Aspects of Efficiency Improvement in Bayesian Networks

Uffe Kjærulff

AALBORG UNIVERSITY

Institute of Electronic Systems
Department of Mathematics and Computer Science

Aspects of Efficiency Improvement in Bayesian Networks

A thesis submitted to the
Faculty of Technology and Science at Aalborg University
for the degree of Doctor of Philosophy.

Uffe Kjærulff

AALBORG UNIVERSITY

Institute of Electronic Systems
Department of Mathematics and Computer Science
Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark

Preface

The objective of the PhD study, resulting in the present thesis, as formulated in the original proposal, was to address some issues of efficiency of inference in Bayesian networks. Thus, in its very foundation the study was not meant to be dealing with a single well-defined and coherent subject. Rather, the idea was to address a number of different issues which might have no other common denominator than 'Bayesian network'. The actual number of issues dealt with appeared to be three. Therefore the thesis consists of three fairly independent parts:

Part I addresses the issue of finding (sub)optimal decomposable hypergraph covers of Bayesian networks which is vital to the efficiency of inference.

Part II presents a method for approximating the probabilistic description of a Bayesian network through selective removal of edges.

Part III describes an extension of the Bayesian-network paradigm enabling the modelling of dynamic systems.

The individual parts are self-contained and not necessarily mutually consistent regarding terminology. Each of the three parts has been published either as technical reports, in conference proceedings, or in journals, as indicated in the abstract of the individual parts.

The study has been partly theoretical and partly practical. As the result of the practical part, a large number of software programs or packages have been produced. Three of these have been documented in the form of manuals or user guides which appear in Appendices A–C.

Acknowledgements

The study has been funded in part by Judex Datasystemer A/S, the company at which the study was initiated and at which I was employed for more than four years including the first year of the PhD study. I am very grateful to Judex for providing funding and access to computing equipment.

The study has been funded in part by the Danish Academy of Technical Sciences and the Danish Research Councils through the PIFT programme.

I wish to thank Finn V. Jensen for encouraging me to apply for the original scholarship, for his effort to get funding for my continued study at the Department of Mathematics and Computer Science at Aalborg University, and for many stimulating discussions and valuable comments on earlier drafts on Parts I and III.

I am indebted to Steffen L. Lauritzen whose comments and support during the preparation of Part II were indispensable, and I am grateful to his many valuable comments on earlier drafts on all three parts of the thesis.

I also wish to express my gratitude to the ODIN group at Aalborg University for providing a very stimulating environment.

From April to October 1990, I was a visiting student at Section on Medical Informatics at Stanford University, California. I wish to thank all the people there for a very stimulating and memorable experience, and for making me feel very welcome. Special thanks to Paul Dagum for many stimulating discussions and to Irene Zagazeta and her family for their hospitality and caring about the well-being of me and my family. During my visit, computing support was provided in part by the SUMEX resource, National Library of Medicine Grant # LM-05208.

Finally, I am greatly indebted to my wife Dorte for having carried more than her share of the burden of housekeeping during my four years of doctoral studies, and from whom I have never heard a single word of disapproval for spending too much time at the university.

Aalborg, Denmark
April 1993

Uffe Kjærulff

Contents

Introduction

Quantitative modelling of knowledge domains including complex stochastic phenomena was, until about a decade ago, considered inadequate when based on probability theory, and certainly, no computationally feasible methods for making inference based on such models existed. Approaches to handling uncertainty which were not based strictly on probability theory were therefore developed. Two of the most well-known examples of such approaches are certainty factors in rule-based systems (Shortliffe and Buchanan 1975) and fuzzy logic (Zadeh 1983).

Modelling of uncertainty based on probability theory has, however, experienced a remarkable renaissance in the last decade. Two factors played a key role in this successful comeback. First, a branch of statistical modelling, where emphasis on the qualitative structure is the prevailing trait, provided a graphical means of expressing conditional independence assumptions (Darroch, Lauritzen and Speed 1980, Wermuth and Lauritzen 1983). In such graphical models the (stochastic) variables are represented as vertices in a graph and the statistical dependences among the variables are represented as edges in the graph. Second, by exploiting the conditional independence properties of graphical models, the solution of the inference problem becomes computationally feasible for a large class of non-trivial models.

A number of different methods for probabilistic inference in graphical models have been proposed and implemented, and some of them have been quite successful in solving real-world problems involving hundreds of variables. The fact that most graphical models related to the discipline of knowledge engineering is based on directed, acyclic graphs is reflected in the range of methods supplied. The methods include propagation techniques (belief updating) based on the original directed graph (Kim and Pearl 1983), propagation techniques based on a related undirected graph (Cannings, Thompson and Skolnick 1976, Cannings, Thompson and Skolnick 1978, Lauritzen and Spiegelhalter 1988, Jensen, Olesen and Andersen 1990, Jensen, Lauritzen and Olesen 1990, Shafer and Shenoy 1990), propagation on a different but related directed graph (Cooper 1990a, D'Ambrosio

1989, Shachter, D'Ambrosio and Favero 1990), conditioning (Pearl 1986a), and node reductions (Shachter 1988, Shachter 1990). Except for the propagation technique of Kim and Pearl (1983), these methods are fundamentally equivalent, since they perform similar operations on a similar undirected graphical structure (Shachter, Andersen and Szolovits 1991).

In the statistical community, graphical models based on directed, acyclic graphs is often referred to as recursive graphical models (Wermuth and Lauritzen 1983). With the invention of efficient inference methods, this knowledge representation scheme has become increasingly popular in the expert systems community. Here, however, the scheme is often termed differently. The most well-known terms are influence diagram (Howard and Matheson 1984), probabilistic causal network (Cooper 1984), Bayesian belief network (Pearl 1986b), causal probabilistic network (Andreassen, Woldbye, Falck and Andersen 1987), causal network (Lauritzen and Spiegelhalter 1988). We shall use the term Bayesian network.

The graph underlying a Bayesian network captures all (conditional) independence properties of the associated probability model, say p. That is, properties of the forms: 'the sets A and B of variables are independent', or 'A and B are conditionally independent given the values of a third set, C'. This graph is therefore often called the independence graph of p.

As indicated above, all exact methods for probabilistic inference has a complexity determined by an underlying undirected graph. This graph belongs to a class of graphs known as triangulated graphs, chordal graphs, or decomposable graphs, which are all synonyms for graphs not containing cycles of length greater than three without a chord. The choice of triangulated graph is constrained only in the way that it must contain the independence graph as a subgraph. That is, the triangulated graph must define a decomposable (hypergraph) cover of the independence graph. Since there can be a vast difference between the complexities associated with different decomposable covers, the choice of cover is essential to the efficiency of inference. Part I is devoted to the issue of selecting optimal or near-optimal decomposable covers.

Even if an optimal decomposable cover is found, inference may in some cases still be intractable, since the inference problem is generally NP-hard for Bayesian networks (Cooper 1990b). Thus the need to compromise the exactitude of representation and inference arises frequently. Either more parsimonious methods (if any) for representation and inference can be employed or (further) approximations of the description can take place. In Part II we present a method of the latter kind. The method is based on approximation under edge removals (i.e. altering the conditional independence properties of a model).

A characteristic feature of many successful applications of the Bayesian paradigm is the static nature of the resulting model. That is, each observable quantity is observed at most once, and the value and influence of such an observation does not change over time. However, domains involving repeated observations of a collection of random quantities arise in many fields of science (e.g. medical, economic, biological). For such domains a static model is not very useful, since the estimation of probability distributions of domain variables based on appropriate prior knowledge and observation of other domain variables is reliable only for a limited period of time, and further, upon arrival of new observations, both these and the old observations must be taken into account in the reasoning process. Thus, to cope with such dynamic systems using Bayesian networks we need to interconnect multiple instances of static networks. Obviously, as time evolves, new 'slices' must be added to the model and old ones cut off. This introduces the notion dynamic Bayesian networks which is the subject of Part III.

Part I

Optimal Decomposable Covers

Preface to Part I

As mentioned in the introduction, Shachter et al. (1991) has shown that all exact methods for probabilistic inference in Bayesian networks have fundamentally identical complexities, since they perform similar operations on a similar underlying undirected graph. This graph belongs to a class of graphs known as triangulated graphs (also: chordal graphs and decomposable graphs). The choice of triangulated graph is constrained only in the way that it must contain the independence graph as a subgraph. That is, the triangulated graph must define a decomposable (hypergraph) cover of the independence graph. Since there can be a vast difference between the complexities associated with different decomposable covers, the choice of cover is essential to the efficiency of inference.

We consider algorithms for finding decomposable covers yielding (near) minimum complexity. Wen (1990) has shown that the problem of finding a minimum decomposable cover is generally NP-complete. In Chapter 1 we consider various heuristic algorithms and in Chapter 2 a method based on a Monte-Carlo technique, known as simulated annealing, is presented.

A software program including most of the deterministic algorithms of Chapter 1 and the simulated annealing algorithm of Chapter 2 has been constructed. A user's guide (Kjærulff 1993e) to the program is included in Appendix A.

Chapter 1

Heuristic Search Algorithms

This chapter includes a survey and evaluation of existing triangulation algorithms most of which are found to be rather useless w.r.t. finding triangulations yielding minimum or near minimum state space. Simple heuristic methods are presented which in practice are found to produce triangulations yielding near-minimum solutions. Moreover, we present an efficient method for rendering a non-minimal triangulation minimal.

The contents of this chapter is identical to Chapters 1 and 2 in Kjærulff (1990).

1.1 Background

The notion of triangulated (chordal, decomposable, ...) graphs applies to problem solving within as widely different areas as solution of sparse symmetric systems of linear equations, pedigree analysis, and evidence propagation in Bayesian networks. Our interest in graph triangulation originates from the latter of these areas. The evidence propagation technique we shall be referring to is the one described by Jensen, Lauritzen and Olesen (1990) which is based on the work of Lauritzen and Spiegelhalter (1988). This technique is based on an underlying triangulated graph of the network. The present paper deals with the problem of achieving approximate optimal triangulations of (undirected) graphs. In the context of Bayesian networks the notion of optimality relates to the sum total of the state space sizes of the cliques of the triangulated graph. This optimality task has been proved to be NP-complete (Wen 1990).

The basic technique applied to triangulate a graph G is to add extra edges T produced by eliminating the vertices of G one by one. (A vertex ν is eliminated by adding edges such that the vertices adjacent to ν are pairwise adjacent and by subsequent deletion of v and its incident edges.) This technique is not guaranteed to produce minimal triangulations neither in terms of number of edges nor in terms of the size of the state space when the vertices are selected at random. Furthermore, in average these quantities are typically far above those of a minimum triangulation.

Triangulating graphs using the elimination technique is essentially a problem of establishing a sequential ordering of the vertices specifying the order in which they should be eliminated. Hence the term *elimination ordering* (or *sequence*) refers to such an ordering. Several algorithms have been suggested for establishing elimination orderings which have certain desired properties. As the task of establishing an elimination ordering may be perceived as a sorting problem such algorithms are quite often referred to as *sorting algorithms*. Since most of the algorithms apply some sort of search strategy the term *search algorithms* is also frequently used.

1.2 Terminology

A graph G consisting of a finite set of *vertices* V and a set of *edges* E (vertex pairs from V) is denoted $G = (V, E)$. Throughout this paper we shall use $n = |V|$ to denote the number of vertices and $e = |E|$ to denote the number of edges. Let A be a set of vertices and B be a set of edges. Then $G(A)$ and $G(B)$ denote the subgraphs of G induced by A and B, respectively. $V(B)$ ($E(A)$) denotes the subset of vertices (edges) induced by B (A). An *ordering* of V is a bijection $\# : V \leftrightarrow \{1, 2, \ldots, n\}$. $G_\# = (V, E)$ is an *ordered graph*.[1] A graph G is called a *belief graph*, if a finite set of elements, called *states*, is attached to each vertex in G. The base 2 logarithm of the number of states (state size) of a vertex v is denoted $w(v)$ (the *weight* of v).

Two vertices $v, w \in V$ are *adjacent* (or *neighbours*) if $\{v, w\} \in E$. The set

$$\mathrm{adj}(v, G) = \{w \in V \mid \{v, w\} \in E\}$$

is the vertices adjacent to v in G. The set of vertices *monotonely adjacent* to v in G is defined by

$$\mathrm{madj}(v, G) = \mathrm{adj}(v, G) \cap \{w \in V \mid \#(v) < \#(w)\}$$

A *complete subgraph* of G is a set of vertices all of which are pairwise adjacent. A *clique* is a maximal complete subgraph of G. A vertex $v \in V$ is said to be *simplicial* if $\mathrm{adj}(v, G)$ induce a complete subgraph of G (which then together with v form a clique in G).

[1] When it appears clearly from the context that a graph is ordered we shall often drop the #-index.

A vertex $v \in V$ is said to be *eliminated* if the adjacency set of v is made a complete subgraph of $G = (V, E)$ and v and its incident edges are removed from G. Let $G_i = (V_i, E_i)$, where $i = 1, \ldots, n$, denote the ordered graph obtained by eliminating in order the vertices $\#^{-1}(1), \ldots, \#^{-1}(i-1) \in V$ from the ordered graph $G = (V, E)$, where

$$\begin{aligned} V_i &= \{v \in V \mid \#(v) > i\} \\ E_i &= \{\{v, w\} \in E_{i-1} \mid v, w \in V_i\} \cup \\ &\quad\; \{\{v, w\} \subseteq V_i \mid v, w \in \mathrm{madj}(u), \text{ where } \#(u) = i\} \\ E_0 &= E \end{aligned}$$

By $C_i(G)$ we denote the clique, if any, produced by eliminating vertex $\#^{-1}(i)$ in G_{i-1}. If $C_i(G)$ is a clique it consists of the set $\{\#^{-1}(i)\} \cup \{\mathrm{adj}(\#^{-1}(i), G_{i-1})\}$.

A *path* in $G = (V, E)$ is a sequence of distinct vertices $\langle v_1, \ldots, v_k \rangle$ such that $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, k-1$. A *cycle* is a path $\langle v_1, \ldots, v_k \rangle$ with the exception that $v_1 = v_k$ and such that $k > 3$. A *chord* in a cycle is an edge $\{v_i, v_j\}$ such that $1 < |i - j| < k - 2$ (i.e. $\{v_i, v_j\}$ connects the two non-consecutive vertices v_i and v_j in the cycle). G is *triangulated* (or *chordal*) if any cycle of length greater than 3 has a chord.

Let $G_\# = (V, E)$ be an ordered graph. Then G^t denotes the triangulated graph obtained by eliminating the vertices of G in the order defined by $\#$. G^t has vertex set V and edge set $E \cup T$, where the set T of *fill edges* is a *triangulation* of G. A fill edge $f \in T$ is *redundant* if $T \setminus \{f\}$ is also a triangulation of G. T is *minimal* if for any $T' \subset T$ (proper containment) $G' = (V, E \cup T')$ is not triangulated. T is *minimum* if for any triangulation T' of G, $|T| \leq |T'|$. $T_i(G)$ denotes the fill edges produced by eliminating vertex $\#^{-1}(i)$ in G_{i-1} and

$$T(G) = \bigcup_{i=1}^{n} T_i(G)$$

Note that $G^t = (V, E \cup T(G))$.

Let $C = \{v_1, \ldots, v_k\}$ represent a clique of a triangulated graph G. The *size* of C is $s(C) = k$ and the *weight* (base 2 logarithm of the state size) of C is

$$w(C) = \sum_{i=1}^{k} w(v_i)$$

The size of G, $s(G)$, is the sum of the sizes of all of its cliques, and the weight of G is

$$w(G) = \log_2 \sum_{C} 2^{w(C)}$$

For $C \subseteq V$,

$$\mathrm{adj}(C,G) = \{v \mid v \in V \setminus C, v \in \bigcup_{w \in C} \mathrm{adj}(w,G)\}$$

denotes the set of vertices separating G into components C and $V \setminus (C \cup \mathrm{adj}(C,G))$.

Throughout this paper we assume that the graphs considered are *connected* (i.e. for each pair of distinct vertices v, w there is a path $\langle v, \ldots, w \rangle$). Note that connectivity is invariant under elimination.

1.3 Well-known Triangulation Methods

Rose, Tarjan and Lueker (1976) provided the following useful characterisation of the fill edges produced by the elimination technique.

Theorem 1.1 (Rose et al. (1976)) *Let $G = (V, E)$ be an ordered graph. Then $\{v, w\}$ is an element in $T(G)$ if and only if there is a path $\langle v, v_1, \ldots, v_k, w \rangle$ in G such that $\#^{-1}(v_i) < min(\#^{-1}(v), \#^{-1}(w))$ for all $1 \leq i \leq k$.*

Note that for $k = 0$ (i.e. $\{v, w\} \in E$) the condition is regarded as satisfied. From that theorem we may formulate the following corollaries.

Corollary 1.1 *Let $G = (V, E)$ be an ordered graph and $\langle v_1, \ldots, v_k, w \rangle$ be a path in G, where $\#^{-1}(v_i) < \#^{-1}(w)$ for all $i \leq k$. Then $N \setminus \{v_1, \ldots, v_k\} \subseteq adj(w, G^t)$, where $N = \bigcup_{v \in \{v_1, \ldots, v_k\}} adj(v, G)$.*

Proof: Follows directly from Theorem 1.1. □

Corollary 1.2 *Let $G_{\#_a} = (V, E)$ and $G'_{\#_b} = (V, E)$ be ordered graphs, where $\#_a^{-1}(i+1) = \#_b^{-1}(i+1)$. Then $G_i = G'_i$ for all $k \leq i \leq n$, where $0 \leq k \leq n$.*

Proof: Clearly G_i and G'_i consist of the same vertex set, and from Theorem 1.1 it is equally clear that they also have identical edge sets. □

It appears from Theorem 1.1 that in order to profit by it in establishing an elimination ordering (e.g. to keep the size of T as small as possible) the ordering should be constructed backwards. The two most well-known ordering algorithms (lexicographic search by Rose et al. (1976) and maximum cardinality search by Tarjan and Yannakakis (1984)) are based on this principle.

The basic idea in the backwards approach is to separate the vertices already ordered in a way such that there is no paths between them of the type described above. The maximum cardinality search does that by continuously selecting a vertex (to be ordered next) with the highest number of ordered neighbours. The lexicographic search does something similar, but in addition it is guaranteed to produce minimal triangulations (in $O(ne)$ time), whereas the maximum cardinality search is not. Maximum cardinality search has the property of not adding edges to an already triangulated graph and hence may be used as an efficient test for graph chordality as it runs in $O(n + e)$ time.

Fujisawa and Orino (1974) have described an extended elimination technique which does not necessarily add fill edges between all non-adjacent vertices in the adjacency set of a vertex being eliminated. This extended elimination strategy produces minimal triangulations no matter which ordering of vertices is employed.

By comparing the ordinary elimination technique with theirs it appears that there are two distinct cases in which the ordinary technique fails to produce minimal triangulations. The two cases are described below.

Let $G = (V, E)$ be a graph and let $v \in V$ be the vertex to be eliminated next.

> **Case I.** $\{v\} \cup \mathrm{adj}(v, G) = V$. If T is a triangulation of $G' = G \setminus \{v\}$ then it is also a triangulation of G. Hence we need not complete $\mathrm{adj}(v, G)$ when v is eliminated, it suffices to just remove v from G.
>
> **Case II.** (See Figure 1.1) $\mathrm{adj}(v, G)$ separates G into components $\{v\}$, $C_1, \ldots, C_m$, $m > 1$. If
>
> $$\begin{aligned} G_0 &= (\mathrm{adj}(v, G), E_0), \\ G_1 &= (C_1 \cup \mathrm{adj}(C_1, G), E_1), \\ &\vdots \\ G_m &= (C_m \cup \mathrm{adj}(C_m, G), E_m) \end{aligned}$$
>
> are triangulated so is $G'' = (V, E \cup T)$, where $T = \bigcup_{i=0}^{m} E_i \setminus E$ is a triangulation of $G' = \bigcup_{i=0}^{m} G_i$. Again we need not complete $\mathrm{adj}(v, G)$, it suffices to complete each $\mathrm{adj}(C_i, G)$.

Fujisawa and Orino (1974) have proved that by extending ordinary vertex elimination by the results from these two observations any elimination ordering produces a minimal triangulation. More precisely they have formulated and proved the following theorems.

Theorem 1.2 (Fujisawa and Orino (1974)) *Let v be a vertex of a connected graph $G = (V, E)$ such that $\{v\} \cup adj(v, G) = V$, and let T be a minimal triangulation of the section graph $G \setminus \{v\}$. Then T is a minimal triangulation of G.*

Theorem 1.3 (Fujisawa and Orino (1974)) *Let v be a vertex of a connected graph $G = (V, E)$ such that $\{v\} \cup adj(v, G) \subset V$, and let $adj(v, G)$ be a separator with*

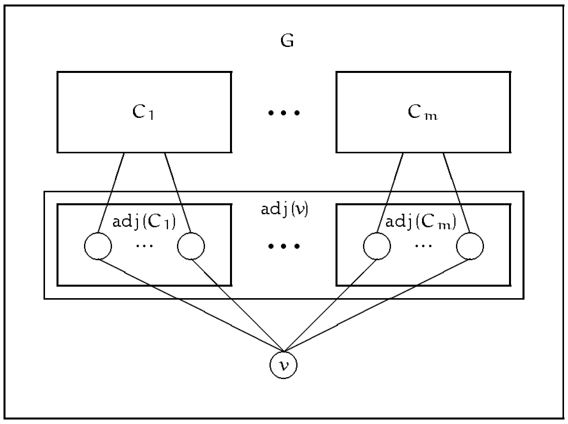

Figure 1.1: By eliminating v we need not complete adj(v), it suffices to complete each adj(C_i).

components $\{v\}, C_1, \ldots, C_m$*,* $m \geq 1$*. Furthermore, let* F *be the set of edges needed to make each adj*(C_i, G) *a complete subgraph and let* $G \setminus \{v\}$*. If* T *is a minimal triangulation of* G' *then* $T \cup F$ *is a minimal triangulation of* G*.*

Based on these theorems they have formulated an $O(n(e + f))$ time algorithm for finding a minimal triangulation T of a graph $G = (V, E)$, where $f = |T|$.

Both the lexicographic search and the extended elimination algorithms produce minimal orderings, but such orderings are not necessarily close to a *minimum fill*[2] ordering. On the other hand a minimum fill ordering is not necessarily a minimum weight ordering. If, for example, we let the solid circles in Figure 1.2 represent vertices with large relative weights then only (a) is a minimum weight triangulation, whereas both (a) and (b) are minimum fill triangulations. However, we claim that a triangulation T_1 is very unlikely to have weight less than a triangulation T_2 if $|T_1|$ is significantly greater than $|T_2|$. As will be demonstrated in Section 1.5 this claim seems empirically reasonable.

Depending on the problem to be solved by graph triangulation, optimality may be defined differently. Three typical candidates are

- minimum fill
- minimum size

[2]We shall use the terms minimum fill, minimum weight, and minimum size triangulations as short for triangulations consisting of a minimum number of fill edges, triangulations producing triangulated graphs with minimum weight, and triangulations producing triangulated graphs for which the sum of the clique sizes is minimum, respectively.

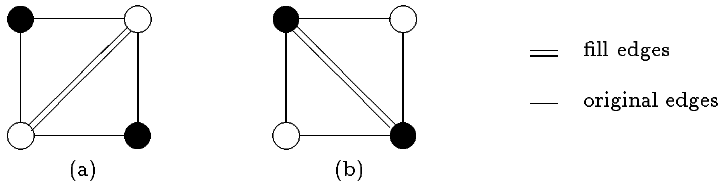

Figure 1.2: Both of the triangulations (a) and (b) are minimum fill, but only (a) is minimum weight (solid circles represent vertices with large relative weights).

- minimum weight

In order to meet these optimality criterions using the elimination technique the most immediate strategy would be to successively choose the next vertex to be eliminated to be one which produces as few fill edges as possible, the smallest clique, or the clique with the least weight, respectively. Henceforth we will call the algorithms based upon these three heuristic rules the *minimum fill*, the *minimum size*, and the *minimum weight* heuristic, respectively. Rose (1973) described the first two of these heuristics under the names *minimum deficiency* and *minimum degree* algorithms and referred to other papers (e.g. Sato and Tinney (1963) and Tinney and Walker (1967)) in which it is assumed that these algorithms produce near optimal orderings.

As noted by Rose (1973) the minimum size heuristic is fast (can be implemented to run in $O(n+e')$ time, where $e' = e+|T|$), but has some disadvantages:

- does not, in general, produce a perfect ordering[3] if the graph is triangulated
- does not, in general, produce minimal triangulations
- there exist examples for which the triangulation produced is arbitrarily greater than a minimum (fill) triangulation

However, this last feature does not really count as a disadvantage in the context of belief networks as we do not care about the number of fill edges.

The minimum fill heuristic has the advantage of producing a perfect ordering when the graph is triangulated, but has the following disadvantages:

- slightly slower than the minimum size heuristic, because the adjacency set of each vertex must be edge tested

[3]If $G_\# = (V, E)$ is an ordered graph then $\#$ is a perfect ordering if $T(G_\#) = \emptyset$. Any triangulated graph $G'_\# = (V, E \cup T(G_\#))$ has a perfect ordering, since $\#$ is a perfect ordering of this graph.

- does not, in general, produce minimal triangulations

The minimum weight heuristic has exactly the same advantages and disadvantages as the minimum size heuristic. Note that if all vertices have equal weights these two heuristics are identical.

As the complexity of finding a minimum triangulation grows like $n!$ we are not able to run a straightforward exhaustive search, except for very small n. However, by constructing an elimination ordering successively and cutting off when the sum of the weights of the cliques produced so far exceeds the current least weight of a complete ordering we might be able to run an exhaustive search even for graphs of moderate sizes. Usually this cutting and branching technique is called *branch-and-bound*. As we are dealing with an NP-complete problem we should not, in general, expect a branch-and-bound algorithm to find an optimal ordering within certain acceptable time bounds. That is, the algorithm must terminate prematurely when the number of vertices exceeds a certain limit or when the number of permutations cut off grows too slowly. Of course, the initial ordering heavily influences the success of this algorithm. So a branch-and-bound algorithm should preferably be cascaded (as the last element) with other faster ordering algorithms to set-up a "good" initial ordering for it in order to avoid checking too many useless orderings and in order to minimise the distance to some minimum ordering (assuming that a low cost ordering is closer to a minimum one than a high cost ordering is).

1.4 Minimal Triangulations by Recursive Thinning

When an elimination ordering has been established and the corresponding triangulation, T, of a graph, G, has been determined it might turn out that there is a subset, $T' \subset T$, such that T' is also a triangulation of G. For the minimum fill criterion T' is, of course, strictly better than T. For the minimum weight criterion T' is no worse than T, and most often T' is significantly better than T. But for the minimum size criterion T is normally better than T'. The fact that T' is most often better than T for the minimum weight criterion will become evident in the following.

In order to develop an algorithm to remove redundant fill edges we first characterise a minimal triangulation in the following way.

Theorem 1.4 (Rose et al. (1976)) *Let $G = (V, E)$ be a graph and $G' = (V, E \cup T)$ be triangulated. Then T is minimal if and only if each edge in T is a unique chord of a 4-cycle in G'.*

An equivalent formulation of Theorem 1.4 is provided by the following corollary.

Corollary 1.3 *Let* $G = (V, E)$ *be a graph and* $G' = (V, E \cup T)$ *be triangulated. Then* T *is minimal if and only if for each edge* $\{v, w\} \in T$ *there is a pair of distinct vertices* $\{x, y\} \subseteq adj(v, G') \cap adj(w, G')$ *such that* $\{x, y\} \notin E \cup T$.

Proof: As $\{v, w\}$ is a chord of a 4-cycle $\langle v, x, w, y, v\rangle$ in G', x and y will both be adjacent to both v and w in G'. By the uniqueness of the $\{v, w\}$ chord it follows that $\{x, y\} \notin E \cup T$. □

By formulating Theorem 1.4 that way we easily get the next corollary.

Corollary 1.4 *Let* T *be a triangulation of a graph* $G = (V, E)$, *let* $G' = (V, E \cup T)$ *be the corresponding triangulated graph, and let* $\{v, w\} \in T$. *Then* $T \setminus \{\{v, w\}\}$ *is a triangulation of* G *if and only if the set of vertices* $N = adj(v, G') \cap adj(w, G')$ *induces a complete subgraph* $G'(N)$ *of* G'.

Proof: Let us first prove that if $T \setminus \{\{v, w\}\}$ is a triangulation of G then $G'(N)$ is complete. By assuming $G'(N)$ is not complete (e.g. $x, y \in N$ are non-adjacent) we get the unchorded 4-cycle $\langle v, x, w, y, v\rangle$ which contradicts the definition of triangulated graphs.

Secondly we shall prove that completeness of $G'(N)$ implies that $G'(E \cup T \setminus \{\{v, w\}\})$ is triangulated. Assuming $G'(E \cup T \setminus \{\{v, w\}\})$ is not triangulated there must be an unchorded cycle of length 4 in it because if it had length > 4 there would be an unchorded cycle of length > 3 before $\{v, w\}$ was removed. As G' is triangulated this cycle must include both v and w and as the cycle has length 4 it must be a $\langle v, x, w, y, v\rangle$ cycle, where $x, y \in N$. But as $G'(N)$ is complete the cycle has a chord. □

Notice that Theorem 1.4 and Corollary 1.3 characterises the fill edges of a minimal triangulation $T' \subseteq T$, whereas Corollary 1.4 characterises the fill edges in $T \setminus T'$. Thus an alternative proof of Corollary 1.4 could be based on the results of Corollary 1.3.

The above propositions tell us that the redundant edges of a triangulation are characterised by not being unique chords of 4-cycles. Note that this set of edges is identical to the set $\{e_i\}$ for which all e_i is a proper subset of a unique clique in the graph. To be precise we formalise this observation as:

Lemma 1.1 *Let* $G = (V, E)$ *be a triangulated graph, let* $E_{\overline{u}} = \{e \in E \mid e$ *is not a unique chord of a 4-cycle in* $G\}$, *and let* $\mathcal{C} = \{C_i\}$ *be the set of cliques in* G. *Then for all* $e \in E_{\overline{u}}$ *there is a unique* $C \in \mathcal{C}$ *such that* $e \subset C$.

Proof: Assume $e = \{v, w\} \in E_{\overline{u}}$ is a subset of more than one clique. Then there is an unchorded cycle $\langle v, \ldots, w, \ldots, v\rangle$ of length greater than 3. Thus the graph is no longer triangulated which contradicts the fact that e is redundant. □

As a redundant (fill) edge, $e = \{v, w\}$, can only be subset of a single clique, C, it is now clear that this clique splits into two new cliques $C_1 = C \setminus \{v\}$ and $C_2 = C \setminus \{w\}$ the sum of weights of which is typically significantly less than the weight of C and in any case no worse than that.

A non-minimal triangulation T may become minimal by simply dropping the edges in T that do not fulfil the conditions of Corollary 1.3. However, it does not suffice in general to sweep through the edges just once, which is illustrated by the following example. In Figure 1.3 the vertices b and c have a common pair of non-adjacent neighbours (i.e. $\{b, c\}$ is a unique chord of the 4-cycle $\langle a, b, c, d, a \rangle$) and hence we are not allowed to remove edge $\{b, c\}$. The vertices c and d do not have such a pair in common and we may remove edge $\{c, d\}$. Examining the reduced graph we now find that $\{b, c\}$ may be removed.

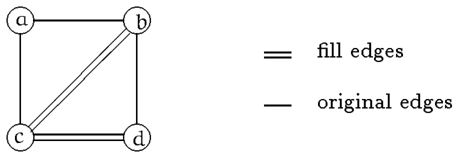

Figure 1.3: A non-minimal triangulation produced by the elimination ordering a, b, c, d.

Clearly for a fill edge which in a previous sweep was found to be a unique chord of a 4-cycle to become redundant, according to Corollary 1.4, one of the four edges in that 4-cycle must have been removed. So, assuming it takes k sweeps to minimise a triangulation, for $i = 2, \ldots, k$ it suffices in sweep i to test the fill edges which intersect (i.e. share an end-point (vertex)) with an edge removed in sweep $i - 1$. Depending on the number of sweeps required this observation may substantially improve the minimisation process.

Based on Corollary 1.4 and the above discussion we may formulate the following $O(c^2|T|^2)$ time[4] algorithm which removes redundant fill edges from a non-minimal triangulation T.

MINT $(T, G = (V, E \cup T), R)$ (initially $R = T$)
    $R' = \{e_1 \in T \mid \exists e_2 \in R : e_1 \cap e_2 \neq \emptyset\}$
    $T' = \{\{v, w\} \in R' \mid G(\mathrm{adj}(v, G) \cap \mathrm{adj}(w, G))$ is complete$\}$
    if $T' \neq \emptyset$ **then**
        return (MINT $(T \setminus T', G = (V, E \cup T \setminus T'), T')$)
    else
        return (T)

[4] The constant c expresses the density of the graph and equals the cardinality of the intersection of the adjacency sets of the vertices which constitute a fill edge.

Obviously the number of sweeps required heavily depends on the order in which we arrange the fill edges and will typically be much less than $|T|$. The clearest example appears in the generalisation of the graph in Figure 1.3 (a *string graph*[5] triangulated by successively eliminating vertices in the middle of the string), where only one fill edge may be removed in each sweep if the edges are ordered according to the order in which they have been generated, whereas all of them may be removed in just one sweep if they are investigated in the reverse order. This gives us the clue of a considerably more efficient minimisation algorithm. The existence of such an algorithm is verified by the proof of Theorem 1.5. However, before presenting this theorem let us have a closer look at the rationale leading to it.

An immediate idea emerging from the above discussion is: Is one sweep sufficient in general provided that the fill edges are ordered appropriately? There are two obvious ways of ordering the fill edges of a triangulated graph. The first being the order of creation (i.e. corresponding to the elimination ordering) and the second being the reversed of the first. As became clear from the string graph example above the first ordering strategy is not applicable, whereas the second could be. Notice that these orderings are not uniquely defined as the fill edges are created in groups (one group, possibly empty, for each vertex elimination). There seems not to be any obvious way of ordering fill edges belonging to the same group. So in order for reversed orderings to be minimised in one sweep it is a requirement that each group of edges can be minimised in one sweep no matter how they are ordered. The example in Figure 1.4 shows that this requirement cannot be satisfied. The elimination of vertex a produces three fill

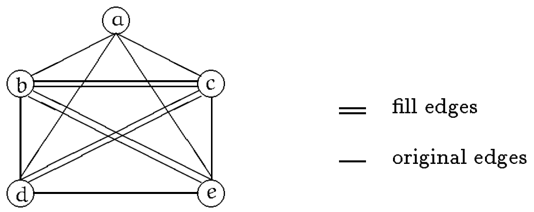

Figure 1.4: The group of fill edges generated by the elimination of a single vertex cannot be minimised in one sweep.

edges (none of which are necessary as the graph is already triangulated). In order to minimise this group (which is assumed to be the last group of a series) we first apply MINT on edge $\{d, c\}$ and find that it is redundant and we remove it. Then we apply MINT on $\{b, e\}$ and find that it cannot be removed. Finally

[5] The term *string graph* denotes a graph which topologically resembles a string of pearls.

$\{b, c\}$ is found to be redundant and it is removed. Now $\{b, e\}$ becomes redundant as well. This shows us that in order to (locally) minimise a group of fill edges MINT must keep its recursive mode of operation. Furthermore, the example in Figure 1.3 demonstrates that the groups have to be ordered according to the reverse ordering scheme if a single sweep through the sequence of groups should be enough to minimise a triangulation. These were the considerations leading to Theorem 1.5.

Theorem 1.5 *Let $G = (V, E)$ be an ordered graph, $T = T(G)$, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ where $F_i = \text{MINT}(T_i(G), G')$, $G' = (V, E \cup F)$, and $F = \bigcup_{j=1}^{i} T_j(G) \cup \bigcup_{k=i+1}^{n} F_k$. Then $T' = \bigcup_i F_i$ is a minimal subset of T.*

Proof: Let $G' = (V, E \cup T)$ be the initial triangulated graph. First we observe that since G' is triangulated the removal of a fill edge that according to Corollary 1.4 is redundant does not introduce unchorded 4-cycles (the resulting graph will still be triangulated).

Now, assume we have just finished the recursive thinning of $T_i(G)$, and that $\{u, v\} \in F_i$. Note that $\{u, v\}$ was once produced by the elimination of a vertex x and hence is a unique chord of the 4-cycle $\langle u, w, v, x, u \rangle$. We will prove that $\{u, v\}$ is not redundant in the final triangulated graph $G'' = (V, E \cup T')$ either. Thus T' will be minimal. Let us therefore assume that $\{u, v\}$ is in fact redundant in G'' (which should lead to a contradiction). As we do not add new edges during the thinning process the only way in which $\{u, v\}$ could have become redundant is that one or more of the edges in the 4-cycle $\langle u, w, v, x, u \rangle$ have been removed. Let us consider the case where just one edge, $\{v, x\}$ say, has been removed. According to Theorem 1.1 there is a path $\langle v, v_1, \ldots, v_k, x \rangle$ from v to x in G (and hence also in G'') such that all vertices $v_1, \ldots, v_k$ are eliminated before both v and x. Next, Corollary 1.1 tells us that $\{u, v, w\} \cap \bigcup_{y \in \{v_1, \ldots, v_k\}} \text{adj}(y, G) = \emptyset$. Otherwise, there would be a fill edge $\{w, x\}$ (which would make $\{u, v\}$ redundant) and/or a fill edge $\{u, v\}$ which contradicts the fact that the elimination of x produced $\{u, v\}$. Thus, there is an unchorded cycle $\langle u, w, v, v_1, \ldots, v_k, x, u \rangle$ the length of which is clearly greater than 3. Repeating this argument in the cases where two, three or all four edges have been removed result in similar conclusions. □

Based on Theorem 1.5 we may now formulate a significantly more efficient minimisation algorithm, FMINT.

FMINT $(\mathcal{F}, G = (V, E \cup T))$
  for $i = n$ **downto** 1 **do**
    $R = \mathcal{F}[i] \setminus$ MINT $(\mathcal{F}[i], G = (V, E \cup T))$
    $T = T \setminus R$
  return (T)

$\mathcal{F}$ is an array of length R of sets of fill edges ordered according to the order of creation when applying the elimination procedure.

The magnitude of the improved performance achieved by FMINT compared to MINT depends on the distribution of the fill edges over the series of the n groups. The more equal they are distributed the better performance is achieved. In the worst case (i.e. all edges in the same group) no improvement is obtained. If the edges are equally distributed ($\frac{|T|}{|V|}$ edges in each group) FMINT runs in $O((c\frac{|T|}{|V|})^2)$ time. However, it is the size of the greatest group that determines the performance of FMINT. If the size of the greatest group is $\frac{|T|}{m}$, $m \geq 1$, FMINT runs in $O((c\frac{|T|}{m})^2)$ time. So for example even if one fourth of all fill edges belongs to the same group, FMINT will run 16 times as fast as MINT.

