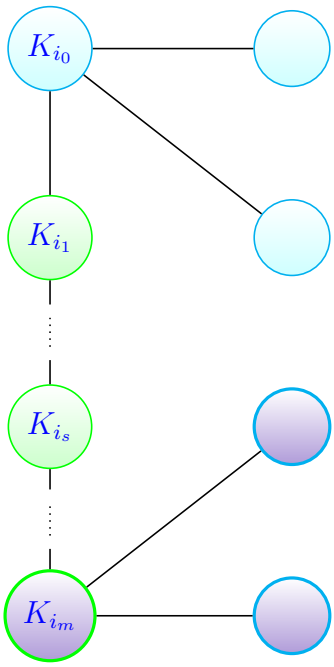
$$K_{i_0}, K_i, K_1, \ldots, K_r$$

er en sti i T. Hvis $A \in K_i \subseteq K_{i_m}$, giver punkt 2., at A ligger i alle kliker på stien

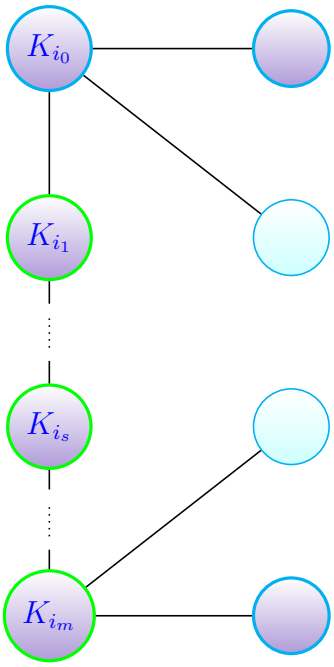
$$K_{i_s}, \ldots, K_{i_m},$$

og punkt 1. giver, at A også ligger i alle kliker på stien

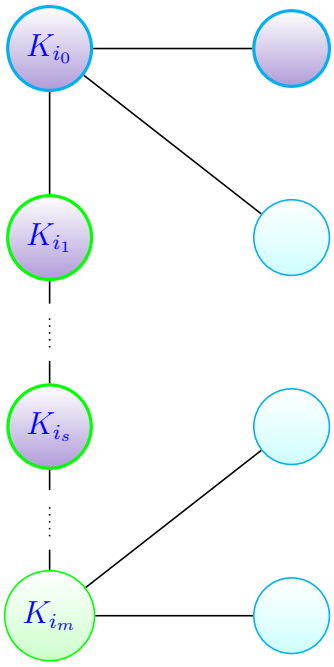
$$K_i, K_1, \ldots, K_r.$$



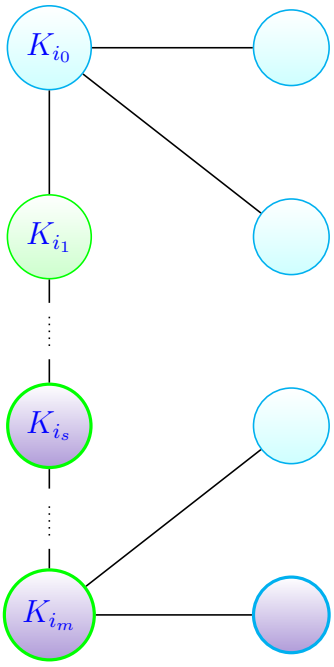
(a) Stien i punkt (a) er markeret med lilla midte.



(b) Stien i punkt (b) er markeret med lilla midte.



(c) Stien i punkt (c) er markeret med lilla midte.



(d) Stien i punkt (d) er markeret med lilla midte.

Figur 6.1.3: Udsnit af tilslutningstræer uden separatorer til illustration af punkt 3. Oprindelige kliker er markeret med blå, nye kliker er markeret med grøn, og kliker på stien er markeret med lilla midte.

Dette dækker alle tilfælde, så den resulterende graf er altså et tilslutningstræ for et k'te ordens t-kirsebærtræ.

■

Sætning 6.1.7 giver, at tilslutningstræet konstrueret med Algoritme 6.1.4 giver en mindst lige så god approksimation til den sande sandsynlighedsfordeling som det oprindelige k'te ordens tilslutningstræ.

Sætning 6.1.7
Lad T være et k'te ordens tilslutningstræ og $\widetilde{T}$ det resulterende tilslutningstræ, når Algoritme 6.1.4 anvendes på T. Lad P^T være sandsynlighedsfordelingen opnået med T og $P^{\widetilde{T}}$ være sandsynlighedsfordelingen opnået med $\widetilde{T}$. Lad desuden P være den sande sandsynlighedsfordeling. Da er

$$KL(P, P^T) \geq KL(P, P^{\widetilde{T}}).$$

Bevis. Da både T og $\widetilde{T}$ er tilslutningstræer, er det nok at vise, at $\widetilde{T}$ har større vægt end T. Genkald, at vægten for et tilslutningstræ med klikemængde $\mathcal{K}$ og separatormængde $\mathcal{S}$ er

$$\sum_{K \in \mathcal{K}} MI(K) - \sum_{S \in \mathcal{S}} \#_S \cdot MI(S).$$

Det er i denne sammenligning ikke nødvendigt at betragte kliker og separatorer, som indgår i begge tilslutningstræer. Betragt derfor en klike $K_i \in T$ med separator S_i, som bliver ændret i punkt (c) i Algoritme 6.1.4, og benyt notationen herfra. Det er nok at vise, at

$$MI(K_i) - MI(S_i) \leq \sum_{j=1}^{m} (MI(K_{i_j}) - MI(S_{i_j})).$$

Fra (B.0.9) fås det, at

$$\begin{aligned}
&MI(K_i) - MI(S_i) \\
&\quad = MI(A_1, \ldots, A_p, A_{k+1}, \ldots, A_{k+m}) - MI(A_1, \ldots, A_p) \\
&\quad = \sum_{j=1}^{m} H(A_{k+j}) + MI(A_1, \ldots, A_p) - H(A_{k+1}|A_1, \ldots, A_p) \\
&\qquad - \sum_{j=2}^{m} H(A_{k+j}|A_1, \ldots, A_p, A_{k+1}, \ldots, A_{k+j-1}) - MI(A_1, \ldots, A_p) \\
&\quad = \sum_{j=1}^{m} H(A_{k+j}) - H(A_{k+1}|A_1, \ldots, A_p) \\
&\qquad - \sum_{j=2}^{m} H(A_{k+j}|A_1, \ldots, A_p, A_{k+1}, \ldots, A_{k+j-1}).
\end{aligned} \tag{6.1.4}$$

Desuden gælder det, at

$$
\begin{aligned}
&\sum_{j=1}^{m}(MI(K_{i_j}) - MI(S_{i_j})) \\
&\quad = \sum_{j=1}^{m} MI(A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j}) - MI(A_1,\ldots,A_p,\ldots,A_{k-1}) \\
&\qquad - \sum_{j=2}^{m} MI(A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}) \\
&\quad = \sum_{j=1}^{m} H(A_{k+j}) + MI(A_1,\ldots,A_p,\ldots,A_{k-1}) \\
&\qquad + \sum_{j=2}^{m} MI(A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}) \\
&\qquad - H(A_{k+1}|A_1,\ldots,A_p,\ldots,A_{k-1}) \\
&\qquad - \sum_{j=2}^{m} H(A_{k+j}|A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}) \\
&\qquad - MI(A_1,\ldots,A_p,\ldots,A_{k-1}) \\
&\qquad - \sum_{j=2}^{m} MI(A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}) \\
&\quad = \sum_{j=1}^{m} H(A_{k+j}) - H(A_{k+1}|A_1,\ldots,A_p,\ldots,A_{k-1}) \\
&\qquad - \sum_{j=2}^{m} H(A_{k+j}|A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}),
\end{aligned}
\tag{6.1.5}
$$

hvor anden lighed følger af (B.0.8). Ved at sammenligne (6.1.4) og (6.1.5) ses det, at det er tilstrækkeligt at vise

$$
\begin{aligned}
&H(A_{k+1}|A_1,\ldots,A_p) + \sum_{j=2}^{m} H(A_{k+j}|A_1,\ldots,A_p,A_{k+1},\ldots,A_{k+j-1}) \geq \\
&H(A_{k+1}|A_1,\ldots,A_p,\ldots,A_{k-1}) + \sum_{j=2}^{m} H(A_{k+j}|A_1,\ldots,A_p,\ldots,A_{k-j},A_{k+1},\ldots,A_{k+j-1}),
\end{aligned}
$$

hvilket følger af Proposition B.0.3. Dette beviser, at $\widetilde{T}$ har enten højere end eller samme vægt som T. Dermed har den approksimerede sandsynlighedsfordeling mindst lige så lav Kullback-Leibler-afstand til den sande sandsynlighedsfordeling.

■

Sætning 6.1.8 giver, at for at finde det k'te ordens tilslutningstræ, som giver den bedste approksimation til den sande fordeling, er det tilstrækkeligt at finde det k'te ordens t-kirsebærtræ, der giver den bedste approksimation.

Sætning 6.1.8
Lad $\mathcal{T}_k$ være mængden af alle k'te ordens tilslutningstræer over variablene $A_1, \ldots, A_n$, lad P være den sande sandsynlighedsfordeling, og lad P^T være sandsynlighedsfordelingen approksimeret ved $T \in \mathcal{T}_k$. Lad desuden

$$G = \arg\min_{T \in \mathcal{T}_k} \{KL(P, P^T)\}.$$

Da er G et tilslutningstræ for et k'te ordens t-kirsebærtræ.

Bevis. Lad $\mathscr{T}_k$ være mængden af tilslutningstræer for k'te ordens t-kirsebærtræer, og bemærk, at $\mathscr{T}_k \subseteq \mathcal{T}_k$. Lad

$$\widetilde{G} = \arg\min_{H \in \mathscr{T}_k} \{KL(P, P^H)\}.$$

Lad $T \in \mathcal{T}_k$ være vilkårligt valgt, og lad $\widetilde{T} \in \mathscr{T}_k$ være det tilslutningstræ, der opnås ved at benytte Algoritme 6.1.4 på T. Da gælder det, at

$$KL(P, P^T) \geq KL(P, P^{\widetilde{T}}) \geq KL(P, P^{\widetilde{G}}),$$

hvor første ulighed følger af Sætning 6.1.7 og sidste følger af, at $\widetilde{G}$ minimerer Kullback-Leibler-afstanden blandt alle tilslutningstræer for k'te ordens t-kirsebærtræer. Dette viser, at $\widetilde{G}$ også minimerer Kullback-Leibler-afstanden blandt alle k'te ordens tilslutningstræer, så

$$\widetilde{G} = \arg\min_{T \in \mathcal{T}_k} \{KL(P, P^T)\} = G,$$

hvilket viser resultatet.

■

Når et tilslutningstræ for et k'te ordens t-kirsebærtræ er fundet, kan dette eventuelt udvides til et tilslutningstræ for et $(k+1)$'te ordens t-kirsebærtræ.

6.2 Udvidelse af *k*'te ordens *t*-kirsebærtræer til *(k+1)*'te ordens *t*-kirsebærtræer

Dette afsnit er baseret på Kovács & Szántai (2012: Kap. 3.4).
Algoritme 6.2.1 giver en måde, hvorpå et tilslutningstræ for et k'te ordens t-kirsebærtræ kan udvides til et tilslutningstræ for et $(k+1)$'te ordens t-kirsebærtræ.

Algoritme 6.2.1 `Tilslutningstræ for et` $(k+1)$`'te ordens` t`-kirsebærtræ konstrueret ud fra et tilslutningstræ for et` k`'te ordens` t`-kirsebærtræ`

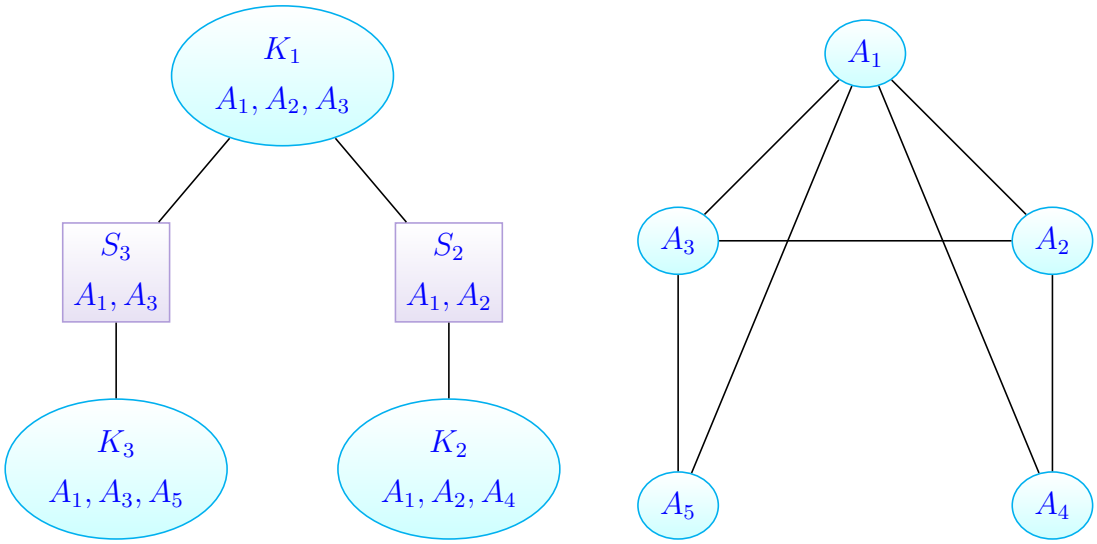
`Input`: Tilslutningstræ for et k'te ordens t-kirsebærtræ T_k

1. Vælg to nabokliker K_1 og K_2 fra T_k.
2. Dan et $(k+1)$'te ordens tilslutningstræ T_{k+1} ved at erstatte K_1 og K_2 med kliken $K_{ny} = K_1 \cup K_2$ i T_k.
   (Den nye klike har $k+1$ knuder, da T_k er et tilslutningstræ for et k'te ordens t-kirsebærtræ.)
3. Dan et tilslutningstræ $\widetilde{T}_{k+1}$ for et $(k+1)$'te ordens t-kirsebærtræ ved at benytte Algoritme 6.1.4 på T_{k+1}.

`Output`: Tilslutningstræ for et $(k+1)$'te ordens t-kirsebærtræ $\widetilde{T}_{k+1}$.

Bemærk, at T_{k+1} stadig er et tilslutningstræ, da alle stier gennem den nye klike går gennem enten K_1 eller K_2 i T_k. Proposition 6.1.6 giver derfor, at $\widetilde{T}_{k+1}$ er et tilslutningstræ for et $(k+1)$'te ordens t-kirsebærtræ. Et eksempel på brugen af Algoritme 6.2.1 ses i Eksempel 6.2.2.

Eksempel 6.2.2
Betragt tilslutningstræet for et 3. ordens t-kirsebærtræ i Figur 6.2.1a og den tilhørende graf i Figur 6.2.1b.

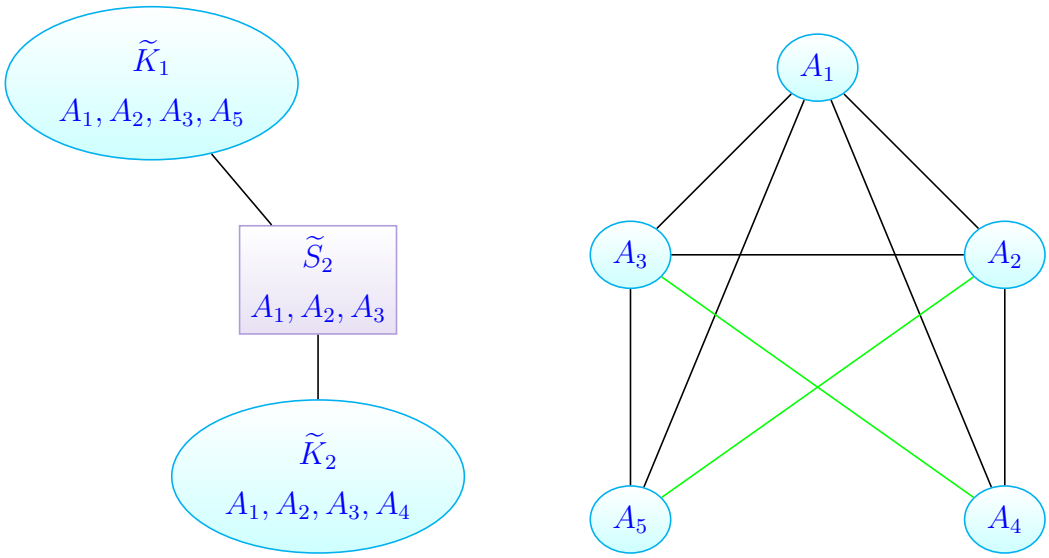


(a) 3. ordens tilslutningstræ.

(b) Tilhørende 3. ordens t-kirsebærtræ.

Figur 6.2.1: Et eksempel på et 3. ordens tilslutningstræ og den tilhørende graf.

Først forenes klikerne K_1 og K_3 til en ny klike $\widetilde{K}_1$. Herefter ændres kliken K_2 med Algoritme 6.1.4, og resultatet bliver kliken $\widetilde{K}_2$ og separatoren $\widetilde{S}_2$. Det resulterende tilslutningstræ for et 4. ordens t-kirsebærtræ ses i Figur 6.2.2a, og den tilhørende graf ses i Figur 6.2.2b.



(a) 4. ordens tilslutningstræ.

(b) Tilhørende 4. ordens t-kirsebærtræ, hvor de nye kanter er markeret med grønt.

Figur 6.2.2: Et eksempel på et 4. ordens tilslutningstræ og den tilhørende graf konstrueret med Algoritme 6.2.1.

Bemærk, at der ved konstruktionen af en ny klike altid kun tilføjes én ny kant. Algoritme

6.2.1 kan derfor betragtes som en generalisering af Algoritme 4.1.6 til højere orden.

Sætning 6.2.3 giver, at $\widetilde{T}_{k+1}$ giver en mindst lige så god approksimation af den sande sandsynlighedsfordeling som T_k.