Letting $G = (V, E)$ be an ordered non-triangulated graph it appears from the above discussion that by approaching the graph triangulation problem by means of vertex elimination the fill edges $T'' = \bigcup_{i=k}^{n} T_i(G)$ seem to prevent a proper thinning of the fill edges $T' = \bigcup_{i=1}^{k-1} T_i(G)$ before T'' has been thinned out. An idea emerging naturally from this observation is: Is it possible, in general, to obtain minimal triangulations by applying MINT on each group $T_i(G)$ before proceeding to the elimination of vertex $\#^{-1}(i+1)$? In case of an affirmative answer we have found that the global search necessary to implement a triangulation algorithm based on the results by Fujisawa and Orino can be replaced by a local search. This seems intuitively very unlikely to be the case. The following example clearly shows that the question unfortunately must be returned a negative answer. Consider the example in Figure 1.5 and assume that vertex v is to be eliminated (cf. Case II above) which will produce a fill edge $\{u, w\}$. Theorem 1.3 states that $\{u, w\}$ is necessary if u and w are connected by a path not including v and that it is redundant otherwise. Since the problem of determining if such a path exists cannot be solved locally our previous results which apply to triangulated graphs are insufficient when we deal with non-triangulated graphs.

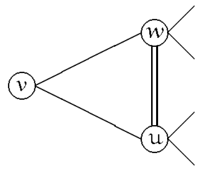

Figure 1.5: Fill edge $\{u, w\}$ is only necessary if u and w are connected by a path not including v.

However, note that in the special case where u and w are eliminated last we may remove $\{u, w\}$ irrespective of the topology of the rest of the graph as $\{u, w\}$ will be reproduced if u and w *are* connected by a path not including v (cf. Theorem 1.1).

1.5 Evaluating the Algorithms

In order to assess the virtues and vices of the various triangulation algorithms we want to perform a series of tests. These tests will be based on two "real world" graphs and on two artificial ones.

The two "real world" graphs, called Medianus-I[6] and Medianus-II originate from the development of the MUNIN[7] system (Andreassen, Jensen, Andersen, Falck, Kjærulff, Woldbye, Sørensen, Rosenfalck and Jensen 1989, Olesen, Kjærulff, Jensen, Jensen, Falck, Andreassen and Andersen 1989). Medianus-II is a modified version of Medianus-I (see Figures 1.6-1.7). Both of these graphs have a layered structure, which means that the number of fill edges required should be moderate.

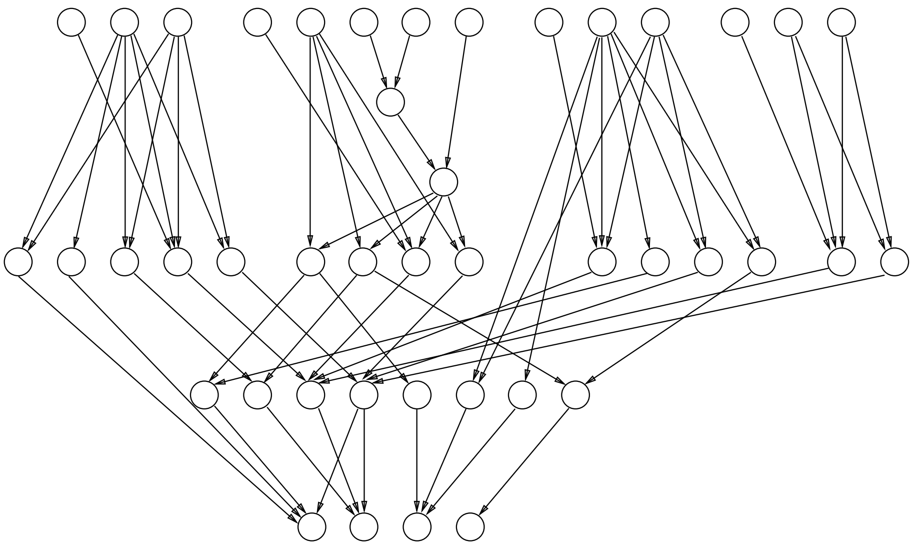

Figure 1.6: The Medianus-I graph.

[6]Bayesian network describing relations between disorders, pathophysiological features and measurements for the human median nerve.

[7]Knowledge based system for diagnosing diseases and malfunctions in the human neuromuscular system.

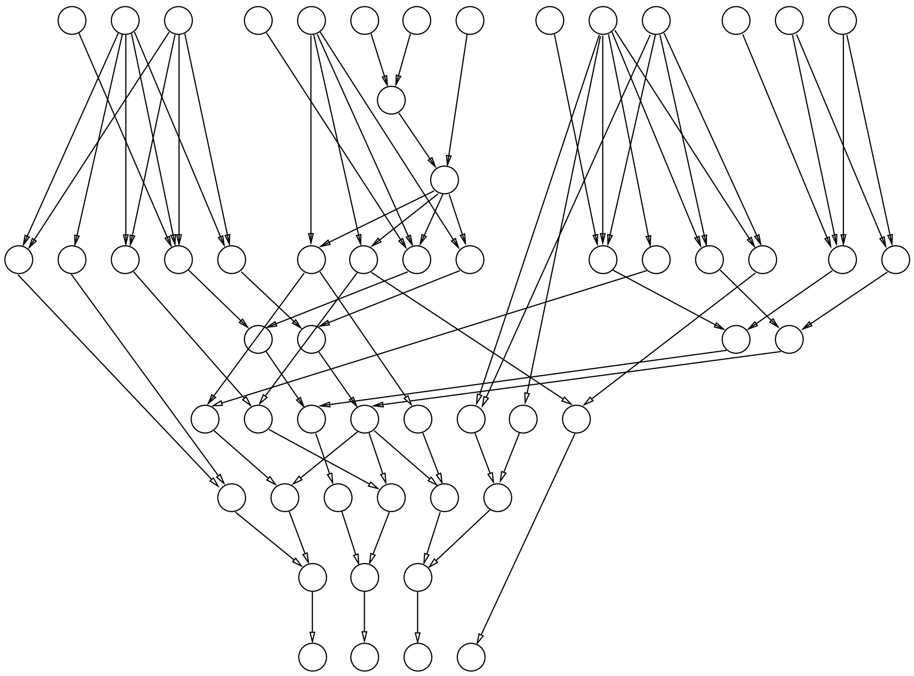

Figure 1.7: The Medianus-II graph.

For a graph $G = (V, E)$ to be connected a minimum of $n-1$ edges are required. The maximum number of edges in G is

$$\binom{n}{2} = \frac{n^2 - n}{2}$$

in which case G is complete.

The artificial graphs consist of a sparse and a dense graph. The notions of sparsity and density are based on the following definition.

Definition 1.1 Let $G = (V, E)$ be a (connected) graph. G is sparse if $n - 1 \leq e \leq \frac{1}{10}\left(\frac{n^2-n}{2}\right)$ (i.e. the number of edges is less than or equals one tenth of the maximum number of edges). G is dense if $e \geq \frac{1}{4}\left(\frac{n^2-n}{2}\right)$. □

From this definition follows that sparse graphs are only defined for $n \geq 20$, whereas dense graphs are defined for all $n \geq 0$.

We let a computer generate the graphs by first making a string graph (to assure that the graphs will be connected) and then adding edges at random by visiting the vertices one by one to assure a fairly equal distribution.

In the implementation of the algorithms for which the next vertex to be enumerated is not selected at random, but according to some ranking strategy, ties are solved by picking one of the equally ranked vertices using a random number generator.

For the Medianus graphs the algorithms were ran 400 times, whereas for the artificial graphs (let us call them Sparse and Dense) the number of runs were reduced to 100, but as will become quite clear below the conclusion is not influenced by that.

The algorithms tested are simple random elimination, lexicographic search, maximum cardinality search, extended random elimination, the FMINT algorithm applied to the triangulations produced by simple random elimination and by maximum cardinality search, and finally the three heuristic algorithms minimum fill, minimum size, and minimum weight. The minimum weight heuristic is the only one that takes the weight of the vertices into account when selecting the next vertex to be eliminated.

The algorithms were first ran on the graphs with the vertices having different weights (for the Medianus graphs the state sizes lies in the range from 3 to 21 with an average of approximately 6, and for the artificial graphs the state sizes are uniformly distributed between 2 and 5). Subsequently all orderings first obtained, except those produced by the minimum weight heuristic, were applied to the graphs with all vertices being of equal weight (state size 6 for the Medianus graphs and 2 for the artificial ones).

For each algorithm we record the following two key figures for a triangulated graph $G = (V, E \cup T)$: Number of fill edges ($|T|$) and weight of G ($w(G)$). Tables 1.1-1.4 show the minimum, average/median and maximum of these two key figures for the four graphs mentioned above. Note that the weight figures in the middle column represent the median weights (i.e. the base 2 logarithm of the average size of the state space).

The algorithms are ordered in Tables 1.1-1.4 according to the median weight of the triangulated graphs in the case where the vertices are of different weight (the reason being that this is the key figure from the point of view of belief graphs).

	Minimum			Average/median			Maximum		
Algorithm[1]	$\|T\|$	w(G)[2]	w(G)[3]	$\|T\|$	w(G)[2]	w(G)[3]	$\|T\|$	w(G)[2]	w(G)[3]
Random elimination	100	27.75	30.28	156	41.66	48.36	244	47.25	54.61
Ext. random elim.	63	22.84	24.65	100	30.94	36.26	175	36.19	41.44
Lexicographic search	68	23.44	25.05	100	30.29	35.30	177	36.50	41.47
Thinning (random)	62	22.86	24.66	98	30.14	35.51	162	35.41	41.69
Max. cardinality	68	23.63	25.27	100	29.00	31.97	136	32.63	36.49
Thinning (max card)	63	22.86	24.58	82	26.42	28.56	111	28.30	31.47
Min. size heuristic	57	22.71	24.55	58	23.27	24.88	60	24.17	26.15
Min. fill heuristic	57	22.71	24.55	58	23.14	24.80	61	24.79	26.31
Min. weight heuristic	57	22.94	– – –	57	22.99	– – –	57	23.08	– – –

1: 400 runs
2: Different vertex state sizes (between 3 and 21)
3: Equal vertex state sizes (= 6)

Table 1.1: Key figures for the triangulation of the Medianus-I graph.

	Minimum			Average/median			Maximum		
Algorithm[1]	$\|T\|$	w(G)[2]	w(G)[3]	$\|T\|$	w(G)[2]	w(G)[3]	$\|T\|$	w(G)[2]	w(G)[3]
Random elimination	134	29.28	32.08	213	46.35	50.32	345	53.86	56.94
Lexicographic search	98	22.28	23.61	145	30.59	33.71	253	35.21	39.57
Ext. random elim.	89	21.67	23.27	130	30.07	33.36	198	36.01	39.77
Thinning (random)	83	21.09	22.99	129	29.97	32.53	209	35.74	37.94
Max. cardinality	101	22.38	23.96	133	29.17	31.88	188	35.28	38.82
Thinning (max card)	87	21.61	23.17	116	26.93	28.87	153	31.68	34.85
Min. size heuristic	74	20.50	22.11	79	22.19	24.27	86	24.88	26.28
Min. fill heuristic	74	20.57	22.15	78	21.78	23.47	84	23.49	24.89
Min. weight heuristic	76	20.65	– – –	76	20.65	– – –	76	20.65	– – –

1: 400 runs
2: Different vertex state sizes (between 3 and 21)
3: Equal vertex state sizes (= 6)

Table 1.2: Key figures for the triangulation of the Medianus-II graph.

The most immediate result of the test runs is the overwhelming superiority of the heuristic algorithms no matter which kind of measure is employed (i.e. $|T|$ or

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]
Random elimination	283	32.86	19.72	390	48.31	26.19	565	54.77	31.46
Max. cardinality	222	27.24	16.51	305	36.61	20.57	406	41.46	23.42
Lexicographic search	217	26.41	16.49	297	33.77	19.84	405	38.25	22.82
Thinning (random)	184	25.01	15.59	254	32.95	18.83	339	37.04	22.20
Ext. random elim.	204	24.72	15.88	260	32.86	18.91	343	37.20	21.09
Thinning (max card)	182	24.62	15.42	224	31.36	17.53	300	35.97	20.53
Min. size heuristic	149	23.05	14.36	157	25.93	15.11	169	28.67	16.37
Min. weight heuristic	156	25.06	− − −	158	25.34	− − −	160	25.92	− − −
Min. fill heuristic	149	24.82	14.57	151	24.97	14.60	153	25.61	14.74

1: 100 runs
2: Different vertex state sizes (between 2 and 5)
3: Equal vertex state sizes (= 2)

Table 1.3: Key figures for the triangulation of the Sparse graph.

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]
Random elimination	538	62.84	37.33	677	73.57	41.99	795	77.40	44.83
Max. cardinality	499	59.14	34.82	567	66.41	38.21	615	67.92	39.21
Lexicographic search	432	54.39	32.59	538	65.41	36.73	644	70.84	40.80
Ext. random elim.	439	53.72	31.70	525	63.92	36.29	633	69.94	41.00
Thinning (random)	412	52.88	31.74	528	62.73	36.25	638	66.87	39.66
Thinning (max card)	438	56.42	32.92	481	59.57	34.26	545	62.53	35.72
Min. size heuristic	384	55.01	31.69	387	56.37	32.06	394	57.09	32.22
Min. fill heuristic	374	54.40	31.82	390	56.35	32.31	398	57.81	32.73
Min. weight heuristic	388	53.48	− − −	388	53.48	− − −	388	53.48	− − −

1: 100 runs
2: Different vertex state sizes (between 2 and 5)
3: Equal vertex state sizes (= 2)

Table 1.4: Key figures for the triangulation of the Dense graph.

w(G)). This superiority becomes even more distinct when the results of the test runs are displayed graphically as in Figure 1.8 which shows the graph weights

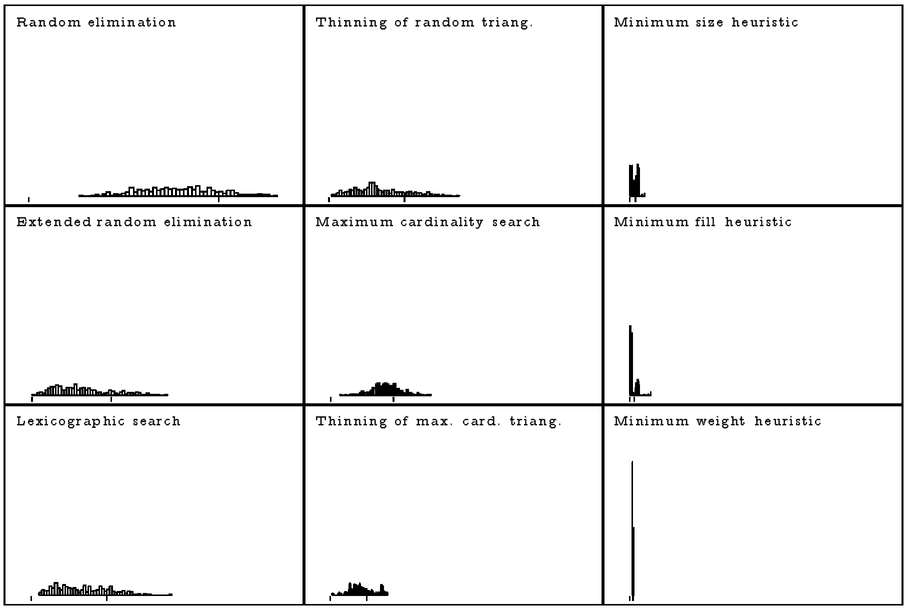

Figure 1.8: Distribution of graph weights obtained by 400 runs on the Medianus-I graph with different vertex weights.

obtained for the Medianus-I graph with different vertex weights (a similar representation of graph weights employing equal vertex weights yields essentially the same result). The 400 figures are ordered in 50 intervals and displayed in a bar chart with the height of each bar being proportional to the number of figures in the corresponding interval. (However, if the interval width in terms of pixels[8] is less than 1, given the width of the display area provided, the number of intervals for the particular set of figures is reduced such that 1 pixel represents the width of one interval.)

The tiny marks in Figure 1.8 indicate the overall minimum weight (in this case found by the minimum size heuristic) and the median weights for each algorithm.

A similar graphical display of the number of fill edges for the Medianus-I triangulations looks very much like Figure 1.8. The main difference is that for the heuristic algorithms the dispersion is even less than in Figure 1.8.

[8] A pixel is the atomic element in computer graphics

Representing the results obtained from the Medianus-II, the Sparse, and the Dense graphs in the same fashion provide bar charts of very similar appearance.

Figure 1.9 shows enlarged representations of the graph weight results for the four graphs tested obtained by the heuristic methods. The most striking result being that the range and dispersion of results for the minimum weight heuristic is very small, except maybe for the Sparse graph. This seems, however, to be a mixed blessing as for the Medianus and the Sparse graphs it is not able to find triangulations producing graph weights that are as small as those produced by some of the triangulations found by the minimum size heuristic which on the other hand produces very dispersed results.

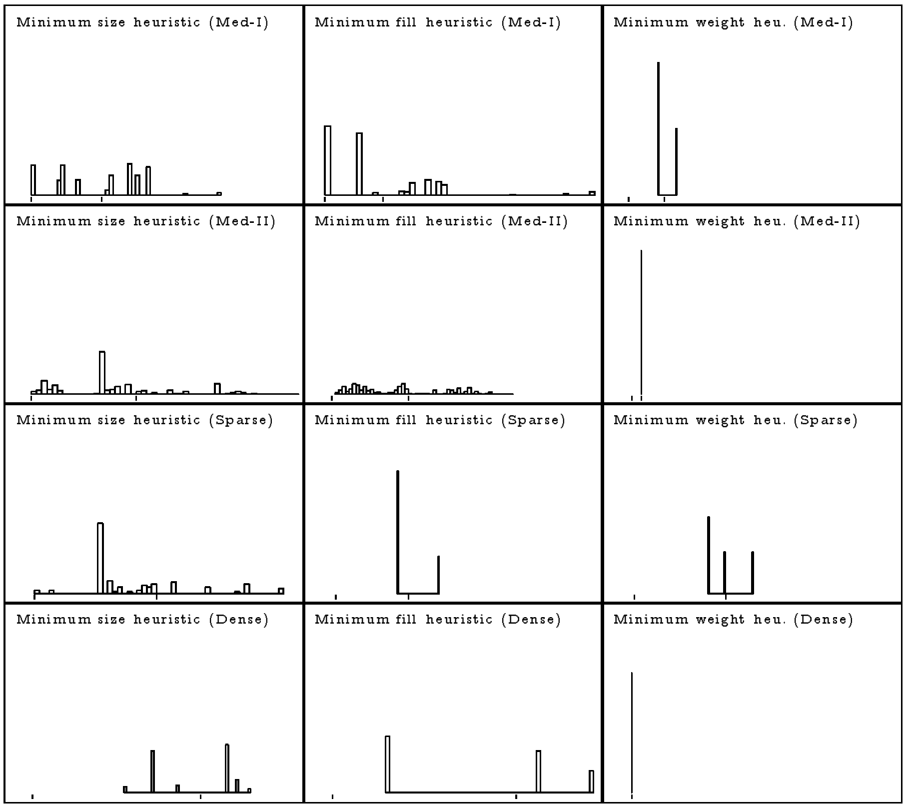

Figure 1.9: Distribution of graph weights obtained by running the heuristic algorithms 400 times on the Medianus graphs and 100 times on the artificial graphs. Different vertex weights.

Instead of breaking ties at random it might be possible to achieve less dispersed results by applying some sort of strategic tie breaking. To find a good strategy we should be able to identify some characteristic features of the choices made during the construction of the best orderings. However, by inspecting the elimination process pertaining to these orderings it seems impossible to find any pattern, at least from a local point of view, in the way ties are broken.

A way of improving the performance of the heuristic algorithms would be to try out all orderings that might possibly be produced by the algorithms. That is, for every tie to record the candidate vertices and then to perform a depth-first search in the tree of orderings successively constructed. However, by counting the number of candidate vertices for every tie, it was experimentally found that for the Medianus-I graph we may expect the number of possible orderings which can be achieved by the minimum size and the minimum fill heuristics to be as high as $2 \cdot 10^7$. This means that an exhaustive search in the tree of candidate orderings is absolutely infeasible for these two algorithms. The tree size for the minimum weight heuristic amounts to only 24 orderings for the Medianus-I graph. On the other hand, not much can be gained by an exhaustive search in this case as the dispersion is rather modest.

A total of 1000 triangulations were produced by the heuristic algorithms and, surprisingly, it appeared that all of these were minimal. However, as mentioned in Section 1.3 none of the heuristic algorithms are, in general, guaranteed to produce minimal triangulations; although in practice there seems to be no reason to run the FMINT algorithm to ensure minimality.

Although the heuristic algorithms and especially the minimum weight seem to produce fairly satisfactory triangulations we still face a major question: How far are we from a minimum weight triangulation?

Assuming the distance[9] between the elimination orderings produced by the heuristic algorithms and a minimum weight ordering, say $\#$, is modest the branch-and-bound algorithm (cf. Section 1.3) might provide an answer to our question. This algorithm has the essential property of searching in a very systematic way as all permutations with identical prefix $(v_1, \ldots, v_k)$ are tested before permutations with prefix $(v_1, \ldots, v_i)$, where $i < k$. This has the important implication that the number of iterations needed by the branch-and-bound algorithm to reach $\#$ might be orders of magnitude greater than the minimum distance to $\#$. This might, however, not necessarily mean that the algorithm does not have any practical significance. If the cardinality of the set, S, of orderings the weights of which differs insignificantly from the minimum weight, is large enough and the distances between the members of S are generally relatively large there might be a chance that the branch-and-bound algorithm is

[9] Here we define the distance between two orderings in terms of the minimum number of interchanges between neighbouring (i.e. adjacent in the ordering) vertices in one of the orderings required to reach the other (cf. Chapter 2).

a practical solution to refinement of orderings found by heuristic methods. On the other hand, if the cardinality of S is small we should, in general, expect an algorithm which chooses the next permutation in a non-deterministic manner to perform considerably better.

The best orderings known yet for the Medianus-I graph were found by the minimum size heuristic. About one seventh (56 out of 400) of the orderings found by this heuristic produced triangulated graphs with practically identical weights. A sample of 10 of these orderings were tested two by two to measure the distances between them (i.e. a total of 45 measure were calculated). It turned out that no two of these orderings were identical and that the distances between them were equally distributed in the interval from 20 to 58. The result of this little test indicates that we might expect a lot of orderings to be minimum and the distances between them to be quite significant.

Still, in order to get answers to the above questions we need to know, at least by high probability, what the minimum weight triangulation of a graph is. In Chapter 2 we consider a stochastic algorithm which with probability arbitrarily close to 1 finds a minimum triangulation, and hence we postpone the discussion of the utility of the branch-and-bound algorithm until the *simulated annealing* algorithm has been evaluated.

Chapter 2

Stochastic Search Algorithm

This chapter investigates the applicability of a Monte-Carlo technique known as 'simulated annealing' to achieve near-optimum decompositions of Bayesian networks under bounded resources. High-quality decompositions are essential for performing efficient inference in Bayesian networks. Optimum decomposition of Bayesian networks is known to be NP-hard (Wen 1990). We prove that cost-function changes can be computed locally, which is essential to the efficiency of the annealing algorithm. Pragmatic control schedules which reduce the running time of the annealing algorithm are presented and evaluated. Apart from the conventional temperature parameter, these schedules involve the radius of the search space as a new control parameter. The evaluation suggests that the inclusion of this new parameter is important for the success of the annealing algorithm for the present problem.

The contents of this chapter is identical to Kjærulff (1992b).

2.1 Introduction and terminology

Major advances in probabilistic inference methods based on graphical representations have been realized in recent years (Jensen, Lauritzen and Olesen 1990, Lauritzen and Spiegelhalter 1988, Pearl 1988, Shachter 1988, Shafer and Shenoy 1988, Suermondt and Cooper 1988). However, probabilistic inference has the inherent disadvantage of being NP-hard (Cooper 1987). Hence probabilistic reasoning may require excessive computational resources. By exploiting the conditional independence relations among the discrete random variables of a Bayesian network the underlying joint probability space may be decomposed into smaller subspaces corresponding to cliques in a triangulated graph (Lauritzen and Spiegelhalter 1988, Spiegelhalter 1986). Probabilistic networks are also frequently referred to as 'belief networks', 'Bayes networks', 'relevance diagrams', 'causal networks' and 'influence diagrams'.

A Bayesian network is built on a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$ of vertices $V = \{1, \ldots, n\}$ interconnected by a set of edges $E \subseteq V \times V$ of ordered pairs of distinct vertices. If $(u, v) \in E$ and $(v, u) \notin E$, then the edge from u to v is *directed*. If both $(u, v) \in E$ and $(v, u) \in E$, then the edge between u and v is *undirected*. If there is an edge between u and v, u and v are *adjacent* or *neighbours*. The neighbours of u are denoted $\mathrm{adj}(u)$. If there is a directed edge from u to v, u is called a *parent* of v and v a *child* of u. A graph containing only undirected edges is said to be *undirected*; otherwise it is *directed*. Unless explicitly stated, graphs are assumed to be undirected.

A subset $A \subseteq V$ induces a subgraph $\mathcal{G}_A = (A, E_A)$ of $\mathcal{G}$, where $E_A = E \cap (A \times A)$. A graph is *complete* if all vertices are pairwise adjacent. A subset $A \subseteq V$ is complete if it induces a complete subgraph. A maximal complete subset is called a *clique*. A *path* $\langle u = u_1, \ldots, u_k = v\rangle$ is an ordered sequence of distinct vertices such that $\{u_i, u_{i+1}\} \in E$ for $i = 1, \ldots, k-1$. An m-cycle is a path of length m with the exception that $u = v$. A *chord* in an m-cycle is an edge $\{u_i, u_j\}$ such that $1 < |i - j| < m - 1$ (i.e. $\{u_i, u_j\}$ connects the two non-consecutive vertices u_i and u_j in the cycle).

A graph is *triangulated* (also: *decomposable* or *chordal*) if any cycle of length greater than 3 has a chord. A vertex $v \in V$ is said to be *eliminated* from a graph $\mathcal{G} = (V, E)$ when the edges $(\mathrm{adj}(v) \times \mathrm{adj}(v)) \setminus E$ are added to E such that $\mathrm{adj}(v)$ becomes complete, the edges between v and its neighbours are removed from E, and v is removed from V. The set, say T, of edges added by eliminating all vertices in V in any order is called a *triangulation* of $\mathcal{G}$ as $\mathcal{G}' = (V, E \cup T)$ is triangulated (Rose 1970). The edges of T are called *fill edges* or *fill-ins*. Formally, an ordering of V is a bijection $\# : V \leftrightarrow \{1, \ldots, n\}$. $\mathcal{G}_\#$ is an *ordered graph*. We shall refer to $\#$ as both the bijection and the actual ordering determined by it, and when it appears clearly from the context that a graph is ordered we shall often drop the $\#$-index. The triangulation $T(\mathcal{G}_\#)$ is the set of edges produced by eliminating the vertices of $\mathcal{G}$ in the order defined by $\#$.

A Bayesian network can be decomposed by sequentially performing three operations on the underlying DAG: First, transform the graph into its corresponding *moral graph* (Lauritzen and Spiegelhalter 1988) by adding undirected edges between pairs of parents with one or more common children. Next, drop the orientation of the edges; and finally, triangulate the moral graph. The cliques of the triangulated graph then define a decomposition of the Bayesian network.

As indicated above, triangulating a graph is essentially a problem of establishing an elimination ordering. Several algorithms have been suggested for establishing elimination orderings which have certain desired properties (Fujisawa and Orino 1974, Kjærulff 1990, Rose et al. 1976, Tarjan and Yannakakis 1984). As the task of establishing an elimination ordering may be perceived as a sorting problem, such algorithms are quite often referred to as *sorting algorithms*. Since most of the algorithms apply some sort of search strategy, the term *search algorithms* is also frequently used.

The present paper deals with the problem of achieving approximate optimum decompositions of Bayesian networks. In this context the notion of optimality relates to the sum total of the state-space sizes of the cliques of the decomposable network. Wen (1990) has proved this optimality task to be NP-hard.

In Section 2.2 the actual sorting task is formulated in terms of simulated annealing and it is proved that cost-function changes can be computed locally, which is essential to the efficiency of the annealing algorithm. Section 2.3 suggests and extensively evaluates some pragmatic control schedules to reduce the running time of the algorithm. Apart from the conventional temperature parameter these schedules involve the radius of the search space as a new control parameter. The evaluation is based on four different graphs. Finally, Section 2.5 summarizes the results and discusses the applicability of simulated annealing for decomposing Bayesian networks by comparing the results of the evaluation with those of other search algorithms.

2.2 Graph triangulation by simulated annealing

Simulated annealing is a stochastic optimization algorithm used for finding global minimum cost configurations of NP-complete combinatorial problems with cost functions having many local minima. The algorithm was first described by Kirkpatrick, Gelatt and Vecchi (1983) and may be thought of as a combination of deterministic descent search and a Monte Carlo method. While a strictly descent algorithm often terminates at a local minimum, the addition of Monte Carlo techniques provides a way of escaping local minima. That is, the annealing algorithm accepts cost-function increases with positive probability dependent on the state of the search process as described below. The act of accepting cost-function increases is often referred to as *hill climbing*. In the original formulation of the annealing process the evolution of the state of the search process is based solely on a control schedule involving exponential decrease of a 'temperature' parameter. In this paper we suggest extending this control schedule by an additional parameter such that the 'step size' of the search decreases along with the temperature. This approach has the advantages of narrowing the focus of attention as attractive subspaces are being identified, thereby avoiding large and obviously unprofitable transitions; and of significantly reducing the running time of the annealing algorithm.

For a given optimization problem to be suitable for the annealing algorithm it should be possible to describe it in terms of a *solution space*, S, the elements of which all relate to some cost-function value $c(s)$, $s \in S$, and which are connected by some appropriate neighbouring structure. That is, S has a certain topology which has the essential feature that for every pair of configurations $s_i, s_j \in S$, s_j should be reachable from s_i. Reachability is defined as a series of *transitions*, $s_k \rightarrow s_l$, each of which consists of a *perturbation* imposed on configuration s_k such that it changes to a neighbouring configuration s_l. The *distance* between two configurations $s_i, s_j \in S$ is defined as the smallest number of transitions required to reach s_j from s_i. The *radius* of a space S is defined as the maximum distance between two configurations in S. Of course, the radius depends on the perturbation scheme employed. In our case S is the set of all possible (elimination) orderings of V in a graph $\mathcal{G} = (V, E)$ (i.e. $|S| = n!$, where $n = |V|$); and a transition $s_i \rightarrow s_j$ is simply defined as the interchange of two vertices in ordering s_i, resulting in ordering s_j.

Henceforth we shall refer to the size of the total state-space of a decomposable Bayesian network as the *weight* of the corresponding triangulated graph $\mathcal{G}^t = (V, E \cup T(\mathcal{G}_\#))$, where $\mathcal{G}_\# = (V, E)$. For convenience the weight, $w(\mathcal{G}^t)$, of $\mathcal{G}^t$ is defined as the base-2 logarithm of the state-space size. As our endeavour is to minimize the weight of $\mathcal{G}^t$ the cost-function is defined as

$$c(\#) = w(\mathcal{G}^t) \tag{2.1}$$

and we should therefore find an optimum ordering $\#^*$ such that $c(\#^*) \leq c(\#)$ for all $\# \in S$.

The annealing algorithm starts at some, possibly random, configuration $\#_i$ ($i = 0$), selects two vertices v and w (according to a perturbation scheme as discussed below), and computes the cost $c(\#_p)$ of the potential new configuration $\#_p$ obtained by interchanging v and w. Now, $\#_p$ is accepted if

$$c(\#_p) \leq c(\#_i) - t_i \log u, \tag{2.2}$$

where u is a uniformly distributed random number on the interval $]0; 1]$ and t_i is the *temperature*, a control parameter, at time i. This means that an uphill step of λt_i will be allowed with probability $e^{-\lambda}$. In other words, the new configuration $\#_{i+1}$ is set to $\#_p$ with probability

$$p = \min\left\{1, e^{-\Delta/t_i}\right\} \tag{2.3}$$

and to $\#_i$ with probability $1 - p$, where $\Delta = c(\#_p) - c(\#_i)$. Large values of t_i allow large cost-function increases, and small values allow only small increases.

The initial temperature, t_0, should be set to some suitable large value such that all perturbations are accepted (this ensures free motion in the search space, that is, all configurations can be reached). Thus, if $\#_i$ and $\#_j$ are neighbouring configurations in S and if we define

$$U = \max_{i,j} |c(\#_i) - c(\#_j)| \tag{2.4}$$

to be the maximum cost-function increase, then an uphill step of U should be accepted with probability close to 1. If $U = \alpha t_0$ we get the rule

$$e^{-\alpha} \approx 1 \tag{2.5}$$

meaning that t_0 should be large compared to U or $0 < \alpha \ll 1$.

Lundy and Mees (1986) concluded in their study on convergence properties of the annealing algorithm that the temperature should be lowered exponentially as

$$t_{i+1} = \frac{t_i}{1 + \beta t_i}, \tag{2.6}$$

where $\beta \ll 1/U$, and that the final temperature, t_f, should (essentially) obey the following inequality:

$$t_f \leq \frac{\varepsilon}{\log |S|}, \tag{2.7}$$

where ε is the acceptable error (i.e. $\varepsilon = c(\#_f) - c(\#^*)$).

An equivalent formulation of Equation 2.6 is

$$t_{i+1} = \frac{t_0}{1 + i\frac{\gamma}{\alpha}}, \tag{2.8}$$

where $\alpha = U/t_0$, $0 < \gamma \ll 1$ and $\gamma/\alpha = \beta t_0$. By solving Equation 2.8 for i it appears that the number of iterations required to reach a temperature t_{i+1} is less than $U/(\gamma t_{i+1})$ and hence to reach the final temperature t_f takes I iterations, where

$$I < \frac{U \log |S|}{\gamma \varepsilon}. \tag{2.9}$$

2.2.1 An additional control parameter

Initially, essentially free motion in the search space, S, should be allowed to avoid becoming trapped in local minima at an early stage. Apart from allowing large uphill steps it is also important that the configurations visited in the initial phase are uniformly distributed over S such that the topographic map of S (w.r.t. configuration costs) generated during the search process represents the entire search space. However, in addition the density of configurations visited in S should be sufficiently high to guarantee that all major basins are adequately represented. Obviously, assuming uniform distribution, the density of configurations visited depends solely on the number of iterations of the annealing algorithm in the initial phase (i.e. when virtually any uphill step is accepted). Of course, the uniformity of the distribution may be ensured by lowering the temperature sufficiently slowly (i.e. spending a long time in the initial phase). This may, however, not be the optimal schedule as the density of configurations visited may then become unnecessarily high. Thus we need an additional control parameter to ensure a uniform distribution of configurations without being forced to lower the temperature unnecessarily slowly.

An obvious candidate providing this feature is the radius of S, that is, by maintaining a small radius the distance between any two configurations becomes small, implying a more uniform distribution of configurations visited given a fixed number of iterations. Thus, in the initial phase the radius of the search space should be kept as small as possible to allow large steps in S. Later, when the algorithm has found an attractive subspace (basin) containing low-cost configurations, it seems more convenient to have a large radius (i.e. narrowing down the focus of attention by reducing the number of neighbours for each configuration allowing only small steps in S). The process of increasing the radius of S might be thought of as a magnification process allowing a more fine-grained search process (cf. the inflation of a balloon).

Apart from narrowing the focus of attention, a large radius (small step size) has the additional advantage of significantly reducing the running time of the annealing algorithm, as will become clear in the following.

2.2.2 Perturbation scheme

The neighbouring structure of the search space, S, is determined by the perturbation scheme employed. That is, given a configuration, $\# \in S$, the set of neighbouring configurations, $N(\#) \subseteq S$, is defined as the set of all possible configurations which can be obtained by applying the perturbation scheme on $\#$. Thus the radius of S is determined by the perturbation scheme. Initially, the only constraint put on the perturbation scheme is that a perturbation is defined to be the interchange of only two vertices u and v in a configuration. With that constraint in mind, the scheme providing the smallest possible radius chooses the pair u, v completely arbitrarily from a configuration, $\#$, and the scheme providing the largest possible radius chooses a pair u, v for which $|\#(u) - \#(v)| = 1$ (i.e. u and v are adjacent in $\#$). For both schemes the pair of configurations $\#_i, \#_j \in S$ between which the distance equals the radius of S is characterized by the one being the reverse of the other (i.e. $\#_i(v) \neq \#_j(v)$ for all $v \in V$ except for at most one vertex). Thus for the two perturbation schemes the radii r_{min}

and r_{max} of S are given by

$$r_{min} = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (n-1)/2 & \text{if } n \text{ is odd} \end{cases}, \tag{2.10}$$

$$r_{max} = \sum_{i=1}^{n-1} n - i = \frac{n^2 - n}{2}. \tag{2.11}$$

Apparently $r_{max} = (n-1)r_{min}$ if n is even and $r_{max} = nr_{min}$ if n is odd, and hence there is a substantial difference between r_{min} and r_{max} even for a moderate number of vertices. However, both of these perturbation schemes may be described in terms of the general scheme for which the interchange of u and v is restricted by the condition $|\#(u) - \#(v)| < \omega$, where $2 \leq \omega \leq n$. That is, we define a 'window' of width ω within which the perturbations take place. When the annealing process starts we set $\omega = n$ and decreases it gradually until it reaches the lower bound $\omega = 2$. A reasonable way of narrowing the window width (i.e. increasing the radius of the search space — decreasing the step size) could be to follow a reduction scheme similar to the one applied to the temperature, i.e. exponential reduction.

2.2.3 Local computation of cost-function change

It is crucial to the success of the annealing algorithm that the cost of a candidate new configuration, $\#_p$, can be computed efficiently. That is, with

$$c(\#_p) = c(\#_i) + \Delta$$

the cost-function increase, Δ, should be locally computable. For that requirement to be fulfilled we must develop a method for identifying the cliques not altered by a given perturbation. By Theorem 2.1, only cliques produced by eliminating vertices $\#^{-1}(i), \ldots, \#^{-1}(j)$, $i < j$, need to be re-established when interchanging $\#^{-1}(i)$ and $\#^{-1}(j)$ in the elimination order given by $\#$.

In the following we denote by $\mathcal{G}_i = (V_i, E_i)$, where $i = 1, \ldots, n$, the ordered graph obtained by eliminating in order the vertices $\#^{-1}(1), \ldots, \#^{-1}(i) \in V$ from the ordered graph $\mathcal{G} = \mathcal{G}_0 = (V, E)$. By $C_i(\mathcal{G})$ we denote the clique, if any, produced by eliminating vertex $\#^{-1}(i)$ in $\mathcal{G}_{i-1}$. Let $C_i(\mathcal{G}) = \emptyset$ if the elimination of $\#^{-1}(i)$ does not produce a clique. In the same spirit we let $T_i(\mathcal{G})$ denote the fill edges produced by eliminating vertex $\#^{-1}(i)$ in $\mathcal{G}_{i-1}$.

Lemma 2.1 (Rose *et al.*) *Let $\mathcal{G} = (V, E)$ be an ordered graph. Then $\{u, v\} \in T(\mathcal{G})$ if and only if there is a path $\langle u, u_1, \ldots, u_k, v\rangle$ in $\mathcal{G}$ such that $\#^{-1}(u_i) < \min(\#^{-1}(u), \#^{-1}(v))$ for all $1 \leq i \leq k$.*

For a proof of Lemma 2.1, see Rose et al. (1976).

Lemma 2.2 *Let* $\mathcal{G}_{\#_a} = (V, E)$ *and* $\mathcal{H}_{\#_b} = (V, E)$ *be ordered graphs, where we have* $\#_a^{-1}(i) = \#_b^{-1}(i)$ *for all* $k < i \leq n$ *with* k *being an arbitrary non-negative integer. Then* $\mathcal{G}_i = \mathcal{H}_i$ *for all* $k \leq i \leq n$.

Proof: Clearly $\mathcal{G}_i$ and $\mathcal{H}_i$ consist of the same set of vertices, and from Lemma 2.1 it is equally clear that they also have identical sets of edges. □

Theorem 2.1 *Let* $\mathcal{G}_{\#_a} = (V, E)$ *and* $\mathcal{H}_{\#_b} = (V, E)$ *be ordered graphs such that for* $u, v \in V$, $\#_a(u) = i < \#_a(v) = j$ *and for* $1 \leq k \leq n$

$$\#_a^{-1}(k) = \begin{cases} \#_b^{-1}(j) & \textit{if } k = i, \\ \#_b^{-1}(i) & \textit{if } k = j, \\ \#_b^{-1}(k) & \textit{otherwise.} \end{cases}$$

Then $C_k(\mathcal{G}) = C_k(\mathcal{H})$ *for* $1 \leq k < i$ *and* $j < k \leq n$.

Proof: The proof consists of two parts: First, we shall prove that all cliques $C_k(\mathcal{G})$ for $1 \leq k < i$ and $j < k \leq n$, are unaffected by the interchange of u and v in the elimination ordering. Knowing that the triangulation of a graph determines how the cliques of the resulting triangulated graph are composed the problem is equivalent to the problem of proving that the fill edges $T_k(\mathcal{G}) = T_k(\mathcal{H})$ for all $1 \leq k < i$ and for all $j < k \leq n$. For $1 \leq k < i$ the problem is trivial as the set of fill edges produced by eliminating a vertex solely depends on the original edges E and on the fill edges already produced. The validity of the equation for $j < k \leq n$ follows immediately from Lemma 2.1.

Next, we shall prove that for every new clique $C_k(\mathcal{H})$, $i \leq k \leq j$, C_k cannot be a superset of a clique $C_l(\mathcal{G}) = C_l(\mathcal{H})$, $1 \leq l < i$ and $j < l \leq n$. For $1 \leq l < i$ the problem is trivial as $\#_a^{-1}(l) = \#_b^{-1}(l) \in C_l$ and hence cannot be a member of C_k. To prove it for the case where $j < l \leq n$, we assume there is a clique C_l such that $C_l \subset C_k$. For this to be true the vertex $\#_b^{-1}(k)$ (the elimination of which created C_k) must have been adjacent to all vertices in C_l in $\mathcal{H}_{k-1}$. By Lemma 2.2 we see that there would also have been such a vertex in $\mathcal{G}_{k-1}$ and hence C_l would not have been a clique. □

The consequence of Theorem 2.1 may be explained as follows. Let $\mathcal{G}_\#$ be an ordered graph, let $\mathcal{G}^t = (V, E)$ be the corresponding triangulated graph produced by eliminating the vertices in order $\#$, and let $\mathcal{G}^t_A = (A, E_A)$ be the subgraph of $\mathcal{G}^t$ induced by $A \subseteq V$ such that $\#^{-1}(i) \in A$ for all $a \leq i \leq a + k$, where $a = \min_{v \in A} \#(v)$ and $|A| = k + 1$. That is, A consists of a set of vertices comprising a consecutive subset of the ordering $\#$; or, more generally, A consists of a set of vertices comprising a consecutive subset of an ordered set defining an elimination order corresponding to the fill edges of $\mathcal{G}^t$. Note that $\mathcal{G}^t_A$ is triangulated. Now the fill edges of $\mathcal{G}^t_A$ are removed and the resulting graph is retriangulated resulting in the graph (A, E^*_A). By Theorem 2.1 we find that substituting E_A in $\mathcal{G}^t$ by E^*_A does not compromise the chordality of the graph nor alter the cliques produced by the elimination of vertices $V \setminus A$. This means that modifications of a triangulated graph may be carried out locally.

It is clear from the above discussion that the time complexity of the annealing algorithm is strongly dependent on the window width, ω. In the worst case the graph must be totally retriangulated for $\omega = n$. On the other hand, for $\omega = 2$ a perturbation affects at most two cliques. Thus an optimal control schedule under bounded resources involves a trade-off between mobility (ω large) and speed (ω small).

2.3 Annealing in practice

This section begins with a discussion of the need to compromise the control schedule presented in Section 2.2 due to excessive time complexity, and then turns to pragmatic ways of reducing the running time of the annealing algorithm. An extensive evaluation is carried out to identify an optimal control schedule under bounded resources.

2.3.1 The excessive time complexity of ideal annealing

To get a feeling for the time complexity of the annealing algorithm, given the control schedule in Section 2.2, assume $\alpha = 10^{-2}$ (satisfying Approximation 2.5) and $\gamma = 10^{-4}$ (so that $\gamma/\alpha \ll 1$). By accepting an error $\varepsilon = 10^{-3}U$ we find that $I \approx 10^7 n(\log n - 1)$ which for $n = 50$ is approximately 1.5×10^9. An actual implementation of the annealing algorithm in the C programming language gave a maximum speed ($\omega = 2$) of approximately 1000 iterations per second on a Sun SPARCstation SLC. Even at that speed the example above would still require a running time of more than 400 hours. Therefore, in order to obtain reasonable response times from the annealing algorithm, we have to compromise the convergence properties of the algorithm and adopt pragmatic control schedules which trade off efficiency against precision.

2.3.2 Pragmatic control schedules

There are three parameters which determine the number of iterations required: the initial temperature, t_0; the cooling rate controlled by γ; and the final temperature, t_f.

As discussed in Section 2.2, to allow essentially free motion in the search space, S, in the initial phase, the probability that any uphill step is accepted should be close to 1 implying the rule $t_0 \gg U$. The validity of this rule might, however, be highly dubious as it depends heavily on the topography of S. If, for example, the search space does not contain 'highland basins' with no configurations having costs close to the global minimum there is no need for a high initial temperature.

If the phenomenon of highland basins is scarce and the basin depths are moderate then the cooling rate may be increased without seriously compromising the convergence properties of the algorithm.

Specifying a reasonable acceptable error ε often leads to extremely small values of both t_f and temperature changes $t_i - t_{i+1}$ when t_i is close to t_f. This means that the algorithm may have reached configurations $\#$ for which $c(\#) - c(\#^*) \leq \varepsilon$ long before t_i has reduced to t_f. Further, by constantly keeping in mind the least-cost configuration visited, t_f can definitely be set at a considerably higher value than recommended by Inequality 2.7.

From the above discussion it appears that the total number of attempted configuration transitions before reaching the final temperature, given a fixed control schedule, depends on the size, $|S|$, of the search space and the maximum cost difference, U, between any two configurations. Concerning the problem of low-cost graph triangulations, this means that in practice there is a limit in terms of number of vertices and dispersion of vertex weights (loosely speaking) above which the annealing algorithm is inapplicable. This is, of course, a rather unsatisfactory situation which can be avoided by allowing more 'coarse-grained' solutions for large graphs. That is, if we adopt such a rather pragmatic approach the stopping criterion must be defined such that it becomes independent of $|S|$ and U. From Equation 2.8 we get

$$I = \frac{\alpha}{\gamma}\left(\frac{t_0}{t_f} - 1\right) = \frac{1}{\gamma}\left(\frac{U}{t_f} - \alpha\right), \tag{2.12}$$

which says that if the final temperature is defined in terms of U the number of iterations required depends solely on the cooling rate. One way of defining t_f is to say that it is the temperature at which an uphill step of cU, where $0 < c < 1$, is accepted with probability p, where $0 < p < 1$. Recalling that an uphill step of $\lambda t_f = cU$ is accepted with probability $e^{-\lambda} = p$ at temperature t_f, we get

$$I = -\frac{1}{\gamma}\left(\frac{\log p}{c} + \alpha\right), \quad p < e^{-\alpha c} \tag{2.13}$$

as desired. By fixing α and c we observe that the cooling rate may be given either explicitly via γ or implicitly via I and p. Similarly, I may be given explicitly or implicitly via γ and p.

The annealing algorithm performs a search in the graph of configurations (the solution space) which is normally controlled solely by the temperature parameter. However, by allowing a gradual increase of the radius of the graph during the search process we have introduced an additional control parameter as discussed in Section 2.2. Ideally the final temperature, t_f, should be low (relative to the acceptable error, ε) and the final radius should be large (i.e. ω should be small). However, to determine the effect of the two control parameters separately a series of test runs must be conducted. One of the parameters is kept at a fixed value which we define to be either small, medium or large.

The three temperature levels are defined in the same way as t_f above with $c = 0.1$ and probabilities $p_s = 0.05$, $p_m = 0.5$, and $p_l = 0.95$. Thus

$$t_s = \frac{U}{10 \log 20}, \tag{2.14}$$

$$t_m = \frac{U}{10 \log 2}, \tag{2.15}$$

$$t_l = \frac{U}{-10 \log 0.95}. \tag{2.16}$$

The radius levels are defined as

$$r_s = r_{min}, \tag{2.17}$$

$$r_m = \frac{r_{min} + r_{max}}{2}, \tag{2.18}$$

$$r_l = r_{max}. \tag{2.19}$$

As the radius of the search space is determined by ω, r is not an explicit control parameter. So we need an algebraic relationship between r and ω. This is, however, rather hard to find if we insist on employing the perturbation scheme described above. By accepting a slightly different scheme based on a 'move-and-push' procedure (i.e. an element $x = L[i]$ at position i in sequence L is moved to position j by pushing each element $L[k]$ to position $k+\delta$ for $k = i+\delta, i+2\delta, \ldots, j$, where $\delta = -1$ if $i < j$ and $\delta = 1$ if $i > j$, and placing x at position j), it can be shown that ω can be expressed in terms of r and n as

$$\omega = \left\lceil \frac{(n-1)^2}{2r - n + 1} + 0.5 \right\rceil. \tag{2.20}$$

The interchange and the 'move-and-push' schemes are identical for $\omega = 2$. For $\omega = n$ 'move-and-push' yields $r_{min} = n - 1$. Thus

$$r_m = \frac{r_{min} + r_{max}}{2} = \frac{n^2 + n - 2}{4}, \tag{2.21}$$

$$\omega_m = \left\lceil \frac{2(n-1)^2}{n(n-1)} + 0.5 \right\rceil = \left\lceil 2\left(1 - \frac{1}{n}\right) + 0.5 \right\rceil . \tag{2.22}$$

The free control parameter may be subjected to a slow, medium or fast reduction (temperature) or increase (radius) schedule.

The rate by which the temperature parameter is decreased is determined by the ratio γ/α. Recalling that $e^{-\alpha}$ expresses the probability of accepting uphill steps of U we select $\alpha = 0.1$ (i.e. with probability $p \approx 0.9$ an uphill step of U is accepted when $t = t_0$). The rule of essentially free motion in the initial phase is not violated, which is clearly demonstrated by the fact that at temperature t_i uphill steps of U are accepted with probability e^{-U/t_i} (i.e. even with the temperature lowered to $t_0/10$ more than one-third of the potential uphill steps of size U are accepted). So, with α fixed the cooling rate is determined by γ. A slow cooling rate is defined such that the temperature equals t_l when only 10% of the iterations are left, for a medium rate the final temperature is defined to be reached at the last iteration, and we define a fast cooling rate such that the temperature equals t_s when only 10% of the iterations are done. In the case of a medium cooling rate we set $t_f = t_s$. Thus

$$\gamma_s = -\frac{1}{0.9I}\left(\frac{\log p_l}{c} + \alpha\right) = \frac{-10\log 0.95 - \alpha}{0.9I}, \tag{2.23}$$

$$\gamma_m = -\frac{1}{I}\left(\frac{\log p_s}{c} + \alpha\right) = \frac{10\log 20 - \alpha}{I}, \tag{2.24}$$

$$\gamma_f = -\frac{1}{0.1I}\left(\frac{\log p_s}{c} + \alpha\right) = \frac{10\log 20 - \alpha}{0.1I}. \tag{2.25}$$

Even though we argued above that ε can be set to a value which is considerably higher than the actual acceptable error, provided the least-cost configuration visited is remembered, it seems hard to justify $t_f = t_s$ as ε then becomes comparable with U. The discussion of whether $t_f = t_s$ is a reasonable choice is continued in Section 2.4.

Now we should assign values to either I or γ (recall that values of p have already been determined). As I is a derived parameter which is defined in terms of the cooling rate and the final temperature we naturally choose the latter. Knowing that $\gamma/\alpha \ll 1$ or $\gamma \ll \alpha$ a value of $\gamma = 10^{-3}$ seems reasonable, and we find from Equation 2.24 that

$$I = \frac{10\log 20 - \alpha}{\gamma_m} \approx 10^4 \log 20 \approx 3 \times 10^4 \tag{2.26}$$

corresponding to running times in terms of minutes instead of hours or even days (cf. Section 2.3.1).

In the spirit of Equation 2.8 the window width, ω, is decreased exponentially as

$$\omega_{i+1} = \frac{\omega_0}{1 + i\tau}. \tag{2.27}$$

Thus the rates by which r is increased are determined by

$$\tau_s = \frac{\omega_0/\omega_m - 1}{I}, \tag{2.28}$$

$$\tau_m = \frac{\omega_0/\omega_s - 1}{0.8I}, \tag{2.29}$$

$$\tau_f = \frac{\omega_0/\omega_s - 1}{0.2I}. \tag{2.30}$$

Note that ω_m corresponds to r_m, $\omega_0 = n$, and ω_s corresponds to r_l. Equations 2.28–2.30 suggest that, at a slow rate the radius of the search space is medium when the annealing terminates, at a medium rate the radius is maximum when the annealing is 80% done, and at a fast rate it is maximum when 20% done.

Now, the only undefined parameter is the maximum cost-function increase, U, which, depending on the problem at hand, can be found either analytically or experimentally. For the graph-triangulation problem it seems very hard to find U analytically as it depends on a number of parameters such as the size and density of the graph, the homogeneity of the graph w.r.t. the distribution of edges and number of states of the individual vertices, and the window width, ω. So we will stick to the experimental approach.

2.4 Evaluation

In order to find the optimal schedule among the pragmatic control schedules presented above, a selected set of schedules is first tested on a particular graph and then a subset of these schedules which appear to produce the best results is then applied to three other graphs. Then, hopefully, we should be able to determine which control schedule(s) is (are) generally preferable for the network decomposition problem. Also the accuracy of the approximate solutions is discussed and compared to results obtained by other graph-triangulation algorithms.

2.4.1 Evaluating the pragmatic control schedules

In the first test cycle we select the control schedules for which at least one of the control parameters is varying, giving a total of 27 distinct combinations. As the target graph we use the Medianus-I network known from the MUNIN expert system (Andreassen et al. 1989, Jensen, Andersen, Kjærulff and Andreassen 1987, Olesen et al. 1989). The Medianus-I network consists of 43 vertices and 110 edges and describes relations between disorders, pathophysiological features, and measurements for the human median nerve. For Medianus-I we find $U \approx 7$. We cannot expect, of course, to find least-cost configurations with identical costs every time the algorithm is run. We thus have to run the algorithm several times for each control schedule. Table 2.1 shows the minimum, median, and maximum least costs found for the 27 different control schedules with ten runs for each schedule. The schedules have been ordered according to the median least-cost value which is the quantity we shall use in our comparison of the schedules. Note that the median cost is defined as the logarithm of the average state-space size. As the state-space size is the quantity of interest and as $E(\log s) \leq \log E(s)$ (cf. Jensen's inequality), where s denotes the state-space size, we focus on the median cost rather than the average cost.

	Min.	Med.	Max.		Min.	Med.	Max.
1. t_s, τ_m	22.63	22.71	22.92	15. γ_m, τ_f	22.71	23.41	24.40
2. γ_m, r_s	22.68	22.73	22.81	16. γ_m, r_l	22.69	23.46	24.23
3. t_s, τ_s	22.66	22.78	23.14	17. t_m, τ_m	23.17	23.63	24.37
4. t_s, τ_f	22.65	22.95	23.87	18. t_m, τ_f	22.98	23.81	24.43
5. γ_f, r_s	22.63	22.96	23.44	19. γ_s, r_s	23.99	24.95	25.83
6. γ_m, τ_m	22.70	22.99	23.47	20. t_l, τ_s	24.07	25.16	26.03
7. γ_f, τ_s	22.63	23.00	23.44	21. γ_s, τ_s	24.35	25.23	26.18
8. γ_f, τ_m	22.70	23.03	24.06	22. t_l, τ_m	24.12	25.34	26.33
9. γ_m, τ_s	22.74	23.07	23.63	23. γ_s, τ_m	24.07	25.66	27.05
10. γ_f, r_m	22.63	23.13	24.06	24. γ_s, τ_f	25.36	26.43	27.28
11. γ_f, τ_f	22.63	23.19	23.78	25. t_l, τ_f	25.06	26.67	28.41
12. γ_m, r_m	22.68	23.25	24.19	26. γ_s, r_m	25.64	27.18	30.46
13. γ_f, r_l	22.63	23.35	24.06	27. γ_s, r_l	25.07	27.44	29.12
14. t_m, τ_s	22.79	23.36	23.92				

Table 2.1: Minimum, median, and maximum least costs found for 27 different control schedules applied ten times to the Medianus-I graph.

A varying control parameter is indicated by a γ for the temperature and a τ for the radius and the rate is indicated by indices s, m, f (i.e. slow, medium, fast). A fixed control parameter is indicated by a t for the temperature or an r for the radius and the level is indicated by indices s, m, l (i.e. small, medium, large).

Considering the results provided by the top third control schedules in Table 2.1 we observe that for all of these the final temperature is low (i.e. $t_f \leq t_s$) and that the initial radius is small (i.e. $r_0 = r_s$). Three additional test graphs are employed: Medianus-II, a later version of Medianus-I containing 56 vertices and 110 edges; Sparse, a computer generated graph of 50 vertices and 100 edges; and Dense, a computer generated graph of 50 vertices and 359 edges. The results obtained by applying the top third control schedules in Table 2.1 100 times to all

	Medianus-I			Medianus-II			Sparse			Dense		
	Min.	Med.	Max.	Min.	Med.	Max.	Min.	Med.	Max.	Min.	Med.	Max.
1. t_s, τ_m	22.63	22.89	23.82	20.46	20.82	22.45	22.67	22.92	24.10	50.89	51.94	53.98
2. γ_m, r_s	22.67	22.79	23.47	20.46	20.57	20.88	22.70	22.96	23.39	50.90	51.41	53.81
3. t_s, τ_s	22.63	22.80	23.46	20.47	20.67	22.44	22.66	22.84	23.33	50.88	51.80	54.94
4. t_s, τ_f	22.63	23.12	24.11	20.49	20.95	22.88	22.69	23.34	25.21	50.90	53.62	56.78
5. γ_f, r_s	22.63	22.96	24.02	20.45	20.64	22.42	22.61	22.68	23.29	50.88	51.42	53.18
6. γ_m, τ_m	22.69	23.22	24.18	20.50	21.30	22.96	22.81	23.89	25.70	51.17	54.25	56.87
7. γ_f, τ_s	22.63	22.97	24.08	20.45	20.61	21.50	22.61	22.77	24.12	50.88	51.84	54.44
8. γ_f, τ_m	22.63	22.99	24.07	20.45	20.83	22.86	22.61	22.99	25.37	50.88	52.34	54.86
9. γ_m, τ_s	22.67	23.04	23.60	20.48	20.87	21.97	22.67	23.38	25.00	50.96	52.80	54.55

Table 2.2: Minimum, median, and maximum least costs found for nine different control schedules applied 100 times to the four test graphs.

four graphs are displayed in Table 2.2. In Table 2.1 the most striking observation is that the three schedules with a constant low temperature belong to the top five schedules. Therefore an interesting experiment would be to test a strictly descent version of the annealing algorithm (i.e. $t = 0$). However, before doing that we will add a few more comments on Tables 2.1 and 2.2. The last third of the schedules is characterized by a high final temperature. For the middle third the final temperature is either low or medium. Note that the average state-space size is more than doubled from the 18th best to the 19th best schedule (i.e. the boundary between the middle and last third).

By calculating the average relative deviation of the median values from the least median for each control schedule in Table 2.2 we find that the ranking of the schedules is 5, 2, 3, 7, 1, 8, 9, 4, 6 (i.e. schedule number 5 is the best), where the differences between the results of schedules 2 and 5 are insignificant (see Figure 2.1). Schedules 4, 6 and 9 are significantly worse than the rest: for all

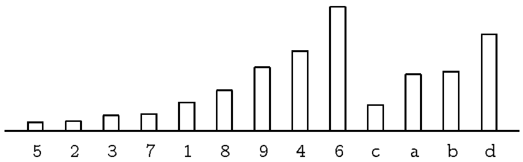

Figure 2.1: Average relative deviation from least medians for control schedules 1–9 and the four descent schedules a–d for the four test graphs.

four graphs these three schedules are the three worst. The reason why schedule 4 performs badly appears to be the fast radius increase: For schedules 1, 3 and 4 the temperature is constantly low and the radius is increased at medium, slow and fast rates, respectively. Thus, fast radius increase is not recommendable and not even medium-rate radius increase seems to be recommendable as all of the control schedules including medium-rate radius increase (i.e. schedules 1, 6 and 8) perform relatively badly, though with schedule 1 as a possible exception. It is interesting to compare schedules 2, 6 and 9 as they are based on medium-rate temperature decrease (relatively high final temperature), but only schedule 2 produces acceptable results. From that comparison we must conclude that if the final temperature is (relatively) high only a constantly small radius is recommendable. Similarly, by comparing schedules 5, 7 and 8 we must conclude that with a high initial temperature a constant small radius is preferable.

So far we have not seen much evidence that the introduction of the new control parameter has resulted in improved annealing schedules. However, the differences between schedules 3 and 7 and between schedules 2 and 5 are very small, but due to the constantly small radius in schedules 2 and 5 the algorithm runs significantly slower with these control schedules than it does with schedules 3 and 7: with schedules 3 and 7 it runs approximately 2.6 times as fast as it does with schedules 2 and 5. Thus, seen in the light of bounded time resources, schedules 3 and 7 would be preferable to schedules 2 and 5. Even schedule 1 might be of interest as the annealing algorithm then runs an additional 30% faster without seriously compromising the median least cost.

To test the utility of a descent algorithm we perform four additional tests: constant small radius, slow increasing radius, medium increasing radius, and fast increasing radius. Table 2.3 shows the resultant minimum, median and maximum least costs obtained by these four descent schedules applied 100 times to each of the four graphs. Not surprisingly, constant small radius (schedule

	Medianus-I			Medianus-II			Sparse			Dense		
	Min.	Med.	Max.	Min.	Med.	Max.	Min.	Med.	Max.	Min.	Med.	Max.
a. $t = 0, \tau_s$	22.63	23.08	24.25	20.45	20.85	22.41	22.61	22.85	24.66	50.88	53.27	56.86
b. $t = 0, \tau_m$	22.63	23.09	24.25	20.45	20.98	22.41	22.61	23.03	24.93	50.88	52.93	55.37
c. $t = 0, r_s$	22.63	23.06	24.25	20.45	20.74	22.41	22.61	22.74	24.51	50.88	51.99	54.55
d. $t = 0, \tau_f$	22.63	23.16	24.25	20.45	21.10	22.97	22.61	23.57	26.73	50.88	53.78	57.29

Table 2.3: Minimum, median, and maximum least costs found for four different descent control schedules applied 100 times to the four test graphs.

c) is found to be superior, although schedules a and b are not significantly worse. By comparing the median values in Table 2.3 by the least medians in Table 2.2 we calculate the average relative deviation as above (see Figure 2.1). It is interesting to compare schedules a, b, c and d with schedules 3, 1, 5 and 4, respectively, as they have pairwise identical radius schedules. Note that the temperature for schedules 1, 3 and 4 are constantly low and that for schedule 5 the temperature becomes low quite rapidly. So the comparison provides a means of evaluating the effect of hill climbing. The comparison quite clearly demonstrates the superiority of the annealing algorithm over the descent algorithm. Thus the phenomenon of highland basins appears to have a significant effect on the results of the descent algorithm.

2.4.2 Accuracy of solutions

Having run the annealing algorithm several hundred times with different control schedules on each of the four test graphs, we can be pretty sure that we have found the global minimum cost or at least one very close to the global minimum. Each of the 13 control schedules was run 100 times giving a total of $1300 \times 3 \times 10^4 = 3.9 \times 10^7$ iterations for each of the four graphs. A histogram of the resulting 1300 least costs for the Medianus-I graph is shown in Figure 2.2, where the height of each bar corresponds to the number of runs producing minimal configurations with exactly the same cost. This shows clearly a unique lower value for these runs

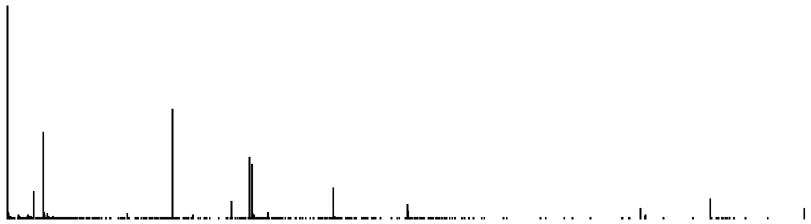

Figure 2.2: Distribution of the least costs for all runs on the Medianus-I graph.

which was obtained in roughly one out of every eight runs. Histograms with this characteristic appearance are also obtained for the other three graphs. In order to further substantiate the assertion of the four lower values (i.e. one for each graph) being the global minimum costs, the annealing algorithm was run with a control schedule complying much better with the criteria for convergence than do the pragmatic ones of Section 2.3. For none of the graphs did the algorithm with this 'almost ideal' control schedule (10^7 iterations) give better results.

2.4.3 Comparison with other methods

To get a fair and useful evaluation of the performance of the annealing algorithm we should compare it with other graph-triangulation algorithms. In the following discussion it is implicitly assumed that the term *annealing* means simulated annealing applied with control schedule 3 (t_s, τ_s).

An extensive evaluation of various heuristic graph-triangulation algorithms has been carried out by Kjærulff (1990). This evaluation was based on the same four graphs as used in Section 2.4.1. Some of the heuristic algorithms are well-known: maximum cardinality search (Tarjan and Yannakakis 1984), lexicographic search (Rose et al. 1976), minimum degree and minimum deficiency heuristics (Rose 1973). Kjærulff (1990) found a heuristic called the *minimum weight heuristic* to be superior.

By comparing the figures from Kjærulff (1990) with those of Table 2.2 it is found that the median cost produced by annealing for the Medianus-I graph is slightly better than the median cost produced by the minimum weight heuristic, whereas a similar comparison for Medianus-II shows that the heuristic algorithm is slightly better than annealing. In both cases the difference is insignificant. For the Sparse and Dense graphs the comparison clearly turns out in favour of annealing (the average state-space sizes obtained by the minimum weight heuristic are 5.5 and 3 times larger than those obtained by annealing, respectively).

It should be emphasized that the comparisons are based on average values from a large number of tests. This introduces another important parameter, namely the dispersion of the results. In Kjærulff (1990) an extremely small dispersion was found for the results of the minimum weight heuristic. A detailed study of the test runs for the annealing algorithm showed that more than 80% of the runs resulted in costs equal to or less than the median costs for the Medianus graphs. A similar study for the Sparse and Dense graphs gave 63% and 68%, respectively. Comparisons of worst-case behaviours of the annealing algorithm and the minimum weight heuristic showed the following. For Medianus-I, Medianus-II and Dense, annealing produced worst-case state-space sizes which are 2.0, 4.8 and 1.8 times as large as the corresponding worst-case results of the minimum weight heuristic, respectively. For the Sparse graph, however, the heuristic algorithm produced results which are 4.3 times as large as the worst-case results of the annealing algorithm. Similar comparisons of best-case behaviours showed that annealing produces state-space sizes which are 81%, 88%, 19% and 16% of those of the heuristic algorithm for Medianus-I, Medianus-II, Sparse and Dense, respectively.

2.5 Conclusion

This paper has investigated practical issues of applying simulated annealing to achieve approximate optimum decomposition of Bayesian networks and suggested control schedules, including the radius of the search space as an additional control parameter. Although the annealing algorithm has been successfully applied to provide high-quality approximate solutions to many NP-complete problems (Bohachevsky, Johnson and Stein 1986, Geman and Geman 1984, Lundy 1985, Thomas 1986) convergence of the algorithm in less than exponential time cannot be guaranteed for general problems (Lundy and Mees 1986). The purpose of this paper, however, has been to investigate the applicability of the algorithm under bounded resources. The specific aim was to develop an optimal control schedule such that the running time of the algorithm becomes independent of the size and topography of the search space. Of course, an important consequence of this is more coarse-grained solutions for large networks, i.e. larger absolute distances between adopted and optimum solutions. However, by assuming independence between the relative rate of convergence and the problem size this strategy implies identical relative distances. Further, keeping the relative distance at a constant level is quite acceptable for the purpose of the present problem.

With the pragmatic control schedules developed in Section 2.3 it appeared that the annealing algorithm performs best (under bounded resources) with a constant low temperature and a slowly increasing search-space radius. Strictly speaking, it is not correct to call it simulated annealing then, but to avoid confusion we will stick to that term anyway. Using the radius of the search space as an additional control parameter has a significant effect on the time consumption of the annealing algorithm and does not significantly compromise the quality of the solutions if the radius is increased only slowly.

The lesson to be learnt from the comparisons in Section 2.4.3 is that if time is not a crucial parameter the use of annealing might be recommended provided it is run more than once. The clearest example in which annealing is superior appears with the Sparse graph, where even the worst-case state-space size obtained by annealing is only 30% of the best-case size obtained by the minimum weight heuristic. On the other hand, if triangulation has to be fast we can by no means recommend the annealing algorithm which is orders of magnitude slower than the heuristic algorithms.

Thomas (1986) applied the annealing algorithm to establish optimum peeling sequences in pedigree analysis. Interestingly, his finding is that annealing performs significantly better than a greedy algorithm (which appeared to be identical to the minimum weight heuristic) even though the two applications are fairly similar. The main differences lie in a different cost-function definition (maximum clique weight instead of sum of clique weights) and the special structures of the pedigree networks.

Since simulated annealing performs best with a constant low temperature (under bounded resources) there is no reason for starting with an arbitrary vertex ordering, which would waste a lot of time testing useless orderings. Instead, it is highly recommended to apply one of the heuristic ordering algorithms first and then let the annealing process start with the ordering produced by the heuristic. This might, first, stabilize the performance of the algorithm (i.e. reduce dispersion), and second, reduce the running time.

Kjærulff (1990) also evaluated a branch-and-bound algorithm, which in a way is the deterministic equivalent of the annealing algorithm. Comparisons with Medianus-I as the target graph showed that branch-and-bound could not at all compete with annealing. A test was conducted by running the annealing and the branch-and-bound algorithms for fixed amounts of time on a heuristically ordered start configuration. In nine out of the ten cases annealing improved the initial ordering down to a level very close to the global minimum cost. Branch-and-bound, on the other hand, made modest improvements in just two of the ten cases.

Acknowledgements

I wish to thank Finn V. Jensen and Steffen L. Lauritzen for their valuable comments and suggestions during the process of producing this paper. I am also grateful to Frank Jensen who gave me his source code of the branch-and-bound algorithm. In my implementation of the simulated annealing algorithm I benefited from reusing some of the ideas from this code.

The research was conducted mainly while I was employed by Judex Datasystemer A/S, Aalborg, Denmark. It was financially supported by Judex Datasystemer A/S and The Danish Academy of Technical Sciences.

Part II

Approximation by Edge Removal

Chapter 3

Approximation of Bayesian networks through edge removals

Due to the general inherent complexity of inference in Bayesian networks, the need to compromise the exactitude of inference arises frequently. A scheme for reduction of complexity by enforcing additional conditional independences is investigated. The enforcement of independences is achieved through edge removals in a triangulated graph. The removal of a single edge may imply an enormous reduction of complexity, since other edges may become superfluous by its removal. The approximation scheme presented has several appealing features. Most notably among these, a bound on the overall approximation error can be computed locally, the bound on the error by a series of approximations equals the sum of the bounds of the errors of the individual approximations, and the influence of an approximation attenuates with increasing 'distance' from edge removed. The scheme compares in some cases very favorably with the approximation method suggested by Jensen and Andersen (1990).

The contents of this chapter is identical to Kjærulff (1993a).

3.1 Introduction

For domains of knowledge with inherent uncertainty there has been a long tradition among statisticians to model such domains in terms of random Markov fields, where the variables are represented as vertices in a graph and the statistical dependences among the variables are represented as edges of the graph (see e.g. Darroch et al. (1980)). Joint probability functions are then specified for clusters of variables. When the dependences have a causal interpretation the edges become directed and a conditional probability function is specified for each variable. In the statistical community, graphical models of this kind is often referred to as recursive graphical models (Wermuth and Lauritzen 1983). With the invention of efficient inference methods for recursive graphical models, first in the singly-connected case (Kim and Pearl 1983) and later in the general case (Lauritzen and Spiegelhalter 1988, Jensen, Lauritzen and Olesen 1990), this knowledge representation scheme has become increasingly popular in the expert systems community. Here, however, the scheme is often termed differently. The most well-known of these are influence diagram (Howard and Matheson 1984), probabilistic causal network (Cooper 1984), Bayesian belief network (Pearl 1986b), causal probabilistic network (Andreassen et al. 1987), causal network (Lauritzen and Spiegelhalter 1988). We shall use the term Bayesian network in this paper.

The directed graph of a Bayesian network is acyclic, i.e. it has no directed cycles, and captures all conditional independences between pairs of sets, say A and B, of variables given the values of a third set, C. By exploiting the conditional independence relations among the variables of a network, the underlying joint probability space may be decomposed into smaller subspaces (belief universes) corresponding to the cliques of a triangulated graph (Spiegelhalter 1986, Lauritzen and Spiegelhalter 1988). These belief universes can then be connected in a tree structure called a junction tree (Jensen 1988, Jensen, Lauritzen and Olesen 1990). However, despite the decomposition and despite the success of Bayesian networks in recent years, probabilistic inference in these networks has the inherent disadvantage of being NP-hard (Cooper 1990b) and may hence require excessive computational resources. Thus the need to compromise the exactitude of representation and inference arises frequently. In these cases the complexity of the probabilistic description inhibits or severely restricts representation and inference. Therefore, either more parsimonious methods (if any) for representation and inference should be employed or (further) approximations of the description could take place. We shall concentrate on the second alternative and specifically on approximation under edge removals (i.e. altering the conditional independence properties of a model).

Another scheme for approximating the junction tree representation of Markov models has been described by Jensen and Andersen (1990) who suggested a method based on annihilation of small probabilities by setting the k smallest probabilities to zero for each belief universe of a junction tree, where k is chosen such that the sum of the k smallest probabilities is less than a predetermined value ϵ. After the annihilation, the belief tables of the belief universes are compressed in order to take advantage of the introduced zeros.

Removing an edge has positive and negative consequences. The complexity is reduced on the positive side, whereas, on the negative side, a possibility for erroneous inference is introduced. These two conflicting interests should be reconciled through a trade-off between utility in terms of reduced complexity and loss in terms of possible erroneous inference. To maximize the reduction of complexity we obviously want to remove edges belonging to large belief universes, whereas to minimize the possible error, the dependences neglected (or the edges removed) should be the ones representing the weakest mutual information. As the measure of mutual information we employ the Kullback-Leibler divergence statistic.

It seems obvious that the impact on the system belief by approximating the probability distribution of a belief universe (clique) C cannot be entirely local in general (i.e. restricted to C), but propagates to C's neighbours via the corresponding clique separators. However, if no evidence is received and retransmitted by C after the approximation has been carried out we show that the effect of the approximation does not spread at all, and that for a certain class of edges the effect does not spread even in presence of posterior evidence. For the complementary class of edges the effect propagates throughout the network upon receiving and retransmitting evidence. Intuitively, however, it seems reasonable that the effect of an approximation in C decreases monotonically with increasing distance from C. Proofs are given that evidence received by a clique attenuates monotonically with increasing distance from the clique, and that the Kullback-Leibler divergence is additive over a series of approximations.

The rest of the paper is organized as follows. In Section 3.2 we briefly review some terminology concerning graphs, Bayesian networks, Markov fields, and junction trees. In Section 3.3 we characterize the edges which can be removed, and a discussion of the consequences of edge removal in terms of altering the potential representation, the junction tree, and the independence graph is presented in Section 3.4. In Section 3.5 we define a criterion for selecting the best edge to remove by reconciling reduction in complexity and possible erroneous inference. We show in Section 3.6 that a bound on the overall approximation error can be computed locally, that the bound on the error by a series of approximations equals the sum of the bounds of the errors of the individual approximations, and that the influence of an approximation attenuates with increasing 'distance' from edge removed. In Section 3.7 we compare the edge-removal approach with the approximation method suggested by Jensen and Andersen (1990) by applying the methods on a couple of semi-realistic models. Finally, we conclude the work in Section 3.8 and discuss possible new areas of research within the presented framework.

3.2 Terminology

3.2.1 Graphs and hypergraphs

A (finite) graph $\mathcal{G} = (V, E)$ consists of a finite set of *vertices* V and a set of *edges* $E \subseteq V \times V$ of ordered pairs of distinct vertices. If $(\alpha, \beta) \in E$ and $(\beta, \alpha) \notin E$ the edge from α to β is *directed*, which we shall denote by $\alpha \rightarrow \beta$ or $\beta \leftarrow \alpha$. If both $(\alpha, \beta) \in E$ and $(\beta, \alpha) \in E$ the edge between α and β is *undirected* in which case we shall use the shorthand notation $\{\alpha, \beta\} \in E$ or $\alpha - \beta$. A graph is said to be directed if at least one edge is directed, and undirected if all edges are undirected.

A (finite) *hypergraph* $\mathcal{H} = (V, \mathcal{C})$ consists of a finite set of vertices V and a set of edges $\mathcal{C}$, where $C \subseteq V$ and $|C| \geq 2$ for each $C \in \mathcal{C}$. A hypergraph is an undirected graph if and only if each edge has cardinality two. $\mathcal{C}$ is called a *hypergraph cover* of an undirected graph $\mathcal{G} = (V, E)$ if there is a $C \in \mathcal{C}$ for each $e \in E$ such that $e \subseteq C$. The (undirected) graph $G(\mathcal{C})$ of a hypergraph has vertex set $\bigcup_{C \in \mathcal{C}} C$ and edge set $\{\{\alpha, \beta\} \mid \exists C \in \mathcal{C} : \alpha, \beta \in C\}$.