Sætning 6.2.3
Lad T_k være et tilslutningstræ for et k'te ordens t-kirsebærtræ og $\widetilde{T}_{k+1}$ være resultatet, når Algoritme 6.2.1 benyttes på T_k. Lad P være den sande sandsynlighedsfordeling og P^{T_k} samt $P^{\widetilde{T}_{k+1}}$ være de approksimerede sandsynlighedsfordelinger med de to tilslutningstræer. Da gælder det, at

$$KL(P, P^{T_k}) \geq KL(P, P^{\widetilde{T}_{k+1}}).$$

Bevis. Lad T_{k+1} være som i punkt 3. i Algoritme 6.2.1. Hvis det kan vises, at

$$KL(P, P^{T_k}) \geq KL(P, P^{T_{k+1}}), \tag{6.2.1}$$

giver Sætning 6.1.7, at

$$KL(P, P^{T_k}) \geq KL(P, P^{T_{k+1}}) \geq KL(P, P^{\widetilde{T}_{k+1}}),$$

hvilket viser sætningen. Derfor vises det, at (6.2.1) gælder. Til dette formål sammenlignes de to tilsutningstræers vægt. Det bemærkes, at langt størstedelen af kliker og separatorer i de to tilslutningstræer er de samme. Den eneste forskel er, at der hvor T_k har klikerne K_1 og K_2 samt deres separator S, har T_{k+1} i stedet kliken $K_{ny} = K_1 \cup K_2$. Det er derfor nok at vise følgende:

$$MI(K_1) + MI(K_2) - MI(S) \leq MI(K_{ny}).$$

Lad til dette formål

$$\begin{aligned} K_1 &= \{A_1, \ldots, A_k\} \\ K_2 &= \{A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}\} \\ S &= \{A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k\}. \end{aligned}$$

Dette giver, at $K_{ny} = \{A_1, \ldots, A_{k+1}\}$. Ved at bruge (B.0.8) fås det, at

$$\begin{aligned} &MI(K_1) + MI(K_2) - MI(S) \\ &\quad = MI(A_1, \ldots, A_k) + MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}) \\ &\qquad - MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) \end{aligned}$$

$$
\begin{aligned}
&= H(A_p) + MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) - H(A_p|A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) \\
&\quad + MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}) - MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) \\
&= H(A_p) - H(A_p|A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) \\
&\quad + MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}).
\end{aligned} \tag{6.2.2}
$$

Tilsvarende gælder det, at

$$
\begin{aligned}
MI(K_{ny}) &= MI(A_1, \ldots A_p, \ldots, A_{k+1}) \\
&= H(A_p) + MI(A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}) \\
&\quad - H(A_p|A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}).
\end{aligned} \tag{6.2.3}
$$

Ved at sammenligne (6.2.2) og (6.2.3) ses det, at det er nok at vise

$$H(A_p|A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k) \geq H(A_p|A_1, \ldots, A_{p-1}, A_{p+1}, \ldots, A_k, A_{k+1}),$$

hvilket følger af Proposition B.0.3. Hermed er det bevist, at $\widetilde{T}_{k+1}$ giver en approksimeret sandsynlighedsfordeling, der er mindst lige så god, som den T_k giver.

■

6.3 Udvidelse og optimering af konstruktionsalgoritmer

I dette afsnit udvides konstruktionsalgoritmerne fra Afsnit 4.2 fra 3. ordens til vilkårlig ordens t-kirsebærtræer. Desuden præsenteres til sidst tre nye konstruktionsalgoritmer. Algoritmerne i dette afsnit er alle implementeret i vores **R**-pakke `tcherry`.

6.3.1 k'te ordens t-kirsebærtræ konstrueret direkte fra data

Algoritme 6.3.1 beskriver en udvidelse af Algoritme 4.2.2 til vilkårlig orden k.

Algoritme 6.3.1 *k*'te ordens *t*-kirsebærtræ konstrueret direkte fra data

`k_tcherry_step(data, k, ...)`

(`data`: Et datasæt med realisationer af variablene, `k`: Ordenen af t-kirsebærtræet, der ønskes konstrueret, ...: Øvrige argumenter til beregning af mutual information.)

1. Regn den estimerede *mutual information* for alle kombinationer af k variable.
2. Lad kombinationen med højest *mutual information* være det første kirsebær.
3. Tilføjelse af resterende knuder:
   (a) Lad D være et tomt datasæt og $\mathcal{H}$ være en mængde med alle nuværende hyperkanter.
   (b) Gentag for alle knuder A_k, som endnu ikke er i k'te ordens t-kirsebærtræet:
      – Gentag for alle hyperkanter $\{A_1, \ldots, A_{k-1}\} \in \mathcal{H}$:
         i. Regn $MI(A_1, \ldots, A_{k-1}, A_k) - MI(A_1, \ldots, A_{k-1})$ for kirsebærret $(\{A_1, \ldots, A_{k-1}\}, A_k)$, og gem værdien i D.
   (c) Tilføj kirsebærret fra D med den højeste værdi til k'te ordens t-kirsebærtræet.
   (d) Fjern rækker fra D, som ikke længere repræsenterer nye mulige kirsebær.
   (e) Lad nu $\mathcal{H}$ være mængden kun bestående af de nye hyperkanter, dannet ved tilføjelsen af det nye kirsebær.
   (f) Gentag punkt (c) – (f), indtil alle knuder er i k'te ordens t-kirsebærtræet.

`Output`: Nabomatricen, vægten, klikerne, separatorene, antal kanter i grafen.

I algoritmen indeholder datasættet D altid de kliker og deres bidrag til vægten, som det er muligt at tilføje i et aktuelt skridt. Efter en klike er accepteret, opdateres denne kun ved at fjerne kliker, som nu er ugyldige tilføjelser, og tilføje de nye mulige tilføjelser, der er opstået.

Bemærk, at Algoritme 6.3.1 ikke kun er en udvidelse af Algoritme 4.2.2, men også afviger en smule i strukturen. Dette skyldes et forsøg på at gøre funktionen hurtigere. Til dette formål er der gjort brug af funktionen `profvis` fra pakken af samme navn (Chang & Luraschi 2018), hvilket afslørede, at det i funktionen tager længst tid at regne *mutual information*. Derfor er det i funktionen `k_tcherry_step` forsøgt at regne *mutual information* så få gange som muligt.

Betragt til sammenligning følgende udsnit af funktionen `tcherry_step`, der viser den løkke, hvor nye kirsebær tilføjes. I dette er `tcherry_nodes` de variable, som allerede er

inkluderet i træet, `tcherry_edges` kanterne i træet, og `n_var` antal variable i data.

```
idx_list <- 1

while (length(tcherry_nodes) != n_var) {
  weight_next_step <- rep(NA, nrow(tcherry_edges) *
                          length(nodes_remaining))
  new_cliques_list <- new_seps_list <- new_var_list <-
    as.list(weight_next_step)
  idx <- 1
  for (i in 1:nrow(tcherry_edges)){
    for (var in nodes_remaining){
      new_sep <- unlist(tcherry_edges[i, ])
      names(new_sep) <- NULL
      new_clique <- unlist(c(new_sep, var))
      MI2_sep <- MI2_fun(new_sep[1], new_sep[2])
      MI3_clique <- MI3_fun(new_clique[1], new_clique[2],
                            new_clique[3])
      weight_next_step[idx] <- weight + MI3_clique - MI2_sep
      new_cliques_list[[idx]] <- new_clique
      new_seps_list[[idx]] <- new_sep
      new_var_list[[idx]] <- var
      idx <- idx + 1
    }
  }

  # Udeladt kode, der vælger et nyt kirsebær og opdaterer elementer.
}
```

Det bemærkes, at hver gang denne løkke gennemløbes, regnes alle *mutual information* på ny, selvom langt de fleste nye kirsebær også var aktuelle i forrige gennemløb. Derfor er den tilsvarende løkke i `k_tcherry_step` modificeret, så denne information kan genbruges under vejs. Følgende viser den tilsvarende løkke i `k_tcherry_step`:

```
n_nodes_remaining_median <- floor((n_var - k) / 2 + 1)
n_hyp_edges_median <- 1 + (k - 1) * ((n_var - n_nodes_remaining_median)
                                     - (k - 1))
```

```
  weight_cliq_sep <- MI_cliq <- MI_sep <- new_var <-
    rep(NA, n_nodes_remaining_median * n_hyp_edges_median)

  new_cliques_list <- new_seps_list <-
    as.list(weight_cliq_sep)

  dat_new_poss <- data.frame(new_cliq = I(new_cliques_list),
                             new_sep = I(new_seps_list),
                             new_var = new_var,
                             MI_cliq = MI_cliq,
                             MI_sep = MI_sep,
                             weight_increase = weight_cliq_sep)

while(length(tcherry_nodes) != n_var){
  for (i in 1:length(tcherry_hyperedges)){
    for (var in nodes_remaining){
      new_sep <- tcherry_hyperedges[[i]]
      dat_new_poss$new_sep[[idx.dat]] <- new_sep
      new_cliq <- c(dat_new_poss$new_sep[[idx.dat]], var)
      dat_new_poss$new_cliq[[idx.dat]] <- new_cliq

      dat_new_poss$MI_sep[idx.dat] <- MIk(new_sep, data, ...)
      dat_new_poss$MI_cliq[idx.dat] <- MIk(new_cliq, data, ...)

      dat_new_poss$weight_increase[idx.dat] <-
        dat_new_poss$MI_cliq[idx.dat] - dat_new_poss$MI_sep[idx.dat]

      dat_new_poss$new_var[[idx.dat]] <- var
      idx.dat <- idx.dat + 1
    }
  }

  # Udeladt kode, der vælger et nyt kirsebær og opdaterer elementer.

  new_hyper_edges <- utils::combn(new_clique, k - 1)
```

```
new_hyper_edges <- split(new_hyper_edges,
                         rep(1:ncol(new_hyper_edges),
                             each = nrow(new_hyper_edges)))
idx.new <- sapply(new_hyper_edges, function(e){
  length(setdiff(e, new_var)) != k - 1
})
tcherry_hyperedges <- new_hyper_edges[idx.new]

idx.new.var <- dat_new_poss$new_var == new_var &
  !is.na(dat_new_poss$new_var)
dat_new_poss[idx.new.var, ] <- NA
ord <- order(dat_new_poss$weight_increase, decreasing = TRUE)
dat_new_poss <- dat_new_poss[ord, ]

idx.dat <- which(is.na(dat_new_poss$new_var))[1]
}
```

Bemærk, at `dat_new_poss` er D i Algoritme 6.3.1. Datasættet `dat_new_poss` er desuden initialiseret til at have det antal rækker, som bliver nødvendigt i det gennemløb, hvor der er flest muligheder. Dette sikrer, at størrelsen ikke ændres undervejs, hvilket en profilering også viste tager tid.

Bemærk også, at denne metode stadig regner nogle mutual information værdier flere gange. For eksempel regnes mutual information for alle kliker, når den første tilføjes, men denne information gemmes ikke til senere brug. Dette skyldes, at det med vores kendskab til programmering tager længere tid at genfinde denne information end at regne den igen. Der tages heller ikke højde for, at den samme separator kan indgå flere gange i træet. Under alle omstændigheder udregnes mutual information langt færre gange i denne nye implementering.

6.3.2 *(k+1)*'te ordens *t*-kirsebærtræ konstrueret fra et *k*'te ordens

Algoritme 6.3.2 giver en udvidelse af Algoritme 4.2.1 til vilkårlig orden.

Algoritme 6.3.2 $(k+1)$'te ordens t-kirsebærtræ konstrueret ud fra et k'te ordens t-kirsebærtræ

`increase_order1(tch_cliq, data, ...)`

(`tch_cliq`: Klikerne i et k'te ordens t-kirsebærtræ, `data`: Et datasæt med realisationer af variablene, ...: Øvrige argumenter til beregning af mutual information.)

1. Genskab først grafen med klikerne `tch_cliq`, og kald denne T_k.
2. Regn den estimerede *mutual information* for alle kombinationer af $(k+1)$ variable. Sorter kombinationerne efter aftagende *mutual information*, og gem dem i en liste L.
3. Bestemmelse af første kirsebær:
   (a) Betragt den første kombination C i L, og lad $T_{k+1} = T_k$.
   (b) Gør midlertidigt C komplet i T_{k+1} ved at tilføje manglende kanter.
   (c) Hvis der kun tilføjes én kant, accepter T_{k+1}, og gem C i mængden $\mathcal{N}$.
   (d) Ellers, sæt C til at være næste kombination i L.
   (e) Gentag punkt (b) – (d), indtil et kirsebær accepteres.
   (f) Fjern C fra L.
4. Tilføj resterende knuder ved at gentage følgende indtil alle knuder er i $\mathcal{N}$:
   (a) Betragt den første kombination C i L.
   (b) Hvis k af knuderne fra C er i $\mathcal{N}$:
      i. Gør midlertidigt C komplet i T_{k+1}.
      ii. Hvis kun én kant tilføjes, accepter T_{k+1}, sæt $\mathcal{N} = \mathcal{N} \cup C$, og fjern alle kombinationer fra L, som er i $\mathcal{N}$.
   (c) Hvis ikke kirsebærret accepteres, betragt næste kombination i L.
   (d) Gentag punkt (b) – (c), indtil et kirsebær accepteres.

`Output:` Nabomatricen, klikerne, separatorene, antal kanter i grafen.

Algoritme 6.3.2 forsøger ligesom Algoritme 4.2.1 ikke at optimere hele vægten for tilslutningstræet, men kun summen af *mutual information* for klikerne. Dette skyldes, at implementeringen i Algoritme 4.2.1 følger fremgangsmåden fra Kovács & Szántai (2010), og Algoritme 6.3.2 er en direkte udvidelse af denne. Det ønskes dog at have en tilsvarende funktion, der

forsøger at optimere hele vægten, hvilket Algoritme 6.3.3 gør.

Algoritme 6.3.3 $(k+1)$'te ordens t-kirsebærtræ konstrueret ud fra et k'te ordens t-kirsebærtræ

`increase_order2(tch_cliq, data, ...)`

(`tch_cliq`: Klikerne i et k'te ordens t-kirsebærtræ, `data`: Et datasæt med realisationer af variablene, ...: Øvrige argumenter til beregning af mutual information.)

1. Genskab først grafen med klikerne `tch_cliq`, og kald denne T_k.
2. Sæt $T_{k+1} = T_k$.
3. Dan alle kombinationer af $(k+1)$ variable.
4. Regn den estimerede *mutual information* for de kombinationer, der kun giver anledning til tilføjelsen af én kant i T_{k+1}.
5. Lad kombinationen med højest *mutual information* være det første kirsebær, og gem denne i mængden $\mathcal{N}$.
6. Lad D være et tomt datasæt og $\mathcal{H}$ være mængden af hyperkanter i det accepterede kirsebær.
7. Tilføj resterende knuder ved at gentage følgende, indtil alle knuder er i $\mathcal{N}$:
   (a) Gentag for alle knuder A_{k+1}, som endnu ikke er i $\mathcal{N}$:
      – Gentag for alle hyperkanter $\{A_1, \ldots, A_k\} \in \mathcal{H}$:
         i. Hvis kliken $\{A_1, \ldots, A_k, A_{k+1}\}$ kun giver én ny kant i T_{k+1}:
            ∘ Regn $MI(A_1, \ldots, A_k, A_{k+1}) - MI(A_1, \ldots, A_k)$ for kirsebærret $(\{A_1, \ldots, A_k\}, A_{k+1})$, og gem værdien i D.
   (b) Tilføj kirsebærret fra D med den højeste værdi til T_{k+1}, og tilføj knuderne til $\mathcal{N}$.
   (c) Fjern rækker fra D, som ikke længere repræsenterer nye mulige kirsebær.
   (d) Lad nu $\mathcal{H}$ være mængden kun bestående af de nye hyperkanter, dannet ved tilføjelsen af det nye kirsebær.

`Output`: Nabomatricen, vægten, klikerne, separatorene, antal kanter i grafen.

Bemærk, at denne algoritme ikke kun afviger i, hvad der optimeres, men også lidt i strukturen af algoritmen, idet *mutual information* ikke regnes for alle klike-separator-par i starten af

algoritmen. Dette skyldes, at det også er forsøgt at gøre denne funktion hurtigere ved kun at regne *mutual information*, når det er nødvendigt. Dette ses i følgende udsnit af koden, hvor `tch_adj` er nabomatricen for det givne k'te ordens t-kirsebærtræ:

```
# Tilføjelse af første kirsebær.
  poss_cliq <- utils::combn(nodes, k + 1)
  poss_cliq <- split(poss_cliq, rep(1:ncol(poss_cliq),
                                    each = nrow(poss_cliq)))
  dat_first_cherry <- data.frame(cliq = I(poss_cliq),
                                 MI = rep(NA, length(poss_cliq)),
                                 adj_mat = I(as.list(
                                   rep(NA, length(poss_cliq)))))
  n_edges <- sum(tch_adj) / 2

  for (i in 1:nrow(dat_first_cherry)) {
    adj_matrix_temp <- tch_adj
    adj_matrix_temp[dat_first_cherry$cliq[[i]],
                    dat_first_cherry$cliq[[i]]] <- 1
    diag(adj_matrix_temp[dat_first_cherry$cliq[[i]],
                         dat_first_cherry$cliq[[i]]]) <- 0
    if ((sum(adj_matrix_temp) / 2 ) == (n_edges + 1)){
      dat_first_cherry$MI[i] <- MIk(dat_first_cherry$cliq[[i]],
                                    data, ...)
      dat_first_cherry$adj_mat[[i]] <- adj_matrix_temp
    }
  }

  # Udeladt kode, der vælger et nyt kirsebær og opdaterer elementer.

  # Tilføjelse af resterende knuder.
  n_edges <- n_edges + 1

  idx.dat <- 1
  idx.list <- 1

  n_nodes_remaining_median <- floor((n_var - (k + 1)) / 2 + 1)
  n_hyp_edges_median <- 1 + (k) * (
```

```
  (n_var - n_nodes_remaining_median) - (k))

weight_cliq_sep <- MI_cliq <- MI_sep <- new_var <-
  rep(NA, n_nodes_remaining_median * n_hyp_edges_median)

new_cliques_list <- new_seps_list <-
  as.list(weight_cliq_sep)

dat_new_poss <- data.frame(new_cliq = I(new_cliques_list),
                           new_sep = I(new_seps_list),
                           new_var = new_var,
                           MI_cliq = MI_cliq,
                           MI_sep = MI_sep,
                           weight_increase = weight_cliq_sep)

while (length(tcherry_nodes) != n_var){
  for (i in 1:length(tcherry_hyperedges)){
    for (var in nodes_remaining){
      new_sep <- tcherry_hyperedges[[i]]
      dat_new_poss$new_sep[[idx.dat]] <- new_sep
      new_cliq <- c(dat_new_poss$new_sep[[idx.dat]], var)
      dat_new_poss$new_cliq[[idx.dat]] <- new_cliq


      adj_matrix_temp <- adj_matrix
      adj_matrix_temp[new_cliq, new_cliq] <- 1
      diag(adj_matrix_temp[new_cliq, new_cliq]) <- 0

      if ((sum(adj_matrix_temp) / 2 ) == (n_edges + 1)){
        dat_new_poss$MI_sep[idx.dat] <- MIk(new_sep, data, ...)
        dat_new_poss$MI_cliq[idx.dat] <- MIk(new_cliq, data, ...)

        dat_new_poss$weight_increase[idx.dat] <-
        dat_new_poss$MI_cliq[idx.dat] - dat_new_poss$MI_sep[idx.dat]
      }
```

```
      dat_new_poss$new_var[[idx.dat]] <- var
      idx.dat <- idx.dat + 1
    }
  }

# Udeladt kode, der vælger et nyt kirsebær og opdaterer elementer.

  new_hyper_edges <- utils::combn(new_clique, k)
  new_hyper_edges <- split(new_hyper_edges,
                           rep(1:ncol(new_hyper_edges),
                               each = nrow(new_hyper_edges)))
  idx.new <- sapply(new_hyper_edges, function(e){
    length(setdiff(e, new_var)) != k
  })
  tcherry_hyperedges <- new_hyper_edges[idx.new]

  adj_matrix[new_clique, new_clique] <- 1
  diag(adj_matrix[new_clique, new_clique]) <- 0

  idx.new.var <- dat_new_poss$new_var == new_var &
    !is.na(dat_new_poss$new_var)
  dat_new_poss[idx.new.var, ] <- NA
  ord <- order(dat_new_poss$new_var)
  dat_new_poss <- dat_new_poss[ord, ]

  idx.dat <- which(is.na(dat_new_poss$new_var))[1]
  n_edges <- n_edges + 1
}
```

Bemærk, at den store forskel i forhold til `k_tcherry_step` er, at *mutual information* kun udregnes for de mulige nye kliker. I dette tilfælde er klikerne kun mulige tilføjelser, hvis de kan sættes på det nuværende $(k+1)$'te ordens t-kirsebærtræ ved kun at tilføje én kant udover de eksisterende kanter i det givne k'te ordens t-kirsebærtræ. Dette er i tråd med, at fremgangsmåden i Algoritme 6.2.1 benyttes til at udvide den givne graf.