If there is an edge (directed or undirected) between α and β in a graph $\mathcal{G} = (V, E)$, denoted by $\alpha \sim \beta$, then α and β are *adjacent*. The set of vertices adjacent to α is denoted $\mathrm{adj}(\alpha)$, and the set of edges between α and $\mathrm{adj}(\alpha)$ are called the edges *incident* to α. The set $\mathrm{cl}(\alpha) = \mathrm{adj}(\alpha) \cup \{\alpha\}$ is called the *closure* of α. If $\alpha \rightarrow \beta$, then α is called a *parent* of β and β a *child* of α. The sets of parents and children of α are denoted $\mathrm{pa}(\alpha)$ and $\mathrm{ch}(\alpha)$, respectively. If $\mathrm{pa}(\alpha) = \emptyset$, then α is a *founder*. For any subset $A \subseteq V$ we define $\mathrm{pa}(A) = \bigcup_{\alpha \in A} \mathrm{pa}(\alpha) \setminus A$, $\mathrm{ch}(A) = \bigcup_{\alpha \in A} \mathrm{ch}(\alpha) \setminus A$, etc. The subset

$$\mathrm{nb}(\alpha) = \mathrm{adj}(\alpha) \setminus \big(\mathrm{pa}(\alpha) \cup \mathrm{ch}(\alpha)\big)$$

of $\mathrm{adj}(\alpha)$ is called the *neighbours* of α. The *block* of α is the union of $\{\alpha\}$, $\mathrm{nb}(\alpha)$, $\mathrm{nb}(\beta)$ for all $\beta \in \mathrm{nb}(\alpha)$, $\mathrm{nb}(\gamma)$ for all $\gamma \in \mathrm{nb}(\beta)$, etc, and is denoted $\mathrm{blk}(\alpha)$. In other words, $\mathrm{blk}(\alpha)$ is the set of vertices reachable from α by traversing undirected edges only.

A *path* $\pi = \langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$ from α to β of length $k-1$, denoted by $\alpha \rightsquigarrow \beta$, is an ordered sequence of distinct vertices such that $\alpha_i \sim \alpha_{i+1}$ for all $i = 1, \ldots, k-1$; π is *weakly directed*, denoted by $\alpha \rightsquigarrow \beta$, if $(\alpha_i, \alpha_{i+1}) \in E$ for all $i = 1, \ldots, k-1$; π is *(strongly) directed* if it is weakly directed and $\alpha_i \rightarrow \alpha_{i+1}$ for some i $(1 \leq i \leq k-1)$; finally, π is *undirected* if it is weakly directed but not strongly directed (i.e. if $\alpha_i - \alpha_{i+1}$ for all $i = 1, \ldots, k-1$). For $A, B, C \subseteq V$, C is said to *separate* A *from* B if for all paths $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$, where $\alpha \in A$ and $\beta \in B$, $\{\alpha_1, \ldots, \alpha_k\} \cap C \neq \emptyset$. A graph $\mathcal{G}$ is *connected* if there is a path between each pair of vertices of $\mathcal{G}$. Unless otherwise stated, connectivity shall be assumed throughout this paper.

An m-cycle is a path $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$ of length m with the exception that $\alpha = \beta$. A *chord* in an m-cycle is an edge $\alpha_i — \alpha_j$ such that $1 < |i-j| < m-1$ (i.e. $\alpha_i — \alpha_j$ connects the two non-consecutive vertices α_i and α_j of the cycle). A graph $\mathcal{G}$ with no (strongly) directed cycles is called a *chain graph*. Using traditional terminology, $\mathcal{G}$ is called a *directed, acyclic graph* or DAG for short, when all edges are directed. Although a chain graph may also be directed and acyclic using our terminology, we shall stick to the conventional interpretation of the term 'DAG'. Note that for a chain graph $\mathcal{G} = (V, E)$ there exists an ordered k-tuple $(A_1, \ldots, A_k)$ of subsets of V such that (i) for each pair $\alpha, \beta \in A_i$ either $\alpha \not\sim \beta$ or $\alpha — \beta$ for all $i = 1, \ldots, k$, and (ii) for each pair α, β, where $\alpha \in A_i$ and $\beta \in A_j$, either $\alpha \not\sim \beta$ or $\alpha \rightarrow \beta$ for all $1 \leq i < j \leq k$. If there is no $l < k$ such that there exists an ordered l-tuple of subsets of V with that property, then $A_1, \ldots, A_k$ are called *chain components* of $\mathcal{G}$ and can be identified as the connected components generated by the removal of all directed edges from $\mathcal{G}$. If $\mathcal{G}$ is a DAG, then $k = |V|$ (i.e. one vertex per chain component).

In a chain graph, $\mathcal{G} = (V, E)$, the set $\mathrm{de}(\alpha)$ for which $\alpha \rightsquigarrow \beta$ for all $\beta \in \mathrm{de}(\alpha)$ is called the *descendants* of α. The set $\mathrm{an}(\alpha)$ for which $\beta \rightsquigarrow \alpha$ in $\mathcal{G}$ for all $\beta \in \mathrm{an}(\alpha)$ is called the *ancestors* of α. A subset $A \subseteq V$ is an *ancestral set* if $\mathrm{an}(\alpha) \subseteq A$ for all $\alpha \in A$. By $\mathrm{An}(A)$ we denote the smallest ancestral set containing A.

A subset $A \subseteq V$ induces a *subgraph* $\mathcal{G}_A = (A, E_A)$ of $\mathcal{G}$, where $E_A = E \cap (A \times A)$. Note that any subgraph of a chain graph is a chain graph. A graph is *complete* if all vertices are pairwise adjacent. A subset $A \subseteq V$ is complete if it induces a complete subgraph, and if A is maximal (i.e. there is no complete subset $B \subseteq V$ such that $A \subset B$), then it is called a *clique*. A vertex $\alpha \in V$ is *simplicial* if $\mathrm{adj}(\alpha)$ is complete. A hypergraph $\mathcal{H} = (V, \mathcal{C})$ is *conformal* if every clique of $G(\mathcal{C})$ is contained in an edge of $\mathcal{H}$. Thus for any conformal hypergraph with edge set $\mathcal{C}$, $\mathcal{C}$ is the clique set of $G(\mathcal{C})$.

For a chain graph $\mathcal{G} = (V, E)$ with $\mathcal{A}$ being the set of chain components of $\mathcal{G}$, $\mathcal{G}^m$ denotes its *moral graph* obtained in a two-step process: (i) add $\{\alpha, \beta\}$ to E for each pair of distinct vertices $\alpha, \beta \in V$ for which $\alpha \not\sim \beta$ and there exists a chain component $A \in \mathcal{A}$ and a pair of (not necessarily distinct) vertices $\gamma, \delta \in A$ such that $\alpha \rightarrow \gamma$ and $\beta \rightarrow \delta$, and (ii) make the resulting graph undirected (i.e. for each $(\alpha, \beta) \in E$, where $(\beta, \alpha) \notin E$, add (β, α) to E). For clarity, a moral edge between α and β shall often be denoted $\alpha \overset{m}{—} \beta$.

An undirected graph $\mathcal{G} = (V, E)$ is *triangulated* (also: *decomposable* or *chordal*) if each cycle of length greater than 3 has a chord. Another way of characterizing triangulated graphs is in terms of sequences of *(proper) decompositions*. A proper decomposition is defined as a triple (A, B, C) of disjoint and non-empty subsets of V such that $V = A \cup B \cup C$, C separates A from B, and C is a complete subset of V. A decomposition (A, B, C) *decomposes* $\mathcal{G}$ into subgraphs $\mathcal{G}_{A \cup C}$ and $\mathcal{G}_{B \cup C}$. A recursive definition of decomposability may now be stated as follows. A graph $\mathcal{G}$ is decomposable either if it is complete or if there exists a decomposition (A, B, C) into decomposable subgraphs $\mathcal{G}_{A \cup C}$ and $\mathcal{G}_{B \cup C}$. Note that the class of triangulated graphs is equivalent to the class of decomposable graphs. For a proof see e.g. Lauritzen, Speed and Vijayan (1984).

Any set T of undirected edges added to an undirected graph $\mathcal{G} = (V, E)$ such that $\mathcal{G}^t = (V, E \cup T)$ is triangulated is called a *triangulation* of $\mathcal{G}$. The edges of T are called *fill edges*, and for clarity of presentation, a fill edge between α and β shall often be denoted $\alpha \overset{f}{-} \beta$. If there is no proper subset $T' \subset T$ such that T' is a triangulation of $\mathcal{G}$, then T is said to be a *minimal triangulation*.

A hypergraph $\mathcal{H} = (V, \mathcal{C})$ is *decomposable* if and only if $G(\mathcal{C})$ is decomposable. Any decomposable hypergraph is conformal (Lauritzen et al. 1984). If $\mathcal{C}$ is a cover of some undirected graph $\mathcal{G} = (V, E)$ and $\mathcal{H}$ is decomposable, then $\mathcal{C}$ is called a *decomposable (hypergraph) cover* of $\mathcal{G}$. $\mathcal{C}$ is a *minimal decomposable cover* of $\mathcal{G}$ if there is a minimal triangulation, T, of $\mathcal{G}$ such that $\mathcal{C}$ is the clique set of $\mathcal{G} = (V, E \cup T)$.

3.2.2 Bayesian networks and Markov fields

For a chain graph $\mathcal{G} = (V, E)$ we consider a collection of discrete random variables $(X_\alpha)_{\alpha \in V}$ (i.e. a random field with V as index set) taking values in probability spaces $\mathrm{Sp}(X_\alpha)$. For brevity we shall interchangeably refer to $\alpha \in V$ as both a vertex and a variable. Thus we shall write e.g. α instead of X_α. For a subset $A \setminus V$ we let $\mathrm{Sp}(A) = \times_{\alpha \in A} \mathrm{Sp}(\alpha)$ (i.e. the Cartesian product of the state spaces of the variables in A).

A probability function p is said to *factorize* according to a chain graph $\mathcal{G} = (V, E)$ if there exist non-negative functions ϕ_A defined on $\mathrm{Sp}(A)$ such that

$$p = p_V \propto \prod_{A \in \mathcal{A}} \phi_A,$$

where $\mathcal{A}$ is the set of cliques of $\mathcal{G}^m$. The functions ϕ_A are called *factor potentials* of p. If p factorizes according to $\mathcal{G}$, then $\mathcal{G}$ is said to be an *independence graph* of p, and p is a probability function of a Bayesian network with $\mathcal{G}$ as underlying graph. The independence graph captures all conditional independence statements involving pairs of sets $A, B \subseteq V$ of variables given a third set C w.r.t. p. We shall use the notation $A \perp\!\!\!\perp B \mid C$ for any triple (A, B, C) of disjoint subsets of V for which A and B are conditionally independent given C w.r.t. p.

If the underlying graph, $\mathcal{G} = (V, E)$, of the Bayesian network is a DAG, then all conditional independence statements captured by $\mathcal{G}$ can be found using the d-separation criterion of Pearl (1988) or the equivalent criterion of Lauritzen, Dawid, Larsen and Leimer (1990). In the general case, where $\mathcal{G}$ is a chain graph, the associated probability model is called a *block recursive model* (Lauritzen and Wermuth 1989, Højsgaard and Thiesson 1992). The Markov properties (i.e. conditional independence properties) captured by a chain graph has been described by Frydenberg (1989), and is expressed by the following theorem.

Theorem 3.1 (Frydenberg (1989)) *Let* p *factorize according to a chain graph,* $\mathcal{G} = (V, E)$*. Then* $A \perp\!\!\!\perp B \mid C$ *w.r.t.* p *for any subsets* $A, B, C \subseteq V$ *whenever* C *separates* A *from* B *in* $(\mathcal{G}_{An(A \cup B \cup C)})^m$.

See Frydenberg (1989) for a proof. Note that the formulation of this theorem, which describes the *global chain Markov property*, is identical to the theorem of Lauritzen et al. (1990) describing the *directed global Markov property*, except that $\mathcal{G}$ is here a chain graph instead of a DAG.

Given a probability function (model) p obeying the conditional independence statements encoded by a chain graph $\mathcal{G} = (V, E)$, the corresponding collection of random variables (indexed by V) is called a *(chain) Markov field*, and p is said to be *(chain) Markov* w.r.t. $\mathcal{G}$. If $\mathcal{G}$ is a DAG, then the terms *directed Markov field* and *directed Markov* are often used.

3.2.3 Decomposable hypergraph covers and junction trees

By exploiting the conditional independence relations among the variables of a Bayesian network, the underlying joint probability space may be decomposed into a set of subspaces corresponding to a decomposable cover of the moralized network such that exact inference can be performed through a simple message passing scheme in a maximal spanning tree of the cover (Spiegelhalter 1986, Lauritzen and Spiegelhalter 1988, Jensen 1988, Jensen, Lauritzen and Olesen 1990). Technically, a decomposable cover of a Bayesian network with underlying chain graph $\mathcal{G}$ is created by triangulating $\mathcal{G}^m$. That is, the set of cliques of the triangulated graph is a decomposable cover of the network.

Jensen (1988) has shown that any maximal spanning tree of a decomposable cover, $\mathcal{C}$, can be used as the basis for a simple inward/outward message-passing scheme for propagation of evidence (belief updating) in Bayesian networks, where the nodes in such a tree contain collections of variables given as edges in the associated hypergraph (i.e. cliques in the corresponding triangulated graph) and where maximality is defined in terms of the sum of cardinalities of the intersections between adjacent nodes in the tree. Jensen (1988) named these trees *junction trees*. The nodes of a junction tree are called *(belief) universes* and the intersections between neighbouring belief universes are called *separators* (Jensen, Lauritzen and Olesen 1990). We shall henceforth refer to a junction tree by the pair $(\mathcal{C}, \mathcal{S})$ of belief universes and separators, and we shall use the terms 'belief universe' and 'clique' interchangeably. It can be shown that for each path $\langle C = C_1, \ldots, C_k = D \rangle$ in a junction tree, $C \cap D \subseteq C_i$ for all $1 \leq i \leq k$, which implies that $A \perp\!\!\!\perp B \mid S$ for each separator S, where A and B are the sets of variables of the two subtrees (except S) induced by the removal of the edge corresponding to S (Jensen 1988).

A junction tree $(\mathcal{C}, \mathcal{S})$ is *minimal* if there is no set $\mathcal{S}'$ such that $(\mathcal{C}, \mathcal{S}')$ is a junction tree and $\sum_{S \in \mathcal{S}'} \prod_{s \in S} \|s\| < \sum_{S \in \mathcal{S}} \prod_{s \in S} \|s\|$, where $\|s\| = |\mathrm{Sp}(s)|$. The *cost*, $C(\mathcal{C})$, of a decomposable cover, $\mathcal{C}$, is given as

$$C(\mathcal{C}) = \sum_{C \in \mathcal{C}} \prod_{c \in C} \|c\| + \sum_{S \in \mathcal{S}} \prod_{s \in S} \|s\|, \tag{3.1}$$

where $(\mathcal{C}, \mathcal{S})$ is minimal. Obviously, to minimize the complexity of inference we should preferably choose a decomposable cover of minimum cost. Unfortunately, however, the problem of determining a minimum-cost decomposable cover is NP-hard (Wen 1990). In practice, though, simple heuristic algorithms often provide near-optimum solutions (Rose 1973, Kjærulff 1992b).

To each belief universe and each separator a *belief potential* ϕ_A is associated. The joint probability distribution, p_V, associated with a junction tree with universe set $\mathcal{C}$ and separator set $\mathcal{S}$ is proportional to the *joint (system) belief* ϕ_V given by

$$p_V \propto \phi_V = \frac{\prod_{C \in \mathcal{C}} \phi_C}{\prod_{S \in \mathcal{S}} \phi_S}.$$

A belief potential ϕ_A is *normalized* if $\sum_A \phi_A = 1$. If all belief potentials of a junction tree are normalized, then ϕ_V is normalized (i.e. $p_V = \phi_V$).

A junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ is said to be *consistent* if

$$\sum_{C \setminus D} \phi_C \propto \sum_{D \setminus C} \phi_D \quad \text{for all } C, D \in \mathcal{C}.$$

That is, the marginal potentials for $C \cap D$ w.r.t. ϕ_C and ϕ_D are proportional. Consistency of Υ shall interchangeably be referred to as consistency of its associated joint belief, ϕ_V.

The basic operation of belief updating (i.e. calculation of updated potentials given incoming evidence) in junction trees is called *absorption* and is defined as follows. A belief universe C is said to *absorb* from a neighbouring universe D if the potentials ϕ_S and ϕ_C are changed to ϕ^*_S and ϕ^*_C, where $S = C \cap D$,

$$\phi^*_S = \sum_{D \setminus S} \phi_D, \quad \text{and} \quad \phi^*_C = \phi_C \frac{\phi^*_S}{\phi_S}. \tag{3.2}$$

For more detailed descriptions of junction trees including calculation of the initial potentials, belief updating, proofs of correctness, etc., see e.g. Jensen, Olesen and Andersen (1990), Jensen (1988), and Jensen, Lauritzen and Olesen (1990)

3.3 Characterization of removable edges

In this section we consider the problem of characterizing the set of edges which may be "legally" removed.

Since we consider probabilistic representations provided by decomposable covers of the moralized independence graph, the cover given by removing one of the edges must also be decomposable.

Definition 3.1 (Removable edges) For any decomposable cover let $\mathcal{G} = (V, E)$ be the corresponding triangulated graph. Then

$$\mathrm{rem}(\mathcal{G}) = \{e \in E \mid (V, E \setminus \{e\}) \text{ is triangulated}\}$$

denote the set of *removable* edges of $\mathcal{G}$. □

Thus, if $\{\alpha, \beta\} \in \mathrm{rem}(\mathcal{G})$, then removing α—β does not produce unchorded 4-cycles $\langle \alpha, \cdot, \beta, \cdot, \alpha \rangle$ or, put differently, none of the members of $\mathrm{rem}(\mathcal{G})$ are unique chords of 4-cycles. The set of removable edges is identical to the set of edges for which each member is a subset of a unique clique in the graph. These observations are formalized by

Theorem 3.2 *Let $\mathcal{G} = (V, E)$ be a triangulated graph with clique set $\mathcal{C}$ and intersection set $\mathcal{I} = \{C \cap D \mid C, D \in \mathcal{C}, C \neq D\}$, and let $E' = \mathrm{rem}(\mathcal{G})$. Let E_1, E_2 and E_3 be subsets of E given by*

(i) $E_1 = \{\{\alpha, \beta\} \in E \mid \mathit{adj}(\alpha) \cap \mathit{adj}(\beta) \textit{ is complete}\}$
(ii) $E_2 = \{e \in E \mid e \textit{ is member of one clique only}\}$
(iii) $E_3 = \{e \in E \mid \nexists I \in \mathcal{I} : e \subseteq I\}$.

E_1, E_2 and E_3 are equivalent characterizations of E', i.e. $E' \equiv E_1 \equiv E_2 \equiv E_3$.

Proof: To prove $E' \equiv E_1$ let $A = \mathrm{adj}(\alpha) \cap \mathrm{adj}(\beta)$ be the set of neighbours common to α and β. First assume $|A| < 2$, where $\{\alpha, \beta\} \in E_1$. Since $\mathcal{G}$ is triangulated there can be no 4-cycles including both α and β. Hence removing the edge $e = \{\alpha, \beta\}$ cannot produce unchorded 4-cycles and consequently $\mathcal{G}' = (V, E \setminus \{e\})$ is triangulated. Second, for $|A| \geq 2$ there is for each pair $\{\gamma, \delta\} \subseteq A$ of distinct common neighbours of α and β a 4-cycle $\langle \alpha, \gamma, \beta, \delta, \alpha \rangle$. If $\{\gamma, \delta\} \in E$ for all such pairs, then removing $e = \{\alpha, \beta\}$ clearly does not produce an unchorded 4-cycle and hence $\mathcal{G}'$ is triangulated. For each element $\{\alpha, \beta\}$ of the complementary set $E \setminus E_1$, $|A| \geq 2$ and there is a pair $\{\gamma, \delta\} \subseteq A$ such that

$\{\gamma, \delta\} \notin E$. Hence removing the edge $e = \{\alpha, \beta\}$ creates an unchorded 4-cycle $\langle \alpha, \gamma, \beta, \delta, \alpha \rangle$ rendering $\mathcal{G}' = (V, E \setminus \{e\})$ non-triangulated. Thus $E_1 \equiv E'$.

To prove $E_1 \equiv E_2$ assume $\{\alpha, \beta\} \in E_1$. Then $A = \text{adj}(\alpha) \cap \text{adj}(\beta)$ induces a complete subgraph of $\mathcal{G}$. $C = A \cup \{\alpha, \beta\}$ induces a clique; otherwise there would be a clique $C' \supset C$ such that $C' \setminus C \subset A$ implying $C' = C$. Now, assume there is another clique C^* such that $\{\alpha, \beta\} \subset C^*$. But as α—β is not a unique chord of a 4-cycle, $A \cup C^*$ must induce a complete subgraph. This, however, contradicts the fact that C is a clique. Thus C uniquely contains $\{\alpha, \beta\}$.

Alternatively, E_3 may be given as $\{\{\alpha, \beta\} \in E \mid \nexists C_1, C_2 \in \mathcal{C} : \{\alpha, \beta\} \subseteq C_1, \{\alpha, \beta\} \subseteq C_2\}$. Thus for each element $e \in E_3$ there is a unique clique $C \in \mathcal{C}$ such that $e \subseteq C$ and hence $E_2 \equiv E_3$. □

From Theorem 3.2 it follows that $\text{rem}(\mathcal{G}) \cap T = \emptyset$, where T is a minimal triangulation of (V, E) and $\mathcal{G} = (V, E \cup T)$. Thus in a minimally triangulated graph the set of removable edges is a subset of the edges of the original moral graph.

An important observation to make at this point is that any triangulated graph contains at least one removable edge (provided disconnecting the graph is allowed). In order to formally prove this statement, we first prove the following two lemmas.

Lemma 3.1 *If a graph $\mathcal{G} = (V, E)$ is triangulated, then there is a decomposition of $\mathcal{G}$.*

Proof: Follows immediately from the fact that a graph is triangulated if and only if it is decomposable (Lauritzen et al. 1984). □

Lemma 3.2 (Dirac (1961)) *Let $\mathcal{G} = (V, E)$ be a triangulated graph with $|V| \geq 2$. Then $\mathcal{G}$ has at least two simplicial vertices. If $\mathcal{G}$ is not complete, these can be chosen to be non-adjacent.*

Proof: The proof is by induction on $|V|$. For $|V| = 2$ the result is obvious. Assume that the result holds for all $|V| \leq k$ and let $|V| = k + 1$. If $\mathcal{G}$ is complete, the result is vacuously true. Otherwise, by Lemma 3.1, there exists a decomposition (A, B, C) of $\mathcal{G}$. If $A \cup C$ is complete, then $\mathcal{G}_{A \cup C}$ contains at least two simplicial vertices since A and C are disjoint and non-empty. At least one of these, say α, must be in A. If $A \cup C$ is not complete, then the induction hypothesis used on $\mathcal{G}_{A \cup C}$ yields a pair (α, β) of non-adjacent vertices that are simplicial in $\mathcal{G}_{A \cup C}$. Since C is complete, at least one of these, say α, must be in A. By symmetry there is a vertex $\gamma \in B$ that is simplicial in $\mathcal{G}_{B \cup C}$. Since C separates A from B, (α, γ) must be a pair of non-adjacent simplicial vertices in $\mathcal{G}$. □

Theorem 3.3 *Let $\mathcal{G} = (V, E)$ be a triangulated graph. Then there is at least one edge $e \in E$ such that $\mathcal{G}' = (V, E \setminus \{e\})$ is also triangulated.*

Proof: According to Lemma 3.2, $\mathcal{G}$ has at least two simplicial vertices. Let α be a simplicial vertex of $\mathcal{G}$, and let E_α be the set of edges incident to α. As $\mathrm{adj}(\alpha)$ induces a complete subgraph of $\mathcal{G}$, any subset $E'_\alpha \subseteq E_\alpha$ can be removed without compromising the chordality of $\mathcal{G}' = (V, E \setminus E'_\alpha)$. □

3.4 Consequences of edge removal

As conceived in this paper, any Bayesian network model with underlying probability model, p, consists of three distinct components:

1. a potential representation of p,
2. a decomposable hypergraph cover, and
3. an independence graph of p,

where, as discussed in Section 3.2.3, the potential representation is given as a set of belief potentials defined on the belief universes and separators of a junction tree given by a decomposable hypergraph cover of the (moralized) independence graph.

We shall assume that the decomposable covers considered are minimal. Thus, only moral edges or edges of the independence graph may be removed. Therefore, the removal of an edge manifests itself in modifying all of the three above-mentioned components of the Bayesian model. Below we discuss these three kinds of modifications separately.

3.4.1 Potential representation

First we show that only the belief potential of the universe exposed to approximation by removing an edge gets affected. A straightforward consequence thereof, is that the approximated joint belief is consistent if the joint belief before the removal was consistent.

Lemma 3.3 *For the belief potentials ϕ_C and ψ_C of a belief universe C, where*

$$\psi_C \propto \frac{\sum_\alpha \phi_C \sum_\beta \phi_C}{\sum_{\alpha,\beta} \phi_C} \quad \Leftrightarrow \quad \alpha \perp\!\!\!\perp \beta \mid C \setminus \{\alpha, \beta\} \text{ w.r.t. } \psi. \tag{3.3}$$

and $\{\alpha, \beta\} \subseteq C$, the following relations hold:

$$\sum_\alpha \phi_C \propto \sum_\alpha \psi_C, \tag{3.4}$$

$$\sum_{\beta} \phi_C \propto \sum_{\beta} \psi_C. \tag{3.5}$$

Proof: By definition

$$\sum_{\alpha} \psi_C \propto \sum_{\alpha} \frac{\sum_{\alpha} \phi_C \sum_{\beta} \phi_C}{\sum_{\alpha,\beta} \phi_C} = \frac{\sum_{\alpha} \phi_C \sum_{\alpha,\beta} \phi_C}{\sum_{\alpha,\beta} \phi_C} = \sum_{\alpha} \phi_C.$$

Since (3.3) is symmetric in α and β, (3.5) follows from (3.4). □

Thus, if C is the unique universe containing $\{\alpha, \beta\}$ and $C_1, \ldots, C_k$ are the neighbouring universes of C with $S_1 \ldots, S_k$ being the associated separators, then the removal of α—β does not affect the separator potentials $\phi_{S_1} \ldots, \phi_{S_k}$, since $S_i \subseteq C \setminus \{\alpha\}$ or $S_i \subseteq C \setminus \{\beta\}$ for all $1 \leq i \leq k$. Therefore we easily get the following corollary.

Corollary 3.1 *If ϕ is a consistent joint belief of a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$, where $C \in \mathcal{C}$ is the unique universe containing $\{\alpha, \beta\}$ and*

$$\psi_{C'} \propto \begin{cases} \dfrac{\sum_{\alpha} \phi_C \sum_{\beta} \phi_C}{\sum_{\alpha,\beta} \phi_C} & \text{if } C' = C, \\ \phi_{C'} & \text{otherwise,} \end{cases} \tag{3.6}$$

where $C' \in \mathcal{C}$, then ψ is also a consistent joint belief of Υ.

Proof: Let $\mathrm{adj}(C) = \{C_1, \ldots, C_k\}$ and let $S_i = C \cap C_i$ for all $i = 1, \ldots, k$. Since C is the unique universe containing $\{\alpha, \beta\}$ either $S_i \subseteq C \setminus \{\alpha\}$ or $S_i \subseteq C \setminus \{\beta\}$ for all $1 \leq i \leq k$. Hence by Lemma 3.3, $\phi_{S_i} \propto \psi_{S_i}$ and the result follows. □

3.4.2 Junction tree

When an edge $\{\alpha, \beta\} \subseteq C \in \mathcal{C}$ is removed from a triangulated graph $\mathcal{G} = (V, E)$ with clique set $\mathcal{C}$ one of the following three mutually exclusive statements must hold for the complete sets $C_\alpha = C \setminus \{\beta\}$ and $C_\beta = C \setminus \{\alpha\}$.

(i) Both C_α and C_β are cliques of $\mathcal{G}' = (V, E \setminus \{\{\alpha, \beta\}\})$, or

(ii) C_α (or C_β) is a clique of $\mathcal{G}'$ and C_β (or C_α) is not, or

(iii) neither C_α nor C_β are cliques of $\mathcal{G}'$.

To see that all three cases can occur consider the examples of Figure 3.1. The cost reduction (i.e. reduction of the sum of the state space sizes of the belief universes — cf. (3.1)) achieved by the removal is

$$\begin{aligned} \sigma(\alpha, \beta) &= \|C\| - \left(I_{\mathcal{C}^-}(C_\alpha)\|C_\alpha\| + I_{\mathcal{C}^-}(C_\beta)\|C_\beta\|\right) \\ &= \|C\| \left(1 - I_{\mathcal{C}^-}(C_\alpha)\|\beta\|^{-1} - I_{\mathcal{C}^-}(C_\beta)\|\alpha\|^{-1}\right), \end{aligned} \tag{3.7}$$

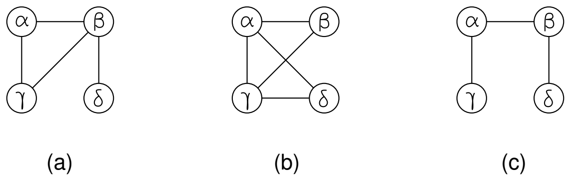

Figure 3.1: Removal of α—β. In part (a), both $\{\alpha, \gamma\}$ and $\{\beta, \gamma\}$ are new cliques. In part (b), $\{\beta, \gamma\}$ is a new clique, whereas $\{\alpha, \gamma\}$ is not. In part (c), neither $\{\alpha\}$ nor $\{\beta\}$ are new cliques.

where $\mathcal{C}^- = \mathcal{C} \setminus \{C\}$ and I is an indicator function given by

$$I_Z(z) = \begin{cases} 0 & \text{if } \exists z' \in Z : z \subseteq z' \\ 1 & \text{otherwise.} \end{cases}$$

Note that $0 \le \sigma \le \|C\|$, where $\sigma = 0$ when $\|\alpha\| = \|\beta\| = 2$ and $I_{\mathcal{C}^-}(C_\alpha) = I_{\mathcal{C}^-}(C_\beta) = 1$, and $\sigma = \|C\|$ when $I_{\mathcal{C}^-}(C_\alpha) = I_{\mathcal{C}^-}(C_\beta) = 0$.

Let $C_1, \ldots, C_k$ be the neighbours of C in a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ corresponding to $\mathcal{G}$, let $S_1, \ldots, S_k$ be the associated separators, and let ϕ be the joint belief of Υ. The new junction tree $\Upsilon' = (\mathcal{C}', \mathcal{S}')$ corresponding to $\mathcal{G}'$ has joint belief ψ and is constructed as follows, where the procedures are described according to the three above-mentioned situations.

Re (i) When C_α and C_β are both cliques of $\mathcal{G}'$ we let $\mathcal{C}' = \mathcal{C}^- \cup \{C_\alpha, C_\beta\}$ and $\mathcal{S}' = \mathcal{S} \cup \{S\}$, where $S = C_\alpha \cap C_\beta = C \setminus \{\alpha, \beta\}$, and connect C_i $(1 \le i \le k)$ to either C_α or C_β (note that S_i is a subset of either C_α or C_β for all $i = 1, \ldots, k$). To be precise,

1. Remove C from Υ.
2. Create a separator $S = C \setminus \{\alpha, \beta\}$ and let $\psi_S = \phi_S = \sum_{\alpha,\beta} \phi_C$.
3. Create two belief universes on C_α and C_β, respectively, and let $\psi_{C_\alpha} = \phi_{C_\alpha} = \sum_\beta \phi_C$ and $\psi_{C_\beta} = \phi_{C_\beta} = \sum_\alpha \phi_C$.
4. Connect C_α and C_β via S.
5. For $i = 1$ to k do
   if $\alpha \in S_i$ or ($\beta \notin S_i$ and $\|C_\alpha\| < \|C_\beta\|$) then
     connect C_i and C_α via S_i
   else
     connect C_i and C_β via S_i.

Since $S_i \cap S_j \subseteq S$ for all $1 \le i \ne j \le k$, Υ' is clearly a junction tree. Figure 3.2(i) shows Υ', where new objects and links are shown in bold.

Re (ii) Assume that C_α is a clique of $\mathcal{G}'$ (the situation is symmetric if C_β is a clique of $\mathcal{G}'$). Thus, $\mathcal{C}' = \mathcal{C}^- \cup \{C_\alpha\}$. Without loss of generality, assume that $C_\beta \subset C_k$ (i.e. $C_\beta = S_k$) such that $||C_k|| \le ||C_i||$ for all $C_i \supset C_\beta$. Therefore, we connect C_α and C_k via a new separator $S = C_\alpha \cap C_k = C \setminus \{\alpha, \beta\}$ and repeat the above procedure with C_β replaced by C_k:

1. Remove C and S_k from Υ.
2. Create a separator $S = C \setminus \{\alpha, \beta\}$ and let $\psi_S = \phi_S = \sum_{\alpha,\beta} \phi_C$.
3. Create a belief universe on C_α and let $\psi_{C_\alpha} = \phi_{C_\alpha} = \sum_\beta \phi_C$.
4. Connect C_α and C_k via S.
5. For $i = 1$ to $k-1$ do
   if $\alpha \in S_i$ or ($\beta \notin S_i$ and $||C_\alpha|| < ||C_k||$) then
     connect C_i and C_α via S_i
   else
     connect C_i and C_k via S_i.

Again, Υ' is clearly a junction tree, since $S_i \cap S_j \subseteq S$ for all $1 \le i \ne j \le k-1$. Figure 3.2(ii) shows Υ'.

Re (iii) Finally, when neither C_α nor C_β are cliques of $\mathcal{G}'$, assume (without loss of generality) that $C_\alpha \subset C_1$ and $C_\beta \subset C_k$ (i.e. $C_\alpha = S_1$ and $C_\beta = S_k$) such that $||C_1|| \le ||C_i||$ for all $C_i \supset C_\alpha$ and $||C_k|| \le ||C_j||$ for all $C_j \supset C_\beta$. Thus, we connect C_1 and C_k via a new separator $S = C_1 \cap C_k = C \setminus \{\alpha, \beta\}$ and use the procedure of case (i) with C_α replaced by C_1 and C_β replaced by C_k:

1. Remove C, S_1, and S_k from Υ.
2. Create a separator $S = C \setminus \{\alpha, \beta\}$ and let $\psi_S = \phi_S = \sum_{\alpha,\beta} \phi_C$.
3. Connect C_1 and C_k via S.
4. For $i = 2$ to $k-1$ do
   if $\alpha \in S_i$ or ($\beta \notin S_i$ and $||C_1|| < ||C_k||$) then
     connect C_i and C_1 via S_i
   else
     connect C_i and C_k via S_i.

Figure 3.2(iii) shows Υ'.

Note that in all three cases we have $S = C \setminus \{\alpha, \beta\}$ meaning that S separates Υ' into two subtrees $\Upsilon'_A = (\mathcal{C}'_A, \mathcal{S}'_A)$ and $\Upsilon'_B = (\mathcal{C}'_B, \mathcal{S}'_B)$, where A and B are the corresponding sets of variables such that $\alpha \in A$ and $\beta \in B$. From Lemma 3.3 it

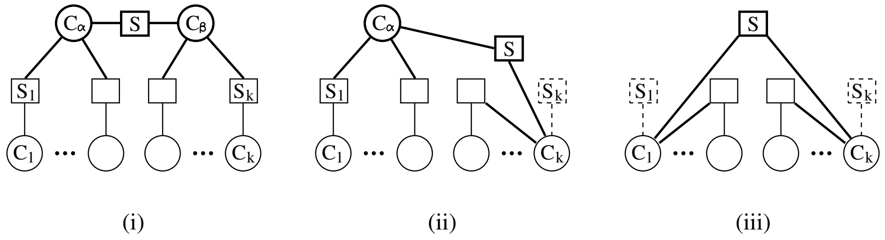

Figure 3.2: Removal of α—β results in a junction tree, Υ', with a new separator $S = C \setminus \{\alpha, \beta\}$ such that the removal of S produces a subtree Υ'_1 containing α but not β and a subtree Υ'_2 containing β but not α.

follows trivially that

$$\psi_{A \cup S} = \frac{\prod_{C' \in \mathcal{C}'_\Lambda} \psi_{C'}}{\prod_{S' \in \mathcal{S}'_\Lambda} \psi_{S'}} = \frac{\prod_{C' \in \mathcal{C}'_\Lambda} \phi_{C'}}{\prod_{S' \in \mathcal{S}'_\Lambda} \phi_{S'}} = \phi_{A \cup S}$$

and similarly $\psi_{A \cup S} = \phi_{A \cup S}$. Therefore,

$$\psi = \frac{\phi_{A \cup S} \phi_{B \cup S}}{\phi_S}. \tag{3.8}$$

3.4.3 Independence graph

Since $\psi_A = \psi_{A \cup S} = \phi_{A \cup S}$, the independence relations among the variables of the set A are unaltered by the removal of α—β, where ϕ is the 'correct' joint belief and ψ is the approximate one given by (3.8). The same applies to the sets B and S. That is, the marginal independence graphs for A, B and S are identical for ϕ and ψ. Therefore, the problem of determining the independence graph of ψ may be formulated as the problem of combining marginal independence graphs such that the independence statements expressed by these are not violated and such that the combined graph represents the fact that $A \perp\!\!\!\perp B \mid S$ (or $A \setminus S \perp\!\!\!\perp B \setminus S \mid S$ to be exact).

Given an independence graph of a probability function (belief potential), $p = p_V$, we need a way of establishing an independence graph of any marginal p_A, $A \subseteq V$. First, however, we shall introduce the notion of minimal directedness.

Definition 3.2 (Minimal directedness) A chain graph $\mathcal{G} = (V, E)$ is *minimally directed* if it does not contain chain components $K, P \subset V$ such that $\mathrm{pa}(K)$ is a non-empty, complete subset of P. $\square$

Theorem 3.4 *Let $\mathcal{G} = (V, E)$ be a chain graph containing chain components $K, P \subset V$ such that $pa(K)$ is a non-empty, complete subset of P (i.e. $\mathcal{G}$ is not minimally directed) and let the graph $\mathcal{G}'$ be obtained by replacing all directed edges between K and P by undirected edges. Then $\mathcal{G}'$ is a chain graph and the independence properties represented by $\mathcal{G}$ and $\mathcal{G}'$ are identical.*

Proof: The addition of (β, α) to E for each $\alpha \to \beta$ ($\beta \in K, \alpha \in \mathrm{pa}(\beta)$) possibly introduces weakly directed cycles $\langle \beta, \alpha = \alpha_0, \dots, \alpha_m = \gamma, \delta = \beta_0, \dots, \beta_n = \beta \rangle$, where $\{\alpha_i\}_{i=0}^m \subseteq P$ and $\{\beta_i\}_{i=0}^n \subseteq K$. But, since all involved edges are undirected, no (strongly) directed cycles are introduced, and hence $\mathcal{G}'$ is a chain graph.