For at vurdere hastigheden af de implementerede funktioner benyttes funktionen `microbenchmark` fra **R**-pakken af samme navn (Mersmann 2018). Denne er sat til at køre funktionen 100

gange på datasættet præsenteret i Afsnit 5, som indeholder 12 variable med mellem 3 og 10 mulige udfald. Resultatet ses i Tabel 6.3.1.

Konstruktion af 3. ordens t-kirsebærtræer	
Funktion	**Median-tid i millisekunder**
`tcherry_step`	684
`k_tcherry_step`	338
`increase_order1`	176
`increase_order2`	100
Konstruktion af 4. ordens t-kirsebærtræer	
Funktion	**Median-tid i millisekunder**
`k_tcherry_step`	734
`increase_order1`	493
`increase_order2`	130

Tabel 6.3.1: Sammenligning af eksekveringstid for forskellige funktioner benyttet på datasættet fra Afsnit 5. Bemærk, at for de to `increase_order` funktioner er tiden uden tilpasning af et lavere ordens t-kirsebærtræ. Tiden beskriver altså kun selve udvidelsen, når denne struktur på forhånd er givet.

Som ønsket er `k_tcherry_step` hurtigere end den ældre implementering `tcherry_step`. Desuden ses det, at `increase_order1` lader til at være langsommere end `increase_order2` på trods af, at den første kun betragter kliker, hvorimod den sidste både regner *mutual information* for kliker og separatorer, og dermed optimerer på hele vægten af tilslutningstræet. Dette indikerer igen, at *mutual information* ikke bør beregnes for kliker, som ikke kan tilføjes på det givne tidspunkt. Det ses også, at det er væsentligt hurtigere at udvide et lavere ordens t-kirsebærtræ med `increase_order2` end at konstruere det fra bunden med `k_tcherry_step`. Bemærk dog, at denne tid kun beskriver selve udvidelsen, det er altså eksklusiv tiden, det tager at tilpasse et lavere ordens t-kirsebærtræ. Tidsbesparelsen skyldes, at når et k'te ordens t-kirsebærtræ blot udvides med flere kanter, kan væsentligt færre nye kliker tilføjes i hvert skridt, hvilket i høj grad begrænser det nødvendige antal gange, *mutual information* skal beregnes.

6.3.3 Grådig algoritme, der tilføjer p kliker ad gangen

I stedet for kun at tilføje én klike ad gangen, som i Algoritme 6.3.1, er det også muligt at tilføje p kliker ad gangen. Dette gør Algoritme 6.3.4.

Algoritme 6.3.4 `Et` k`'te ordens` t`-kirsebærtræ konstrueret direkte fra data ved stepvis at tilføje` p `kliker ad gangen`

`k_tcherry_p_lookahead(data, k, p, ...)`

(`data`: Et datasæt med realisationer af n variable, `k`: Ordenen af t-kirsebærtræet, der ønskes konstrueret, `p`: Antal kliker der tilføjes i hvert step, ...: Øvrige argumenter til beregning af mutual information.)

1. Lav en liste L til mulige modeller, hvor hver model indeholder kliker, separatorer, variable som mangler at blive tilføjet, nabomatricen og vægt.
2. Bestem alle kombinationer af k variable. Indsæt hver kombination som klike i en model i L, og regn vægten.
3. Hvis der kun er k variable i `data`, så returner modellen i L med højest vægt.
4. Ellers, sæt $i = n - k$, og gentag følgende, indtil $i = 0$:
   (i er antal kliker, der mangler at blive tilføjet.)
   - Gentag følgende $j = \min\{p-1, i\}$ gange, første gang dette punkt eksekveres, og $j = \min\{p, i\}$ alle efterfølgende gange:
     - Lav en liste L_{ny} til de nye modeller.
     - For alle modeller M i L og for alle kliker K i M's klikemængde:
       - Dan nye separatorer ved at danne alle kombinationer af $k-1$ variable af de k variable i K.
       - For alle nye separatorer S og for alle ubrugte variable v i M:
         - ⋆ Dan kliken $K_{ny} = S \cup \{v\}$.
         - ⋆ Dan modellen med klikerne fra M samt K_{ny}, separatorerne fra M samt S, de ubrugte variable fra M på nær v og den nye nabomatrix.
         - ⋆ Tilføj den nye model til L_{ny}.
     - Udtynd L_{ny}, så hver model kun indgår én gang, og sæt $L = L_{ny}$.
   - Opdater vægtene for alle modeller i L med bidraget fra de nye kliker og separatorer.
   - Vælg modellen M_{max} fra L, som har højest vægt, og sæt $L = M_{max}$.
   - Sæt $i = i - j$.

`Output`: Nabomatricen, vægten, klikerne, separatorene, antal undersøgte modeller, antal kanter i grafen.

6.3.4 Udtømmende afsøgning

Alle algoritmer præsenteret hidtil har været grådige algoritmer. Dette skyldes, at der er

$$\binom{n}{k-1}((k-1)(n-k+1)+1)^{n-k-1}$$

mulige k'te ordens t-kirsebærtræer med n knuder. Dette resultat følger af Moon (1969), da definitionen for et $(k+1)$'te ordens t-kirsebærtræ også er definitionen på et såkaldt k-træ med n navngivne knuder.

Hvis forskellen mellem n og k er stor, er der for mange kombinationer til effektivt at undersøge dem alle. Det ønskes dog at vurdere kvaliteten af de grådige algoritmer, hvorfor funktionen `tcherry_complete_search`, som undersøger alle muligheder, er implementeret.

Algoritme 6.3.5 `Et` k`'te ordens` t`-kirsebærtræ valgt ved udtømmende afsøgning`

`tcherry_complete_search(data, k, ...)`

(`data`: Et datasæt med realisationer af variablene, `k`: Ordenen af t-kirsebærtræet, der ønskes konstrueret, ...: Øvrige argumenter til beregning af mutual information.)

1. Lav en liste L til mulige modeller, hvor hver model indeholder kliker, separatorer, variable som mangler at blive tilføjet og nabomatricen.
2. Bestem alle kombinationer C af k variable, og indsæt hvert C som klike i en model i L.
3. Lad n være antal variable, og gentag følgende $n-k$ gange:
   - Lav en liste L_{ny} til de nye modeller.
   - For alle modeller M i L og for alle kliker K i M's klikemængde:
     - Dan nye separatorer ved at danne alle kombinationer af $k-1$ variable i K.
     - For alle nye separatorer S og for alle ubrugte variable v i M:
       - ⋆ Dan kliken $K_{ny} = S \cup \{v\}$.
       - ⋆ Udvid M med kliken K_{ny} og separatoren S. Fjern v fra de ubrugte variable.
       - ⋆ Tilføj M til L_{ny}.
   - Udtynd L_{ny}, så hver model kun indgår én gang, og sæt $L = L_{ny}$.
4. Regn vægten for alle tilslutningstræer for alle modeller i L, og vælg modellen med størst vægt.

`Output`: Nabomatricen, vægten, klikerne, separatorene, antal undersøgte modeller, antal kanter i grafen.

I forbindelse med at udvide et k'te ordens t-kirsebærtræ til et $(k+1)$'te ordens t-kirsebærtræ er det kun en delmængde af de mulige t-kirsebærtræer, der er gyldige. Derfor undersøger Algoritme 6.3.6 kun alle muligheder i denne delmængde.

Algoritme 6.3.6 `Udtømmende afsøgning af et` $(k+1)$`'te ordens` t`-kirsebærtræ konstrueret ud fra et` k`'te ordens` t`-kirsebærtræ`

`increase_order_complete_search(tch_cliq, data, ...)`

(`tch_cliq`: Klikerne i et k'te ordens t-kirsebærtræ, `data`: Et datasæt med realisationer af variablene, `...`: Øvrige argumenter til beregning af mutual information.)

1. Gendan grafen T_k med klikerne `tch_cliq`.
2. Lav en liste L til mulige modeller, hvor hver model indeholder kliker, separatorer, variable som mangler at blive tilføjet og nabomatricen.
3. Bestem alle kombinationer af $k+1$ variable, og indsæt dem som klike i en model i L, hvis en sådan klike kun tilføjer én kant i forhold til T_k. Benyt for alle disse modeller nabomatricen for T_k samt den nye kant tilføjet af kliken.
4. Lad n være antal variable, og gentag følgende $n-k$ gange:
   - Lav en liste L_{ny} til de nye modeller.
   - For alle modeller M i L og for alle kliker K i M's klikemængde:
     - Dan nye separatorer ved at danne alle kombinationer af $k-1$ variable af de k variable i K.
     - For alle nye separatorer S og for alle ubrugte variable v i M:
       - ⋆ Dan kliken $K_{ny} = S \cup \{v\}$.
       - ⋆ Udvid M med kliken K_{ny} og separatoren S. Fjern v fra de ubrugte variable.
       - ⋆ Tilføj M til L_{ny}, hvis den nye nabomatrix kun afviger fra den gamle med én ekstra kant.
   - Udtynd L_{ny}, så hver model kun indgår én gang, og sæt $L = L_{ny}$.
5. Regn vægten for alle tilslutningstræer for alle modeller i L.
6. Vælg modellen med størst vægt.

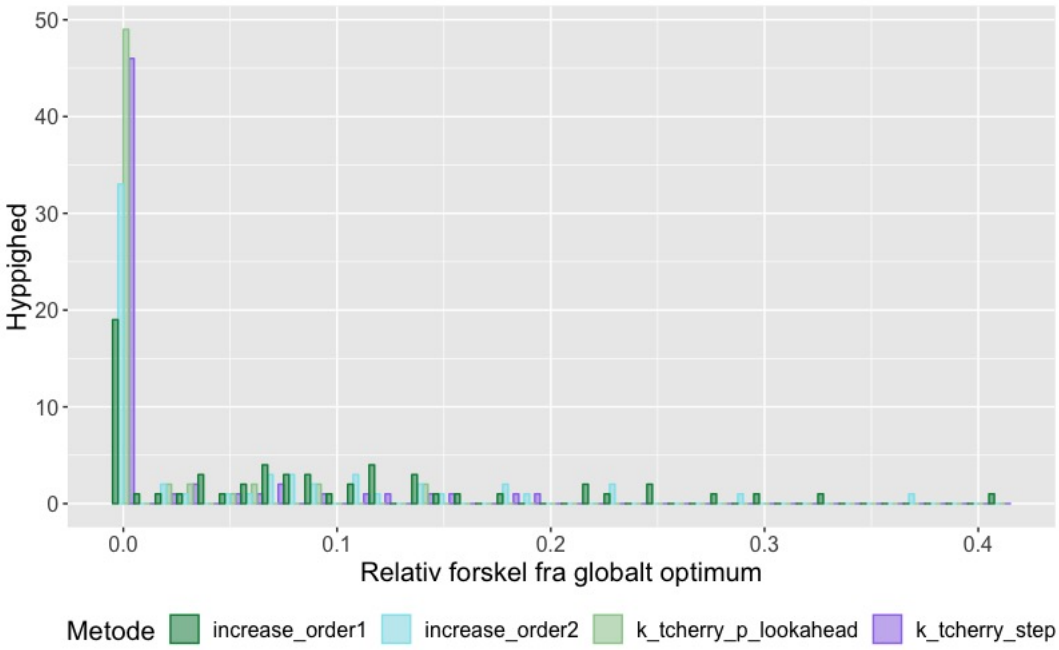
`Output`: Nabomatricen, vægten, klikerne, separatorene, antal undersøgte modeller, antal kanter i grafen.

Bemærk, at denne algoritme næsten er identisk med Algoritme 6.3.5 bortset fra, at nye modeller kun accepteres, hvis den nye klike, udover kanter fra T_k, kun tilføjer én ny kant til modellen.

Disse to funktioner benyttes til at vurdere kvaliteten af de grådige algoritmer i næste afsnit.

6.4 Simulationsstudie til vurdering af grådige konstruktionsalgoritmers kvalitet

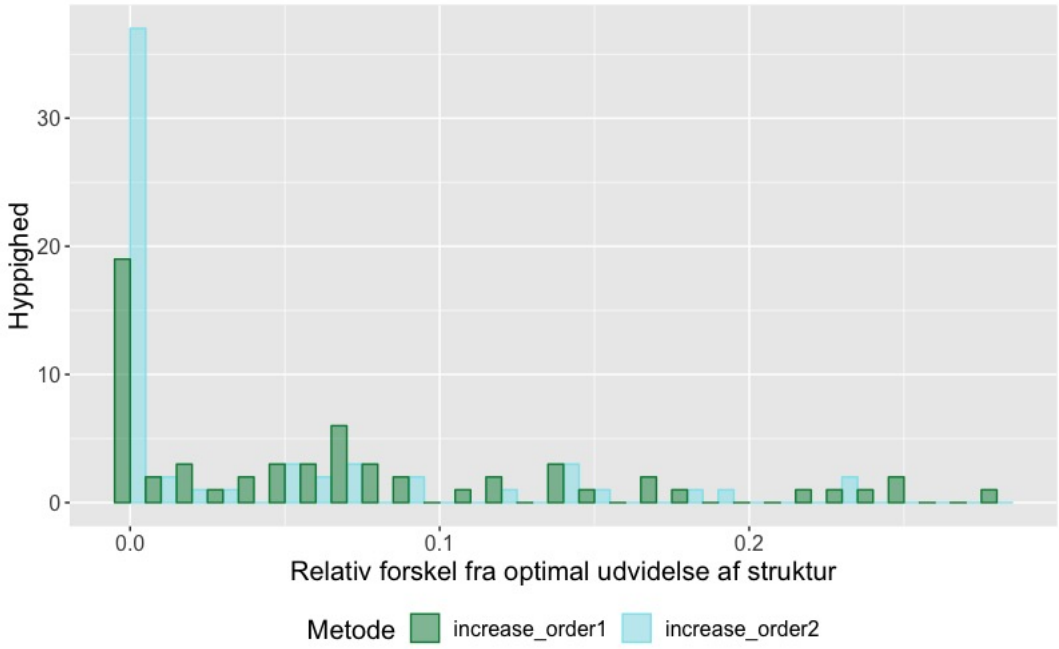
Udgangspunktet i dette afsnit er et simulationsstudie, hvor der for 5, 6 og 7 knuder er genereret data, hvorpå konstruktionsalgoritmer er benyttet til at bestemme 3. ordens t-kirsebærtræer. For funktionerne, der som input tager klikerne fra et lavere ordens t-kirsebærtræ, er disse kliker altid bestemt som klikerne i Chow-Liu-træet ved at benytte `k_tcherry_step`, hvor $k = 2$. Funktionen `k_tcherry_p_lookahead` bruger desuden argumentet $p = 2$. For hver mulighed af antal knuder er der simuleret 20 datasæt med 500 rækker ved at simulere data fra et 3. ordens t-kirsebærtræ genereret med `random_tcherry`. I alt simuleres altså 60 datasæt. For hver konstrueret struktur sammenlignes den opnåede vægt af tilslutningstræet med vægten opnået ved `tcherry_complete_search` ved at udregne differensen. Resultatet af dette ses i Figur 6.4.1.



Figur 6.4.1: Histogram over forskellen fra det globale optimum for forskellige grådige konstruktionsalgoritmer. Forskellen er angivet relativt til vægten opnået med udtømmende afsøgning, og den sande struktur er et 3. ordens t-kirsebærtræ.

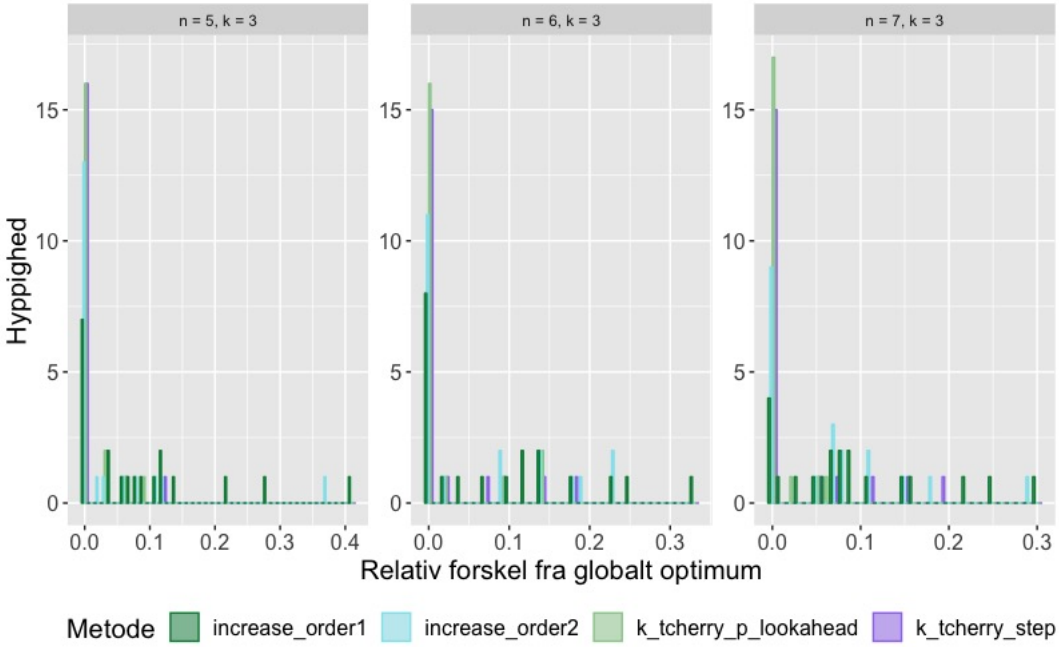
Det ses, at `k_tcherry_p_lookahead` og `k_tcherry_step` generelt er gode til at ramme det globale optimum, hvorimod de to funktioner, der udvider et lavere ordens t-kirsebærtræ, generelt er længere fra. `increase_order2` lader dog til at klare sig bedre end `increase_order1`, hvilket er meget intuitivt, da `increase_order2` faktisk forsøger at optimere vægten i modsætning til `increase_order1`.

For rigtigt at kunne vurdere kvaliteten af `increase_order1` og `increase_order2`, er det dog mere retfærdigt at sammenligne resultatet med `increase_order_complete_search`. Dette er gjort i Figur 6.4.2.

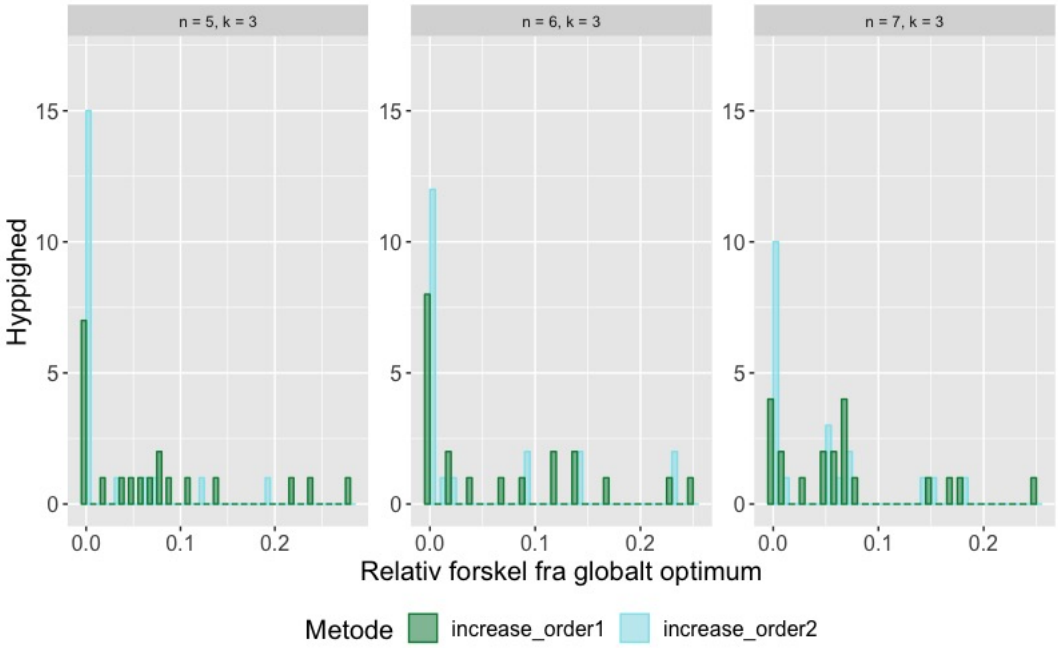


Figur 6.4.2: Histogram over forskellen fra den optimale udvidelse af strukturen for forskellige grådige konstruktionsalgoritmer. Forskellen er angivet relativt til vægten opnået med udtømmende afsøgning, og den sande struktur er et 3. ordens t-kirsebærtræ.

Her ses det, at `increase_order2` generelt er ret god til at ramme den optimale løsning under de begrænsninger, som det lægger på strukturen at udvide et lavere ordens t-kirsebærtræ. De samme analyser som i Figur 6.4.1 og 6.4.2 kan ses opdelt efter antal knuder i Figur 6.4.3 og 6.4.4.



Figur 6.4.3: Histogram over forskellen fra det globale optimum for forskellige grådige konstruktionsalgoritmer opdelt efter antal knuder n. Forskellen er angivet relativt til vægten opnået med udtømmende afsøgning, og den sande struktur er et 3. ordens t-kirsebærtræ.

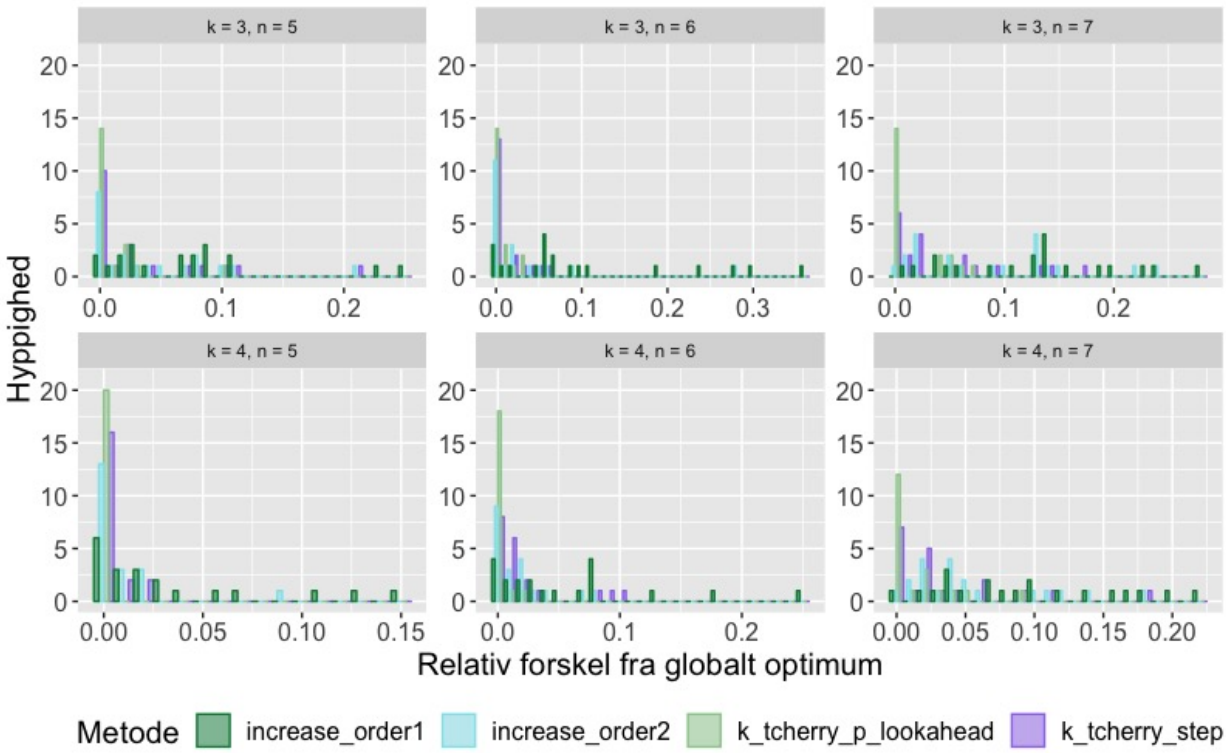


Figur 6.4.4: Histogram over forskellen fra den optimale udvidelse af strukturen for forskellige grådige konstruktionsalgoritmer opdelt efter antal knuder n. Forskellen er angivet relativt til vægten opnået med udtømmende afsøgning, og den sande struktur er et 3. ordens t-kirsebærtræ.

Det ses generelt, at `k_tcherry_p_lookahead` og `k_tcherry_step` klarer sig bedre i alle tilfælde. Desuden ses det, at selvom `increase_order2` ikke klarer sig lige så godt, er denne bedre

end `increase_order1`. Ydermere ses det, at algoritmerne, der danner træet direkte fra data, klarer sig lige godt uanset hvor mange mulige kombinationer, der er. De algoritmer, som udvider lavere ordens træer, bliver derimod dårligere, når der forekommer flere muligheder. Når de to `increase_order` funktioner sammenlignes med `increase_order_complete_search`, ses det igen, at `increase_order2` er den bedste, og det bliver tydeligere jo flere mulige t-kirsebærtræer, der er i alt.

Der er lavet et tilsvarende simulationsstudie, hvor den sande struktur dog ikke er et t-kirsebærtræ, men blot et tilfældigt bayesiansk netværk. I dette studie er der også tilpasset 4. ordens t-kirsebærtræer, og resultatet kan ses opdelt efter antal knuder og orden i Figur 6.4.5.



Figur 6.4.5: Histogram over forskellen fra det globale optimum for forskellige grådige konstruktionsalgoritmer opdelt efter orden k og antal knuder n. Forskellen er angivet relativt til vægten opnået med udtømmende afsøgning, og den sande struktur er et tilfældigt bayesiansk netværk. Fra venstre mod højre er antallet af mulige t-kirsebærtræsstrukturer henholdsvis 70, 1.215, 27.951, 10, 200 og 5.915.

Det ses, at når den sande struktur ikke nødvendigvis er et t-kirsebærtræ, har `k_tcherry_step`, `increase_order1` og `increase_order2` generelt sværere ved at opnå det globale optimum. `k_tcherry_p_lookahead` klarer sig derimod generelt godt, og sammenlignet med de andre funktioner brillerer den, især når der findes mange mulige t-kirsebærtræer. Bemærk, at for tilfældet $k = 4$ og $n = 5$ har et t-kirsebærtræ kun to kliker, så `k_tcherry_p_lookahead` for $p = 2$ svarer i dette tilfælde til udtømmende afsøgning.

7 Anvendelse af k'te ordens t-kirsebærtræer og hill-climbing til klassificering

Bayesianske netværk og Markovnetværk kan benyttes til prædiktion ved at bestemme den betingede sandsynlighed for responsvariablen givet observerede kovariater via modellen. For at opnå en klassificering tildeles variablen den klasse med størst sandsynlighed. Dette afsnit har til formål at udforske muligheden for at benytte k'te ordens t-kirsebærtræer og *hill-climbing* på denne måde. Til dette formål er det valgt at analysere følgende fem datasæt fra *UCI machine learning repository* (Dua & Graff 2017), som Cheng & Greiner (1999) også har analyseret med forskellige bayesianske netværk:

- **Adult**: Data til klassificering af hvorvidt en person tjener over 50.000 USD om året. Data har 32.561 observationer og 13 forklarende variable, hvoraf *fnlwgt* dog er udeladt som i (Cheng & Greiner 1999). Der er 5 kontinuerte variable, som er diskretiseret med funktionen `discretizeDF` fra pakken `arules` (Hahsler et al. 2011). For 3 af variablene bruges standardindstillingerne, men for *capital-gain* og *capital-loss* er grupperne valgt manuelt, alt efter om værdien er nul eller ej.

- **Car evaluation**: Data til klassificering af kvaliteten af biler. Data har 6 forklarende variable og 1.728 observationer.

- **Chess**: Data til klassificering af hvorvidt hvid kan vinde et slutspil i skak. Data har 36 forklarende variable og 3.196 observationer.

- **Mushroom**: Data til klassificering af hvorvidt en svamp er giftig. Data har 21 forklarende variable og 8.124 observationer.

- **Nursery**: Datasæt til klassificering af ansøgninger til en børnehave. Data har 8 forklarende variable og 12.960 observationer.

Disse datasæt tilbyder både variation i antal variable og observationer. På alle datasæt tilpasses 2. til 5. ordens t-kirsebærtræer med funktionen `k_tcherry_step`. Ydermere benyttes også `hill-climbing`. Disse modeller benyttes herefter til prædiktion. Kvaliteten af prædiktionen vurderes ved at bestemme gennemsnitlig nøjagtighed og standardafvigelse for denne med udgangspunkt i en 10-fold krydsvalidering. Nøjagtigheden beregnes som andelen af korrekte klassificeringer. Resultatet af dette ses i Tabel 7.0.1, hvor der sammenlignes med de bedste resultater fra Cheng & Greiner (1999).

Adult						
14 variable	2. ordens	3. ordens	4. ordens	5. ordens	Hill-climbing	C & G
Nøjagtighed	0,7985	0,8328	0,8324	0,8313	0,8322	**0,8611**
Standardafvigelse	0,0067	0,0094	0,0060	0,0054	0,0045	0,0027
Car evaluation						
7 variable	2. ordens	3. ordens	4. ordens	5. ordens	Hill-climbing	C & G
Nøjagtighed	0,8642	0,9452	0,9604	**0,9664**	0,8742	0,9404
Standardafvigelse	0,0268	0,0160	0,0145	0,0124	0,0241	0,0044
Chess						
37 variable	2. ordens	3. ordens	4. ordens	5. ordens	Hill-climbing	C & G
Nøjagtighed	0,9020	0,9214	0,9485	**0,9667**	0,9418	0,9465
Standardafvigelse	0,0136	0,0223	0,0188	0,0104	0,0113	0,0069
Mushroom						
22 variable	2. ordens	3. ordens	4. ordens	5. ordens	Hill-climbing	C & G
Nøjagtighed	0,9852	0,9941	0,9971	0,9999	**1**	**1**
Standardafvigelse	0,0051	0,0037	0,0013	0,0004	0	0
Nursery						
9 variable	2. ordens	3. ordens	4. ordens	5. ordens	Hill-climbing	C & G
Nøjagtighed	0,9031	0,9353	0,9670	**0,9806**	0,9161	0,9308
Standardafvigelse	0,0104	0,0073	0,0067	0,0042	0,0089	0,0039

Tabel 7.0.1: Tabel over gennemsnitlig nøjagtighed og standardafvigelse for denne baseret på 10-fold krydsvalidering for forskellige grafiske modeller tilpasset på forskellige datasæt. Det bedste resultat for hvert datasæt er markeret med fed skrift. C & G henviser til det bedste resultat opnået i Cheng & Greiner (1999: Tab. 3). Bemærk, at deres resultater dog enten er for en opsplitning i trænings- og testdata eller resultatet af en 5-fold krydsvalidering.

Generelt set klarer k'te ordens t-kirsebærtræerne sig ganske godt på de udvalgte klassificeringsproblemer. Det lader heller ikke til, at modellerne overtilpasser på trods af et forholdsvist højt valg af orden. Dette kan skyldes, at der i alle tilfælde er forholdsvis mange observationer og et rimeligt antal variable. Hvis dette ikke er tilfældet, risikerer k'te ordens t-kirsebærtræerne dog hurtigt at overtilpasse grundet mange kanter i de fundne grafer. En tilsvarende analyse er lavet for et datasæt **AIMs** med *Ancestry-Informative Markers* (Mogensen et al. 2019) med 443 observationer af 161 variable til klassificering af, hvor i verden mennesker kommer fra ved hjælp af genetisk information. Resultatet af dette ses i Tabel 7.0.2.

AIMs				
161 variable	2. ordens	3. ordens	4. ordens	Hill-climbing
Nøjagtighed	0,8405	0,7617	0,6916	**0,8999**

Tabel 7.0.2: Tabel over gennemsnitlig nøjagtighed fra en 10-fold krydsvalidering for forskellige grafiske modeller tilpasset på **AIMs** datasættet. Det bedste resultat er markeret med fed skrift.

I dette tilfælde lader der til at være betydelige problemer med, at t-kirsebærtræerne overtilpasser. Næste afsnit forsøger at løse problemer med overtilpasning.

8 Udtynding af kanter

Som bemærket i Afsnit 7 kan det høje antal kanter i højere ordens t-kirsebærtræer hurtigt føre til overtilpasning. I dette afsnit undersøges derfor muligheden for at fjerne nogle af kanterne igen efter tilpasningen. For fortsat at kunne approksimere sandsynlighedsfordelingen via grafens kliker, ønskes det stadig, at grafen er triangulær efter udtynding i kanterne. Til at sikre dette benyttes Proposition 8.0.1, som kommer fra Lauritzen (1996: s. 19).

Proposition 8.0.1
Lad G være en trianguleret graf, og lad $\widetilde{G}$ være en graf opnået ved at fjerne præcis én kant $e = \{A_i, A_j\}$ fra G. Da er $\widetilde{G}$ triangulær, hvis og kun hvis e indgår i præcis én klike i G.

Bevis. I dette bevis benyttes Definition 3.4.2 af en trianguleret graf.

Først vises det, at hvis $\widetilde{G}$ er triangulær, kan e kun indgå i én klike i G. Derfor antages det for modstrid, at e indgår i flere kliker i G, og der betragtes to af disse. Er dette tilfældet, må der i hver af de to kliker yderligere være mindst én knude, der ikke indgår i den anden klike. To sådanne knuder, en fra hver klike, må desuden kunne vælges således, at de ikke er naboer, da alle knuderne i de to kliker ellers ville udgøre en komplet punktmængde, hvorved de enten ville udgøre én klike eller en delmængde af en klike. Kald disse to knuder A_k og A_m. Da en klike er en komplet punktmængde, er A_k, A_i, A_m, A_j, A_k en kreds. Uden kanten mellem A_i og A_j bliver dette en kreds med fire knuder uden en genvej i $\widetilde{G}$. Dette er i modstrid med, at $\widetilde{G}$ er triangulær.

Herefter vises det, at hvis e kun indgår i én klike i G, er $\widetilde{G}$ triangulær. Antag for modstrid, at $\widetilde{G}$ ikke er triangulær. Dermed må e være den eneste genvej i en kreds med fire knuder i G. Benævn denne kreds A_k, A_i, A_m, A_j, A_k. Da udgør $\{A_k, A_i, A_j\}$ og $\{A_i, A_m, A_j\}$ to komplette punktmængder i G, og da e er den eneste genvej i kredsen, er disse to punktmængder delmængder af mindst to forskellige kliker. Dette betyder, at kanten e også ligger i mindst to kliker, hvilket er en modstrid.

Dette beviser, at hvis én kant fjernes fra en trianguleret graf, er den resulterende graf triangulær, hvis og kun hvis den pågældende kant kun indgår i én klike.

■

Proposition 8.0.1 giver, at hvis det ønskes at fjerne en kant i en trianguleret graf, således at den resulterende graf også er trianguleret, må denne kant kun indgå i én klike. Algoritme 8.0.2 beskriver, hvordan tilslutningstræet kan opdateres, hvis en sådan kant fjernes.

Algoritme 8.0.2 `Opdatering af et tilslutningstræ hvis én kant i kun én klike fjernes`

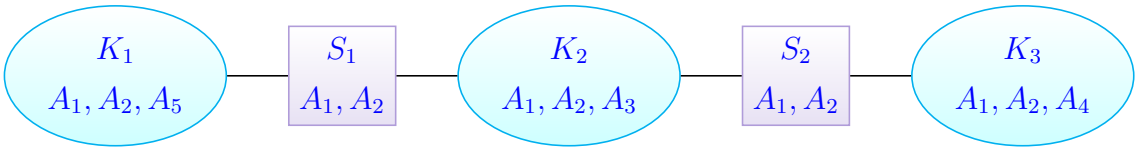
`Input`: Tilslutningstræ T og kanten $e = \{A_i, A_j\}$, der ønskes fjernet.