To see that $\mathcal{G}$ and $\mathcal{G}'$ represent identical independence statements, recall the criterion of Theorem 3.1 stating that $A \perp\!\!\!\perp B \mid C$ whenever C separates A from B in $(\mathcal{G}_{\mathrm{An}(A \cup B \cup C)})^m$. When $\{\alpha, \beta\} \cap \mathrm{An}(A \cup B \cup C) = \emptyset$ in $\mathcal{G}$ (and $\mathcal{G}'$) for all $\beta \in K$ and $\alpha \in \mathrm{pa}(\beta)$, $(\mathcal{G}_{\mathrm{An}(A \cup B \cup C)})^m$ and $(\mathcal{G}'_{\mathrm{An}(A \cup B \cup C)})^m$ are obviously identical. When $\{\alpha, \beta\} \cap \mathrm{An}(A \cup B \cup C) \neq \emptyset$ in $\mathcal{G}$ for some $\beta \in K$ and $\alpha \in \mathrm{pa}(\beta)$, $(\mathcal{G}_{\mathrm{An}(A \cup B \cup C)})^m$ and $(\mathcal{G}'_{\mathrm{An}(A \cup B \cup C)})^m$ are identical with the possible exception of a subgraph $\mathcal{G}'_{K^*}$, where $\mathrm{an}(\alpha) \supset K^* \subset K$, when $(A \cup B \cup C) \cap K^* = \emptyset$. But in that case $\mathcal{G}'_{K^*}$ has no influence on the separability of A and B by C, since K^* and $\mathrm{An}(A \cup B \cup C) \setminus K^*$ are separated by the complete set $\mathrm{pa}(\alpha)$ in $(\mathcal{G}'_{\mathrm{An}(A \cup B \cup C)})^m$. □

Thus, for any independence graph which is not minimally directed, there is a minimally directed graph with identical independence structure obtained by successive replacements of sets of directed edges with undirected edges, where the set of edges to be replaced complies with the restrictions of Theorem 3.4. Frydenberg (1989) shows an equivalent result stating that two chain graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V, E')$ with identical undirected graphs (i.e. graphs obtained by replacing all directed edges with undirected ones) represent the same independence statements if $E \subseteq E'$ (i.e. $\mathcal{G}$ may contain directed edges where the equivalent ones in $\mathcal{G}'$ are undirected) and $\mathrm{pa}(K)$ is complete in $(\mathcal{G}_L)^m$ for each chain component K of $\mathcal{G}'$, where $L = K \cup \mathrm{pa}(K)$.

Definition 3.3 (Correct and perfect independence graphs) Let $\mathcal{G} = (V, E)$ be any chain graph and p any probability function (belief potential) defined on $\mathrm{Sp}(V)$. If $\mathcal{G}$ does not represent (conditional) independence properties inconsistent with p, then $\mathcal{G}$ is a *correct* independence graph of p. If $\mathcal{G}$ is a correct independence graph of p and all (conditional) independence properties of p are represented by $\mathcal{G}$, then $\mathcal{G}$ is a *perfect* independence graph of p. □

We shall often say '$\mathcal{G}$ is an independence graph of p' with correctness being understood. Further, we shall use the shorthands: '$\mathcal{G}$ is correct' and '$\mathcal{G}$ is perfect', when it is understood that $\mathcal{G}$ is an independence graph of a given probability function.

Below, we use the symbol $\otimes$ to denote the binary operator producing the set of all ordered pairs of distinct elements of its arguments.

Definition 3.4 (Marginal independence graph) Let $\mathcal{G} = (V, E)$ be any chain graph and let $\alpha \in V$ be any vertex of $\mathcal{G}$. Then $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}} = (V \setminus \{\alpha\}, E^{\downarrow}_{V\setminus\{\alpha\}})$ is given by $E^{\downarrow}_{V\setminus\{\alpha\}} = E' \cup E^*$, where

$$E' = E \setminus E_\alpha \cup (\mathrm{nb}(\alpha) \otimes \mathrm{nb}(\alpha)) \cup \big(\mathrm{pa}(\alpha) \times (\mathrm{nb}(\alpha) \cup \mathrm{ch}(\alpha))\big) \cup (\mathrm{nb}(\alpha) \times \mathrm{ch}(\alpha)),$$

(E_α is the set of edge incident to α) and E^* denotes the set of edges rendering $\mathrm{ch}(\alpha)$ complete in such a way that

- $\beta — \gamma$ if β and γ belong to the same chain component,
- $\beta \rightarrow \gamma$ if the chain component containing β precedes the component containing γ,
- $\beta \leftarrow \gamma$ otherwise,

for any two non-adjacent vertices $\beta, \gamma \in \mathrm{ch}(\alpha)$ in $\mathcal{G}$. □

In words, $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is constructed by making the neighbours of α complete, adding directed edges from the parents of α to its neighbours and children, adding directed edges from the neighbours of α to its children, making the children of α complete in such a way that no (strongly) directed paths are introduced, and removing α and the edges incident to it. Figure 3.3 illustrates this process.

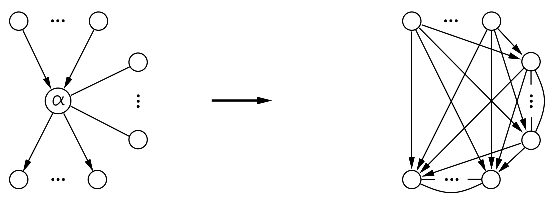

Figure 3.3: Marginalization w.r.t. an unobserved variable α render $\mathrm{ch}(\alpha)$ and $\mathrm{nb}(\alpha)$ complete and adds directed edges from $\mathrm{pa}(\alpha)$ to $\mathrm{nb}(\alpha) \cup \mathrm{ch}(\alpha)$ and from $\mathrm{nb}(\alpha)$ to $\mathrm{ch}(\alpha)$.

Lemma 3.4 *For any chain graph $\mathcal{G} = (V, E)$ and any $\alpha \in V$, $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is a chain graph.*

Proof: Since $\mathrm{nb}(\alpha)$ is a subset of a single chain component in $\mathcal{G}$, completion of $\mathrm{nb}(\alpha)$ by addition of undirected edges cannot introduce directed cycles. The addition of $\pi \rightarrow \mu$ for each $\pi \in \mathrm{pa}(\alpha)$ and $\mu \in \mathrm{nb}(\alpha) \cup \mathrm{ch}(\alpha)$ cannot introduce directed cycles either, since π belongs to a chain component of lower order than the component containing μ (i.e. any path from π to μ is strongly directed). The same is obviously true for the directed edges from $\mathrm{nb}(\alpha)$ to $\mathrm{ch}(\alpha)$. Finally, since the edges added between pairs $\beta, \gamma \in \mathrm{ch}(\alpha)$ are directed only if β and γ belong to different chain components and such that the direction of the edge follows the ordering of the components, we conclude that $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is a chain graph. □

Below we prove that $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is an independence graph of the marginal distribution $\sum_\alpha p$ if $\mathcal{G}$ is an independence graph of a probability function p. Note that correctness of $\mathcal{G}' = \mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ follows if separation of A and B by C in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$ implies separation in $(\mathcal{G}_{\mathrm{An}(A\cup B\cup C)})^m$ as well, and that perfectness of $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ follows if separation in $(\mathcal{G}_{\mathrm{An}(A\cup B\cup C)})^m$ implies separation in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$ provided $\mathcal{G}$ is perfect.

Note: If a path between A and B contains one or more vertices in C we shall say that the path is *blocked* (by C). Similarly, if a path between A and B contains no vertices in C, it is *unblocked*.

Theorem 3.5 *Let the chain graph $\mathcal{G} = (V, E)$ be an independence graph of $p = p_V$, where $\alpha \in V$ is any unobserved variable. Then $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is an independence graph of $p_{V\setminus\{\alpha\}} = \sum_\alpha p_V$.*

Proof: Let $A, B, C \subseteq V \setminus \{\alpha\}$ be any disjoint subsets such that C separates A and B in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$, where $\mathcal{G}' = \mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$, and let $\pi \in A$ and $\mu \in B$ be arbitrary vertices. Assume that there is an unblocked path between π and μ in $(\mathcal{G}_{\mathrm{An}(A\cup B\cup C)})^m$. Since $\mathcal{G}$ is a subgraph of $(V, E^{\downarrow}_{V\setminus\{\alpha\}} \cup E_\alpha)$, the path must include α. Let $\langle \pi, \ldots, \beta, \alpha, \gamma, \ldots, \mu \rangle$ be such a path. Again, since $\mathcal{G}$ is a subgraph of $(V, E^{\downarrow}_{V\setminus\{\alpha\}} \cup E_\alpha)$, the subpaths $\langle \pi, \ldots, \beta \rangle$ and $\langle \gamma, \ldots, \mu \rangle$ must also be paths in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$. By the construction of $E^{\downarrow}_{V\setminus\{\alpha\}}$, $\beta \sim \gamma$ in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$ violating the assumption of A and B being separated by C in $(\mathcal{G}'_{\mathrm{An}(A\cup B\cup C)})^m$. □

The following example shows that $\mathcal{G}^{\downarrow}_{V\setminus\{\alpha\}}$ is not necessarily perfect when $\mathcal{G}$ is perfect. Let $V = \{\alpha, \beta, \gamma, \delta, \epsilon\}$ and $p = p_V$ be Markov w.r.t. the DAG in Figure 3.4a. Since $\beta \not\perp\!\!\!\perp \delta \mid \{\gamma, \epsilon\}$ w.r.t. to p (and $p_{V\setminus\{\alpha\}}$), we must have $\beta \sim \delta$ in an independence graph of $p_{V\setminus\{\alpha\}} = \sum_\alpha p$, and, since $\gamma \perp\!\!\!\perp \epsilon$ and $\gamma \not\perp\!\!\!\perp \epsilon \mid \beta$, a candidate independence graph of $p_{V\setminus\{\alpha\}}$ could be the one in Figure 3.4b. However, since $\gamma \perp\!\!\!\perp \epsilon \mid \delta$ w.r.t. to p (and $p_{V\setminus\{\alpha\}}$), this graph is not perfect, but it is correct, since it does not represent non-existing conditional independences.

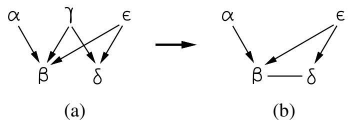

Figure 3.4: $\gamma \perp\!\!\!\perp \epsilon \mid \delta$ w.r.t. to p (and $\sum_\alpha p$) which is Markov w.r.t. the DAG in part (a). However, $\sum_\alpha p$ is not Markov w.r.t. the graph in part (b), since according to that $\gamma \not\!\perp\!\!\!\perp \epsilon \mid \delta$.

Thus, all the independence properties of $p_{V\setminus\{\alpha\}}$ cannot be represented by a single chain graph. If we want a perfect representation, a more sophisticated language must be adopted. One such language may be given by the class of annotated graphs (Geva and Paz 1992). However, in the present paper we shall refrain from pursuing this any further.

Corollary 3.2 *If $\mathcal{G} = (V, E)$ is a chain graph and $A \subseteq V$ is a set of founders, then $\mathcal{G}^{\downarrow}_A = \mathcal{G}_A$.*

Proof: Follows immediately from Definition 3.4. □

Corollary 3.3 *Let $\mathcal{G} = (V, E)$ be a chain graph and $\alpha, \beta \in V$ any pair of non-adjacent vertices. Then $\alpha \sim \beta$ in $\mathcal{G}^{\downarrow}_{V\setminus \mathrm{an}(\{\alpha,\beta\})}$ if $\mathrm{an}(\alpha) \cap \mathrm{an}(\beta) \neq \emptyset$.*

Proof: The proof is by induction on $|\mathrm{an}(\{\alpha, \beta\})|$. If $|\mathrm{an}(\{\alpha, \beta\})| = \{\gamma\}$, then $\mathrm{an}(\alpha) = \mathrm{an}(\beta) = \{\gamma\}$ and, by Definition 3.4, $\alpha \sim \beta$ in $\mathcal{G}^{\downarrow}_{V\setminus\{\gamma\}} = \mathcal{G}^{\downarrow}_{V\setminus \mathrm{an}(\{\alpha,\beta\})}$. Assume that the result holds for $|\mathrm{an}(\{\alpha, \beta\})| < k$, and let $|\mathrm{an}(\{\alpha, \beta\})| = k$. Let γ be the variable to be eliminated first and let $A = (\mathrm{ch}(\gamma) \cup \mathrm{nb}(\gamma)) \cap \mathrm{an}(\{\alpha, \beta\})$.

First, assume that $\gamma \in \mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$. If $A = \{\delta\}$, then $\delta \in \mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$ in $\mathcal{G}^{\downarrow}_{V\setminus\{\gamma\}}$, and, by the induction hypothesis, the result holds. Let $\mu, \pi \in A$ be any two distinct vertices in A. Since $\mu \sim \pi$ in $\mathcal{G}^{\downarrow}_{V\setminus\{\gamma\}}$, at least one of μ, π will be a member of $\mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$ in $\mathcal{G}^{\downarrow}_{V\setminus\{\gamma\}}$ making the result hold.

Second, assume that $\gamma \in \mathrm{an}(\alpha)$ and let $\delta \in \mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$. If $\delta \notin \mathrm{an}(\gamma)$, then the elimination of γ clearly does not affect δ's membership of $\mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$. If $\delta \in \mathrm{an}(\gamma)$, then, by Definition 3.4, the elimination of γ will add edges such that δ's membership of $\mathrm{an}(\alpha) \cap \mathrm{an}(\beta)$ will not be affected. In both cases, the induction hypothesis makes the result hold. □

Definition 3.4 provides a method for constructing a (marginal) independence graph of the marginal distribution $p_{V\setminus\{\alpha\}}$ (i.e. it deals with the case where α is unobserved). However, the construction of an independence graph of the approximate joint belief $\psi = \phi_{A \cup S} \phi_{B|S}$ involves combination of a marginal independence graph and a conditional (marginal) independence graph. The (conditional) independence graph of the conditional distribution $p_{V|\alpha}$ is obtained simply by moralizing the subgraph induced by $An(\alpha)$ and removing α and the edges incident to it.

Theorem 3.6 *Let the chain graph $\mathcal{G} = (V, E)$ be an independence graph of $p = p_V$ and let $E^m_{An(A)}$ be the edge set of $(\mathcal{G}_{An(A)})^m$. Then $\mathcal{G}' = (V, E \cup E^m_{An(A)})$ is a chain graph and a conditional independence graph of $p_{V|A}$.*

Proof: Since $\mathcal{G}$ is a chain graph, $An(A)$ is a chain component of $\mathcal{G}'$, and there are no directed edges from $V \setminus An(S)$ to $An(S)$, $\mathcal{G}'$ is also a chain graph. The fact that $\mathcal{G}'$ is a conditional independence graph of $p_{V|A}$ follows directly from Theorem 3.1. □

By the methods of Definition 3.4 and Theorem 3.6 we can construct any marginal independence graph (possibly conditional on a set of variables) by successive removal of the relevant variables.

Note that the presence of the set A and the edges incident to A in the independence graph of $p_{V|A}$ is unnecessary for a correct interpretation of the conditional independence relations among variables in $V \setminus A$ given A (this is stated formally by Corollary 3.4 below). However, when combining a conditional and a marginal independence graph to obtain a joint independence graph, these vertices and edges are needed. Thus, we shall assume that the variables upon which we condition are included in the conditional independence graphs considered. Theorem 3.7 below states that a joint independence graph can be formed by simple graph union of a conditional and a marginal independence graph.

Corollary 3.4 *Let the chain graph $\mathcal{G}_{V|A} = (V, E)$ be a conditional independence graph of $p_{V|A}$, where $A \subseteq V$. Then $\mathcal{G}'_{V|A} = (V, E \setminus ((E_A \otimes E_A) \cup (nb(A) \times E_A)))$ is a chain graph and a conditional independence graph of $p_{V|A}$.*

Proof: Obviously, the operation does not introduce directed cycles; $\mathcal{G}'_{V|A}$ is therefore a chain graph. Any path involving a variable in A is blocked. Hence, the removal of any subset, $\{(\alpha, \beta)\} \subseteq E$, where $\alpha, \beta \in A$ or $\alpha \in nb(A)$ and $\beta \in A$, does not alter the conditional independence representation given A. □

When combining a conditional and a marginal independence graph, we shall assume that the conditional one has been exposed to the transformation described in Corollary 3.4, since, in general, we cannot otherwise be sure that the resulting graph will be a chain graph.

Theorem 3.7 *Let the chain graph $\mathcal{G}^{\downarrow}_{A\cup S} = (A, E^{\downarrow}_{A})$ be a marginal independence graph of $p_{A\cup S}$ and the chain graph $\mathcal{G}^{\downarrow}_{B\mid S} = (B, E^{\downarrow}_{B})$ a conditional independence graph of $p_{B\mid S}$, where $A, B, S \subseteq V$ such that $A \cap B = S$ and $A \perp\!\!\!\perp B \mid S$ w.r.t. $p = p_V$. Then $\mathcal{G}' = \mathcal{G}^{\downarrow}_{A\cup S} \cup \mathcal{G}^{\downarrow}_{B\mid S}$ is a chain graph and an independence graph of p. Further, $\mathcal{G}'$ is perfect if both $\mathcal{G}^{\downarrow}_{A\cup S}$ and $\mathcal{G}^{\downarrow}_{B\mid S}$ are perfect.*

Proof: For $\mathcal{G}'$ to be an independence graph of p we must require that

(i) any path from $\alpha \in A \setminus S$ to $\beta \in B \setminus S$ is blocked by S in $(\mathcal{G}'_{\mathrm{An}(\{\alpha,\beta\}\cup S)})^m$ (i.e. the fact that $A \perp\!\!\!\perp B \mid S$ w.r.t. p must be reflected by $\mathcal{G}'$), and

(ii) $\mathcal{G}'^{\downarrow}_{A\cup S}$ and $\mathcal{G}'^{\downarrow}_{B\cup S}$ are independence graphs of $p_{A\cup S}$ and $p_{B\cup S}$, respectively.

Since S is a founder set of $\mathcal{G}'_{B\cup S}$, there can be no moral edge between α and β in $(\mathcal{G}'_{\mathrm{An}(\{\alpha,\beta\}\cup S)})^m$ whenever $\alpha \in A \setminus S$ and $\beta \in B \setminus S$. Thus, since $A \cap B = S$ and $\mathcal{G}' = \mathcal{G}^{\downarrow}_{A\cup S} \cup \mathcal{G}^{\downarrow}_{B\mid S}$, (i) is fulfilled.

Now, there can be no directed cycles in $\mathcal{G}'$, since (i) is fulfilled, S is a founder set of $\mathcal{G}'_{B\cup S} = \mathcal{G}^{\downarrow}_{B\mid S}$, and both $\mathcal{G}^{\downarrow}_{A\cup S}$ and $\mathcal{G}^{\downarrow}_{B\mid S}$ are chain graphs. That is, $\mathcal{G}'$ is a chain graph.

Since S is a founder set of $\mathcal{G}'_{B\cup S} = \mathcal{G}^{\downarrow}_{B\mid S}$, it follows by Corollary 3.2 that $(\mathcal{G}'^{\downarrow}_{A\cup S})_S = \mathcal{G}'_S$. Therefore $\mathcal{G}'^{\downarrow}_{A\cup S} = \mathcal{G}^{\downarrow}_{A\cup S}$ implying that $\mathcal{G}'^{\downarrow}_{A\cup S}$ is an independence graph of $p_{A\cup S}$.

For $\mathcal{G}'^{\downarrow}_{B\cup S}$ to be an (not necessarily perfect) independence graph of $p_{B\cup S}$, we must require that

$$\left((\mathcal{G}^{\downarrow}_{B\cup S})_{\mathrm{An}(F\cup G\cup H)}\right)^m \subseteq \left((\mathcal{G}'^{\downarrow}_{B\cup S})_{\mathrm{An}(F\cup G\cup H)}\right)^m$$

for any $F, G, H \subseteq B$. Let $B' = \mathrm{An}(S)$ w.r.t. $\mathcal{G}^{\downarrow}_{B\cup S}$, $B'' = B \setminus B'$, and $N = \mathrm{ch}(S) \cap B'$, where $\mathrm{ch}(S)$ is determined w.r.t. $\mathcal{G}'^{\downarrow}_{B\cup S}$. Since $\mathcal{G}'^{\downarrow}_{A\cup S} = \mathcal{G}^{\downarrow}_{A\cup S}$ and by the construction of $\mathcal{G}^{\downarrow}_{B\mid S}$, it follows that $(\mathcal{G}'^{\downarrow}_{B\cup S})_{B''\cup S} = (\mathcal{G}^{\downarrow}_{B\cup S})_{B''\cup S}$. Further, since

$$(\mathcal{G}'^{\downarrow}_{B\cup S})_{B'\setminus S} = \left((\mathcal{G}^{\downarrow}_{B\cup S})_{B'\setminus S}\right)^m,$$

a possible difference between $((\mathcal{G}^{\downarrow}_{B\cup S})_{\mathrm{An}(F\cup G\cup H)})^m$ and $((\mathcal{G}'^{\downarrow}_{B\cup S})_{\mathrm{An}(F\cup G\cup H)})^m$ can only be caused by differences between the subgraphs induced by $S\cup N$, and, more specifically, only by differences in the two sets of edges between S and N. Let E' and E'' be these two sets pertaining to $\mathcal{G}^{\downarrow}_{B\cup S}$ and $\mathcal{G}'^{\downarrow}_{B\cup S}$, respectively. We note that $N \subseteq \mathrm{An}(S)$ in $\mathcal{G}^{\downarrow}_{B\cup S}$ and $N \subseteq \mathrm{ch}(S)$ in $\mathcal{G}'^{\downarrow}_{B\cup S}$. Thus, for $\mathcal{G}'^{\downarrow}_{B\cup S}$ to be an independence graph of $p_{B\cup S}$, we must require that $\alpha \sim \beta$ in $\mathcal{G}'^{\downarrow}_{B\cup S}$ whenever there are (not necessarily distinct) variables $\gamma, \delta \in N$ such that $\alpha \to \gamma$ and $\beta \to \delta$

and distinct variables $\alpha, \beta \in S$. However, by Corollary 3.3 it follows that $\alpha \sim \beta$ in $\mathcal{G}^{\downarrow}_{A \cup S}$, and (again), since $\mathcal{G}'^{\downarrow}_{A \cup S} = \mathcal{G}^{\downarrow}_{A \cup S}$, it follows that $(\mathcal{G}'^{\downarrow}_{B \cup S})_S = (\mathcal{G}^{\downarrow}_{B \cup S})_S$. Thus, the independence relations represented by $\mathcal{G}'^{\downarrow}_{B \cup S}$ are indifferent to the (lack of) direction of the edges between S and N. Therefore, $\mathcal{G}'^{\downarrow}_{B \cup S}$ and $\mathcal{G}^{\downarrow}_{B \cup S}$ are equivalent representations of the independence properties of $p_{B \cup S}$. Since the combination of $\mathcal{G}^{\downarrow}_{A \cup S}$ and $\mathcal{G}^{\downarrow}_{B | S}$ did not involve addition of edges, $\mathcal{G}'$ is perfect if both $\mathcal{G}^{\downarrow}_{A \cup S}$ and $\mathcal{G}^{\downarrow}_{B | S}$ are perfect. □

3.4.4 Some examples

In this section we present a few examples illustrating the modification of the independence graph when an edge is removed and show that the new decomposable cover may become significantly simpler.

The first example shows an independence graph consisting of a 4-cycle as illustrated in Figure 3.5. A decomposable cover may be generated by adding a

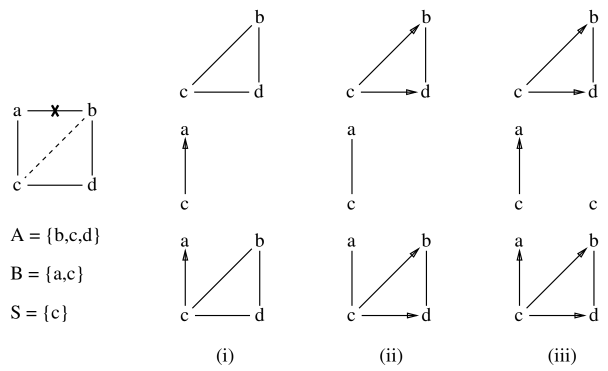

Figure 3.5: By removing a—b the independence graph of the associated approximate joint belief can be created by combining the marginal independence graph for $A \cup S$ and the conditional one for $B \mid S$ (i), the marginal graph for $B \cup S$ and the conditional one for $A \mid S$ (ii), or the conditional graphs for $A \mid S$ and $B \mid S$ and the marginal one for S (iii).

fill edge between b and c. Each of the four edges of the independence graph is removable. Suppose we remove a—b. Using the above notation we get e.g. $A = \{b, c, d\}$, $B = \{a, c\}$ and $S = \{c\}$. Following the above results we should now determine a marginal independence graph and a conditional one, and then combine these into a new joint independence graph. This combination can involve three principally different sets of marginal and conditional graphs:

1. marginal graph for $A \cup S$ plus conditional one for $B \mid S$,
2. marginal graph for $B \cup S$ plus conditional one for $A \mid S$,
3. conditional graphs for $A \mid S$ and $B \mid S$ plus marginal one for S.

Parts (i)–(iii) in Figure 3.5, respectively, show these three possibilities, where e.g. the upper graph in part (i) shows the marginal graph for $A \cup S$, the middle one the conditional graph for $B \mid S$, and the lower one the resulting combined graph. In this example the resulting new independence graphs appear to be equivalent, since, by Theorem 3.4, they are all equivalent to the graph obtained by substituting all the directed edges by undirected ones.

This small example shows an interesting feature, namely that the choice of decomposable cover affects the topology of the resulting independence graph. If, for example, the alternative (minimal) cover given by substituting $b \overset{f}{—} c$ with $a \overset{f}{—} d$ was applied, the resulting independence graph obtained by removing $a — b$ will be different, as shown in Figure 3.6. In both examples we observe that the

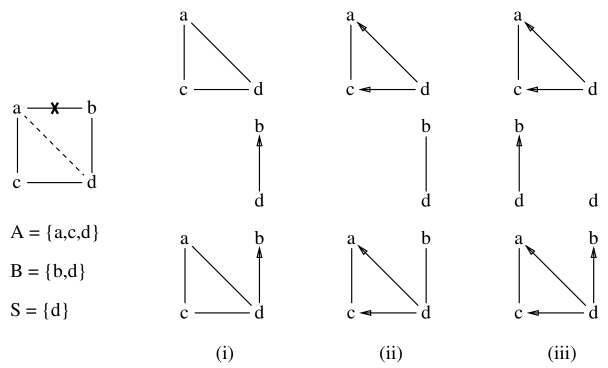

Figure 3.6: Similar to Figure 3.5 except that a different decomposable cover is used. This results in a different combined independence graph.

fill edge becomes part of the independence graph.

A bit more complicated example is shown in Figure 3.7, where a minimal decomposable cover is given by addition of the fill edges indicated by the dashed lines. The three possible ways of generating the independence graph of the approximate joint belief are shown in parts (i)–(iii) corresponding to the possibilities 1–3 above (note: none of the combined graphs are minimally directed).

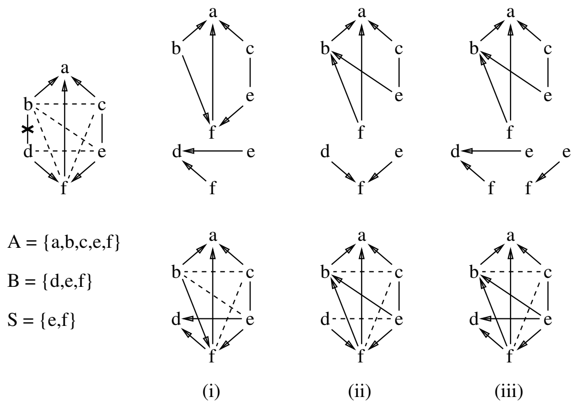

Figure 3.7: Removal of b—d result in the corresponding independence graph of part (iii) if the two conditional graphs for $A \mid S$ and $B \mid S$ and the marginal one of S are combined, which contains one edge more than those of parts (i)–(ii). None of the graphs are perfect, since the fact that $d \perp\!\!\!\perp e$ and $b \perp\!\!\!\perp e$ are not represented.

In both of the conditional graphs for $A \mid S$ and $B \mid S$ one edge gets added by moralization. In the graph for $A \mid S$, b—e gets added (see parts (ii)–(iii)), and in the graph for $B \mid S$, d—e gets added (see parts (i) and (iii)). This means that the combined independence graph of part (iii) contains one edge more than those of parts (i)–(ii). Thus, one of the first two are preferable, since they are closer representations of the independence properties of the approximate joint belief. However, none of the graphs are perfect representations, since, by the marginal graphs for $A \cup S$ and $B \cup S$, $d \perp\!\!\!\perp e$ and $b \perp\!\!\!\perp e$, which all of the combined graphs fail to express. In fact, the three independence statements $d \perp\!\!\!\perp e$, $b \perp\!\!\!\perp e$, and $b \perp\!\!\!\perp d \mid (e, f)$ cannot simultaneously by represented in a single chain graph. (Note that the conditional independence of the variables between which the edge was removed given the remaining variables (the set called S above) of the relevant belief universe ($b \perp\!\!\!\perp d \mid (e, f)$ in this example) will always be properly represented in any of the resulting combined graphs.) The statement $b \perp\!\!\!\perp e$ is represented in the combined graph of part (i) while $d \perp\!\!\!\perp e$ is not, and vice versa for part (ii).

In selecting one of the three alternatives for determining an independence graph of the approximate joint belief we should generally choose the conditional graph for which we have to add the fewest number of moral edges, since the more edges are added the less perfect the resulting combined independence graph will be. Thus alternative 3 in the above list will never provide better results than will alternatives 1 and 2. In all of the previous examples the decomposable cover of the approximate independence graph simply contains one edge less than the original cover, namely the edge removed. However, this is generally not true, since the removal of an edge may render some of the moral and fill edges superfluous. That is, a minimal cover of the approximate independence graph will not contain these edges. Figure 3.8 shows an example, where the fill edges $\mathrm{b} \stackrel{f}{—} \mathrm{c}$

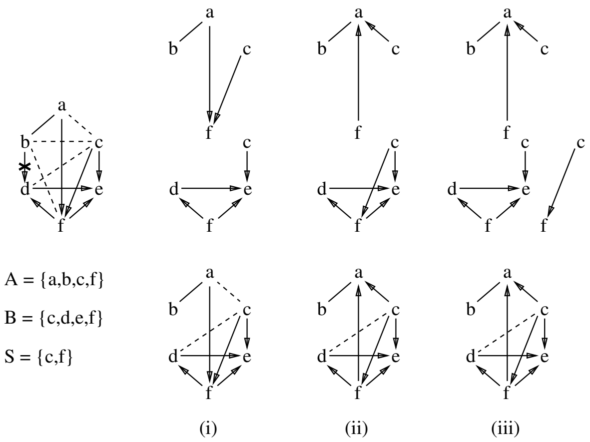

Figure 3.8: The independence graph obtained by combining the marginal graph for $A \cup S$ and the conditional one for $B \mid S$ is perfect (i). In all three cases the fill edges $\mathrm{b} \stackrel{f}{—} \mathrm{c}$ and $\mathrm{b} \stackrel{f}{—} \mathrm{f}$ are rendered superfluous.

and $\mathrm{b} \stackrel{f}{—} \mathrm{f}$ are rendered superfluous by the removal of $\mathrm{b} — \mathrm{d}$. Again, for pedagogical reasons, minimal directedness has not been imposed. This example clearly demonstrates that alternative 1 (i.e. marginal graph for $A \cup S$ and conditional one for $B \mid S$) is preferable, since no moral edges get added in creating the conditional graph for $B \mid S$. A heuristic for choosing the optimal marginal graph upon which to perform conditioning w.r.t. S could be to select the one where $|\mathrm{pa}(S)|$ is smallest.

The final example in Figure 3.9 shows the resulting independence graphs by a series of edge removals, where minimal directedness is imposed. The two first removals each render two fill edges superfluous. The third does not give rise to removal of superfluous edges, but it makes the moral edge between h and i become an edge of the new independence graph and hence, the directed edges

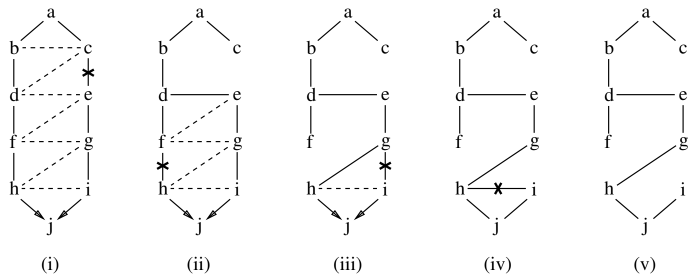

Figure 3.9: The removal of c—e renders $\mathrm{b}\overset{\mathrm{f}}{—}\mathrm{c}$ and $\mathrm{c}\overset{\mathrm{f}}{—}\mathrm{d}$ superfluous and makes $\mathrm{d}\overset{\mathrm{f}}{—}\mathrm{e}$ become part of the new independence graph. Similarly, the removal of f—h renders $\mathrm{e}\overset{\mathrm{f}}{—}\mathrm{f}$ and $\mathrm{f}\overset{\mathrm{f}}{—}\mathrm{g}$ superfluous and makes $\mathrm{g}\overset{\mathrm{f}}{—}\mathrm{h}$ become an edge of the new independence graph. By the removal of g—i, $\mathrm{h}\overset{\mathrm{m}}{—}\mathrm{i}$ becomes an edge of the new independence graph which in turn makes $\mathrm{h}\rightarrow\mathrm{j}$ and $\mathrm{i}\rightarrow\mathrm{j}$ undirected in a minimal directed representation.

$\mathrm{h}\rightarrow\mathrm{j}$ and $\mathrm{i}\rightarrow\mathrm{j}$ become undirected in a minimal directed representation which is also a decomposable cover of itself. Therefore the removal of any further edges cannot give rise to superfluous edges.

3.5 Selection criterion

Neglecting dependences (i.e. removing edges) has positive consequences in terms of cost reduction, but possible negative consequences in terms of erroneous inference. These two conflicting interests should be reconciled through a selection criterion trading off reduction of complexity for reduced 'information content' of the edge removed. To maximize the cost reduction we obviously want to remove edges belonging to belief universes with large state spaces, whereas to minimize the possible error, the dependences neglected should be the ones representing the weakest mutual information.

3.5.1 Information divergence

When removing an edge $\{\alpha, \beta\} \in E$ from a graph $\mathcal{G} = (V, E)$ as described above, we need a quantitative measure of the information lost by assuming $\alpha \perp\!\!\!\perp \beta \mid \gamma$, where $\gamma = V \setminus \{\alpha, \beta\}$. Put differently, we need an expression for the divergence between p and q, where p is the probability function for V before the removal of α—β and q is the probability function after the removal. In terms of p, q is given as

$$q = \frac{\sum_\alpha p \sum_\beta p}{\sum_{\alpha,\beta} p} = \frac{p(\beta,\gamma)p(\alpha,\gamma)}{p(\gamma)} = p(\alpha \mid \gamma)p(\beta \mid \gamma)p(\gamma), \tag{3.9}$$

implying that $\alpha \perp\!\!\!\perp \beta \mid \gamma$ under q.

Several divergence statistics exist. Perhaps the most well-known of these is the Kullback-Leibler information divergence statistic, also known as cross entropy or entropy distance. There are several good reasons for choosing this statistic as our measure of divergence, as we shall see below. The Kullback-Leibler divergence between two probability functions f and g is defined as

$$C(f,g) = \begin{cases} E\log(f/g) = \sum f \cdot \log(f/g) & \text{if } f > 0 \text{ and } g > 0 \\ 0 & \text{if } f = 0 \text{ and } g = 0 \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{3.10}$$

and expresses the expected divergence between f and g, where the expectation is taken with respect to f. In the literature this measure is also frequently referred to as the directed divergence between f and g since, in general, $C(f,g) \neq C(g,f)$. Kullback (1959) refers to $C(f,g)+C(g,f)$ as the (undirected) information divergence. This measure, however, does not possess the as many properties as the undirected ones. When appropriate, the more precise notation $C(f,g;A)$ or $C(f_A, g_A)$ shall be used to indicate for which random variable (or vector) the divergence is taken.

In the special case where $f = p$ and $g = q$, with q given by (3.9), (3.10) is called the *conditional mutual information* between α and β given γ and is written

$$I(\alpha,\beta \mid \gamma) = C(p,q). \tag{3.11}$$

Properties of I are discussed in detail in Section 3.6.

The Kullback-Leibler divergence statistic has a number of important properties some of which shall be mentioned briefly in the following.

Theorem 3.8 *The Kullback-Leibler information divergence statistic satisfies*

(i) $C(f,g) \geq 0$,

(ii) $C(f_{A\cup B}, g_{A\cup B}) = C(f_A, g_A) + C(f_B, g_B)$ *if* $A \perp\!\!\!\perp B$ *w.r.t. to both* f *and* g.

Statement (i) follows directly from Jensen's inequality and statement (ii) may be verified by simple algebraic manipulations.

The following important relationship between the maximum absolute divergence and the Kullback-Leibler divergence has been described by e.g. Csiszár (1967), Kemperman (1967), and Kullback (1967).

Theorem 3.9 *Let V be any set of discrete random variables, and let f and g be probability functions defined on V. Then for any subset $A \subseteq V$*

$$|f_A - g_A| \leq \sqrt{\tfrac{1}{2}C(f_V, g_V)}. \tag{3.12}$$

Proofs may be found in the above references, or in Whittaker (1989).

3.5.2 Reduction of complexity

From the above discussion it appears that when deciding which edge to remove we should consider not only the mutual information represented by the edges and the immediate cost reduction (cf. (3.7)), but also the consequences of the removals in terms of rendering moral and fill edges superfluous (the removal of which further reduces the cost).

The total benefit in terms of cost reduction by removing an edge from a minimal decomposable cover, $\mathcal{C}$, is given by the cost difference between $\mathcal{C}$ and a minimal decomposable cover of the new moralized independence graph. Let the total cost reduction be

$$\Sigma(\alpha, \beta) = \sigma(\alpha, \beta) + \rho(\alpha, \beta), \tag{3.13}$$

where $\sigma(\alpha, \beta)$ is given by (3.7) and $\rho(\alpha, \beta)$ is the possible additional cost reduction obtained by rendering some (moral and fill) edges superfluous.

3.5.3 Combined measure

It might seem natural to choose an edge selection criterion based on a trade-off between the total cost reduction, $\Sigma(\alpha, \beta)$, obtained by removing α—β and the mutual information $I(\alpha, \beta \mid \cdot)$. However, an edge of a large belief universe might be rendered removable by first removing an edge yielding a relatively small cost reduction. Thus, given an amount by which the cost of a decomposable cover should be reduced, it is generally not possible to select an optimum sequence of edges to be removed using a selection scheme based on local considerations alone. To avoid a time-consuming search process one might wish to employ a heuristic selection criterion reflecting the trade-off between Σ and I. Such a heuristic might take the form

$$\lambda\Sigma - I, \tag{3.14}$$

where λ is a constant to be chosen such that $\lambda\Sigma$ is comparable with I. As $I \geq 0$ both terms in (3.14) are non-negative. Since we want I to be as small as possible and Σ as large as possible, the edge to be removed should be chosen such that the combined measure is maximized.

3.6 Propagation, accumulation, and attenuation of error

In this section we study the implications of edge removal in terms of propagation, accumulation, and attenuation of error. That is, we describe the circumstances under which an approximated belief potential does not influence the correctness of other belief potentials, prove that the overall error (information divergence) imposed by a series of edge removals equals the sum of the divergences associated with the individual edge removals, and prove that the influence of an approximation attenuates with increasing distance from the approximated belief universe.