1. Lad K_e være kliken, som e indgår i.
2. Efter e fjernes, giver K_e anledning til de to komplette punktmængder $K_{A_i} = K_e \setminus \{A_j\}$ og $K_{A_j} = K_e \setminus \{A_i\}$.
3. Forbind K_{A_i} og K_{A_j} med separatoren $S = K_{A_i} \cap K_{A_j} = K_e \setminus e$.
4. Fjern K_e fra T, og forbind alle dens naboseparatorer til enten K_{A_i} eller K_{A_j} således, at separatorerne stadig er mængdernes fællesmængde.
   (Det er altid muligt at forbinde de eksisterende separatorer på denne måde, da det eneste problem ville være, hvis e indgik i en separator, hvilket ikke er muligt, da den kun indgår i én klike.)
5. Hvis K_{A_i} er lig nogle af naboseparatorerne, fjernes K_{A_i} og en sådan separator S_{A_i}. Lad S_{A_i}'s anden nabo tage K_{A_i}'s plads i T. Gør det samme for K_{A_j}.
   (*Dette fjerner punktmængderne, hvis de er delmængder af andre komplette punktmængder, og derfor ikke er kliker. Bemærk, at hvis en af de nye punktmænger er en delmængde af en anden klike, vil den altid være lig en naboseparator grundet RIP-egenskaben.*)

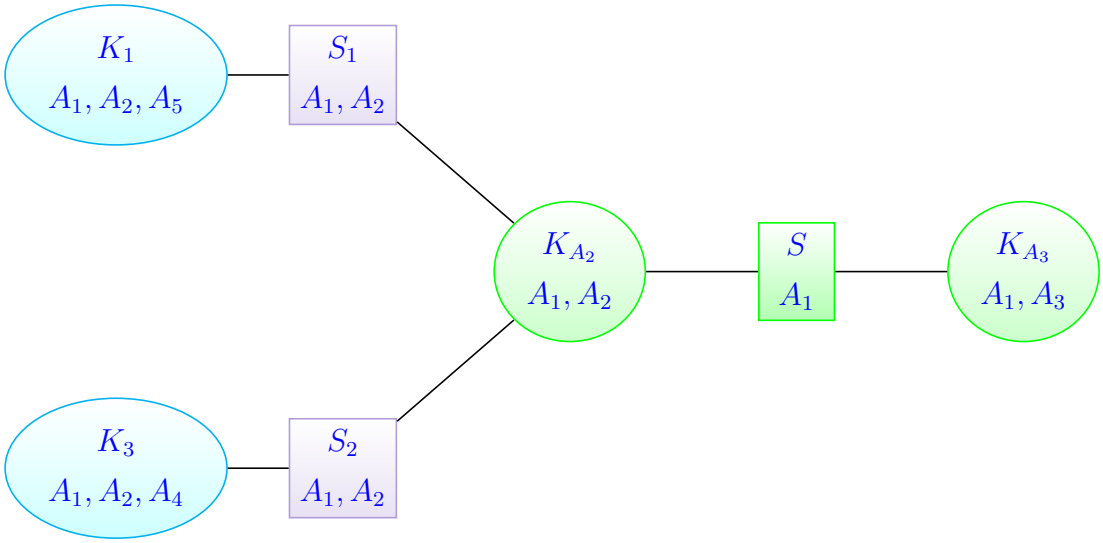
`Output`: Et tilslutningstræ T efter fjernelsen af e.

Bemærk, at efter punkt 5. passer separatorene og klikerne stadig sammen grundet RIP-egenskaben i det initialiserende tilslutningstræ T. Brugen af Algoritme 8.0.2 illustreres i Eksempel 8.0.3.

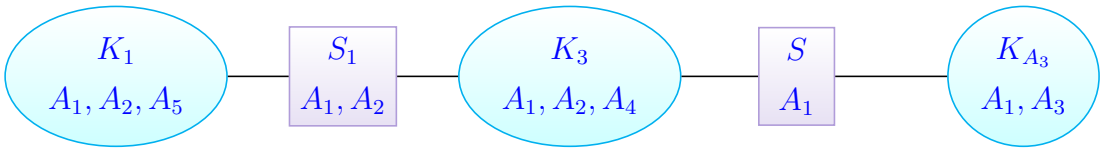
Eksempel 8.0.3
Betragt tilslutningstræet i Figur 8.0.1a. Det ønskes at fjerne kanten mellem A_2 og A_3, som kun indgår i kliken K_2. Efter punkt 4. i Algoritme 8.0.2 fås træet i Figur 8.0.1b. Efter punkt 5., hvor punktmængder, som ikke er kliker, fjernes, fås tilslutningstræet i Figur 8.0.1c.



(a) Initialiserende tilslutningstræ.



(b) Træet efter punkt 4.



(c) Det resulterende tilslutningstræ.

Figur 8.0.1: Illustration af Algoritme 8.0.2.

Proposition 8.0.4 giver, at det resulterende træ faktisk er et tilslutningstræ.

Proposition 8.0.4
Algoritme 8.0.2 returnerer et tilslutningstræ.

Bevis. Benyt gennem beviset notationen fra Algoritme 8.0.2.

For at vise, at træet er et tilslutningstræ, er det tilstrækkeligt at vise, at det konstruerede træ opfylder RIP-egenskaben. Det er tilstrækkeligt at vise, at RIP-egenskaben er opfyldt for træet efter punkt 4., da den udeladte klike netop er en delmængde af den klike, der erstatter den. Det er derfor nok at betragte stier gennem K_{A_i} eller K_{A_j}, da alle andre stier også

eksisterede i det oprindelige tilslutningstræ. Betragt en sti $K_1, \ldots, K_p, K_{A_i}, K_{p+1}, \ldots, K_n$, og lad $A_m \in K_1 \cap K_n$. Det er nok at vise, at $A_m \neq A_j$, da alle andre tilfælde er dækket af RIP-egenskaberne i det oprindelige tilslutningstræ. Antag for modstrid, at $A_m = A_j$. $K_1, \ldots, K_p, K_e, K_{p+1}, \ldots, K_n$ er en sti i det oprindelige tilslutningstræ, og A_j vil derfor ligge i separatoren mellem K_p og K_e samt mellem K_e og K_{p+1}. Dermed kan de per konstruktion ikke være naboer til K_{A_i} som antaget, hvilket er en modstrid. Det er dermed bevist, at det resulterende træ opfylder RIP-egenskaben. Et tilsvarende argument gælder for stier gennem K_{A_j} og stier gennem både K_{A_i} og K_{A_j}.

■

For at vurdere hvorvidt en kant kan fjernes uden, at modellen forringes væsentligt, vælges det at benytte en likelihood ratio test. Lad $\mathcal{D}$ være en mængde af uafhængige observationer af data med elementer på formen $d = (d_1, \ldots, d_n)$ og genkald at likelihoodfunktionen for et Markovnetværk $M(\theta)$ er givet ved

$$L(M(\theta); \mathcal{D}) = \prod_{d \in \mathcal{D}} \frac{\prod\limits_{\{A_1, \ldots, A_m\} \in \mathcal{K}} P(A_1 = d_1, \ldots, A_m = d_m)}{\prod\limits_{S = \{A_1, \ldots, A_p\} \in \mathcal{S}} P(A_1 = d_1, \ldots, A_p = d_p)^{\#_S}},$$

hvor $\mathcal{K}$ og $\mathcal{S}$ er mængden af henholdsvis kliker og separatorer.

Lad $M(\theta)$ være det oprindelige Markovnetværk og $M_{-e}(\theta)$ være det netværk, hvor kanten e er fjernet. Bemærk, at det i likelihoodfunktionen for $M_{-e}(\theta)$ ikke gør nogen forskel, om punkt 5. i Algoritme 8.0.2 er gennemført, idet punktmængder, som ikke er kliker, alligevel går ud med en identisk separator. Hvis notationen fra Algoritme 8.0.2 benyttes, bliver likelihood ratioen derfor

$$\begin{aligned}
\frac{L\left(M_{-e}(\hat{\theta}^L); \mathcal{D}\right)}{L\left(M(\hat{\theta}^L); \mathcal{D}\right)} &= \prod_{d \in \mathcal{D}} \frac{\left(\frac{P^L\left(K_{A_i}\right) P^L\left(K_{A_j}\right)}{P^L(S)}\right)}{P^L(K_e)} \\
&= \prod_{K_e} \left(\frac{\left(\frac{P^L\left(K_{A_i}\right) P^L\left(K_{A_j}\right)}{P^L(S)}\right)}{P^L(K_e)} \right)^{N(K_e)} \\
&= \prod_{K_e} \left(\frac{\left(\frac{P^L\left(K_{A_i}\right) P^L\left(K_{A_j}\right)}{P^L(S) P^L(S)}\right)}{\frac{P^L(K_e)}{P^L(S)}} \right)^{N(K_e)} \\
&= \prod_{K_e} \left(\frac{P^L\left(A_i | S\right) P^L\left(A_j | S\right)}{P^L(A_i, A_j | S)} \right)^{N(K_e)},
\end{aligned}$$

hvor produktet er over alle tilstande af variablene i K_e, og $N(K_e)$ angiver antal gange, den nuværende konfiguration er set i data. Bemærk, at når d indgår i udtrykket, sættes de pågældende realisationer lig de tilsvarende variable i sandsynlighederne. Likelihood ratio teststørrelsen bliver

$$-2\log\left(\frac{L\left(M_{-e}(\hat{\theta}^L);\mathcal{D}\right)}{L\left(M(\hat{\theta}^L);\mathcal{D}\right)}\right) = 2\sum_{K_e} N(K_e)\log\left(\frac{P^L(A_i,A_j|S)}{P^L\left(A_i|S\right)P^L\left(A_j|S\right)}\right). \tag{8.0.1}$$

Under nulhypotesen, at $M_{-e}(\theta)$ er en tilstrækkelig model, er denne teststørrelse asymptotisk chi-i-anden fordelt, hvor antal frihedsgrader er forskellen i antal frie parametre (Madsen & Thyregod 2011: s. 26). For at bestemme frihedsgraderne lad n_i og n_j være antal udfald af henholdsvis A_i og A_j. Lad desuden n_S være antal konfigurationer af separatoren S. I $P(A_i|S)$, $P(A_j|S)$ og $P(A_i,A_j|S)$ er der henholdsvis $(n_i-1)n_S$, $(n_j-1)n_S$ og $(n_in_j-1)n_S$ frie parametre. De to modeller har altså

$$\begin{aligned}(n_in_j-1)n_S-((n_i-1)n_S+(n_j-1)n_S) &= (n_in_j-n_i-n_j+1)n_S\\ &= (n_i-1)(n_j-1)n_S\end{aligned} \tag{8.0.2}$$

parametre til forskel, hvilket er antal frihedsgrader.
Bemærk, at denne likelihood ratio test faktisk er en test for, hvorvidt A_i og A_j er betinget uafhængige givet S. Dette er i tråd med Proposition 8.0.5.

Proposition 8.0.5
Lad G være en graf tilhørende et tilslutningstræ returneret af Algoritme 8.0.2. Hvis notationen fra algoritmen benyttes, går alle stier mellem A_i og A_j i G gennem S. A_i og A_j er altså betinget uafhængige givet S i det tilsvarende Markovnetværk.

Bevis. Lad T være tilslutningstræet returneret af Algoritme 8.0.2. Stier mellem A_i og A_j kan bestemmes ud fra tilslutningstræet. Dette gøres ved at vælge en klike, som indeholder A_i og en anden, som indeholder A_j. Bemærk, at A_i og A_j ikke længere indgår i samme klike på noget tidspunkt. Betragt stien mellem de to valgte kliker i T. En sti mellem A_i og A_j findes nu ved at bevæge sig enten mellem knuder i samme klike eller via knuder i en separator til en anden klike på stien i T. Det er derfor tilstrækkeligt at vise, at for ethvert gyldigt valg af to sådanne kliker går stien mellem dem i T gennem S.

Lad K_i og K_j være klikerne placeret på henholdsvis K_{A_i}'s og K_{A_j}'s plads i punkt 5. i Algoritme 8.0.2. Hvis S fjernes fra T, splittes det op i to træer T_i og T_j, hvor K_i ligger i T_i og K_j i T_j. Det er nu tilstrækkeligt at vise, at $A_i \notin T_j$, og $A_j \notin T_i$. Antag for modstrid, at

$A_i \in T_j$. Lad K_{Tj_i} være den klike i T_j, som A_i indgår i. Da er der en sti mellem K_{Tj_i} og K_i, som går gennem K_j i T. Men da T er et tilslutningstræ, giver RIP-egenskaben, at $A_i \in K_j$, hvilket er en modstrid. Et tilsvarende argument giver, at $A_j \notin T_i$. Dette beviser, at alle stier i G mellem A_i og A_j går gennem S.

■

Det giver god mening at tilpasse et tilstrækkeligt stort Markovnetværk for efterfølgende at udtynde kanterne på grund af resultatet i Proposition 8.0.7, som stammer fra Lauritzen (1996: s. 20-21). Dette benytter Lemma 8.0.6.

Lemma 8.0.6
Lad $G = (\mathcal{V}, \mathcal{E})$ være en trianguleret graf, $\widetilde{\mathcal{V}} \subseteq \mathcal{V}$ og $\widetilde{\mathcal{E}}$ være mængden af kanter fra $\mathcal{E}$, som kun involverer knuder i $\widetilde{\mathcal{V}}$. Da er $\widetilde{G} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ også trianguleret.

Bevis. Benyt Definition 3.4.2 på at være en trianguleret graf. Alle kredse i $\widetilde{G}$ er også kredse i G, og for knuderne i $\widetilde{\mathcal{V}}$ har de to grafer desuden de samme kanter. Da G er triangulær, har enhver kreds med fire eller flere knuder en genvej, og denne egenskab bevares for de tilbageværende kredse i $\widetilde{G}$, som derfor er triangulær.

■

Proposition 8.0.7
Lad $G = (\mathcal{V}, \mathcal{E})$ og $\widetilde{G} = (\mathcal{V}, \widetilde{\mathcal{E}})$ være triangulerede grafer, hvor $\widetilde{\mathcal{E}} \subset \mathcal{E}$. Da findes der en sekvens af triangulerede grafer, således $\widetilde{G} \subset G_1 \subset \cdots \subset G$, hvor hver graf kun har én kant mere end den forrige.

Bevis. Det er nok at vise, at G_1 kan findes, fordi argumentet da også gælder for sekvensen startende med G_1. Det ønskes derfor at finde en kant $e \in \mathcal{E} \setminus \widetilde{\mathcal{E}}$ således, $G_1 = (\mathcal{V}, \widetilde{\mathcal{E}} \cup \{e\})$ er triangulær. Der laves induktion over antallet af knuder. Enhver graf med 3 eller færre knuder er altid triangulær. I dette tilfælde gælder resultatet derfor trivielt. Antag, at det er sandt for en trianguleret graf med n knuder. Det vises nu, at så er det også sandt for en graf med $n + 1$ knuder.

Da $\widetilde{\mathcal{E}}$ er en ægte delmængde af $\mathcal{E}$, kan $\widetilde{G}$ ikke være komplet. Da findes der to simple knuder A_i og A_j, som ikke er naboer i $\widetilde{G}$ jævnfør Lemma 3.4.4. Dan delgraferne G_{-A_i}, $\widetilde{G}_{-A_i}$, G_{-A_j} og $\widetilde{G}_{-A_j}$, hvor henholdsvis A_i og A_j er fjernet fra den pågældende graf. Bemærk, at disse grafer har n knuder, og at de stadig er triangulerede jævnfør Lemma 8.0.6.

Hvis $G_{-A_i} = \widetilde{G}_{-A_i}$, må alle kanter i $\mathcal{E} \setminus \widetilde{\mathcal{E}}$ involvere A_i. Hvis det ligeledes gælder, at $G_{-A_j} = \widetilde{G}_{-A_j}$, må det samme gøre sig gældende for A_j. Derfor er den eneste kant i $\mathcal{E} \setminus \widetilde{\mathcal{E}}$ kanten mellem A_i og A_j, som kan sættes uden problemer, og resultatet gælder.

Antag nu, at $G_{-A_i} \neq \widetilde{G}_{-A_i}$. Da graferne stadig er triangulerede, giver induktionsantagelsen, at der findes en kant $e \in \mathcal{E}_{-A_i} \setminus \widetilde{\mathcal{E}}_{-A_i}$, som kan tilføjes til $\widetilde{G}_{-A_i}$, således denne stadig er triangulær. Det mangles nu bare at vise, at grafen stadig er triangulær, når A_i og dennes kanter i $\widetilde{G}$ igen tilføjes denne graf. Det eneste, der kan forhindre, at grafen er triangulær, er, hvis der er en kreds med 4 eller flere knuder, heriblandt A_i, uden en genvej. En sådan kreds må nødvendigvis indeholde mindst to af A_i's naboer. Bemærk, at A_i har de samme naboer som i $\widetilde{G}$, og at alle disse er forbundet, da A_i er simpel i $\widetilde{G}$. Derfor er der en genvej i kredsen, og grafen er derfor triangulær. Resultatet kunne tilsvarende vises, hvis det var $G_{-A_j} \neq \widetilde{G}_{-A_j}$. Dermed er resultatet bevist.

■

For at udtynde kanterne i et tilpasset Markovnetværk udføres en likelihood ratio test for alle kanter, som kun indgår i én klike. Hvis nulhypotesen ikke kan forkastes, vælges det at fjerne den pågældende kant. Der fortsættes, til alle mulige kanter i den resulterende graf er testet. Dette koncept formaliseres i næste afsnit.

8.1 Algoritme til udtynding af kanter

I vores **R**-pakke `tcherry` er der implementeret en funktion, `thinning_edges`, som udtynder kanterne i et Markovnetværk. Den benyttede fremgangsmåde er givet i Algoritme 8.1.1.

Algoritme 8.1.1 `Udtynding af kanter i et Markovnetværk`

`thinning_edges(cliques, separators, data, alpha, ...)`

(`cliques`: Klikerne i et tilslutningstræ, `separators`: Separatorerne i et tilslutningstræ, `data`: Et datasæt med realisationer af variablene, `alpha`: Signifikansniveau for hypotesetest, `...`: Øvrige argumenter til udregning af teststørrelse.)

1. Find alle kanter, som kun er i én klike. Gem disse og deres klike i L.
2. Lad e være den første kant i L og K_e den tilhørende klike.
3. Udregn teststørrelsen givet ved (8.0.1), hvor $S = K_e \setminus e$, og udregn frihedsgraderne givet ved (8.0.2).
4. Udregn p-værdien p via den asymptotiske chi-i-anden fordeling.
5. Hvis $p <$ `alpha`:
   (a) Gem e i en liste $L_{forkastet}$.
   (b) Sæt e til at være næste kant i L.
6. Hvis $p \geq$ `alpha`:
   (a) Fjern e, og opdater tilslutningstræet. Find på ny alle kanter, som kun ligger i én klike. Erstat elementerne i L med disse kanter og deres klike. Fjern elementer fra $L_{forkastet}$ fra L.
   (b) Sæt e til at være den første kant i L.
7. Gentag punkt 3. - 6. indtil L er tom eller testen er forkastet for alle kanter i L.

`Output`: Nabomatricen, klikerne, separatorerne, antal fjernede kanter, antal kanter i grafen.

Bemærk, at denne algoritme fjerner kanter i vilkårlig rækkefølge. Rækkefølgen, som kanterne fjernes i, påvirker hvilke kanter, der kan testes næst, hvilket kan påvirke den endelige struktur. Det kunne også vælges at regne teststørrelsen for alle mulige kanter i den aktuelle graf hver gang og herefter fjerne den med højest p-værdi, hvis den ligger over signifikansniveauet. Dette er dog ikke valgt, da det ville kræve udregning af flere teststørrelser end i Algoritme 8.1.1.

8.2 Anvendelse af algoritme til udtynding af kanter

I Tabel 7.0.2 kunne det tyde på, at de tilpassede t-kirsebærtræer overtilpasser. Derfor forsøges det at udtynde kanterne i disse. For alle modeller er BIC og antal frie parametre udregnet med funktionerne `BIC_junction_tree` og `n_params_junction_tree` fra vores **R**-pakke. Resultatet ses i Tabel 8.2.1.

AIMs				
161 variable	Funktion	2. ordens	3. ordens	4. ordens
Nøjagtighed	`k_tcherry_step`	0,8405	0,7617	0,6916
	`thinning_edges`	0,8309	0,8225	0,6643
BIC	`k_tcherry_step`	-105.959,9	-147.950,4	-329.517,9
	`thinning_edges`	-105.889,6	-107.077,9	-127.801,8
Antal parametre	`k_tcherry_step`	3.767	15.199	59.903
	`thinning_edges`	3.746	4.066	8.197

Tabel 8.2.1: Tabel over gennemsnitlig nøjagtighed fra en 10-fold krydsvalidering, BIC og antal frie parametre for t-kirsebærtræer tilpasset med `k_tcherry_step` og efterfølgende udtyndet med `thinning_edges` på **AIMs** datasættet.

Den gennemsnitlige nøjagtighed af 3. ordens t-kirsebærtræet forbedres ved udtynding, men blandt disse modeller er det stadig 2. ordens t-kirsebærtræet, der giver den største nøjagtighed. Disse har desuden også den højeste BIC og det laveste antal parametre. Dette tyder på, at det stadigvæk er en god idé at vælge en passende lav orden af t-kirsebærtræet i stedet for at tilpasse et træ af meget høj orden og efterfølgende udtynde det. Dog erindres det, at den bedste nøjagtighed i Tabel 7.0.2 var den opnået med *hill-climbing*, som var på 0,8999.

Det er desuden forsøgt at tilpasse 2. til 5. ordens t-kirsebærtræer med `k_tcherry_step` og efterfølgende udtynde disse med `thinning_edges` på de øvrige datasæt i Afsnit 7, men her er nøjagtighederne i forvejen så høje, at der ikke sker de store ændringer. Derfor analyseres i stedet datasættet **Parkinsons** fra Dua & Graff (2017) som et eksempel på, at det kan være en god idé at tilpasse et t-kirsebærtræ af højere orden og efterfølgende udtynde dette frem for at tilpasse et lavere ordens t-kirsebærtræ. Dette datasæt er til klassificering af, hvorvidt patienter har Parkinsons sygdom. Datasættet har 193 observationer og 22 forklarende variable. De forklarende variable er alle kontinuerte. De variable, der ikke kan antage negative værdier, er log-transformeret, og alle forklarende variable er herefter diskretiseret med funktionen `discretizeDF` med standardindstillingerne fra pakken `arules`. Resultatet af analysen ses i Tabel 8.2.2.

Parkinsons					
23 variable	Funktion	2. ordens	3. ordens	4. ordens	5. ordens
Nøjagtighed	`k_tcherry_step`	0,8073	0,7622	0,8277	0,7824
	`thinning_edges`	0,8163	0,8072	0,8247	**0,8966**
BIC	`k_tcherry_step`	**-4.241,25**	-4.866,22	-7.134,78	-14.205,31
	`thinning_edges`	**-4.241,25**	-4.565,99	-4.995,51	-5.143,84
Antal parametre	`k_tcherry_step`	**131**	377	1.079	3.077
	`thinning_edges`	**131**	281	431	437

Tabel 8.2.2: Tabel over gennemsnitlig nøjagtighed fra en 10-fold krydsvalidering, BIC og antal frie parametre for t-kirsebærtræer tilpasset med `k_tcherry_step` og efterfølgende udtyndet med `thinning_edges` på **Parkinsons** datasættet. De bedste resultater er markeret med fed skrift.

Når `thinning_edges` benyttes, er nøjagtighederne enten forbedret eller på samme niveau som før udtyndingen. Den højeste nøjagtighed opnås med et udtyndet 5. ordens t-kirsebærtræ, hvor der opnås en tydelig forbedring. Den højeste BIC opnås dog med et Chow-Liu-træ, selvom det for de fleste modeller giver en væsentlig forbedring af BIC at udtynde modellen.

Selvom det ikke altid giver en stor forbedring af nøjagtigheden, når der udtyndes i kanterne, bliver modellerne mere simple, da antallet af kanter mindskes. Dette betyder, at der estimeres færre parametre, og de estimerede sandsynlighedstabeller bliver mindre. Dermed bliver modellen både nemmere at fortolke, og det er hurtigere at sende information gennem netværket.

9 Konklusion

I dette speciale er der undersøgt to typer af grafer til at beskrive afhængigheder mellem variable: Orienterede acykliske grafer (bayesianske netværk) og ikke-orienterede triangulerede grafer (Markovnetværk). Formålet med begge typer af modeller er både at kunne visualisere afhængighedsstrukturer og at kunne opstille variablenes simultane sandsynlighedsfordeling med udgangspunkt i afhængighedsstrukturen. For bayesianske netværk opstilles sandsynlighedsfordelingen ved at betragte knudernes forælderrelationer, og for Markovnetværk benyttes et produkt af potentialer over grafens kliker. For at lære strukturen i bayesianske netværk har dette speciale behandlet metoderne *hill-climbing* og Chow-Liu-træer. Hvad angår Markovnetværk har fokus udelukkende været på at lære strukturen af k'te ordens t-kirsebærtræer, som eventuelt kan udtyndes efterfølgende.

Til at lære strukturen af k'te ordens t-kirsebærtræer er der i dette speciale præsenteret to former for algoritmer: Algoritmer, som udvider et k'te ordens t-kirsebærtræ til at have orden $k+1$, og algoritmer, som konstruerer et k'te ordens t-kirsebærtræ direkte fra givet data. Alle konstruktionsalgoritmer forsøger at maksimere vægten for et tilslutningstræ for grafen. Resultater har vist, at hvis vægten regnes eksakt, giver dette et k'te ordens t-kirsebærtræ med minimal Kullback-Leibler-afstand til den sande sandsynlighedsfordeling, og når vægten estimeres fra data, giver det en k'te ordens t-kirsebærtræsstruktur af maksimal likelihood. Numeriske resultater har vist, at det generelt er hurtigere at udvide et lavere ordens t-kirsebærtræ end at konstruere det direkte fra data. Et simulationsstudie har dog vist, at hvis den sande struktur er et 3. ordens t-kirsebærtræ, sker det oftere, at t-kirsebærtræerne konstrueret direkte fra data genfinder den sande struktur, og disse træer giver også en lavere Kullback-Leibler-afstand til den sande fordeling, end hvis et lavere ordens t-kirsebærtræ udvides. Studiet viste også, at t-kirsebærtræerne konstrueret direkte fra data genfandt den sande struktur i næsten 90% af tilfældende, hvorimod det samme kun gjaldt i cirka 50% af tilfældende for de udvidede 2. ordens t-kirsebærtræer.

De to former for konstruktionsalgoritmer er yderligere inddelt i to typer, alt efter måden de forsøger at maksimere tilslutningstræets vægt. Den ene type af algoritmer anvender en grådig søgning, hvor en eller eventuelt flere kliker tilføjes stepvis til modellen, og den struktur, der giver den største vægt, vælges i hvert step. Den anden type finder alle mulige strukturer og vælger strukturen med højest vægt. Grundet det høje antal af mulige strukturer er det hurtigere at benytte en grådig søgning. Generelt har et simulationsstudie vist, at de grådige algoritmer ofte finder en struktur af maksimal vægt, hvis den sande struktur er et 3. ordens t-kirsebærtræ. Hvis den sande struktur er ukendt, klarer de sig dårligere, men resultatet kan forbedres ved at tilføje flere kliker ad gangen.

Det er forsøgt at benytte k'te ordens t-kirsebærtræer på forskellige datasæt med henblik på

klassificering. Resultaterne for disse er sammenlignet med resultatet opnået med et bayesiansk netværk tilpasset med *hill-climbing* og analyser med bayesianske netværk fra Cheng & Greiner (1999). Generelt opnås der høj nøjagtighed med t-kirsebærtræer af passende orden sammenlignet med både *hill-climbing* og resultaterne fra Cheng & Greiner (1999). Dog ses der også en tendens til, at disse høje nøjagtigheder kræver et tilstrækkeligt stort datasæt i forhold til antal variable. Hvis der kun er få observationer i forhold til antal variable, ses der en tendens til at højere ordens t-kirsebærtræer hurtigt overtilpasser.

Slutteligt er der præsenteret en algoritme, som kan udtynde en trianguleret graf ved gentagen brug af en likelihood ratio test, således den resulterende graf stadig er trianguleret. Udvalgte analyser har vist, at et højere ordens t-kirsebærtræ med fordel kan udtyndes på denne måde efter tilpasningen. Dette lader til at give modeller med samme eller bedre prædiktive evner samt bedre BIC. Disse er samtidig lettere at fortolke grundet færre parametre og kanter i grafen. Resultater tyder dog også på, at det stadig er nødvendigt at vælge en passende lav orden, når t-kirsebærtræet tilpasses.

Litteraturliste

Andersen, M. M., Curran, J., de Zoete, J., Taylor, D., & Buckleton, J. (2018). Modelling the dependence structure of y-str haplotypes using graphical models. *Forensic Science International: Genetics*, 37, 29–36.

Butler, J. M. (2011). *Advanced Topics in Forensic DNA Typing: Methodology.* Academic Press.

Chang, W. & Luraschi, J. (2018). *profvis: Interactive Visualizations for Profiling R Code.* R package version 0.3.5.

Cheng, J. & Greiner, R. (1999). Comparing bayesian network classifiers. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99 (pp. 101–108). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Chow, C. K. & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions On Information Theory*, IT-14 no. 3, 462–467.

Cohen, S. (2016). *Bayesian Analysis in Natural Language Processing.* Morgan & Claypool publishers.

Cover, T. M. & Thomas, J. A. (2006). *Elements Of Information Theory.* Wiley-Interscience, second edition.

Dua, D. & Graff, C. (2017). UCI machine learning repository.

Hahsler, M., Chelluboina, S., Hornik, K., & Buchta, C. (2011). The arules r-package ecosystem: Analyzing interesting patterns from large transaction datasets. *Journal of Machine Learning Research*, 12, 1977–1981.

Hallenberg, C., Nielsen, K., Simonsen, B., Sanchez, J., & Morling, N. (2005). Announcement of population data y-chromosome str haplotypes in danes. *Forensic Science International 155*, (pp. 205–210).

Højsgaard, S. (2012). Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10), 1–26. `http://www.jstatsoft.org/v46/i10/`.

Jensen, F. V. & Nielsen, T. D. (2007). *Bayesian Networks and Decision Graphs.* Springer, second edition.

Jordan, M. I. (1999). *Learning in Graphical Models.* Kluwer Academic Publishers, first edition.

Kirkeby, K., Knudsen, M., & Vihrs, N. (2019). *tcherry: Learning the structure of tcherry trees.* R package version 0.1.1.

Kovács, E. & Szántai, T. (2008). Some improvements of t-cherry junction trees. *First International Conference on Complexity and Intelligence og the Artifacial and Natural Complex Systems.*, (pp. 117–125).

Kovács, E. & Szántai, T. (2010). On the approximation of a discrete multivariate probability distribution using the new concept of t-cherry junction tree. *Lecture Notes in Economics and Mathematical System*, 633, 39–56.

Kovács, E. & Szántai, T. (2012). Hypergraphs as a mean of discovering the dependence structure of a discrete multivariate probability distribution. *Ann Oper Res*, 193, 71–90. `https://doi.org/10.1007/s10479-010-0814-y`.

Kruskal, Jr., J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7, 48–50.

Lauritzen, S. L. (1996). *Graphical Models.* Oxford University Press.

Madsen, H. & Thyregod, P. (2011). *Introduction to General and Generalized Linear Models.* Champman & Hall.

Mersmann, O. (2018). *microbenchmark: Accurate Timing Functions.* R package version 1.4-6.

Mogensen, H., Tvedebrink, T., Børsting, C., Pereira, V., & Morling, N. (2019). Ancestry prediction efficiency of the software genogeographer using a z-score method and the 165 ancestry informative markers in the precision id ancestry panel. *Forensic Science International: Genetics.* under udarbejdelse.

Moon, J. (1969). The number of labeled k-trees. *Journal of Combinatorial Theory*, 6(2), 196 – 199.

R Core Team (2018). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. `https://www.R-project.org/`.

Rose, D. J. (1970). Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32, 597–609.

Rosen, K. H. (2012). *Discrete Mathematics and Its Applications.* McGraw-Hill, seventh edition.

Schloerke, B., Crowley, J., Cook, D., Briatte, F., Marbach, M., Thoen, E., Elberg, A., & Larmarange, J. (2018). *GGally: Extension to 'ggplot2'*. R package version 1.4.0.

Scutari, M. (2010). Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3), 1–22.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

A Bevis for at (3.2.2) er en sandsynlighedsfordeling

Proposition A.0.1
Udtrykket i (3.2.2) specificerer en sandsynlighedsfordeling.

Bevis. Bemærk, at alle faktorer i (3.2.2) er sandsynlighedsfordelinger, som altid er ikke-negative, hvorfor dette også gælder for produktet. Det mangles derfor bare at vise, at $\sum_{A_1,\ldots,A_n} P(A_1,\ldots,A_n) = 1$, hvor denne notation i summen angiver, at der summes over alle variablenes tilstande. Dette gøres ved induktion over antal variable. For $n = 1$ giver (3.2.2) at $P(A_1) = P(A_1)$, der er specificeret i det bayesianske netværk som en sandsynlighedsfordeling, så det summer til 1. Antag, at

$$\sum_{A_1,\ldots,A_{n-1}} \prod_{i=1}^{n-1} P(A_i|pa(A_i)) = 1,$$

hvor sandsynlighederne er fra et bayesiansk netværk med $n-1$ variable. Lad $A_1,\ldots,A_n$ være en topologisk ordning af knuderne i et bayesiansk netværk med n variable. Da har A_n ingen efterkommere, så

$$\sum_{A_1,\ldots,A_n} P(A_1,\ldots,A_n) = \sum_{A_1,\ldots,A_{n-1}} \prod_{i=1}^{n-1} P(A_i|pa(A_i)) \sum_{A_n} P(A_n|pa(A_n)) = 1,$$

hvor sidste lighed følger af induktionsantagelsen og det faktum, at $P(A_n|pa(A_n))$ er en sandsynlighedsfordeling for A_n. Dette færdiggør beviset for, at (3.2.2) giver en sandsynlighedsfordeling.

■

B Informationsteori

Dette appendiks sammenfatter en del definitioner og resultater fra informationsteori, som dele af specialets indhold benytter undervejs. Indholdet i dette afsnit er baseret på Cover & Thomas (2006).

Lad gennem afsnittet P være en sandsynlighedsfordeling. Det første begreb, der introduceres, er entropien for en diskret stokastisk vektor.

Definition B.0.1 (Entropi)
Lad A være en diskret stokastisk vektor. Da defineres entropien af A til

$$H(A) = -\sum_{A} P(A) \log_2(P(A)). \tag{B.0.1}$$

Det er også muligt at betragte en betinget version af entropi.

Definition B.0.2 (Betinget entropi)
Lad A og B være diskrete stokastiske vektorer. Den betingede entropi defineres ved

$$H(A|B) = -\sum_{A} \sum_{B} P(A, B) \log_2 P(A|B). \tag{B.0.2}$$

For betinget entropi gælder resultatet i Proposition B.0.3, se Cover & Thomas (2006: s. 29) for bevis.

Proposition B.0.3
Lad A, B og C være diskrete stokastiske variable. Da er

$$H(A|B) \geq H(A|B, C). \tag{B.0.3}$$

Et nyttigt mål for afstanden mellem to sandsynlighedsfordelinger er Kullback-Leibler-afstanden.

Definition B.0.4 (Kullback-Leibler-afstanden)
Lad P og Q være to sandsynlighedsfordelinger for den diskrete stokastiske vektor A. Kullback-Leibler-afstanden er givet ved

$$KL(P,Q) = \sum_A P(A) \log_2 \left(\frac{P(A)}{Q(A)} \right). \tag{B.0.4}$$

Bemærk, at $KL()$ ikke er symmetrisk, og det er derfor ikke et ægte afstandsmål. Det har dog alligevel den nyttige egenskab i Sætning B.0.5, for bevis se Cover & Thomas (2006: s. 28).

Sætning B.0.5
Lad $P(A)$ og $Q(A)$ være to sandsynlighedsfordelinger for den diskrete stokastiske vektor A. Da gælder det, at

$$KL(P,Q) \geq 0, \tag{B.0.5}$$

og

$$KL(P,Q) = 0 \quad \Leftrightarrow \quad P(A) = Q(A).$$

Kullback-Leibler-afstanden kan benyttes til at måle afhængigheden mellem to diskrete stokastiske variable via *mutual information*.

Definition B.0.6 (Mutual information for to variable)
Lad A og B være diskrete stokastiske variable. *Mutual information* mellem disse er givet ved

$$MI(A,B) = \sum_A \sum_B P(A,B) \log_2 \left(\frac{P(A,B)}{P(A)P(B)} \right). \tag{B.0.6}$$

Bemærk, at $MI(A,B) = KL\big(P(A,B), P(A)P(B)\big)$, og Sætning B.0.5 giver derfor, at $MI(A,B) = 0$, hvis og kun hvis A og B er uafhængige. Sætning B.0.7 giver nogle nyttige ligheder for *mutual information*, for bevis se Cover & Thomas (2006: s. 21).

Sætning B.0.7
Lad A og B være diskrete stokastiske variable. Da er

$$\begin{aligned}
MI(A,B) &= H(A) - H(A|B),\\
MI(A,B) &= H(B) - H(B|A),\\
MI(A,B) &= H(A) + H(B) - H(A,B),\\
MI(A,B) &= MI(B,A),\\
MI(A,A) &= H(A).
\end{aligned}$$

Angående *mutual information* for flere variable er det valgt at bruge definitionen i Kovács & Szántai (2012).

Definition B.0.8 (Mutual information for n variable)
Lad $A = (A_1, \ldots, A_n)$ være en diskret stokastisk vektor. *Mutual information* for A er givet ved

$$MI(A) = \sum_{A} P(A) \log_2 \left(\frac{P(A)}{P(A_1) \cdots P(A_n)} \right). \tag{B.0.7}$$

Per konvention sættes $MI(A_i, \emptyset) = MI(A_i) = 0$.