3.6.1 The error may spread under arrival of posterior evidence

Now we investigate the effect of removing an edge α—β from a belief universe C_0 under subsequent absorption of evidence from a neighbouring universe C_i. In particular we show that if either C_i does not contain neither α nor β or at least one of α and β is contained in C_0 only, then the error induced by the removal of α—β does not spread to the neighbouring universes of C_0. In general, the error obviously spreads if this condition is not satisfied, since the joint potential for α and β is required for correct transferral of information from C_i via C_0 to any other neighbouring universe C_j when e.g. $\alpha \in C_i$ and $\beta \in C_j$.

Theorem 3.10 *Let ϕ be a joint belief for a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$, where $C_0 \in \mathcal{C}$ is the unique universe containing $\{\alpha, \beta\}$, adj$(C_0) = \{C_1, \dots, C_k\}$, and $S_j = C_0 \cap C_j$ for all $j = 1, \dots, k$. Let further ψ be a joint belief for Υ given by (3.6), $\phi^*_{S_i}$ any potential on S_i ($1 \le i \le k$) to be absorbed by C_0, and $\psi^*_{C_0}$, $\psi^*_{S_j}$, $\phi^*_{S_j}$, etc. the potentials for C_0 and S_j ($j = 1, \dots, k$) after the absorption. If either $\{\alpha, \beta\} \not\subseteq \bigcup_{j=1}^k S_j$ or $\{\alpha, \beta\} \cap S_i = \emptyset$, then $\phi^*_{S_j} \propto \psi^*_{S_j}$ for all $j = 1, \dots, k$.*

Proof: By (3.2) we get

$$\psi^*_{S_j} = \sum_{C_0 \setminus S_j} \psi^*_{C_0} = \sum_{C_0 \setminus S_j} \psi_{C_0} \frac{\phi^*_{S_i}}{\psi_{S_i}} = \sum_{S_i \setminus S_j} \psi_{S_i \cup S_j} \frac{\phi^*_{S_i}}{\psi_{S_i}}.$$

By Lemma 3.3 we have that $\psi_{S_i} \propto \phi_{S_i}$, and since $\{\alpha, \beta\} \not\subseteq (S_i \cup S_j)$ or equivalently $(S_i \cup S_j) \subseteq C_0 \setminus \{\alpha\}$ or $(S_i \cup S_j) \subseteq C_0 \setminus \{\beta\}$, it follows similarly from Lemma 3.3 that $\psi_{S_i \cup S_j} \propto \phi_{S_i \cup S_j}$. Hence

$$\psi^*_{S_j} \propto \sum_{S_i \setminus S_j} \phi_{S_i \cup S_j} \frac{\phi^*_{S_i}}{\phi_{S_i}} = \sum_{C_0 \setminus S_j} \phi^*_{C_0} = \phi^*_{S_j}. \qquad \square$$

Note that in cases where external evidence on $A \subseteq C_0$ is entered into C_0, where $\{\alpha, \beta\} \cap A \neq \emptyset$, A can be perceived as the separator S_i from which C_0 absorbs, and hence the error spreads to the neighbours of C_0 only if the requirement $\{\alpha, \beta\} \not\subseteq \bigcup_{j=1}^{k} S_j$ or $\{\alpha, \beta\} \cap S_i = \emptyset$ is violated.

3.6.2 The Kullback-Leibler divergence can be computed locally

In this section we show that the Kullback-Leibler divergence $C(p_V, q_V)$, due to the removal of an edge α—β from a belief universe C_0, is equivalent to the marginal divergence $C(p_{C_0}, q_{C_0})$. Thus a vast reduction of the computational complexity can be achieved.

Theorem 3.11 *Let p_V be a consistent probability function for a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ with corresponding graph $\mathcal{G} = (V, E)$. Let $C_0 \in \mathcal{C}$ be the unique universe containing $\{\alpha, \beta\}$ and let q_V be a probability function for Υ given by (3.6) with proportionality constant 1. Then*

$$C(p_V, q_V) = C(p_{C_0}, q_{C_0}) = I(\alpha, \beta \mid \gamma),$$

where $\gamma = C_0 \setminus \{\alpha, \beta\}$ is the minimal set separating α from β in $\mathcal{G}' = (V, E \setminus \{\{\alpha, \beta\}\})$.

Proof: First note that q is consistent since p is consistent (cf. Corollary 3.1). With that in mind we easily get

$$C(p_V, q_V) = \sum_V p_V \log \frac{p_V}{q_V} = \sum_V p_V \log \frac{\prod_{C \in \mathcal{C}} p_C / q_C}{\prod_{S \in \mathcal{S}} p_S / q_S} = \sum_V p_V \log \frac{p_{C_0}}{q_{C_0}}$$

$$= \sum_{C_0} p_{C_0} \log \frac{p_{C_0}}{q_{C_0}} = C(p_{C_0}, q_{C_0}) = I(\alpha, \beta \mid \gamma).$$

Now we prove that γ separates α from β in $\mathcal{G}'$ (i.e. the only path $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$, $2 \leq k \leq |V|$, for which $S = \{\alpha_1, \ldots, \alpha_k\} \cap \gamma = \emptyset$ is $\langle \alpha, \beta \rangle$). Assume there is a path $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$ between α and β such that $k > 2$ and $S \neq \emptyset$ (i.e. a path not including a vertex in γ). Since $\mathcal{G}$ is triangulated (i.e. all m-cycles for $m > 3$ have a chord), either $\{\beta, \alpha_2\} \in E$ or $\{\alpha, \alpha_{k-1}\} \in E$; otherwise there would be an m-cycle, $3 < m \leq k$, without a chord. Thus either $\alpha_2 \in \gamma$ or $\alpha_{k-1} \in \gamma$ violating the assumption $S \neq \emptyset$.

Clearly γ is minimal as $\operatorname{adj}(\alpha) \cap \operatorname{adj}(\beta) = \gamma$ in both $\mathcal{G}$ and $\mathcal{G}'$. □

If interest in the belief potential p_{C_0} vanishes and no evidence is going to be received and retransmitted by C_0 then all edges which are unique to C_0 can be removed without affecting the belief potentials of other universes. This is a very useful result for belief updating in dynamic Bayesian networks (Kjærulff 1992a).

3.6.3 The Kullback-Leibler divergence is additive

When approximating a complex model by removing several edges, the following additivity property of Kullback-Leibler divergence is very important. Assume that k edges are removed sequentially from a junction tree Υ_0 with probability function p_0, and let Υ_i and p_i be, respectively, the junction tree and the probability function after the removal of the i^{th} edge. Then the Kullback-Leibler divergence between p_0 and p_k equals the sum over all $i = 1, \dots, k$ of the divergence between p_{i-1} and p_i.

Theorem 3.12 *Let $\{\Upsilon_i\}_{i=1}^k$ be a collection of junction trees with belief universes $\{\mathcal{C}_i\}$, where $\mathcal{C}_i = \{C_i^1, \dots, C_i^{m_i}\}$. Let C_{i-1}^* be the unique universe in $\mathcal{C}_{i-1}$ containing $\{\alpha_i^1, \alpha_i^2\}$, and for $j = 1, 2$ let*

$$A_i^j = \begin{cases} C_{i-1}^* \setminus \{\alpha_i^j\} & \textit{if } \nexists C \in \mathcal{C}_{i-1} : C_{i-1}^* \setminus \{\alpha_i^j\} \subset C, \\ \emptyset & \textit{otherwise.} \end{cases}$$

Further, for $i = 1, \dots, k$ and $j = 1, \dots, m_i$ let the belief universes $\{\mathcal{C}_i\}$ and the probability functions $\{p_i\}$ of $\{\Upsilon_i\}$ be given as

$$\mathcal{C}_i = (\mathcal{C}_{i-1} \setminus \{C_{i-1}^*\}) \cup \{A_i^1, A_i^2\},$$

$$p_i(C_i^j) = \begin{cases} \dfrac{\sum_{\alpha_i} p_{i-1}(C_{i-1}^*) \sum_{\beta_i} p_{i-1}(C_{i-1}^*)}{\sum_{\alpha_i, \beta_i} p_{i-1}(C_{i-1}^*)} & \textit{if } C_{i-1}^j = C_{i-1}^*, \\ p_{i-1}(C_{i-1}^j) & \textit{otherwise.} \end{cases}$$

Then $C(p_0, p_k) = \sum_{i=1}^k C(p_{i-1}, p_i)$.

Proof: First we rewrite $C(p_0, p_k)$ slightly

$$\begin{aligned} C(p_0, p_k) &= \sum_V p_0 \log \frac{p_0}{p_k} \\ &= \sum_V p_0 \log \left[\frac{p_0}{p_1} \frac{p_1}{p_2} \cdots \frac{p_{k-1}}{p_k} \right] \\ &= \sum_{i=1}^k \sum_V p_0 \log \frac{p_{i-1}}{p_i}. \end{aligned}$$

Since p_{i-1} and p_i differ only in clique C_{i-1}^* we have

$$\sum_V p_0 \log \frac{p_{i-1}}{p_i} = \sum_{C_{i-1}^*} p_0 \log \frac{p_{i-1}}{p_i},$$

and further, since the clique marginals are 'backward compatible', i.e.

$$p_i(C_l^j) = p_{i-l}(C_l^j), \quad i = 1, \ldots, k, \quad l = 1, \ldots, i, \quad j = 1, \ldots, m_i,$$

we get

$$\sum_{C_{i-1}^*} p_0 \log \frac{p_{i-1}}{p_i} = \sum_{C_{i-1}^*} p_{i-1} \log \frac{p_{i-1}}{p_i} = C(p_{i-1}, p_i). \qquad \square$$

Among the divergence statistics described in the literature there are some which have the additivity property and some which do not. See e.g. the work of Read and Cressie (1988) for an overview of divergence statistics and their properties.

3.6.4 The error attenuates under increasing distance

Intuitively it seems reasonable that the effect of removing an edge from a belief universe, C, attenuates with increasing distance from C. Below we show this formally, but first we prove a lemma stating that the Kullback-Leibler divergence decreases under partitioning.

Lemma 3.5 *Consider a belief universe* C *with* p *and* q *being probability functions on* $Sp(C)$*, and let* $S \subset C$*. Then*

$$C(p_S, q_S) \leq C(p_C, q_C).$$

Proof: Let $||S|| = k$ and $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be disjoint subsets of $\mathrm{Sp}(C)$ such that $\bigcup_{i=1}^k \mathcal{S}_i = \mathrm{Sp}(C)$, $s_i = \bigvee_{x \in \mathcal{S}_i} x \in \mathrm{Sp}(S)$, $p(s_i) = \sum_{\mathcal{S}_i} p$, and $q(s_i) = \sum_{\mathcal{S}_i} q$ for all $i = 1, \ldots, k$. Consider the partition of C into S as a consecutive series of collapses of $\mathcal{S}_1, \ldots, \mathcal{S}_k$ starting by $\mathcal{S}_1$. The first partition obtained is thus $\{s_1, \mathrm{Sp}(C) \setminus \mathcal{S}_1\}$ representing a smaller space, say $\mathrm{Sp}(C')$. Obviously, $C(p_{C'}, q_{C'})$ equals $C(p_C, q_C)$ minus the divergence between p and q in subspace $\mathcal{S}_1$ plus the divergence between $p(s_1)$ and $q(s_1)$, i.e.

$$C(p_{C'}, q_{C'}) = C(p_C, q_C) - \sum_{\mathcal{S}_1} p \log \frac{p}{q} + p(s_1) \log \frac{p(s_1)}{q(s_1)}. \tag{3.15}$$

The second partition $\{s_1, s_2, \mathrm{Sp}(C) \setminus \mathcal{S}_1 \setminus \mathcal{S}_2\}$ in this process is obtained by collapsing subspace $\mathcal{S}_2$ into s_2, and the divergence between p and q after this collapse is calculated by substituting $C(p_C, q_C)$ by $C(p_{C'}, q_{C'})$ and subscript 1 by subscript 2 on the right-hand side of (3.15). Continuing this process we find

$$\begin{aligned} C(p_S, q_S) &= C(p_C, q_C) - \sum_{i=1}^k \left[\sum_{\mathcal{S}_i} p \log \frac{p}{q} - p(s_i) \log \frac{p(s_i)}{q(s_i)} \right] \\ &= C(p_C, q_C) - \sum_{i=1}^k \left[p(s_i) \sum_{\mathcal{S}_i} p(x \mid s_i) \log \frac{p(x \mid s_i)}{q(x \mid s_i)} \right], \end{aligned}$$

where the inner-most sum is equal to the Kullback-Leibler divergence between $f_i(x) = p(x \mid s_i)$ and $g_i(x) = q(x \mid s_i)$ in space $\mathcal{S}_i$ since $\sum_{\mathcal{S}_i} f_i = \sum_{\mathcal{S}_i} g_i = 1$. Thus

$$C(p_S, q_S) = C(p_C, q_C) - \sum_{i=1}^{k} p(s_i) C(f_i, g_i) \leq C(p_C, q_C)$$

since both $p(s_i)$ and $C(f_i, g_i)$ are non-negative. □

Based on that lemma we can now prove the following theorem.

Theorem 3.13 *Let $\Upsilon = (\mathcal{C}, \mathcal{S})$ be a consistent junction tree with probability function p, and let q be the consistent probability function for Υ obtained by substituting p_{C_0} with the probability function q_{C_0} and performing belief updating in Υ. Then for any path $\langle C_0, \ldots, C_k \rangle$ in Υ*

$$C(p_{C_0}, q_{C_0}) \geq C(p_{C_1}, q_{C_1}) \geq \cdots \geq C(p_{C_k}, q_{C_k}).$$

Proof: It suffices to consider the case $C(p_{C_1}, q_{C_1}) \leq C(p_{C_0}, q_{C_0})$. We have

$$C(p_{C_1}, q_{C_1}) = \sum_{C_1} p_{C_1} \log \frac{p_{C_1}}{q_{C_1}},$$

but since C_0 is the only universe which received evidence, then $q_{C_1} = p_{C_1} \dfrac{q_{S_1}}{p_{S_1}}$ and hence

$$C(p_{C_1}, q_{C_1}) = \sum_{C_1} p_{C_1} \log \frac{p_{S_1}}{q_{S_1}} = \sum_{S_1} p_{S_1} \log \frac{p_{S_1}}{q_{S_1}} = C(p_{S_1}, q_{S_1}).$$

By Lemma 3.5 we have that $C(p_{S_1}, q_{S_1}) \leq C(p_{C_0}, q_{C_0})$ which completes the proof. □

The principle of attenuation of evidence in a junction tree was pointed out by Hilden (1988).

3.7 Some examples

Since, from a theoretical point of view, not much can be said about the practical applicability of edge removal, we employ the method on a couple of real-world networks to gain some additional insight into the applicability of the method. Further, we compare the method with the annihilation technique (i.e. removal of probability mass) by Jensen and Andersen (1990).

3.7.1 Water treatment model

The first network is a probabilistic model of the biological processes of a water treatment plant (Jensen, Kjærulff, Olesen and Pedersen 1989). The model includes eight process parameters and is dynamic in the sense that, at any time instant, the model comprises a series of consecutive slices of time, where any two consecutive slices represent a time interval of 15 minutes. Each time slice t of the model describes the state of the dynamic process at time t and contains therefore eight nodes representing the random variables associated with the eight process parameters. Due to the relatively large time constants of the biological processes there are no within-slice edges, and thus, all probabilistic relationships between the process parameters appear as inter-slice edges (see Figure 3.10).

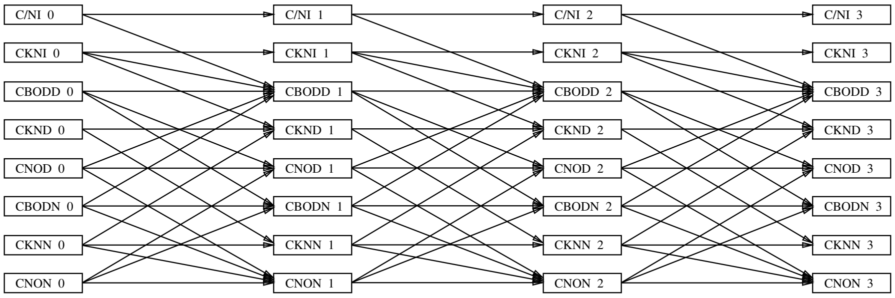

Figure 3.10: First four time slices of a dynamic Bayesian network model for the biological processes of a water treatment plant.

Due to the number of inter-slice edges and the ways they have been arranged, the complexity of the model become extremely high. After compilation in dHUGIN (Kjærulff 1992a) of a model including the first four time slices, the tables of the associated junction tree take up more than 43 megabytes of storage. The fact that the model has been considerably simplified to fit the capacity of workstation computers, indicates that a full-scale model probably would take up hundreds of megabytes of storage. This amount of storage is seldomly provided by today's workstation computers and is definitely not provided by today's low-cost personal computers which might be the target hardware for such a system. Therefore, further simplifications seem indispensable.

We have conducted two simplifications of the model (comprising the first four time slices) using the edge-removal technique. The first simplification (resulting in Model A) involves the removal of 43 edges selected according to a trade-off of the form (3.14), where we take the logarithm of the cost reduction, and the second simplification (resulting in Model C) involve 67 edges in addition (giving a total of 110 edges). The simplifications provided cost reductions by 90.2% and 96.5%, respectively. Approximations providing the same amounts of reductions using the method of Jensen and Andersen (1990) were also conducted for the sake of comparison. The resulting models are called Model B and Model D. The four models were evaluated by running them on three test cases discussed by Jensen et al. (1989) and comparing the resulting marginal distributions for the 32 variables with the corresponding correct distributions.

By recording the square error and the maximum absolute deviation for each variable, we can compute the average square error and the minimum, average and maximum of the maximum absolute deviation among the variables, and thereby get a summary of the merits of the individual models. Table 3.1 shows these figures for each model and each case, where Model A and Model B should be compared, and Model C and Model D should be compared.

Water treatment: all variables					
Model	Case	Min max dist	Ave max dist	Max max dist	Ave sq err
A	1	0.000000	0.000000	0.000000	0.000000
	2	0.000000	0.000000	0.000000	0.000000
	3	0.000000	0.000000	0.000000	0.000000
B	1	0.000000	0.000429	0.001371	0.000001
	2	0.000000	0.001889	0.007110	0.000014
	3	0.000000	0.359477	0.949389	0.469538
C	1	0.000000	0.000002	0.000061	0.000000
	2	0.000000	0.000000	0.000003	0.000000
	3	0.000000	0.006908	0.030908	0.000253
D	1	0.000000	0.003716	0.013097	0.000057
	2	0.000000	0.012682	0.040817	0.000595
	3	0.000000	0.360060	0.950088	0.474214

Table 3.1: Comparison of the edge-removal and the removal-of-probability-mass techniques for the water treatment model.

As indicated in Table 3.1 none of the posterior probabilities for the 32 process parameters differed more than $5 \cdot 10^{-7}$ from the 'correct' one when Model A was applied. When, however, the corresponding removal-of-probability-mass approximation was applied (Model B), Case 3 gave rise to extremely unreliable results. For Case 1 and Case 2, Model B provided quite acceptable result results. By the 96.5% reduction of cost, marginally significant errors can be registered for Case 3 under Model A. However, since the variables have at most 4 states this error can hardly be considered significant. The same is true for Case 1 and Case 2 under Model D, but for Case 3, Model D is useless.

The results provided by edge removal are not surprising since for the approximation producing Model A the (accumulated) Kullback-Leibler deviation is only 2.2×10^{-5} and for the approximation producing Model C it is 1.0×10^{-4}. In both cases the maximum absolute deviation (cf. (3.12)) is less than 0.01 (under lack of posterior evidence). Note that we observe a maximum absolute deviation of 0.03 under Model C (Case 3).

3.7.2 Winter wheat model

The second example is a preliminary model for the influence of temperature, precipitation, solar energy, and spraying against mildew on winter wheat in terms of leaf area index, severity of attack of mildew, production of dry matter, and expected yield. The model consists of four consecutive submodels (time slices) each representing a distinct phase of development (see Figure 3.11).

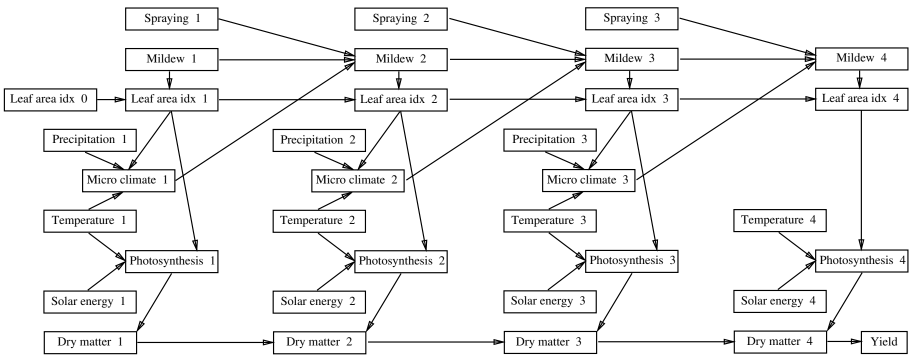

Figure 3.11: A Bayesian network model of growing of winter wheat.

We have conducted a single simplification of the model using the edge-removal technique. The resulting model, called Model A, was obtained by the removal of 26 edges. The simplification provided a cost reduction of 49.2%. An approximation providing the same amount of reduction using the removal-of-probability-mass technique was also conducted (this resulting model is referred to as Model B). These two models were evaluated by running them on three test cases. The resulting distributions for the four Mildew variables and the Yield variable were compared with the corresponding correct distributions. The reason for only considering these five variables is due to the fact that mildew is the only controllable parameter (via spraying) influencing the yield, and the reason for considering the yield should be obvious. The remaining variables determine the degree of attack of mildew and amount of dry matter produced (the accumulation of which determines the yield).

If we consider the approximated distributions of the Mildew variables, then we observe mostly insignificantly small deviations from the 'correct' distributions for both approximations. This pattern is broken in only one occasion, namely when running Model A with Case 1. Then the maximum absolute deviation for the distribution of variable Mildew_4 is about 0.1 (relative deviation 17.6%); see Table 3.2. The graphical display of the correct and approximated distributions in Figure 3.12 shows that even in this case the error is probably not serious since the shapes of the approximated and the correct distributions are roughly identical.

Winter wheat: Mildew						
Model	Case	Min max dist	Ave max dist	Max max dist	%	Ave sq err
A	1	0.006350	0.040844	0.103721	17.6	0.004556
	2	0.001520	0.006445	0.011031	13.7	0.000139
	3	0.000406	0.001486	0.002565	0.3	0.000005
B	1	0.006108	0.009248	0.012461	2.1	0.000151
	2	0.008700	0.015650	0.021074	3.4	0.000410
	3	0.000251	0.000399	0.000546	0.1	0.000000

Table 3.2: Comparison of the edge-removal and the removal-of-probability-mass techniques for the Mildew variables of the winter-wheat model.

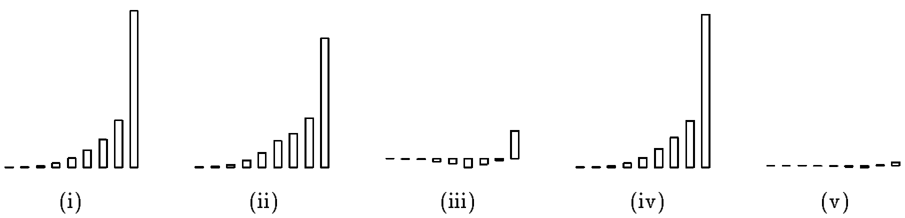

Figure 3.12: (i) The 'correct' distribution of the Mildew_4 variable under Case 1. (ii) The distribution under Model A. (iii) The absolute difference between (i) and (ii). (iv) The distribution under Model B. (v) The absolute difference between (i) and (iv).

A maximum absolute deviation of 0.012 for the Yield variable when applying Model A and Case 1 may seem to be insignificant (see Table 3.3). However, since the Yield variable has 86 states and since the distribution of the variable is fairly flat, the error is after all clearly significant. This is also indicated by the relative deviation of 41.8%, and it becomes very clear indeed when considering the graphical displays in Figure 3.13 of the correct and the approximate distributions of the variable. The approximate distribution (ii) of Figure 3.13 provided by Model A appears to be clearly bimodal and differs significantly from the correct one. The approximate distribution obtained by Model B appears to be a quite close approximation of the correct one. Neither of the approximation techniques gave rise to serious deviations between the approximated distributions and the correct ones for the Yield variable when run with Case 2 and Case 3.

Winter wheat: Yield						
Model	Case	Min dist	Ave dist	Max dist	%	Sq err
A	1	0.000000	0.003537	0.012439	41.8	0.002616
	2	0.000000	0.001601	0.006361	23.0	0.000543
	3	0.000000	0.001345	0.004967	17.0	0.000377
B	1	0.000000	0.000566	0.001885	9.4	0.000053
	2	0.000000	0.000638	0.002676	5.6	0.000073
	3	0.000000	0.000404	0.002239	100	0.000038

Table 3.3: Comparison of the edge-removal and the removal-of-probability-mass techniques for the Yield variable of the winter-wheat model.

The rather discouraging results by the edge-removal technique in this example is not surprising since the accumulated Kullback-Leibler deviation for the 26 edges removed is about 0.2.

3.8 Concluding remarks

We have presented a scheme for approximating Bayesian networks through edge removals. A decomposable hypergraph cover (triangulated graph), of the network to be approximated, is first found and a corresponding junction tree is constructed along with an associated potential representation of the probability function. The removal of an edge amounts to enforcing additional conditional independence assumptions, expressed through modification of the potential representation, the junction tree, and the independence graph. We showed that although the initial independence graph is directed and acyclic, removal of edges from a corresponding triangulated graph may result in an independence graph being a chain graph. Further, using the modified independence graph, edges rendered superfluous by an edge removal are automatically identified. This feature is important, since the computational bottleneck in determining which edge to remove is calculation of the mutual information represented by candidate edges.

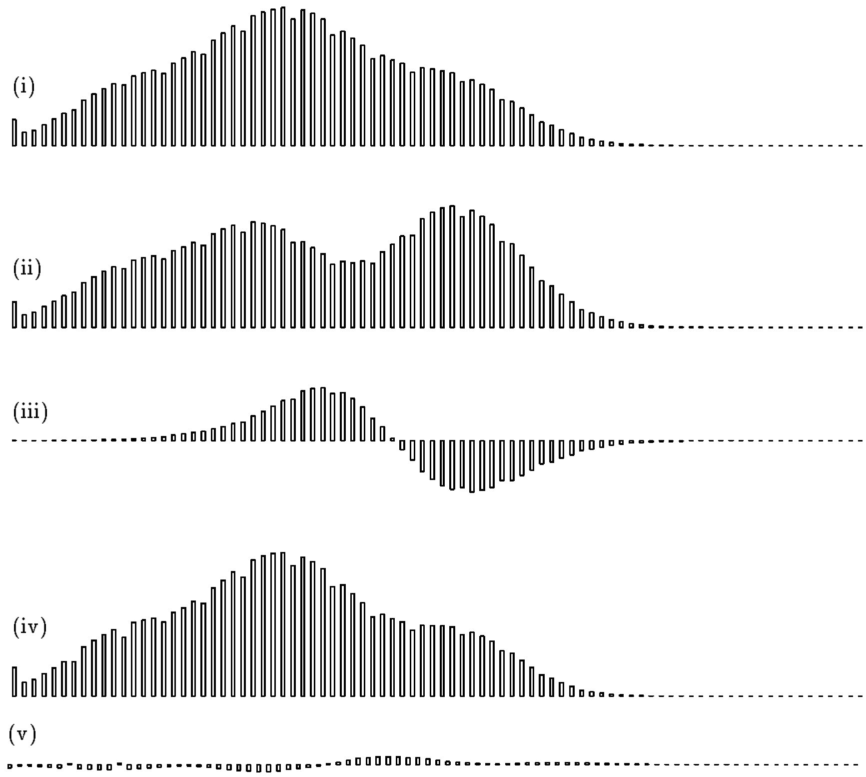

Figure 3.13: (i) The 'correct' distribution of the Yield variable under Case 1. (ii) The distribution under Model A. (iii) The absolute difference between (i) and (ii). (iv) The distribution under Model B. (v) The absolute difference between (i) and (iv).

The Kullback-Leibler information divergence statistic was employed as the measure of mutual information. This measure has a number of important properties w.r.t. edge removal. We showed that the error induced by the removal of an edge affects only the belief universe (clique) from which the edge is removed, provided no evidence enters the universe (after the edge has been removed) from neither neighbouring universes nor the 'outside world'. Even in the presence of such posterior evidence, the error need not spread to neighbouring universes; however, it often does. As a consequence of the error being local, the Kullback-Leibler divergence can be computed using the (normalized) belief potential attached to the relevant universe instead of the (normalized) joint system belief. We also showed that the total divergence by a series of edge removals equals the sum of the divergences associated with the individual edge removals.

Since the key indicator associated with an approximation is most often the maximum absolute error, inequality (3.12) is essential, as it expresses the relationship between the Kullback-Leibler divergence and the maximum absolute error. However, the inequality only deals with the case where no posterior evidence has been received. Under arrival of posterior evidence, the inequality can only be used as a rough guideline (cf. the discussion at the end of Section 3.7.1).

An important feature of all approximation schemes performing local approximations (i.e. approximating single belief universes) is given by the fact that the effect of an approximation attenuates with increasing distance from the belief universe subjected to the approximation. This feature is especially important in connection with dynamic Bayesian networks (Kjærulff 1992a, Kjærulff 1993b). An additional property of the presented scheme, is the property of errors remaining localized in absence of posterior evidence and, under certain conditions, even in presence of posterior evidence. That is, the belief potentials of the separators are unaltered by the removal of an edge. The approximation method by Jensen and Andersen (1990) does not have this property.

As was clearly demonstrated by the example in Section 3.7.1, the edge removal approach may in some cases be significantly better than the annihilation approach of Jensen and Andersen (1990). In other cases, however, a comparison between the two may turn out to the disadvantage of the edge removal approach (cf. Section 3.7.2). Intuitively, this seems quite reasonable, since a model including a lot of edges with a very low information content will be almost equivalent to a model which lacks these edges, but it might be quite different from a model obtained by uniformly removing a corresponding amount of probability mass over the belief universes. On the other hand, the edge removal approach is unsuited in cases where there are no edges with low information content. Thus, to approximate a given network using these two methods, approximation by edge removal should be tried first and when all edges with low information content have been removed, the annihilation technique should take over.

Among directions for future research, the following two points are important. Assessment of a good upper bound on the error upon arrival of posterior evidence is crucial. Such a bound is not provided in the present paper, but it is important to address this problem. As mentioned above, the network to be approximated is first transformed to a corresponding junction tree which may increase the complexity of representation quite seriously. Thus, another area of interest for future research is selection of edges to be removed based solely on the original (conditional) probabilities of a Bayesian network. That is, edges are removed before transformation to a junction tree representation.

Acknowledgements

I am indebted to Steffen L. Lauritzen whose comments and support during the preparation of this paper were indispensable. I also wish to express my gratitude to the ODIN group for providing a very stimulating environment. The research has been funded partly by the Danish Research Councils through the PIFT programme.

Part III

Dynamic Bayesian Networks

Chapter 4

A computational scheme for dynamic Bayesian networks

We present a computational scheme for reasoning about dynamic systems using Bayesian networks. The scheme is based on message passing in junction trees, and may be viewed as a generalization of the inference methods of classical time-series analysis in the sense that it allows description of non-linear, multivariate dynamic systems with complex conditional independence structures. Further, the scheme provides a method for efficient backward smoothing, and an efficient Monte-Carlo algorithm for forecasting is presented.

The scheme has been implemented as an extension of the HUGIN system and is referred to as the dHUGIN system. Appendix B contains a reference manual to the set of functions comprising the Application Program Interface to the dHUGIN system. Appendix C contains a user's guide to the `dhugin` program (implemented entirely on top of the dHUGIN API) which provides a menu-driven user interface to the dHUGIN system.

The contents of this chapter is identical to Kjærulff (1993b).

4.1 Introduction

The application of probabilistic graphical models (Bayesian networks, belief networks, influence diagrams, etc.) for modeling domains with inherent uncertainties has become widespread. A common trait of the domains, where such applications turn out most successfully, is their static nature. That is, each observable quantity is observed once and for all, and confidence in the observations remaining true is not questioned. However, domains involving repeated observations of a collection of random quantities arise in many fields of science (e.g. medical, economic, biological). For such domains a static model is not very useful: the estimation of probability distributions of domain variables based on appropriate prior knowledge and observation of other domain variables is reliable only for a limited period of time, and further, upon arrival of new observations, both these and the old observations must be taken into account in the reasoning process. Thus, to cope with such dynamic systems using Bayesian networks we need to interconnect multiple instances of static networks. Obviously, as time evolves, new 'slices' must be added to the model and old ones cut off. This introduces the notion dynamic Bayesian networks (DBNs).

In general, a dynamic model may be defined as a sequence of submodels each representing the state of a dynamic system at a particular point or interval in time; henceforth, such a time instance will be referred to as a *time slice*. Hence, a DBN consists of a series of, most often structurally identical, subnetworks interconnected by temporal relations. To make estimates of variables of a dynamic system in a way that makes full use of the information about past observations of the system, requires a compact representation of this information. The creation of this representation is part of the process of reducing the dynamic model. This reduction process includes elimination of parts of the model representing past time slices, and should have no effect on future estimates, that is, the information conveyed by the eliminated part of the model should be completely represented in the remaining part. The complementary process of expanding the model must be carried out whenever new time slices have to be included in the model.

In classical time-series analysis (see e.g. Box and Jenkins (1976) or West and Harrison (1989)) the emphasis is on model assessment, i.e. estimation of model parameters given a time series of observations of some stochastic process. The model thereby selected is then used for making predictions about future behaviour of the time series. Although the classical time-series analysis techniques have been quite successful, their ability to cope with such important issues as complex independence structures and non-linear relationships of have appeared to be rather modest. By formulating the analysis in terms of DBNs both of these limitations vanish. Attempts to integrate methods of classical time-series analysis with network representation and inference techniques have been presented by Dagum, Galper and Horvitz (1992). This paper, however, does not address the issue of model assessment, but merely problems related to making inferences (including prediction (forecasting) and backward smoothing, in classical time-series analysis terms). That is, the dynamic model is assumed to be given.

Among research activities applying DBNs, as defined above, are a model for glucose prediction and insulin dose adjustment by Andreassen, Hovorka, Benn, Olesen and Carson (1991), an approach to building planning and control systems by Dean, Basye and Lejter (1990), a model for making judgements concerning persistence of propositions by Dean and Kanazawa (1989), and a model for sensor validation by Nicholson and Brady (1992). However, none of these research activities have dealt with the issues of reasoning in DBNs.

In Section 4.2 we briefly review some relevant graph theoretic concepts as well as some fundamental characteristics of Bayesian networks; furthermore, a precise definition of DBNs and some terminology regarding DBNs are presented. The processes of reducing and expanding DBNs are described in detail in Section 4.3 as well as the processes of backward smoothing and forecasting. Section 4.6 discusses pros and cons of the presented scheme and mentions some issues of possible future research.

4.2 Terminology

4.2.1 Graphs and hypergraphs

A (finite) graph $\mathcal{G} = (V, E)$ consists of a finite set of *vertices* V and a set of *edges* $E \subseteq V \times V$ of ordered pairs of distinct vertices. If $(\alpha, \beta) \in E$ and $(\beta, \alpha) \notin E$ the edge from α to β is *directed*. If both $(\alpha, \beta) \in E$ and $(\beta, \alpha) \in E$ the edge between α and β is *undirected* in which case we shall use the shorthand notation $\{\alpha, \beta\} \in E$. A graph is said to be directed if all edges are directed, and undirected if all edges are undirected. Undirectedness shall be understood unless directedness is explicitly assumed.

A (finite) *hypergraph* $\mathcal{H} = (V, \mathcal{C})$ consists of a finite set of vertices V and a set of edges $\mathcal{C}$, where $C \subseteq V$ and $|C| \geq 2$ for each $C \in \mathcal{C}$. A hypergraph is a graph if and only if each edge has cardinality two. $\mathcal{C}$ is called a *hypergraph cover* of a graph $\mathcal{G} = (V, E)$ if there is a $C \in \mathcal{C}$ for each $e \in E$ such that $e \in C$. The graph $G(\mathcal{C})$ of a hypergraph has vertex set $\bigcup_{C \in \mathcal{C}} C$ and edge set $\{\{\alpha, \beta\} \mid \exists C \in \mathcal{C} : \alpha, \beta \in C\}$.

Let $(\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_k = (V_k, E_k))$ be a total ordering of any k graphs such that $V_i \cap V_{i+1} \neq \emptyset$ for all $i = 1, \ldots, k-1$. Then the *compound graph* of $\mathcal{G}_1, \ldots, \mathcal{G}_k$ is given by

$$\bigcup_{i=1}^{k} \mathcal{G}_i = \left(\bigcup_{i=1}^{k} V_i, \bigcup_{i=1}^{k} E_i \right).$$

If there is an edge between α and β, denoted by $\alpha \sim \beta$, then α and β are *adjacent* or *neighbours*. The set of neighbours of α is denoted $\mathrm{adj}(\alpha)$, and the set of edges between α and $\mathrm{adj}(\alpha)$ are called the edges *incident* to α. If there is a directed edge from α to β, then α is called a *parent* of β and β a *child* of α. The sets of parents and children of α are denoted $\mathrm{pa}(\alpha)$ and $\mathrm{ch}(\alpha)$, respectively. For a directed graph $\mathcal{G} = (V, E)$, $\mathcal{G}^m$ denotes its *moral graph* obtained by adding edges between pairs of vertices with common children and dropping the directions of the edges.

A *path* $\pi = \langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$ from α to β of length $k - 1$ is an ordered sequence of distinct vertices such that $\alpha_i \sim \alpha_{i+1}$ for all $i = 1, \ldots, k - 1$. If $\alpha_i \in \text{pa}(\alpha_{i+1})$ for all $i = 1, \ldots, k - 1$, then π is a *directed path*. In a directed graph, $\mathcal{G} = (V, E)$, the set $\text{de}(\alpha)$ for which there is a directed path from α to β for all $\beta \in \text{de}(\alpha)$ is called the *descendants* of α. The set $\text{an}(\alpha)$ for which there is a directed path from β to α in $\mathcal{G}$ for all $\beta \in \text{an}(\alpha)$ is called the *ancestors* of α. A subset $A \subseteq V$ is an *ancestral set* if $\text{an}(\alpha) \subseteq A$ for all $\alpha \in A$. By $\text{An}(A)$ we denote the smallest ancestral set containing A.

An m-cycle is a path $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$ of length m with the exception that $\alpha = \beta$. A *chord* in an m-cycle is an edge between α_i and α_j such that $1 < |i - j| < m - 1$ (i.e. it connects the two non-consecutive vertices α_i and α_j of the cycle). A directed graph with no directed cycles is called a *directed, acyclic graph* or DAG for short.

For $A, B, C \subseteq V$, C is said to *separate A from B* if for all paths $\langle \alpha = \alpha_1, \ldots, \alpha_k = \beta \rangle$, where $\alpha \in A$ and $\beta \in B$, $\{\alpha_1, \ldots, \alpha_k\} \cap C \neq \emptyset$. A graph $\mathcal{G}$ is *connected* if there is a path between each pair of vertices of $\mathcal{G}$. Unless otherwise stated, connectivity shall be assumed throughout this paper.