Ved at benytte definitionerne af $H()$ og $MI()$ og foretage nogle beregninger fås følgende lighed:

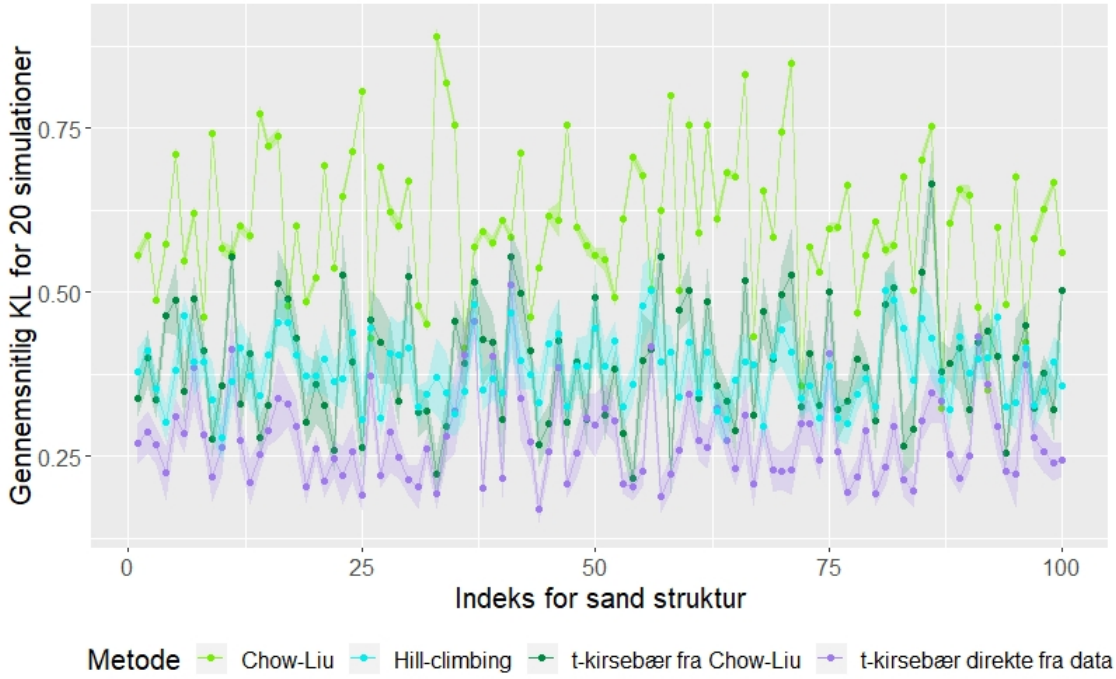
$$MI(A) = H(A_i) + MI(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n) - H(A_i|(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n). \tag{B.0.8}$$

Ved gentagen brug af (B.0.8) fås det også, at

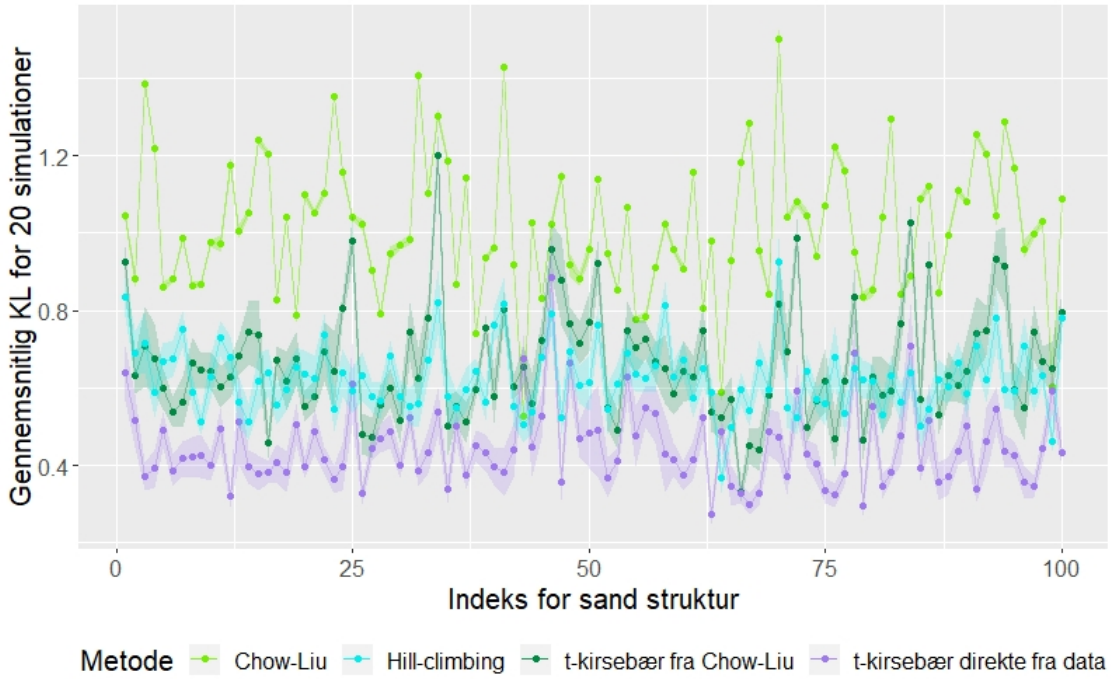
$$MI(A_1, \ldots, A_n) = \sum_{i=p+1}^{n} H(A_i) + MI(A_1, \ldots, A_p) - \sum_{i=p+1}^{n} H(A_i|A_1, \ldots, A_{i-1}), \tag{B.0.9}$$

hvor $p \in \{1, \ldots, n\}$.

C Supplerende simulationer til Afsnit 4.3

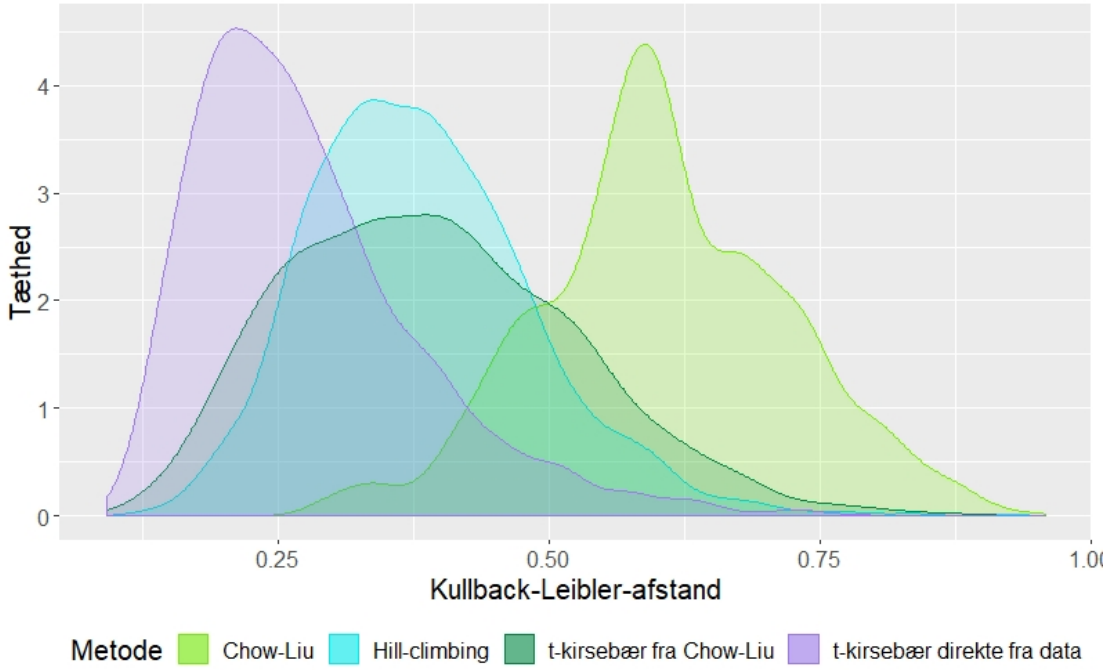


(a) Resultat af simulationsstudie for 7 variable.

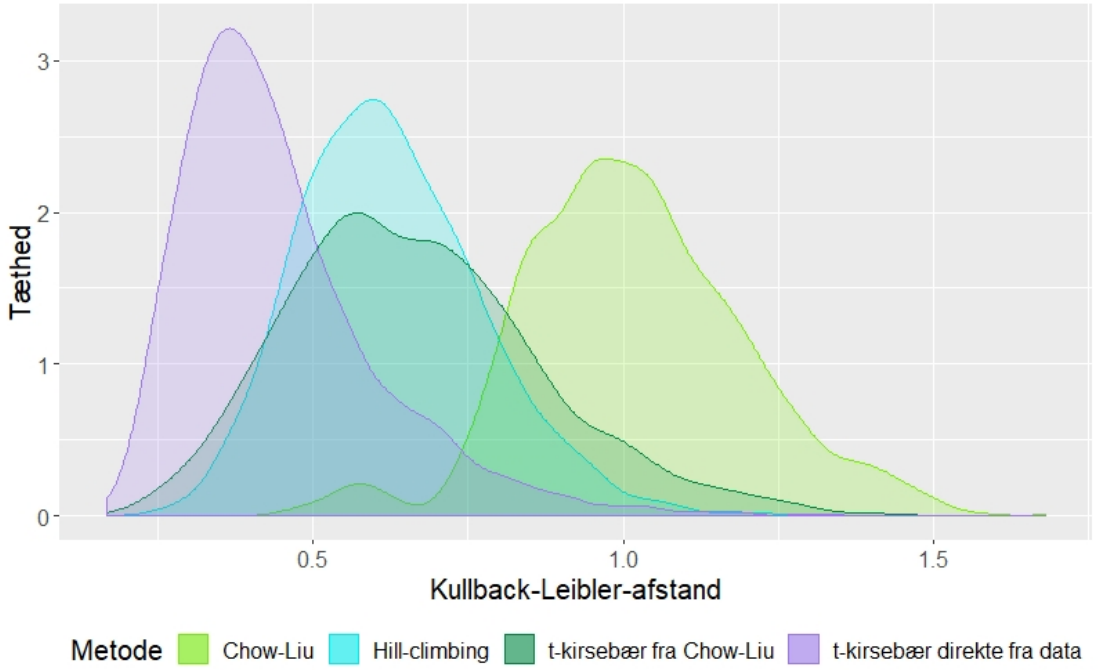


(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.1: Gennemsnitlig Kullback-Leibler-afstand mellem estimeret og sand sandsynlighedsfordeling henover 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.



(a) Resultat af simulationsstudie for 7 variable.



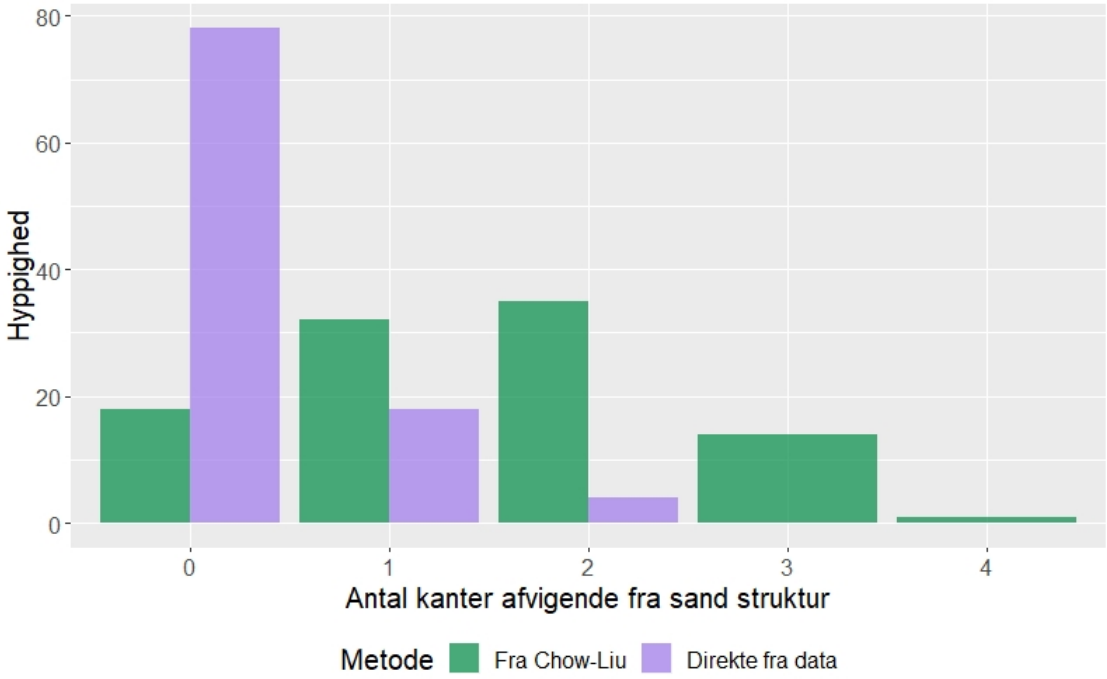
(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.2: Estimeret tæthed for alle Kullback-Leibler-afstande mellem estimerede og sande sandsynlighedsfordelinger for 100 tilfældige t-kirsebærtræer. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Disse algoritmer er for hver sand struktur benyttet på 20 simulerede datasæt.

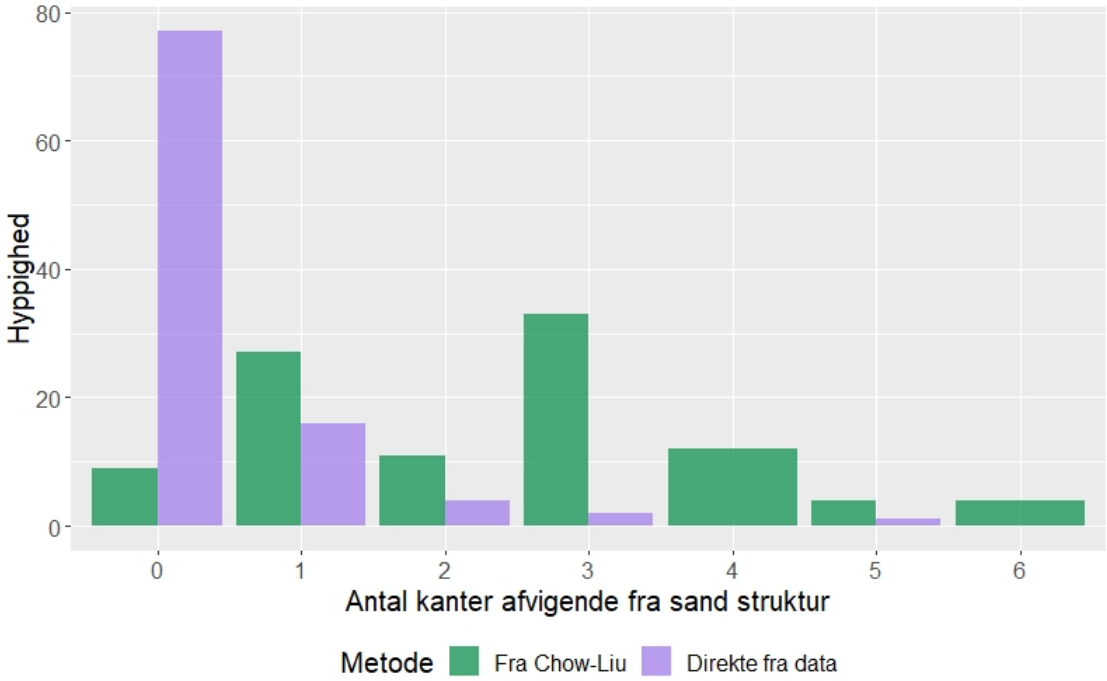
(a) Resultat af simulationsstudie for 7 variable.

(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.3: Gennemsnitlig antal fejlkanter mellem estimeret og sand struktur hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer. Den estimerede struktur er fundet med forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.



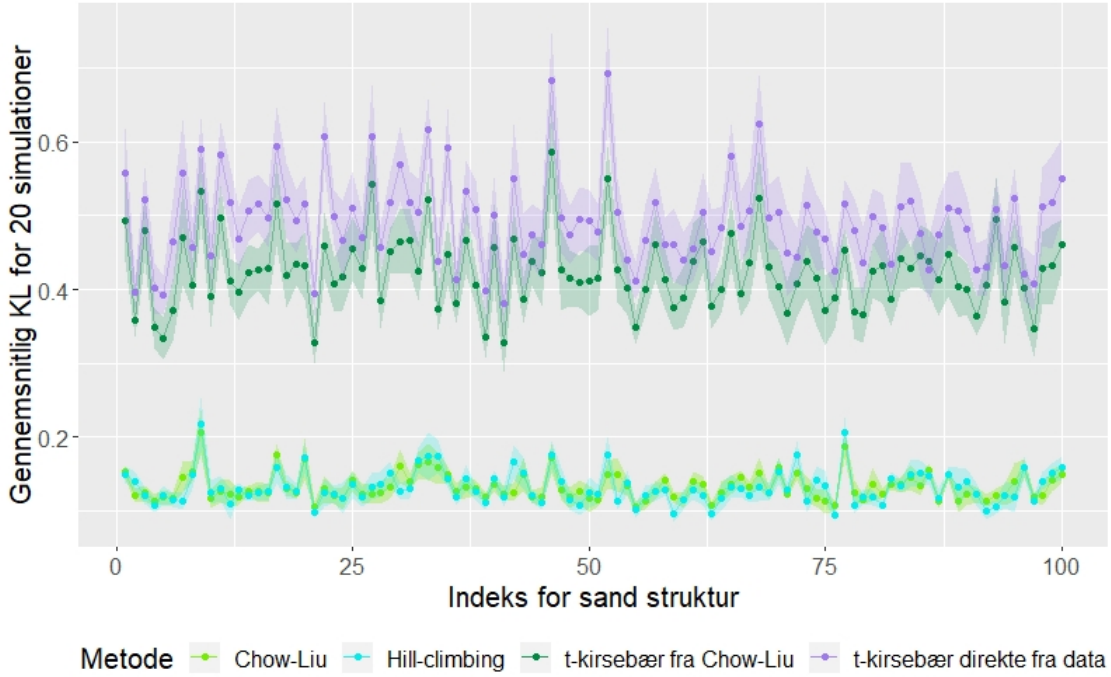
(a) Resultat af simulationsstudie for 7 variable.



(b) Resultat af simulationsstudie for 10 variable.

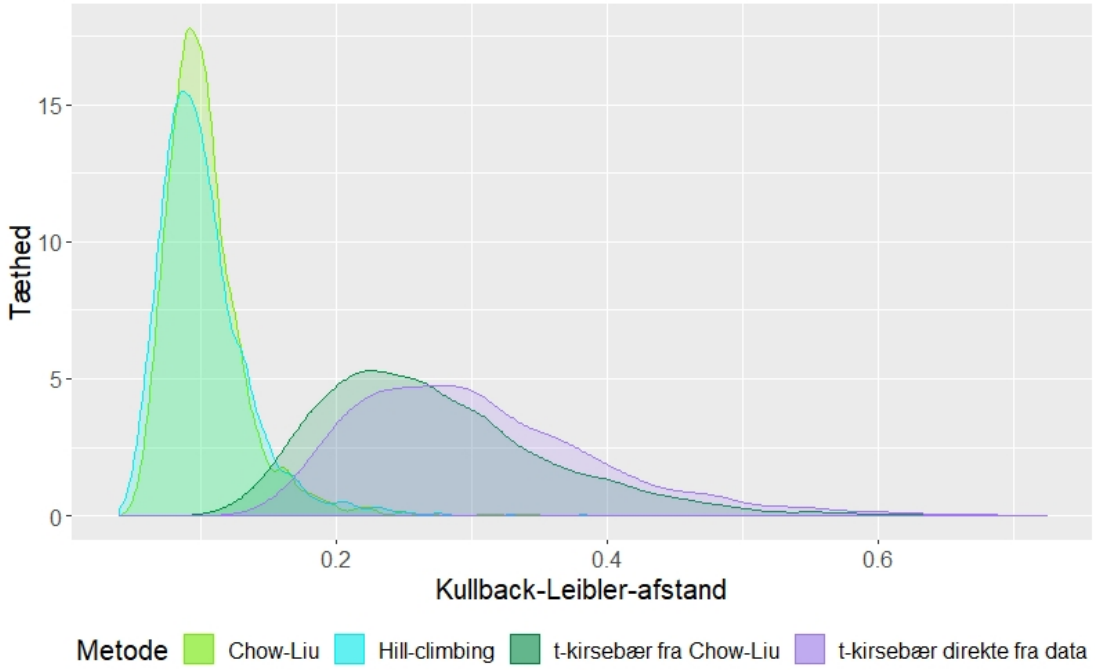
Figur C.0.4: Mest typiske antal fejlkanter for forskellige strukturer hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer. Den estimerede struktur er bestemt ved forskellige konstruktionsalgoritmer.

(a) Resultat af simulationsstudie for 7 variable.

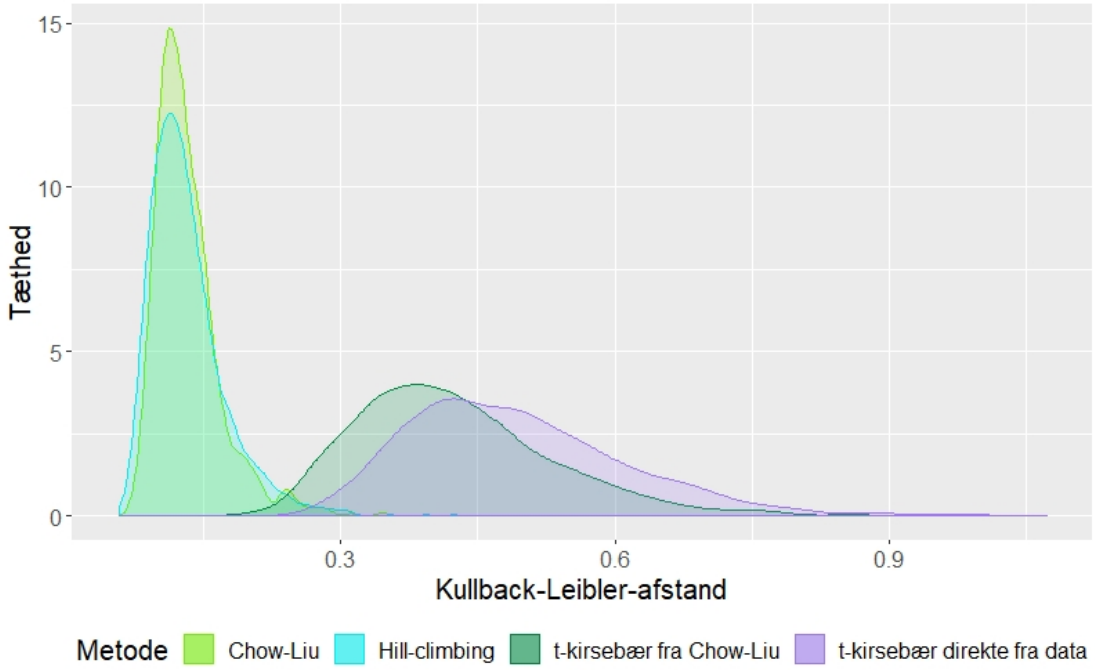


(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.5: Gennemsnitlig Kullback-Leibler-afstand mellem estimeret og sand sandsynlighedsfordeling hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med `noise` = 0,1. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.



(a) Resultat af simulationsstudie for 7 variable.



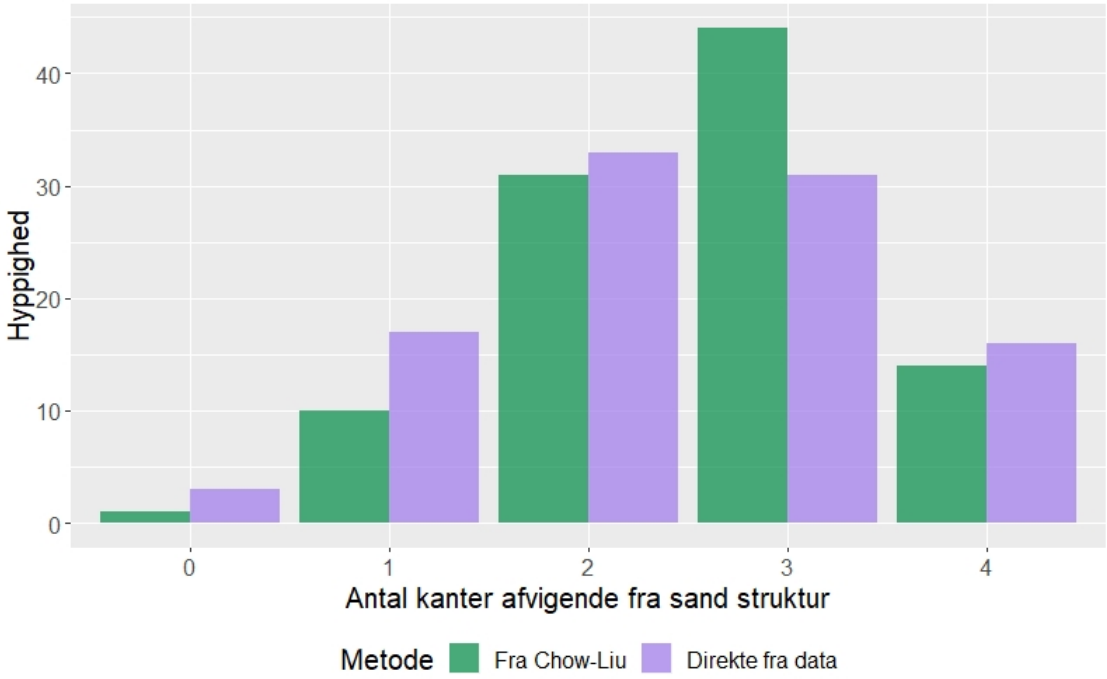
(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.6: Estimeret tæthed for alle Kullback-Leibler-afstande mellem estimerede og sande sandsynlighedsfordelinger for 100 tilfældige t-kirsebærtræer med `noise` = 0,1. Den estimerede sandsynlighedsfordeling er fundet på baggrund af grafiske strukturer bestemt ved forskellige konstruktionsalgoritmer. Disse algoritmer er for hver sand struktur benyttet på 20 simulerede datasæt.

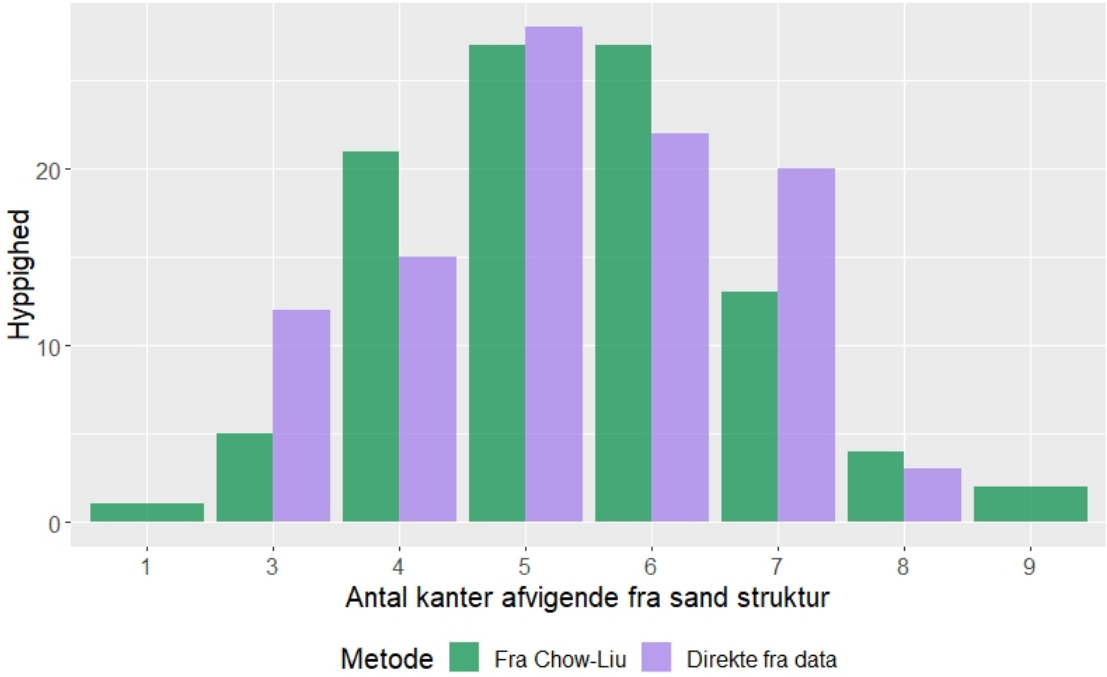
(a) Resultat af simulationsstudie for 7 variable.

(b) Resultat af simulationsstudie for 10 variable.

Figur C.0.7: Gennemsnitlig antal fejlkanter mellem estimeret og sand struktur hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med `noise` = 0,1. Den estimerede struktur er fundet med forskellige konstruktionsalgoritmer. Yderligere er der angivelse af et 95% konfidensinterval.



(a) Resultat af simulationsstudie for 7 variable.



(b) Resultat af simulationsstudie for 10 variable.

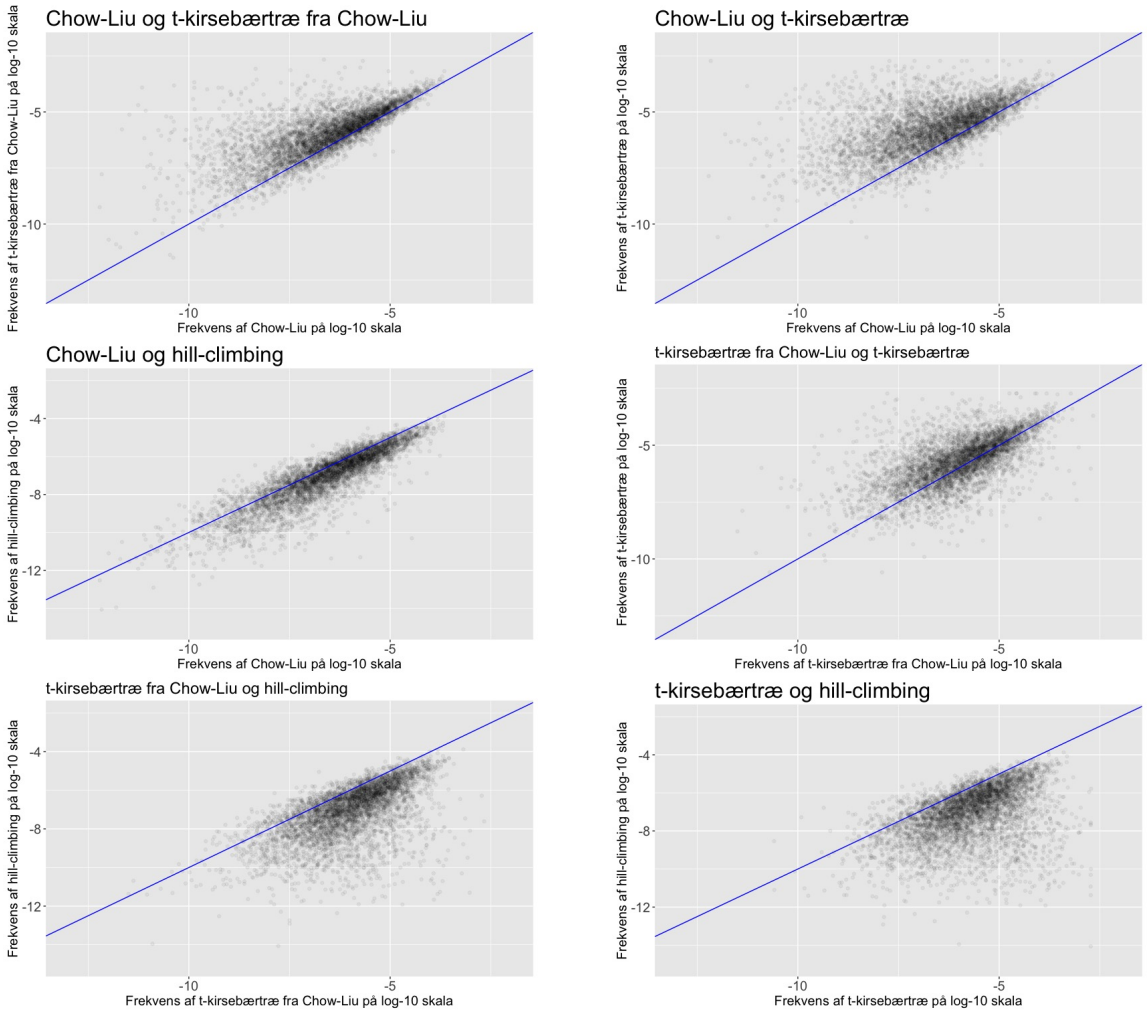
Figur C.0.8: Mest typiske antal fejlkanter for forskellige strukturer hen over 20 simulerede datasæt for 100 tilfældige t-kirsebærtræer med `noise` = 0,1. Den estimerede struktur er bestemt ved forskellige konstruktionsalgoritmer.

D Information om STR loci

Locus	Basesekvens	Allel interval
$DYS393$	$AGAT$	$8-17$
$DYS19$	$TAGA$	$10-19$
$DYS391$	$TCTA$	$6-14$
$DYS437$	$TCTR$	$13-17$
$DYS439$	$AGAT$	$8-15$
$DYS389I$	$TCTR$	$9-17$
$DYS389II$	$TCTR$	$24-34$
$DYS438$	$TTTTC$	$6-14$
$DYS390$	$TCTR$	$17-28$
$DYS385a$	$GAAA$	$7-28$
$DYS385b$	$GAAA$	$7-28$
$DYS392$	TAT	$6-20$

Tabel D.0.1: Tabel over Y-STR loci, kopieret fra Butler (2011: s. 378).

E Supplerende analyser til Afsnit 5



Figur E.0.1: Sammenligning af forskellige metoders estimerede frekvenser på simulationer af databaser af størrelse 500 med kittet PowerPlexY. Den blå linje angiver identitetslinjen.

F Oversigt over funktioner i pakken `tcherry`

`BIC_junction_tree:`
Bestemmer BIC-kriteriet for et Markovnetværk for et givet tilslutningstræ.

`ChowLiu:`
Bestemmer strukturen af Chow-Liu-træet fra data, samt de betingede sandsynlighedstabeller.

`cond_independence_test:`
Likelihood ratio test for betinget uafhængighed.

`CPT:`
Estimerer betingede sandsynlighedstabeller for bayesianske netværk, hvor strukturen er givet ved en nabomatrix.

`diff_edges_tch:`
Bestemmer antal kanter som to t-kirsebærtræer over samme variable har til forskel. Dette defineres som antal kanter, de to grafer har placeret forskelligt.

`increase_order1:`
Bestemmer strukturen af et $(k+1)$'te ordens t-kirsebærtræ ud fra et k'te ordens t-kirsebærtræ, med en grådig algoritme, der forsøger at maksimere summen af *mutual information* for klikerne.

`increase_order2:`
Bestemmer strukturen af et $(k+1)$'te ordens t-kirsebærtræ ud fra et k'te ordens t-kirsebærtræ, med en grådig algoritme, der forsøger at maksimere vægten af tilslutningstræet.

`increase_order_complete_search:`
Bestemmer ved udtømmende afsøgning strukturen af et $(k+1)$'te ordens t-kirsebærtræ med maksimal vægt ved at udvide et k'te ordens t-kirsebærtræ.

`is.acyclic:`
Tester om en ikke-orienteret graf, repræsenteret af en nabomatrix, er acyklisk.

`k_tcherry_p_lookahead:`
Bestemmer strukturen af et k'te ordens t-kirsebærtræ direkte fra data baseret på en grådig stepvis søgning. I hvert step tilføjes p nye kliker.

`k_tcherry_step:`
Bestemmer strukturen af et k'te ordens t-kirsebærtræ direkte fra data baseret på en grådig stepvis søgning, med tilføjelse af én klike ad gangen.

`kullback:`
Bestemmer Kullback-Leibler-afstanden for to diskrete sandsynlighedsfordelinger for samme variable.

`loglikelihood:`
Beregner værdien af log-likelihoodfunktionen for et Markovnetværk med et givet tilslutningstræ med de indsatte estimerede sandsynligheder.

`MI2:`
Bestemmer *mutual information* for to variable.

`MI3:`
Bestemmer *mutual information* for tre variable.

`MIk:`
Bestemmer *mutual information* for k variable.

`n_params_junction_tree:`
Bestemmer antal frie parametre i et Markovnetværk for et givet tilslutningstræ.

`random_tcherry:`
Konstruerer et tilfældigt 3. ordens t-kirsebærtræ med en tilfældig sandsynlighedsfordeling.

`tcherry_CL:`
Bestemmer stukturen af et 3. ordens t-kirsebærtræ konstrueret ud fra et Chow-Liu-træ for data.

`tcherry_complete_search:`
Bestemmer ved udtømmende afsøgning strukturen af et k'te ordens t-kirsebærtræ fra data med maksimal vægt.

`tcherry_step:`
Bestemmer stukturen af et 3. ordens t-kirsebærtræ konstrueret direkte ud fra data, baseret på en grådig stepvis fremgangsmåde.

`thinning_edges:`
Udtynding af kanter i et Markovnetværk baseret på likelihood ratio test.

`weight_junction_tree:`
Regner vægten af et tilslutningstræ.

Liste over notation

$1_{[]}$ Indikatorfunktionen.

$\#_S$ Antal gange separatoren S indgår i et tilslutningstræ.

$\cdot^B$ Estimat opnået ved bayesiansk estimation.

$\cdot^L$ Estimat opnået ved maksimum likelihood estimation.

$\hat{\cdot}$ Estimat.

$\in$ Element i.

$\notin$ Ikke element i.

$\prod \Phi$ Produktet af potentialerne i mængden Φ.

$\subset$ Ægte delmængde af.

$\subseteq$ Delmængde af.

$\sum_{A_i}$ Sum over alle A_i's tilstande.

`#konfig`(X) Antal mulige konfigurationer af variablene i mængden X.

$fa(A)$ Mængden $na(A) \cup \{A\}$ for knuden A.

$KL()$ Kullback-Leibler-afstanden.

$MI()$ Mutual information. Per konvention sættes $MI(A, \emptyset) = MI(A) = 0$.

$N(a)$ Antal datapunkter, hvor konfigurationen a indgår.

$na(A)$ Nabomængde til knuden A.

$P(A)$ Sandsynlighedsfordeling for variablen A. Per konvention er $P(\emptyset) = 1$. Hvis A er en vektor, menes den simultane sandsynlighedsfordeling.

$P(A, B)$ Simultan sandsynlighedsfordeling for variablene A og B. Per konvention er $P(A, \emptyset) = P(A)$.

$P(A|X)$ Betinget sandsynlighedsfordeling for variablen A givet en mængde af variable X. Per konvention er $P(A|\emptyset) = P(A)$.

$pa(A)$ Forældremængden til en knude A.

$|X|$ Antal elementer i mængden X.

$\text{size}(N)$ Antal uafhængige parametre i netværket N.

$\emptyset$ Den tomme mængde.