A subset $A \subseteq V$ induces a *subgraph* $\mathcal{G}_A = (A, E_A)$ of $\mathcal{G} = (V, E)$, where $E_A = E \cap (A \times A)$. A graph is *complete* if all vertices are pairwise adjacent. A subset $A \subseteq V$ is complete if it induces a complete subgraph, and if A is maximal (i.e. there is no complete subset $B \subseteq V$ such that $A \subset B$), then it is called a *clique*.

A *(proper) decomposition* of an undirected graph $\mathcal{G} = (V, E)$ is a triple (A, B, C) of non-empty and disjoint subsets of V such that $V = A \cup B \cup C$, C separates A from B, and C is a complete subset of V (i.e. each pair of vertices in C are neighbours). A decomposition (A, B, C) *decomposes* $\mathcal{G}$ into subgraphs $\mathcal{G}_{A \cup C}$ and $\mathcal{G}_{B \cup C}$ (i.e. subgraphs induced by $A \cup C$ and $B \cup C$, respectively). $\mathcal{G}$ is *decomposable* if and only if (A, B, C) decomposes $\mathcal{G}$ and both $\mathcal{G}_{A \cup C}$ and $\mathcal{G}_{B \cup C}$ are decomposable. $\mathcal{G}$ is *triangulated* (or *chordal*) if each cycle of length greater than 3 has a chord. Note that the class of triangulated graphs is equivalent to the class of decomposable graphs. For a proof see e.g. Lauritzen et al. (1984).

When a vertex $\alpha \in V$ and the edges incident to α are removed from $\mathcal{G} = (V, E)$, α is said to be *deleted*, but when $\text{adj}(\alpha)$ are made a complete subset by adding the necessary edges (if any) to the graph before α and the edges incident to α are removed, then α is said to be *eliminated*. Note that connectivity of a graph is invariant under elimination, but not necessarily under deletion. The set, say T, of edges added by eliminating all vertices in V in any order is called a *triangulation* of $\mathcal{G}$ as $(V, E \cup T)$ is triangulated. If there is no proper subset $T' \subset T$ such that T' is a triangulation of $\mathcal{G}$, then T is said to be a *minimal triangulation*. The edges of T are called *fill edges* (or *fill-ins*). An *elimination ordering* is a bijection $\# : V \leftrightarrow \{1, \ldots, |V|\}$. $\mathcal{G}_{\#}$ is an *ordered graph*. The triangulation $T(\mathcal{G}_{\#})$ is the set of edges produced by eliminating the vertices of $\mathcal{G}$ in order $\#$. An elimination ordering $\#$ is *perfect* if $T(\mathcal{G}_{\#}) = \emptyset$. The triangulated graph $(V, E \cup T(\mathcal{G}_{\#}))$ shall be denoted $\mathcal{G}^{t\#}$; for convenience, the same notation shall be used if $\mathcal{G}$ is a DAG (i.e. moralization is understood).

A hypergraph $\mathcal{H} = (V, \mathcal{C})$ is decomposable if and only if $G(\mathcal{C})$ is decomposable. If $\mathcal{C}$ is a cover of some graph $\mathcal{G} = (V, E)$ and $\mathcal{H}$ is decomposable, then $\mathcal{C}$ is called a *decomposable cover* of $\mathcal{G}$. $\mathcal{C}$ is a *minimal decomposable cover* of $\mathcal{G}$ if there is a minimal triangulation, T, of $\mathcal{G}$ such that $\mathcal{C}$ is the clique set of $\mathcal{G} = (V, E \cup T)$.

4.2.2 Bayesian networks and Markov properties

A Bayesian network, as used in this paper, is built on a DAG, $\mathcal{G} = (V, E)$, where each vertex $\alpha \in V$ corresponds to a discrete random variable X_α with finite state space $\mathcal{X}_\alpha$. For $A \subseteq V$, X_A denotes the vector of variables indexed by A. Similarly, x_A denotes an element of the joint state space $\mathcal{X}_A = \times_{\alpha \in A} \mathcal{X}_\alpha$. Each random variable X_α of a Bayesian network is described in terms of a conditional probability distribution $p(x_\alpha \mid x_{pa(\alpha)})$ defined on $\mathcal{X}_\alpha$, where $p(x_\alpha \mid x_{pa(\alpha)})$ reduces to an unconditional distribution if $pa(\alpha) = \emptyset$. In $\mathcal{G}$, the conditioning variables of X_α are represented by $pa(\alpha)$. The joint probability, $p = p_V$, defined on $\mathcal{X}_V$ *factorizes (recursively)* according to $\mathcal{G}$, since

$$p(x) = \prod_{\alpha \in V} p(x_\alpha \mid x_{pa(\alpha)}). \tag{4.1}$$

$\mathcal{G}$ is called the *independence graph* of p, since it captures all (conditional) independence properties of p. That is, statements of conditional independence between pairs of sets $X_A, X_B \subseteq X_V$ of variables given a third set X_C. We shall use the notation $A \perp\!\!\!\perp B \mid C$ for any triple (A, B, C) of disjoint subsets of V for which X_A and X_B are conditionally independent given X_C w.r.t. p.

All conditional independence statements captured by $\mathcal{G}$ can be found using the d-separation criterion of Pearl (1988) or the equivalent criterion of Lauritzen et al. (1990) expressed by the following theorem.

Theorem 4.1 (Lauritzen et al. (1990)) *Let p factorize according to a DAG, $\mathcal{G} = (V, E)$. Then $A \perp\!\!\!\perp B \mid C$ w.r.t. p for any subsets $A, B, C \subseteq V$ whenever C separates A from B in $(\mathcal{G}_{An(A \cup B \cup C)})^m$.*

See Lauritzen et al. (1990) for a proof. The property described in Theorem 4.1

is referred to as the *global Markov property*. We also say that p is Markov w.r.t. $\mathcal{G}$ when p factorizes according to $\mathcal{G}$.

If a probability function p is Markov w.r.t. a DAG, $\mathcal{G} = (V, E)$, then there exists a *potential representation*

$$p(x) = z^{-1}\phi(x) = z^{-1} \prod_{C \in \mathcal{C}} \phi_C(x_C), \tag{4.2}$$

of p, where ϕ_C is a non-negative function for each C in a set $\mathcal{C}$ of complete subsets of a V (Lauritzen and Spiegelhalter 1988). The ϕ_C's are called *belief potentials*, ϕ a *joint belief*, and z a *normalization constant*. In particular, the product (4.1) is a potential representation with normalization constant 1.

If $\mathcal{C}$ is the set of cliques of the triangulated graph $\mathcal{G}^{t\#}$, then ϕ may be expressed as the joint belief of a *junction tree* $\Upsilon = (\mathcal{C}, \mathcal{S})$ having vertex set $\mathcal{C}$ and edge set $\mathcal{S} \subseteq \mathcal{C} \otimes \mathcal{C}$ called *separators* (Jensen 1988, Jensen, Lauritzen and Olesen 1990), where the symbol $\otimes$ denotes the binary operator producing the set of all ordered pairs of distinct elements of its arguments. A vertex of a junction tree is called a *belief universe*. For each path $\langle C = C_1, \dots, C_k = D\rangle$ in Υ, $C \cap D \subset C_i \cap C_j$ for all $1 \leq i \neq j \leq k$. Thus, calculation of marginal distributions can be done in Υ by a local message-passing scheme described in terms of a two-stage process involving *collection* and *distribution* of separator potentials (i.e. belief potentials on separators) between all pairs of neighbours in the tree (Jensen, Lauritzen and Olesen 1990). These two operations performed in sequence are jointly referred to as *propagation* or *belief updating*. The basic operation in passing a message from a belief universe $C \in \mathcal{C}$ to one of its neighbours $D \in \mathcal{C}$, is referred to as *(belief) absorption*. Let S be the separator associated with C and D. Then D is said to *absorb* from C if the belief potentials ϕ_D and ϕ_S are changed to ϕ^*_D and ϕ^*_S given by

$$\phi^*_S = \sum_{C \setminus D} \phi_C,$$

$$\phi^*_D = \phi_D * \frac{\phi^*_S}{\phi_S}.$$

A junction tree is said to be *consistent* if

$$\sum_{C \setminus D} \phi_C \propto \sum_{D \setminus C} \phi_D$$

for each $C, D \in \mathcal{C}$. Two junction trees $\Upsilon_1 = (\mathcal{C}_1, \mathcal{S}_1)$ and $\Upsilon_2 = (\mathcal{C}_2, \mathcal{S}_2)$ with non-empty and complete intersection $S = C_1 \cap C_2$, where $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$, are said to be *mutually consistent* if both Υ_1 and Υ_2 are consistent and $\phi_{C_1}(x_S) \propto \phi_{C_2}(x_S)$ for all $x_S \in \mathcal{X}_S$. If both Υ_1 and Υ_2 are consistent, but they fail to be mutually consistent, then they are said to be *mutually inconsistent*.

4.2.3 Dynamic Bayesian networks

A DBN represents a finite (though possibly varying) number, say n, of time slices. Let $\mathcal{G} = (V, E)$ be the DAG describing the structure of the dynamic model at a particular point in time. Note that $\mathcal{G}$ is not necessarily an independence graph of the model when the initial time slice (i.e. time slice 0) is not represented explicitly in the model. If t' is the first time slice of the network, then V consists of disjoint subsets, $V(t'), \dots, V(t'+n-1)$. That is,

$$V = V(t', n) = \bigcup_{t=t'}^{t'+n-1} V(t).$$

The time slices of a DBN are assumed to be chosen such that the DBN obeys the Markov property: the future is conditionally independent of the past given the present. (The notion of 'Markov property' should not be confused with the previously defined notion of 'global Markov property'.) Formally this may be written as

$$X_{V(0)}, \dots, X_{V(t-1)} \perp\!\!\!\perp X_{V(t+1)}, \dots, X_{V(t+k)} \,|\, X_{V(t)}$$

for all $t > 0$ and $k > 0$.

The set of directed edges

$$E^{\mathrm{tmp}}(t) = \{(\alpha, \beta) \in E \mid \alpha \in V(t-1), \beta \in V(t)\}, \quad t' \le t \le t' + n - 1,$$

is called the *temporal edges* (or *temporal relations*) of time slice t and define how the distributions of the variables of time slice t are given conditionally on the distributions of the variables of time slice $t-1$. Thus, temporal edges are those between vertices of adjacent time slices in the DAG of a DBN; see the example in Figure 4.1.

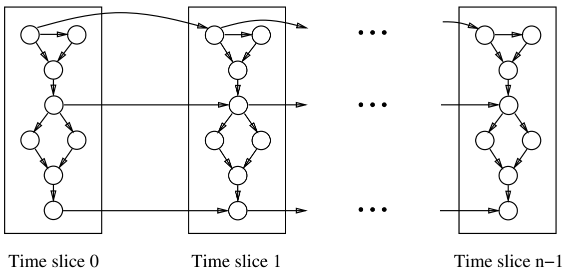

Figure 4.1: Independence graph of a sample DBN.

The subset $\mathrm{int}(t) \subseteq V(t)$ is called the *interface* of time slice t and is defined as

$$\begin{aligned}\mathrm{int}(t) \quad = \quad & \{\beta \in V(t) \mid (\alpha, \beta) \in E^{\mathrm{tmp}}(t) \text{ or} \\ & \qquad \exists \gamma \in \mathrm{ch}(\beta) : (\alpha, \gamma) \in E^{\mathrm{tmp}}(t), \;\; \alpha \in V(t-1)\}.\end{aligned}$$

The moral graph corresponding to the independence graph of the sample DBN in Figure 4.1 appears in Figure 4.2, where the interfaces are indicated by filled circles. Note that $\mathrm{int}(0) = \emptyset$.

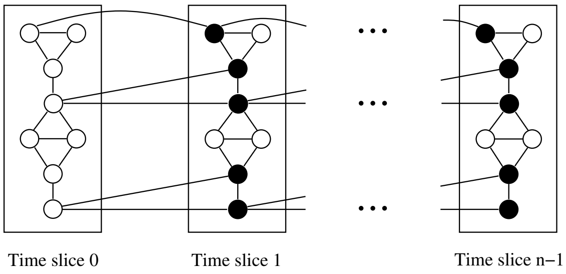

Figure 4.2: Moral graph corresponding to the independence graph of Figure 4.1, where the interfaces are indicated by filled circles.

The set E of edges of $\mathcal{G}$ can now be described as

$$E = E(t', n) = E(t') \cup \bigcup_{t=t'}^{t'+n-1} E^*(t),$$

where

$$E(t) \subseteq V(t) \otimes V(t),$$
$$E^*(t) = E(t) \cup E^{\mathrm{tmp}}(t).$$

At any point in time, there is a series $\mathcal{P}_1, \ldots, \mathcal{P}_{N+1}$ of distinct, but strongly related, models. Each $\mathcal{P}_n$ $(1 \leq n \leq N+1)$ is specified by the quadruple $(p_n, \mathcal{G}_n, t_n, w_n)$, where w_n is the number of time slices represented by $\mathcal{P}_n$, t_n is the oldest time slice represented by $\mathcal{P}_n$, $\mathcal{G}_n = (V_n, E_n)$ is a DAG as defined below, and p_n is a probability function defined on $\mathcal{X}_{V_n}$. $\mathcal{P}_1, \ldots, \mathcal{P}_{N-1}$ are referred to as the *backward smoothing models* and the corresponding time slices $t_1, \ldots, t_{N-1}$ as the *backward smoothing slices*. Thus, $w_n = 1$ whenever $n < N$. The maximal number of backward smoothing slices, which we shall denote by b, is greater than or equal to $N-1$, the actual number of backward smoothing slices. $\mathcal{P}_{N+1}$ is referred to as the *forecast model*, and it comprises $f = w_{N+1}$

time slices $t_{N+1}, \ldots, t_{N+1} + f - 1$ referred to as the *forecast slices*. The model $\mathcal{P}_N$ is referred to as the *(time) window*. The window has width $w = w_N$ and thus includes time slices $t_N, \ldots, t_N + w - 1$. See Figure 4.3 for a summary.

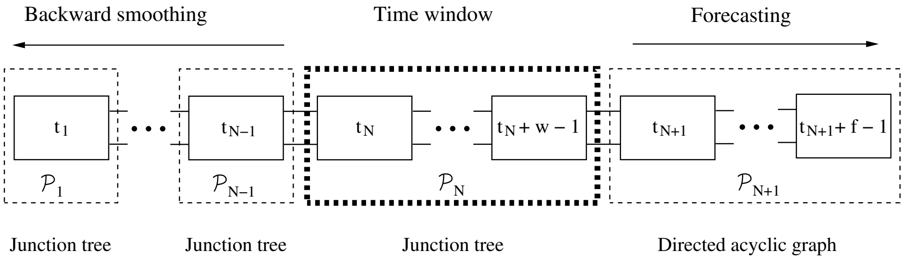

Figure 4.3: A DBN consists of a series of models $\mathcal{P}_1, \ldots, \mathcal{P}_{N+1}$, where the model $\mathcal{P}_N$ defines a time window.

Note that the term 'forecast slices' is slightly imprecise since all inference concerning variables of time slices for which no observations (evidence) have been entered, actually are forecasts, even if such time slices belong to the time window. For similar reasons the term 'backward smoothing slices' is also slightly imprecise.

The DAGs $\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_{N+1} = (V_{N+1}, E_{N+1})$ are given as follows.

$$(V_n, E_n) = \begin{cases} (V(t_n, 1) \cup \mathrm{int}(t_{n+1}), E(t_n, 1) \cup E'(t_{n+1})) & \text{if } n < N \\ (V(t_n, w), E(t_n, w)) & \text{if } n = N \\ (V(t_{N+1}, f) \cup V_N^*, E(t_{N+1}, f) \cup E^{tmp}(t_{N+1})) & \text{if } n = N+1, \end{cases} \tag{4.3}$$

where

$$E'(t_{n+1}) = E^*(t_{n+1}) \cup \Big((\mathrm{int}(t_{n+1}) \otimes \mathrm{int}(t_{n+1})) \cap E(t_{n+1}) \Big),$$

$$V_N^* = \{\alpha \mid (\alpha, \cdot) \in E^{tmp}(t_{N+1})\}.$$

Note that for each $n < N$, $\mathcal{G}_n$ contains the interface of slice t_{n+1}, and $\mathcal{G}_{N+1}$ contains the vertices of $\mathcal{G}_N$ from which there are temporal edges to vertices of slice t_{N+1}. Figure 4.4 shows an example, where $N = 3$ and $w = f = 2$.

4.3 Expansion and reduction of DBNs

Based on the terminologic framework established above, we shall now discuss issues of moving the time window forward. This process involves the two more

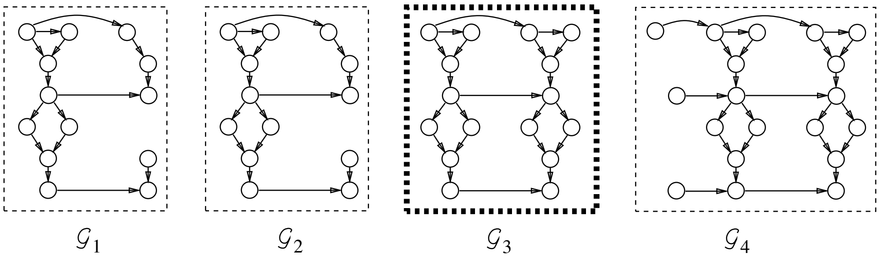

Figure 4.4: A sample DBN consisting of 4 models with associated DAGs $\mathcal{G}_1, \ldots, \mathcal{G}_4$. The backward smoothing models, $\mathcal{P}_1$ and $\mathcal{P}_2$, contain the interface variables of models $\mathcal{P}_2$ and $\mathcal{P}_3$, respectively. The forecast model, $\mathcal{P}_4$, contains variables of the time window needed to make forecasts.

or less separate processes of *window expansion* and *window reduction* discussed in detail in Sections 4.3.2 and 4.3.3. Since the time window is represented by a junction tree, these two processes roughly amount to, respectively, adding a new subtree to the junction tree and cutting off a part of the tree.

Window expansion by, say, k new time slices consists of (a) adding k new consecutive time slices to the forecast model, (b) moving the k oldest time slices of the forecast model to the time window, (c) moralizing the compound graph of the triangulated graph of the window and the DAGs of the k new time slices, (d) triangulating that graph and identifying the new clique set, (e) constructing the new (expanded) junction tree, and (f) computing the new clique potentials with appropriate consideration of the old ones. As discussed in Section 4.3.2, the last step is optional. Expanding the window by k new time slices causes the width, w, of the window to be increased by k, while the number, f, of forecast slices remains unchanged.

In principle, window reduction by k time slices involves elimination of all variables pertaining to time slices $t_N, \ldots, t_N + k - 1$. Recall that elimination of a vertex α forms a complete subset of the vertices $\mathrm{adj}(\alpha)$ unless the set is already complete. The end-product of the elimination process is a belief potential involving the variables $\mathrm{int}(t_N + k)$. This potential, represented in one of the cliques of the reduced junction tree, Υ_N say, represents all information about the past necessary for the reduced window to take full account of the knowledge about the history of the system. If $k \leq b - (N - 1)$ (i.e. k is less than or equal to the difference between the maximal and the actual number of backward smoothing slices), then N is increased by k; otherwise, it is increased by $b - (N - 1)$. In both cases the width, w, of the window is decreased by k, but the number, f, of forecast slices remains unchanged. Note that when the maximal number of backward smoothing slices is decreased by m, say, then N is decreased by $m-(b-(N-1))$ if $m > b-(N-1)$; otherwise, it remains unaltered.

Two issues are of major importance here: (a) if backward smoothing is to be performed, the cliques of the triangulated graph resulting from the reduction process must be linked together in a new junction tree, Υ_{N-1} say, such that backward smoothing can be performed by passing messages from Υ_N to Υ_{N-1} via the belief potential involving variables $\mathrm{int}(t_N)$, and (b) since both the expansion and the reduction process performs a triangulation (i.e. finds an elimination ordering) of (basically) the same graph, these two processes should be coordinated such that the same elimination ordering is employed.

The triangulation carried out as a subtask of the expansion process is unconstrained in the sense that the search space of elimination orderings consists of all permutations of the set V of vertices of the (expanded) window, whereas the reduction process may be perceived as a constrained triangulation, where the eliminated set $A \subset V$ of vertices define the prefix of orderings comprising all vertices in V. Then obviously it might be advantageous to make a constrained decomposition in the first place, rendering the reduction process trivial, provided it is carried out in the fundamental way described above (i.e. assuming the reduction concerns all vertices of the k oldest time slices of the window). This introduces the notion of *a constrained elimination ordering* which is discussed further in Section 4.3.1.

4.3.1 Constrained elimination orderings

A constrained elimination ordering is defined as follows.

Definition 4.1 Let $\mathcal{P}_1, \ldots, \mathcal{P}_N$ be a series of models of a DBN, $\mathcal{G}^N = (V, E)$ the compound graph of $\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_N = (V_N, E_N)$, and $\# : V \leftrightarrow \{1, \ldots, |V|\}$ a bijection defining a total ordering of V. The ordering $\#$ is said to be *constrained* if $\#(\alpha) < \#(\beta)$ for all $1 \leq i < j \leq N$, where $\alpha \in V_i$ and $\beta \in V_j$. Similarly, $T((\mathcal{G}^N_\#)^m)$ is said to be a *constrained triangulation* of $(\mathcal{G}^N)^m$. □

Constrained elimination orderings have a number of important properties which shall be used in Sections 4.3.3 and 4.4.

Rose et al. (1976) have shown that $\alpha \sim \beta$ in a triangulated graph $\mathcal{G}^t_\#$ if and only if all vertices on a path between α and β in $\mathcal{G}_\#$ are of lower order than both α and β (i.e. they are eliminated before α and β). We state that property formally by the following lemma.

Lemma 4.1 (Rose et al. (1976)) *Let $\mathcal{G}_\# = (V, E)$ be an ordered graph. Then $\alpha \sim \beta$ in $\mathcal{G}^t_\#$ if and only if there is a path $\langle \alpha, \alpha_1, \ldots, \alpha_k, \beta \rangle$ in $\mathcal{G}_\# = (V, E)$ such that $\#(\alpha_i) < \min\{\#(\alpha), \#(\beta)\}$ for all $i = 1, \ldots, k$.*

See Rose et al. (1976) for a proof.

Lemma 4.1 can now be used to prove that, under constrained elimination, the interfaces of a DBN induce complete separators of the compound graph of the triangulated graphs corresponding to the DAGs of the DBN.

Lemma 4.2 *Let $\mathcal{P}_1, \ldots, \mathcal{P}_N$ be a series of models of a DBN, $\mathcal{G}^N = (V, E)$ the compound graph of $\mathcal{G}_1, \ldots, \mathcal{G}_N$, and $(\mathcal{G}^N)^*$ the compound graph of the triangulated graphs $\mathcal{G}_1^{t\#}, \ldots, \mathcal{G}_N^{t\#}$, where $\#$ define a constrained ordering of V. Then $\mathrm{int}(t)$ is a complete separator in $(\mathcal{G}^N)^*$ for all $t = t_1, \ldots, t_N + w - 1$.*

Proof: Let n refer to any of the models $\mathcal{P}_1, \ldots, \mathcal{P}_N$ and let t refer to any of the time slices $t_n, \ldots, t_n + w_n - 1$. Since $\#$ defines a constrained ordering, vertices in $V(t-1)$ are eliminated before $\mathrm{int}(t)$ in $\mathcal{G}_n$. Hence, by Lemma 4.1, $\alpha \sim \beta$ in $\mathcal{G}_n^{t\#}$ for each $\alpha, \beta \in \mathrm{int}(t)$ (i.e. $\mathrm{int}(t)$ is complete in $\mathcal{G}_n^{t\#}$). Since, by definition, $\mathrm{int}(t)$ is a separator of $(\mathcal{G}^N)^m$ and vertices in $V(t-1)$ are eliminated before $\mathrm{int}(t)$, then $\alpha \not\sim \beta$ in $(\mathcal{G}^N)^*$ for each $\alpha \in V(t-1)$ and each $\beta \in V(t) \setminus \mathrm{int}(t)$. Thus $\mathrm{int}(t)$ is a complete separator in $(\mathcal{G}^N)^*$. □

This result immediately leads to the following corollary.

Corollary 4.1 *Let $\mathcal{P}_1, \ldots, \mathcal{P}_N$ be a series of models of a DBN, $\mathcal{G}^N = (V, E)$ the compound graph of the triangulated graphs $\mathcal{G}_1^{t\#}, \ldots, \mathcal{G}_N^{t\#}$ associated with the models, where $\#$ defines a constrained ordering of V. Then $\mathcal{G}^N$ is constrainedly triangulated.*

Proof: By Lemma 4.2, $\mathcal{G}^N$ has complete separators $\mathrm{int}(t_1), \ldots, \mathrm{int}(t_N + w - 1)$. Thus, for any $t \in \{t_1, \ldots, t_N + w - 1\}$, $(A, B, \mathrm{int}(t))$ is a decomposition of $\mathcal{G}^N$, where $A = V(t_1) \cup \cdots \cup V(t-1)$ and $B = V \setminus (A \cup \mathrm{int}(t))$. The graphs $\mathcal{G}^N_{A \cup \mathrm{int}(t)}$ and $\mathcal{G}^N_{B \cup \mathrm{int}(t)}$ have complete separators $\mathrm{int}(t_1), \ldots, \mathrm{int}(t)$ and $\mathrm{int}(t), \ldots, \mathrm{int}(t_N + w - 1)$, respectively. Thus, there are decompositions $(\cdot, \cdot, \mathrm{int}(t'))$ and $(\cdot, \cdot, \mathrm{int}(t''))$ for $\mathcal{G}^N_{A \cup \mathrm{int}(t)}$ and $\mathcal{G}^N_{B \cup \mathrm{int}(t)}$, respectively. Continuing this argument we end up with subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_N$ all of which are constrainedly triangulated, and the result follows. □

4.3.2 Window expansion

The operation of window expansion refers to the processes of adding k new time slices $t_{N+1} + f, \ldots, t_{N+1} + f + k$ to the forecast model and subsequent transferral of the k oldest time slices $t_N + w = t_{N+1}, \ldots, t_{N+1} + k - 1$ of the forecast model to the time window. The wish to expand the window may be explicit, or implicit as part of the operation of moving the window k time slices forward.

A new time slice, $t_{N+1} + f$, to be added to the forecast model contains one or more variables for which their associated conditional probability functions are specified in terms of variables belonging to the forecast model. Due to the assumption of compliance with the Markov property, all such variables of the forecast model must be included in time slice $t_{N+1} + f - 1$ (i.e. the last slice of the model). In many applications the time slices added to a dynamic model will be identical. However, apart from compliance with the Markov property, the subnetworks (time slices) added to the model can be specified absolutely freely (i.e. there are no assumptions of the time slices added being identical or similar in some sense, although in many applications this is often the case).

Moving the k oldest time slices of the forecast model to the time window is conceptually straightforward. We simply create the compound graph of the triangulated graph of the time window and the DAG induced by the k oldest time slices of the forecast model, and subsequently adds the subset of vertices of the (expanded) time window which have children belonging to the (reduced) forecast model (i.e. the subset referred to as V_N^* above). In order to produce a junction tree for the expanded window we perform the operations of moralization and triangulation. Moralization is performed in the compound graph consisting of both directed and undirected edges. After (constrained) triangulation, the belief potentials (i.e. conditional probabilities) of the new variables are attached to appropriate belief universes (cliques).

A sample window expansion is shown in Figure 4.5. The time window consists of a single time slice (the initial one) before expansion takes place, and no forecast slices are present. Assume now that we want to expand the window by time slice 1, and that the forecast model should consist of a single time slice (time slice 2). Therefore we first add time slices 1 and 2 to the forecast model and, since the forecast model is currently void, dummy variables a, d, and h pertaining to time slice 0 is added to the forecast model to make it self-contained (i.e. such that forecasts, by means of Monte-Carlo simulation, can be done without reference to the time window once appropriate information has been passed from the window). Secondly, time slice 1 is moved from the forecast domain to the window, and the resulting graph is moralized (moral edges indicated as dashed lines) and triangulated (fill edges indicated as dotted lines). Finally, the new expanded junction tree is created as descibed below.

Obviously, in finding an optimal elimination ordering, we have to consider the topology of the graph as it appears after the addition of time slice $t_N + w$ (i.e. the next one to be added). Since we want the model complexity in terms of the state space size to be as low as possible to minimize the complexity of inference, and since the state space size varies heavily over the range of elimination orderings, a careful analysis must be conducted to establish an appropriate ordering. To

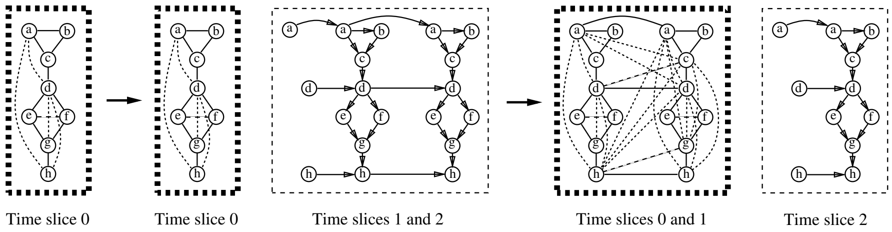

Figure 4.5: Sample window expansion. Before window expansion takes place, the time window consists of time slice 0 only, and the forecast model is void. The expansion of both the window and the forecast model by one time slice requires addition of two new time slices, 1 and 2. Time slice 1 is subsequently moved from the forecast model to the time window which is then moralized (dashed edges) and triangulated (dotted edges).

find an optimum elimination ordering for an arbitrary graph is, however, an NP-hard problem (Wen 1990). Yet, in practice it turns out that near optimum triangulations may be found using simple heuristic ordering strategies (Rose 1973, Kjærulff 1992b). In Figure 4.5 the elimination ordering applied to obtain the triangulated graph of time slice 0 is b, e, f, c, g, d, a, h. (The corresponding DAG and moral graph are shown in Figures 4.1 and 4.2.)

Having identified the cliques of the new triangulated graph of the window, the next step concerns construction of a junction tree with belief universes corresponding to those cliques. As much as possible of the junction tree, $\Upsilon = (\mathcal{C}, \mathcal{S})$, in existence prior to the expansion should be reused in order to minimize the amount of work required to construct the expanded junction tree $\Upsilon' = (\mathcal{C}', \mathcal{S}')$. Note that as a direct consequence of the constrained triangulation scheme, there is for each 'old' universe $C \in \mathcal{C}$ a 'new' universe $C' \in \mathcal{C}'$ such that $C \subseteq C'$. For some universes the containment might be strict. The creation of Υ' can be described as follows.

1. Identify the set $\mathcal{C}'$ of cliques of Υ'.

2. Construct a 'skeleton' of Υ':

   (a) Create belief universes for all members of $\mathcal{C}' \setminus \mathcal{C}$ and separators for all members of $\mathcal{S}' \setminus \mathcal{S}$.

   (b) Create unit-potential tables for these new universes and separators (i.e. tables with all cells instantiated to 1). (The belief potentials of the universes in $\mathcal{C} \cap \mathcal{C}'$ and of the separators in $\mathcal{S} \cap \mathcal{S}'$ remain unchanged.)

3. For each member of $\mathcal{C} \setminus \mathcal{C}'$ and each member of $\mathcal{S} \setminus \mathcal{S}'$ (i.e. 'old' cliques rendered redundant and the separators incident to them), multiply the associated belief potentials to the belief potentials of appropriate universes and separators and attach the result to these objects.

4. Attach the conditional probability tables of the variables of the new time slices to appropriate new belief universes.

(The term 'appropriate' in points 3 and 4 refers to the index set of the table to be attached being a subset of the set of variables of the universe or separator to which it is attached.) The expanded junction tree Υ' has now been created. That is, a potential representation for the joint probability distribution for the expanded window has been established.

Figure 4.6 illustrates the modifications of the time window junction tree corresponding to the sample window expansion in Figure 4.5. Parts (a) and (b) show, respectively, the junction tree before and after the expansion. The numbers attached to the belief universes correspond to the order of creation using the above elimination ordering. In this example, universe no. 5 of part (a) becomes redundant, since it is a proper subset of universe no. 5 of part (b). Thus the belief potential of universe no. 5 of part (a) and the belief potentials of its incident separators are transferred to, respectively, universe no. 5 of part (b) and relevant separators incident to it (indicated by the dashed arrows). The universes (and separators) remaining unchanged are shown in bold.

Now, if we have an immediate interest in the marginal distributions of variables of the k new time slices of the window, a propagation can be performed; otherwise we might postpone the propagation step until e.g. new observations have been recorded. If Υ was consistent immediately before the window expansion was executed, we only need to perform propagation in the subtree induced by the set of new belief universes (see Figure 4.7).

4.3.3 Window reduction

Due to the constrained decomposition scheme employed by the window expansion process, window reduction becomes a relatively easy task as previously discussed. In developing a window reduction scheme it is important to recognize the requirements for convenient backward smoothing beyond the time window. Below we develop a reduction scheme which meets such requirements and which is based on the following result.

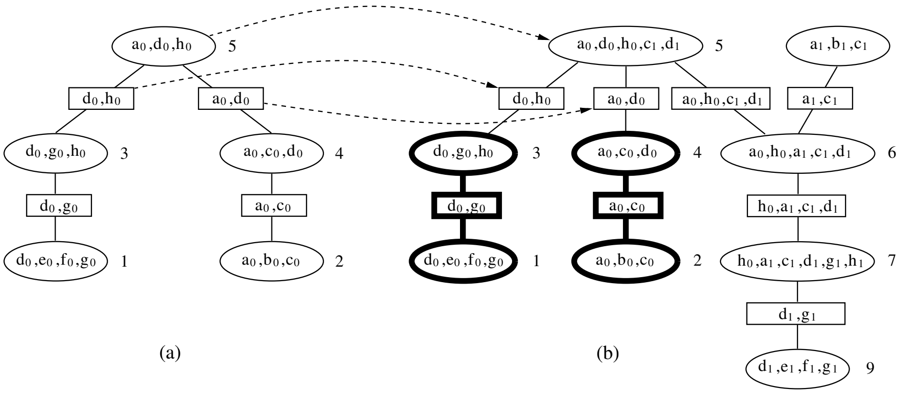

Figure 4.6: Modifications of the time window junction tree corresponding to the window expansion in Figure 4.5. Belief potentials of universe 5 of part (a) and its incident separators are transferred to the new junction tree, since this universe becomes redundant. The parts shown in bold are inherited from the old junction tree.

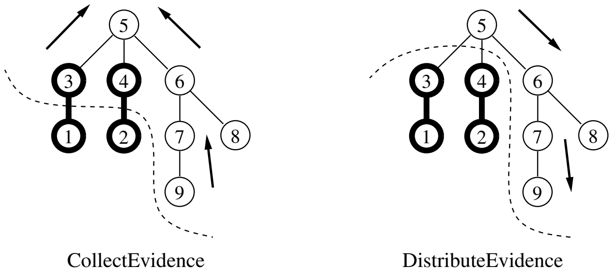

Figure 4.7: If the junction tree was consistent before expansion took place, then to obtain consistency in the expanded junction tree, it is sufficient to collect evidence to a universe in the subtree induced by the new universes *and* the 'old' universes being adjacent to new universes, and subsequently, to distribute evidence from that universe in the subtree induced by the new universes.

Theorem 4.2 *Let $\mathcal{P}_1, \ldots, \mathcal{P}_N$ be a series of (marginally) consistent models, with associated constrainedly decomposable covers. Assume that $\mathcal{P}_{n-1}$ and $\mathcal{P}_n$ are mutually inconsistent for some $1 < n \leq N$. Sufficient information required to render $\mathcal{P}_{n-1}$ and $\mathcal{P}_n$ mutually consistent is given by the marginal belief potentials $\phi_{int(t_n)}$, where there is a universe C_1 of $\mathcal{P}_{n-1}$ and a universe C_2 of $\mathcal{P}_n$ such that $int(t_n) \subset C_1$ and $int(t_n) \subseteq C_2$.*

Proof: Let $\mathcal{G}^t_{n-1}$ and $\mathcal{G}^t_n$ be the (constrainedly) triangulated graphs corresponding to the decomposable covers of $\mathcal{P}_{n-1}$ and $\mathcal{P}_n$, and let $\#$ define the associated constrained elimination ordering. From Lemma 4.2 we have that $\mathrm{int}(t_n)$ is a complete separator of $\mathcal{G}^t_{n-1} \cup \mathcal{G}^t_n$ and hence $\phi_{\mathrm{int}(t_n)}$ contains complete mutual information between $\mathcal{P}_{n-1}$ and $\mathcal{P}_n$. From the definition of $\mathcal{G}_i$, $1 \leq i < N$, (cf. (4.3)) we have that $\mathrm{int}(t_n) \subset V(t_{n-1})$, and since for each pair $\{\alpha, \beta\}$, where $\alpha \in V(t_{n-1})$ and $\beta \in \mathrm{int}(t_n)$, $\#(\alpha) < \#(\beta)$, $\mathrm{int}(t_n)$ induces a complete subgraph of $\mathcal{G}^t_{n-1}$. Hence there is a clique C_1 of $\mathcal{G}^t_{n-1}$ such that $\mathrm{int}(t_n) \subset C_1$. Since $\mathrm{int}(t_n)$ is complete in $\mathcal{G}^t_{n-1}$ it follows immediately that it is also complete in $\mathcal{G}^t_n$ and hence there is a clique C_2 in $\mathcal{G}^t_n$ such that $\mathrm{int}(t_n) \subseteq C_2$. □

Reduction of the window by k time slices partition $\mathcal{P}_N$ into $k+1$ submodels, where the k models become the k consecutive backward smoothing models immediately preceding the new (reduced) time window. The $k+1$'st model is the new time window. If the new actual number of backward smoothing slices, $N-1$, exceeds the maximal number, b, then we dispose of the models representing the $(N-1)-b$ oldest time slices and let $N = b+1$. That is, whenever $\mathcal{P}_N$ is subjected to a reduction of k time slices, the net increase of N is k if $k \leq b-(N-1)$; otherwise, it is $b-(N-1)$.

In terms of operations on $\mathcal{G}_N$ (i.e. the DAG of the time window), the window reduction process produces $k+1$ new DAGs, $\mathcal{G}_{N-k}, \ldots, \mathcal{G}_N$, complying with (4.3) and with N updated as described above. In terms of operations on the junction tree of the time window, the reduction process may be formulated as follows, where consistency shall be assumed before the process is initiated.

1. Let $\Upsilon_0 = (\mathcal{C}_0, \mathcal{S}_0)$ be a junction tree for the window before reduction takes place.

2. For $i = 0$ to $k-1$ do

   (a) Let $\mathcal{C}'_i = \{C \in \mathcal{C}_i \mid C \cap V(t_N + i) \neq \emptyset\}$ be the set of belief universes containing variables of time slice $T_N + i$. Let $\mathcal{C}''_i = \mathcal{C}_i \setminus \mathcal{C}'_i$ be the remaining belief universes.

   (b) Let $\Upsilon'_i = \Upsilon_{\mathcal{C}'_i}$ and $\Upsilon''_i = \Upsilon_{\mathcal{C}''_i}$ be the junction trees induced by $\mathcal{C}'_i$ and $\mathcal{C}''_i$, respectively (see Figure 4.8).

(c) Let $\mathcal{B}_i = \{C \in \mathcal{C}''_i \mid \mathrm{adj}(C) \cap \mathcal{C}'_i \neq \emptyset \text{ in } \Upsilon_i\}$ be the set of belief universes of Υ''_i which are adjacent to universes of Υ'_i.

(d) If there is no $C \in \mathcal{B}_i$ such that $\mathrm{int}(t_N + i + 1) \subseteq C$, then create a belief universe comprising $\mathrm{int}(t_N + i + 1)$, add it to Υ''_i, and let its neighbours be $\mathcal{B}_i$; otherwise, add $\mathcal{B}_i \setminus \{C\}$ to the neighbour set of C.

(e) Let $\Upsilon_{i+1} = (\mathcal{C}_{i+1}, \mathcal{S}_{i+1}) = \Upsilon''_i = (\mathcal{C}''_i, \mathcal{S}''_i)$.

3. Let $N := N + k$. If $N - 1 > b$, then dispose of the models representing the $(N - 1) - b$ oldest time slices and let $N := b + 1$.

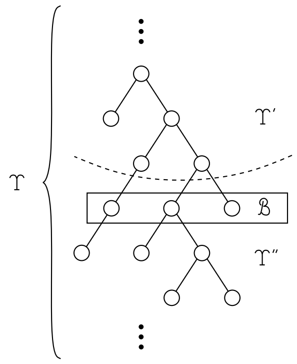

Figure 4.8: Partitioning Υ into Υ'_i and Υ''_i.

After the execution of Steps 1–3, the junction trees $\Upsilon'_0, \ldots, \Upsilon'_{k-1}, \Upsilon''_{k-1}$ are mutually consistent and correspond to models $\mathcal{P}_{N-k}, \ldots, \mathcal{P}_N$, respectively. To see that $\Upsilon'_0, \ldots, \Upsilon'_{k-1}$ are junction trees of $\mathcal{P}_{N-k}, \ldots, \mathcal{P}_{N-1}$ it suffices to realize that any subtree of a junction tree is a junction tree and that $\bigcup_{C \in \mathcal{C}'_i} C$ corresponds exactly to the vertex set of $\mathcal{G}_i$ for all $i = 0, \ldots, k-2$ (cf. (4.3)).

The fact that $\Upsilon_k = \Upsilon''_{k-1}$ is a junction tree of the new time window can be seen as follows, where we let $\mathcal{G}^t_N$ be the (constrainedly) triangulated graph corresponding to the decomposable cover of $\mathcal{P}_N$.

First, assume that the condition of the 'if' part of Step 2d holds. Since the constrained triangulation forces $\mathrm{int}(t_N + k)$ to induce a complete subgraph of $\mathcal{G}^t_N$ and since there is no clique in $\mathcal{C}''_{k-1}$ containing $\mathrm{int}(t_N + k)$, then $\mathrm{int}(t_N + k)$ itself must be a clique of $\mathcal{G}^t_N$. The subset $\mathcal{B}_{k-1} \subseteq \mathcal{C}''_{k-1}$, where for each $B \in \mathcal{B}$ there is a non-empty intersection between the adjacency set of B and $\mathcal{C}'_{k-1}$ in Υ_{k-1}, is then made the adjacency set of the new belief universe comprising $\mathrm{int}(t_N + k)$. Since the path in Υ_{k-1} between any pair of elements in $\mathcal{B}_{k-1}$ includes elements in $\mathcal{C}'_{k-1}$ (i.e. $\mathcal{C}'_{k-1}$ separates the elements of $\mathcal{B}_{k-1}$ from one another), this does not violate the tree structure of Υ_k. Neither does it violate the property of Υ_k being a junction tree, as the intersection of any pair (C', C'') of belief universes, where $C' \in \mathcal{C}'_{k-1}$ and $C'' \in \mathcal{C}''_{k-1}$, is a subset of $\text{int}(t_N + k)$.

Second, assume the condition to fail (i.e. there is a belief universe $C \in \mathcal{C}''_{k-1}$ such that $\text{int}(t_N + k) \subseteq C$) in which case $\mathcal{B}_{k-1} \setminus \{C\}$ is made a subset of the adjacency set of C in Υ_k. With arguments similar to those above it is readily realized that the property of Υ_k being a junction tree is not violated.

4.4 Forward and backward propagation

The above description of the expansion and reduction processes provides a framework for providing a comprehensible account of issues related to propagation of evidence in DBNs. As indicated in Figure 4.3, for each $n \leq N$, inference involving model $\mathcal{P}_n$ is carried out in a junction tree representation of p_n, whereas inference in the forecast model, $\mathcal{P}_{N+1}$, is performed in a DAG using Monte-Carlo methods (see Section 4.5). Bidirectional message passing (transferral of information) between two consecutive models $\mathcal{P}_n$ and $\mathcal{P}_{n+1}$ is allowed only when $n < N$ (i.e. when both models are represented as junction trees). Information can be transferred from the time window to the forecast model, but not vice versa. Message passing involving two junction trees is termed *forward propagation* or *backward propagation* depending on the direction in which the message is passed.

Clearly, the arrival of external evidence (observations) affects not only the calculation of revised beliefs on variables of the time slice(s) pertaining to the observed variables, but may also have significant impact on beliefs on variables of other time slices. Clearly, this impact goes forward as well as backward in time.

The process of revising beliefs of variables of past slices in light of new evidence (retrospective assessment) is often referred to as *backward smoothing*. If the variables for which revised beliefs are required, are all included in the model, $\mathcal{P}_n$, for which evidence has been obtained, then backward smoothing is an implicit part of propagation in the junction tree of $\mathcal{P}_n$. However, if we want to calculate revised beliefs for variables pertaining to model $\mathcal{P}_{n-k}$ $(0 < k < n)$, then sufficient information about the observations should be propagated backward from $\mathcal{P}_n$ to $\mathcal{P}_{n-k}$ through belief updating of the models $\mathcal{P}_{n-1}, \ldots, \mathcal{P}_{n-k-1}$ (in that order). Similarly, if $n < N$, then appropriate information should be propagated forward from $\mathcal{P}_n$ to $\mathcal{P}_N$ via the interjacent models.

From the description of the window reduction scheme (Steps 1–3 in Section 4.3.3) it is clear that for each pair of models $\mathcal{P}_n, \mathcal{P}_{n+1}$ $(1 \leq n < N)$ both of the associated junction trees contain a belief universe including the interface of time slice t_{n+1}, $\mathrm{int}(t_{n+1})$. Let these universes be C and D, respectively. By Theorem 4.2, the marginal belief potentials $\sum_{C\setminus D} \phi_C$ and $\sum_{D\setminus C} \phi_D$ on $\mathrm{int}(t_{n+1})$ both provide sufficient information to render $\mathcal{P}_n$ and $\mathcal{P}_{n+1}$ mutually consistent, provided $\mathcal{P}_n$ and $\mathcal{P}_{n+1}$ are (marginally) consistent. Therefore, transferral of information from e.g. $\mathcal{P}_n$ to $\mathcal{P}_{n+1}$ is performed by absorption

$$\phi_D^* = \phi_D * \frac{\sum_{C\setminus D} \phi_C}{\sum_{D\setminus C} \phi_D}$$

followed by distribution of revised separator potentials from universe D.

To summarize, each model $\mathcal{P}_n$ $(1 \le n < N)$ has two *interface universes* IC_n^- and IC_n^+, where $\mathrm{int}(t_n) \in IC_n^-$ and $\mathrm{int}(t_{n+1}) \in IC_n^+$, and $\mathcal{P}_N$ has one interface universe IC_N^- containing $\mathrm{int}(t_N)$. (If $t_1 = 0$ (i.e. $\mathcal{P}_1$ represent the initial time slice), then IC_1^- is udefined.) When performing forward propagation from $\mathcal{P}_n$ $(1 \le n < N)$ to $\mathcal{P}_m$ $(n < m \le N)$, any model $\mathcal{P}_i$ $(n < i < m)$ absorbs information from IC_{i-1}^+ (through IC_n^-), propagates it, and emits information from IC_n^+; which is absorbed by $\mathcal{P}_{i+1}$ through IC_{n+1}^-, etc; see Figure 4.9. The process of backward propagation is similar.

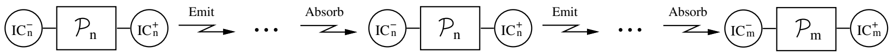

Figure 4.9: Forward propagation from $\mathcal{P}_n$ to $\mathcal{P}_m$.

4.5 Forecasting

In time-series analysis applications there is typically a desire to make forecasts of the random process considered. That is, to calculate estimated distributions of future variables given observations of past and present variables. Within the computational framework presented above, forecasts which do not exceed the extent of the time window are implicit parts of propagation in the junction tree of the window. If the required distributions are not provided by the window, then they be calculated through expansion of the window by an appropriate number of time slices followed by propagation. However, if forecasts are wanted for a large number of time slices ahead of the window, then this method might result in a model the complexity of which exceeds the capacity of the available computing resources. One way to resolve this problem is to advance the window an appropriate number of steps (i.e. a series of alternating expansion/reduction steps), where propagation is performed in each step, and then move it back again. This might, however, be a very time consuming operation, and furthermore, a lot of unnecessary calculations will quite often be carried out, because, typically, forecasts are only wanted for a limited number of variables. Therefore, there is a demand for alternative forecast methods which either avoids the junction tree approach and/or exploits the fact that forecasts are only required for a limited number of variables. We shall address the first of these two alternatives.

4.5.1 Forward sampling

The Monte-Carlo sampling scheme denoted 'forward sampling' is an efficient way of calculating single-variable marginal distributions in directed Markov fields when observations are not available for variables with non-empty parent set.

In general, we have a collection $X = (X_v)_{v \in V}$ of random variables for which we seek marginal probabilities for each $v \in V$. With f being a density defined on $\mathcal{X}$, our objective is to provide an estimate for a parameter

$$\begin{aligned}\theta &= E\phi(X_1) = E\phi(X_2) = \cdots = E\phi(X_m) \\ &= \int \phi(x)f(x)dx \\ &= \lim_{m \to \infty} \frac{1}{m} \sum_{i=1}^{m} \phi(X_i),\end{aligned}$$

where $X_1, \ldots, X_m$ are independent and identically distributed random variables associated with samples from $\mathcal{X}$ w.r.t. f (i.e. x_i is sampled with probability $f(x_i)$). An estimate of θ can be obtained as

$$\hat{\theta} = \frac{1}{m} \sum_{i=1}^{m} \phi(X_i), \tag{4.4}$$

since

$$E\hat{\theta} = \frac{1}{m} \sum_{i=1}^{m} E\phi(X_i) = \frac{1}{m} \sum E\phi(X_1) = E\phi(X_1) = \theta.$$

The variance of the estimate provides a measure of the precision by which the value of the parameter has been determined. The variance of $\hat{\theta}$ is given by

$$\begin{aligned}\mathrm{Var}(\hat{\theta}) &= \frac{1}{m} \int (\phi(x) - \theta)^2 f(x)dx \\ &\approx \frac{1}{m^2} \sum_{i=1}^{m} (\phi(X_i) - \hat{\theta})^2. \end{aligned} \tag{4.5}$$

So the standard deviation of $\hat{\theta}$ is proportional to $1/\sqrt{m}$, which implies that a decrease of the standard deviation (or standard error) by a factor of k requires an increase of the sample size by k^2. Thus, to get forecasts within a small distance from the exact values, we should expect the computing time to be relatively large; in some cases even larger than those required by exact methods, but of course with much less space requirements since the sampling is performed in a DAG structure involving relatively low-dimensional probability tables. Another important feature of sampling methods is that the time complexity grows only linearly in the dimensionality of the tables involved, whereas it grows exponentially for exact methods.

If the parameters, the values of which we are seeking, are given by $\theta = (\theta_v^a = P(X_v = a)), v \in V, a \in \mathcal{X}_v$, we define

$$\phi(x) = \left(\phi_v^a(x) = \begin{cases} 1 & \text{if } x_v = a \\ 0 & \text{otherwise} \end{cases}\right), v \in V, a \in \mathcal{X}_v,$$

such that $\theta_v^a = E\phi_v^a$, where the expectation is taken w.r.t. f, and $\hat{\theta}_v^a = (\sum \phi_v^a)/m$. Given that definition we obviously have

$$f_v(a) = \sum_{x \in \mathcal{X}: x_v = a} f(x) = \sum_{x \in \mathcal{X}} \phi_v^a(x) f(x). \tag{4.6}$$

The fact that ϕ is a mapping from $\mathcal{X}$ to $\{0, 1\}$ enables a substantial reduction of (4.5):

$$\begin{aligned} \mathrm{Var}(\hat{\theta}_v^a) &\approx \frac{1}{m^2} \sum_{i=1}^{m} \left(\phi_v^a(x_i) - \hat{\theta}_v^a\right)^2 \\ &= \frac{1}{m^2} \left[\sum_{i=1}^{n} \left(1 - \hat{\theta}_v^a\right)^2 + \sum_{i=1}^{m-n} \left(\hat{\theta}_v^a\right)^2\right] \\ &= \frac{1}{m} \left[\frac{n}{m} \left(1 - \hat{\theta}_v^a\right)^2 + \frac{m-n}{m} \left(\hat{\theta}_v^a\right)^2\right] \\ &= \frac{1}{m} \left[\hat{\theta}_v^a \left(1 - \hat{\theta}_v^a\right)^2 + \left(1 - \hat{\theta}_v^a\right) \left(\hat{\theta}_v^a\right)^2\right] \\ &= \frac{\hat{\theta}_v^a \left(1 - \hat{\theta}_v^a\right)}{m}, \end{aligned} \tag{4.7}$$

where $n = \sum_{i=1}^{m} \phi_v^a(x_i)$, i.e. the number of times $(x_i)_v = a$.

If f is Markov w.r.t. to a DAG, $\mathcal{G} = (V, E)$, a sample x_i may be obtained as follows. Let $A \subseteq V$ be the set orphan vertices of $\mathcal{G}$. Since no observations are available except possibly for members of A, then A constitute a set of marginally independent variables. Hence, these variables may be instantiated independently. Now, conditional on A the orphan vertices of $\mathcal{G}_{V\setminus A}$ are independent, and may hence be instantiated independently, but conditional on the instantiated values associated with the variables in A. Continuing this process until all variables have been instantiated, we have a sample x_i. Note that there is always at least one orphan vertex of a DAG, due to the absence of directed cycles. This way of sampling from $\mathcal{X}_V$ is denoted *forward sampling*.

4.5.2 Fast forward sampling

In forward sampling the estimate $\hat{\theta}_v^a$ is derived as the proportion of the samples where $(x_i)_v = a$, $i = 1, \ldots, m$. In computing an estimate of the distribution of X_v by forward sampling, we take samples from $\mathcal{X}_v$ w.r.t. $f(X_v \mid x_{pa(v)})$. These samples serve two purposes. First, they (partly) specify the density w.r.t. which samples from $\mathcal{X}_{ch(v)}$ should be drawn. Second, they provide the basis for calculating $\hat{\theta}_v^a$, $a \in \mathcal{X}_v$. However, why are we using the samples for this calculation? Intuitively, it seems more efficient to calculate an estimate for the distribution of X_v by averaging over the sample densities $f(X_v \mid x_{pa(v)})$, i.e.

$$\tilde{\theta}_v = \frac{1}{m}\sum_{i=1}^{m} f\left((X_i)_v \mid (x_i)_{pa(v)}\right),$$

The variance of $\tilde{\theta}$ is given as

$$\begin{aligned}
\mathrm{Var}(\tilde{\theta}_v^a) &= \mathrm{Var}\left(f(x_v^a \mid X_{pa(v)})\right) \\
&= \frac{1}{m}\sum_{y\in\mathcal{X}_{pa(v)}} \left(f(x_v^a \mid y) - f(x_v^a)\right)^2 f(y) \\
&= \frac{1}{m}\sum_{y\in\mathcal{X}_{pa(v)}} f(x_v^a \mid y)^2 f(y) + f(x_v^a)^2 f(y) - 2f(x_v^a \mid y) f(x_v^a) f(y) \\
&= \frac{1}{m}\left\{\left[\sum_{y\in\mathcal{X}_{pa(v)}} f(x_v^a \mid y) f(x_v^a, y)\right] - f(x_v^a)^2\right\}.
\end{aligned}$$

Theorem 4.3 *$\mathrm{Var}(\tilde{\theta}) \leq \mathrm{Var}(\hat{\theta})$.*

Proof: With $\mathrm{Var}(\hat{\theta}_v^a) = f(x_v^a)(1 - f(x_v^a))/m$ we get

$$\begin{aligned}
\mathrm{Var}(\tilde{\theta}_v^a) - \mathrm{Var}(\hat{\theta}_v^a) &= \frac{1}{m}\left\{\left[\sum_{y \in \mathcal{X}_{\mathrm{pa}(v)}} f(x_v^a \mid y) f(x_v^a, y)\right] - \right. \\
&\qquad \left. f(x_v^a)^2 - f(x_v^a)(1 - f(x_v^a))\right\} \\
&= \frac{1}{m} \sum_{y \in \mathcal{X}_{\mathrm{pa}(v)}} f(x_v^a \mid y) f(x_v^a, y) - f(x_v^a, y) \\
&= \frac{1}{m} \sum_{y \in \mathcal{X}_{\mathrm{pa}(v)}} \left(f(x_v^a \mid y) - 1\right) f(x_v^a, y) \\
&\leq 0.
\end{aligned}$$

□

Based on the samples, an approximate expression for $\mathrm{Var}(\tilde{\theta}_v^a)$ is

$$\mathrm{Var}(\tilde{\theta}_v^a) \approx \frac{1}{m^2} \sum_{i=1}^{m} \left(f\left(x_v^a \mid (x_i)_{\mathrm{pa}(v)}\right) - \tilde{\theta}_v^a\right)^2. \tag{4.8}$$

However, calculating $\mathrm{Var}(\tilde{\theta}_v^a)$ using this formula requires an m-dimensional vector of probabilities $f(x_v^a \mid (x_i)_{\mathrm{pa}(v)})$ for each state $a \in \mathcal{X}_v$ of each variable $X_v \in X_V$. To circumvent such an excessive space requirement we might instead calculate the difference between $\mathrm{Var}(\hat{\theta}_v^a)$ and $\mathrm{Var}(\tilde{\theta}_v^a)$ and subtract it from $\mathrm{Var}(\hat{\theta}_v^a)$. Hence, based on (4.8) an approximation of $\mathrm{Var}(\tilde{\theta}_v^a)$ may be obtained as

$$\begin{aligned}
\mathrm{Var}(\tilde{\theta}_v^a) &= \mathrm{Var}(\hat{\theta}_v^a) - \left(\mathrm{Var}(\hat{\theta}_v^a) - \mathrm{Var}(\tilde{\theta}_v^a)\right) \\
&\approx \frac{1}{m}\tilde{\theta}_v^a\left(1 - \tilde{\theta}_v^a\right) - \\
&\quad \frac{1}{m^2} \sum_{i=1}^{m} f\left(x_v^a \mid (x_i)_{\mathrm{pa}(v)}\right) - f\left(x_v^a \mid (x_i)_{\mathrm{pa}(v)}\right)^2,
\end{aligned} \tag{4.9}$$

where the sum can be calculated 'on the fly', enabling a reduction in space requirement by a factor of m.

Superiority of fast-forward sampling

Given an upper bound on the acceptable (average) standard error, it is an interesting question how much faster the fast forward sampling scheme reaches that bound than do the plain forward sampling scheme. Unfortunately, the ratio between the rates of convergence for the two schemes is not constant, but relies heavily on the characteristics of the density, f, from which we sample. The sharper f is the closer the ratio will be to unity. In the limit where the conditional probabilities of the CPN are purely logical the two schemes are equivalent. However, since the computational complexity of the two are identical, there are no reasons for sticking to the plain scheme.

To get an idea of the improvement achieved by replacing the plain forward sampler with the fast forward sampler, consider the figures of Table 4.1. These figures represent the minimum, average, and maximum ratios between $\mathrm{Var}(\hat{\theta}_v^a)$ and $\mathrm{Var}(\tilde{\theta}_v^a)$ for four different networks. *Note:* if $k = \mathrm{Var}(\hat{\theta}_v^a)/\mathrm{Var}(\tilde{\theta}_v^a)$ with a sample size of m, then the fast forward sampler provides forecasts of a precision similar to that of the plain forward sampler with only m/k samples. The

Network	Min. ratio	Ave. ratio	Max. ratio
Diabetes	1.00	1.57	7.56
Wheat	1.06	4.25	9.16
Water	1.60	7.03	69.21
TestNet	15.02	32.57	84.67

Table 4.1: Minimum, average, and maximum ratios between $\mathrm{Var}(\hat{\theta}_v^a)$ and $\mathrm{Var}(\tilde{\theta}_v^a)$.

Diabetes network represents a model-based approach to insulin adjustment (Andreassen et al. 1991); the Wheat network is a preliminary model for the influence of temperature, precipitation, solar energy, and spraying against mildew on winter wheat; the Water network is a probabilistic model of the biological processes of a water treatment plant (Jensen et al. 1989); and the TestNet is the common sample network used in the present paper with more or less arbitrary probabilities attached. Assuming that an acceptable precision using the plain forward sampler is obtained with a sample size of 1000, the average ratios in Table 4.1 indicate that the fast forward sampler needs only sample sizes of 636, 235, 142, and 30 for the networks Diabetes, Wheat, Water, and TestNet, respectively.

Confidence interval for θ_v^a

Given the sample size m, the estimate $\hat{\theta}$, the variance of the estimate $\mathrm{Var}(\hat{\theta})$, and a confidence level γ (0.95, 0.99 or the like), a confidence interval can be computed for θ_v^a, $v \in V, a \in \mathcal{X}_v$, can be computed by utilizing the asymptotic behaviour of the distribution function for $\hat{\theta}$.

The following is a standard cook-book recipe for computing the confidence interval for θ_v^a.

1. Choose a confidence level e.g. $\gamma = 0.95$.
2. Determine c such that $P(-c \leq \theta_v^a \leq c) = \gamma$, i.e. such that $\Phi(c) = (1+\gamma)/2$, where Φ is the distribution function for the standard normal density.
3. Compute $k = c\sqrt{\mathrm{Var}(\hat{\theta}_v^a)}$, where $\mathrm{Var}(\hat{\theta}_v^a)$ is given by (4.7). The confidence interval for θ_v^a at level γ is then

$$\hat{\theta}_v^a - k \leq \theta_v^a \leq \hat{\theta}_v^a + k.$$

4.6 Discussion

We have presented a computational scheme for reasoning in dynamic Bayesian networks featuring description of non-linear, multivariate dynamic systems with complex conditional independence structures and providing mechanisms for efficient forward propagation (incorporation of late observations) and backward propagation (backward smoothing). The phenomenon of late observations appears in numerous applications; for example, in a medical setting delays may be caused by processing time in a laboratory (e.g. analysis of a blood sample). As opposed to a static network representing a fixed number of time slices (i.e. capable of reasoning only about a finite series of observations of a dynamic system) the proposed scheme can handle infinite series of observations. Further, in applying static networks representing a fixed number of time slices as models of dynamic systems, there is typically a desire to include as many time slices as possible in the model. Thus, inference easily becomes time consuming and inflexible (i.e. propagation involves all time slices in the model even if updated distributions are wanted only for a limited number of time slices). The proposed scheme, on the other hand, provides a high degree of flexibility in the reasoning process, since the width of the window of time slices can be changed dynamically as well as the number of 'backward smoothing slices' and the number of 'forecast slices'. In addition, the scheme provides selective inference in the sense that inference can be performed in (i) the window, as (ii) forward propagation, as (ii) backward smoothing, or as (iii) forecasting.

Although we have presented a scheme for reasoning in dynamic networks, we have not addressed all relevant issues. Some of the most important yet unresolved issues are discussed in the following.

4.6.1 Utilization of independence relations induced by observations

The presented scheme is inflexible in the sense that the possibilities for reduction of space complexity imposed by observation of variables are not utilized. Put differently, observations may induce independence relations such that expansion of the time window may induce fewer fill edges and hence smaller belief universes. As demonstrated by the following example, utilization of these possibilities can substantially reduce the space requirements.

Consider the sample window expansion in Figure 4.5 and assume that evidence on variable h is available at each time slice. When evidence on a variable has been entered and a corresponding belief updating has taken place, the associated vertex and its incident edges may be removed without affecting the correctness of subsequent belief updatings. Therefore, assuming that all variables have equal state space sizes, an optimal elimination ordering is now h, g, f, e, b, c, d, a. Consequently, the triangulated graph corresponding to the initial time window covering time slice 0, contains only one fill edge; see Figure 4.10 More importantly, though, the removal of the edge between g and h implies that the number of fill edges of the triangulated graph of the expanded time window is reduced from 16 to 4, and the sum of the state space sizes of the belief universes is reduced from 192 to 96 if all variables are assumed to be binary. With increasing state space sizes of the variables, this size of the reduction becomes larger and larger. If for example the state space size is 5 for each variable, then a reduction by more than a factor of 10 is obtained. Note that, in Figure 4.10, variable h of time slice 0 cannot be removed until slice 1 has been added, since we need the associated evidence when calculating the belief potentials of the expanded window.

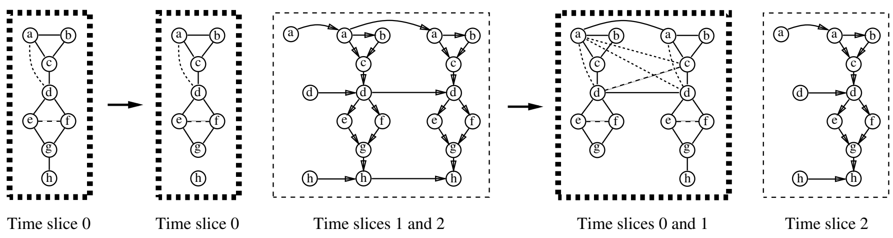

Figure 4.10: Utilization of independence relations induced by observing variable h at each time slice enables significant reduction of the space complexity.

A couple of interesting questions naturally arise. What is the criterion for obtaining significant reductions and does the utilization of evidence to reduce space requirements fit into our computational scheme?

First, if re-triangulation shall be avoided (i.e. finding another (more optimal) triangulation of an already triangulated graph), all evidence available on variables of time slice $t_N + w - 1$ (the newest time slice of the window) must be entered and propagated before time slice $t_N + w$ is added to the window. Thus only one time slice must be added at a time. Further, if $\mathcal{G}$ is the triangulated graph of the time window before expansion takes place, $\mathcal{G}'$ the triangulated graph of the expanded time window, and A the set of variables pertaining to the added time slice, then by Lemma 4.1 and the constrained elimination scheme it follows immediately that the complexity of $\mathcal{G}'_A$ is unaltered as long as $\mathcal{G}$ is connected. Thus, significant reductions of the space requirements are achieved only if $\mathcal{G}$ becomes disconnected.

Second, if $\mathcal{G}$ becomes disconnected, it is easily realized that the interfaces split into subsets, one for each connected component of $\mathcal{G}$. If, for example, g is observed instead of h in our example, then the original interface $\{a, c, d, g, h\}$ is split into subsets $\{a, c, d\}$ and $\{g, h\}$. Further, if window reduction is carried out such that the reduced window contains only time slice 1, then forward and backward propagation between the resulting models $\mathcal{P}_1$ and $\mathcal{P}_2$ would involve three junction trees $\mathcal{J}_{1a}$, $\mathcal{J}_{1b}$, and $\mathcal{J}_2$ ($\mathcal{P}_1$ consists of two (disconnected) junction trees $\mathcal{J}_{1a}$ and $\mathcal{J}_{1b}$), where $\mathcal{J}_2$ would contain two interface universes, one (containing $\{a, c, d\}$) used for transferral of information between $\mathcal{J}_2$ and $\mathcal{J}_{1a}$ and the other (containing $\{g, h\}$) for transferral of information between $\mathcal{J}_2$ and $\mathcal{J}_{1b}$. Thus, basically, this fits into our computational scheme. However, to serve as a platform for implementation, a slight revision is necessary.

4.6.2 Trading off time for reduction of space complexity

Provided a fixed set of variables is guaranteed to be observed at each time slice, the above method is a simple and effective means of reducing the space complexity. However, if different variables are observed at different time slices or anticipated observations are missing from time to time, this method cannot be used.

In Figure 4.5, the triangulated graph associated with initial model contains four fill edges even though the moral graph is already triangulated. As discussed in Section 4.3.2, this apparent waste of space is caused by the elimination ordering b, e, f, c, g, d, a, h which is optimal when considered in the perspective of an expanded window. However, if e.g. variable d is observed at time 0, we would be much better off using the decomposable cover given by the cliques of the moral graph and removing d before expanding the window. On the other hand, if e.g. b and e were observed, then the cover obtained by adding the four fill edges is preferable w.r.t. time complexity, but indifferent w.r.t. space complexity.

Thus, a scheme focusing on minimum-space complexity should use a myopic wait-and-see approach by producing locally optimal covers. The cost of such an approach appears as a possible need to re-triangulate the triangulated graph of the time window before expansion can take place.

4.6.3 Other issues

For some applications neither of the approaches discussed in the two preceding sections will be able to provide sufficient reductions of the space complexity. In such situations more drastic solutions must be addressed. One is to employ inference schemes based (partly) on stochastic simulation (i.e. Monte-Carlo methods). It is well-known, however, that the time complexity of such methods may be extremely high. If we stick to the junction tree approach for inference, we might be forced to accept approximations in the expansion and reduction processes. One obvious, but rather extreme, way to approximate the window expansion process is to assume that the variables of the window being parents of variables to be added to the window are conditional independent given the evidence available so far. An indication of the error hereby made could be calculated by employing e.g. information theoretic metrics providing measures of mutual information among the relevant variables. If we apply this approximate method on the sample window expansion in Figure 4.5, then no fill edges are needed at all, which implies a reduction of the space requirements from 192 to 76 for the expanded time window if all variables are binary. If the state space size is 5 for each variable, then a reduction by more than a factor of 20 is obtained.

Improved methods for forecasting is another field, where further work could be done. Due to the relatively large number of iterations of the forward sampling schemes that might be required for an acceptable precision, the time complexity of these and other Monte-Carlo methods can be fairly high. One alternative approach could be to apply the approximate window expansion method described above. Since no evidence is available for the variables of the forecast model, the approximation is likely to provide quite reliable forecasts. If some of the relevant conditional probabilities exhibit linearity in the sense that they are (approximately) linear functions in the variables upon which they are given, then a hybrid method might be advantageous. Such a hybrid method could be based on a combination of a Monte-Carlo method and computation of marginal distributions for variables with (approximately) linear conditional probability functions by marginalization over product tables of the conditional probability tables and vectors of marginal probabilities of the parents (computed by Monte-Carlo simulation or by this linear approximation method).

4.6.4 Implementation

An implementation of the computational scheme presented in this paper has been built on top of the HUGIN shell. The implementation is provided as an extension of the regular HUGIN API (application program interface) (Kjærulff 1993c). In addition, based on the extended API library, a menu-driven user interface has been built (Kjærulff 1993d). This interface provides facilities for defining a DBN (through specification of conditional time slice models, initial window width, number of forecast slices, etc.), moving the window forward and backward, dynamically changing the window width and the number of forecast and backward smoothing slices, updating beliefs in the window, performing backward and forward propagation, plus the usual facilities of inserting and retracting evidence and viewing beliefs of single variables.

Acknowledgements

I wish to thank Steffen L. Lauritzen for his valuable comments on an earlier draft of this paper and other members of the ODIN group at Aalborg University for stimulating discussions. The research has been funded partly by the Danish Research Councils through the PIFT programme.

Appendix A

User's Manual to `gt`

Preface

This manual provides a User's Guide to the `gt` program for (sub)optimal triangulation of undirected and directed, acyclic graphs. The program offers a variety of heuristic triangulation algorithms, a branch-and-bound algorithm, and an algorithm based a Monte-Carlo technique known as simulated annealing.

For a description of the terminology used and a detailed account of the algorithms and their properties, consult the relevant literature. A reference covering all the algorithms mentioned is Kjærulff (1990).

Section A.1 gives a synopsis of the program (i.e. it shows the fundamental ways to apply the program).

Section A.2 provides a more detailed description of the program and of the triangulation algorithms offered. Further, it provides a brief explanation of the options offered by the program.

Section A.3 describes the on-line help facility provided by the program.

Section A.4 describes the syntax of the program's input files.

Section A.5 provides a detailed description of all the optional arguments of the program.

A.1 Synopsis

The `gt` program allows two different kinds of input formats. The first format requires two input files, one describing the graph structure and the other the number of states of the variables represented by the nodes of the graph:

```
gt [-AaBbCDEeFGgHIiJLMoPpQRrSTtUVWwXZ] <graph_file> <card_file>
```

where `<graph_file>` contains a description of an undirected graph or a directed, acyclic graph (see the description of option `-M` in Section A.5 below), and where `<card_file>` contains a list of pairs of node names and integers specifying the number of states of the associated variable. The syntax of the graph and cardinality files are described in Section A.4 below.

The alternative input format requires only one input file:

```
gt -N [-AaBbCDEeFGgHIiJLoPpQRrSTtUVWwXZ] <net_spec_file>
```

where `<net_spec_file>` is a HUGIN net specification file (Fischer, Abrahamsen and Nielsen 1992).

A.2 Description

The gt program provides a large variety of facilities for achieving (sub)optimal graph triangulations. See e.g. Kjærulff (1990) for a definition of the optimality criterion employed. The triangulation of a graph may be formulated in terms of a node elimination ordering (Rose 1970). Thus the task of finding a good triangulation of a graph is formulated in terms of finding a good elimination ordering of the nodes. In general, the task of finding an optimal elimination ordering is NP-complete (Wen 1990). Therefore the gt program provides a Monte-Carlo algorithm which, with a probability arbitrarily close to 1, is able to find an optimal elimination ordering; see the description of option `-T` in Section A.5 below.

The program contains a number of different ordering algorithms:

- Maximum cardinality search (Tarjan and Yannakakis 1984).
- Minimum weight heuristic.
- Minimum size (degree) heuristic.
- Minimum fill (deficiency) heuristic.
- An exhaustive branch-and-bound algorithm.
- A Monte-Carlo algorithm based on simulated annealing.

These algorithms may furthermore be combined (or cascaded). See options `-P` and `-T` below.

The program also provides facilities for minimising triangulations (i.e. removing redundant fill edges).

The program accepts a myriad of options. All optional arguments are specified by single letters prefixed by '-' (hyphen). Capital letters generally applies to the main program, whereas lower case letters serves the annealing and branch-and-bound (B&B) algorithms. A brief description of the options follows:

Option	Used by (brief explanation)
A	Simulated annealing algorithm (inhibit reporting)
a	Simulated annealing algorithm (alpha)
B	Heuristic ordering algorithms (heuristic tiebreaking)
b	Simulated annealing and B&B algorithms (batch mode)
C	Minimal triangulation (minimise triangulation systematically)
D	Minimal triangulation (minimise triangulation randomly)
E	Main program (print node ordering to file)
e	Simulated annealing algorithm (epsilon)
F	Main program (print fill edges)
G	Main program (print graph)
g	Simulated annealing algorithm (gamma)
H	Produces this help message
I	Main program (number of iterations)
i	Sim. annealing and B&B algorithms (number of iterations)
J	Minimal triangulation (test for minimality)
L	Heuristic ordering algorithms (lookahead)
M	Main program (moral graph read from file)
N	Main program (read graph from Hugin spec-file)
o	Simulated annealing algorithm (tau)
P	Main program (pre-ordering algorithm)
p	Simulated annealing algorithm (report frequency)
Q	Main program (print resulting total state space on file)
R	Main program (report file)
r	Simulated annealing and B&B algorithms (report file)
S	Main program (print summary of resulting triangulation)
T	Main program (ordering algorithm)
t	Simulated annealing algorithm (initial temperature)
U	Heuristic ordering algorithms (heuristic strategy)
V	Main program (print graph weight on file)
W	Simulated annealing algorithm (max window size)
w	Simulated annealing algorithm (min window size)
X	Main program (exclude simplicials)
Z	Minimal triangulation (edge removal strategy)

A.3 On-line help facility

The gt program provides an on-line help facility. Three levels of detail are provided. The highest level is activated as

```
> gt
Usage:
       gt [-AaBbCDEeFGgHIiJLMoPpQRrSTtUVWwXZ] <graph_file> <card_file>
or
       gt -N [-AaBbCDEeFGgHIiJLoPpQRrSTtUVWwXZ] <net_spec_file>
If this summary doesn't provide sufficient help,
try "gt -H | more" for additional help.
>
```

The second level of help is activated as

```
> gt -H
This package provides a large variety of facilities for achieving
(sub)optimal graph triangulations.

The package contains a number of different ordering algorithms:
Maximum cardinality search,
minimum weight heuristic,
minimum size (degree) heuristic,
        :
        :
```

This help is quite similar to the description in the beginning of this section.

At the lowest level of help the user gets detailed help concerning the syntax and semantics of a specific option. For example, to get an explanation on option `-T` type

```
> gt -H T
Option T: (ordering algorithm)
---------

(The following applies for option -P as well.)
Provides a way of specifying the ordering algorithm (or file) to be
used. Option -T must be followed by one of the following:

   min-fill:  Minimum fill heuristic (i.e. successively chooses a
              node producing the least number of fill edges to be
        :
        :
```

A.4 Syntax of input files

The specification of an undirected graph must obey the syntax:

```
{<node> <neighbour>* <new_line>}*
```

If the graph is directed and acyclic the syntax is:

```
{<node> <child>* <new_line>}*
```

Example A.1 Consider the undirected graph given by the 4-cycle $\langle a, b, c, d, a \rangle$. The specification file could look like:

```
a b d
b a c
c b d
d a c
```

Note that this specification contains redundant information, since all edges are specified twice. □

Example A.2 Consider the directed, acyclic graph given by substituting the (undirected) edges a—b, b—c, c—d, a—d of the undirected graph of Example A.1 with the directed edges $a \rightarrow b$, $b \rightarrow c$, $c \leftarrow d$, $a \rightarrow d$. The specification file could look like:

```
a b d
b c
d c
```

Note that nodes with no children need not be specified. □

The syntax of a cardinality file (i.e. a file specifying the number of states of the nodes) is

```
{<node> <cardinality>}*
```

where `cardinality` is a positive integer.

Example A.3 A cardinality file for the graphs of Examples A.1 and A.2 could be

```
a 2 b 2
c 2 d 2
```

if all variables are binary. □

A.5 Options

This section provides a detailed description of all the optional arguments of `gt`.

-A **No reporting — simulated annealing** Inhibits reporting from the simulated annealing module (see option `-r`).

-a **Alpha — simulated annealing** The rate by which the temperature is lowered is controlled by the alpha and gamma parameters (see also option `-g`). A sensible alpha value could be 0.1.

-B **Tiebreak strategy** Provides a way of choosing the tiebreak strategy applied by the heuristic sorting algorithms (i.e. `min-weight`, `min-size`, or `min-fill`). When applying the option it must be followed by one of the symbols:

deterministic Ties are broken by selecting the most recent candidate in the conflict set.

arbitrary A uniformly distributed random variable is used to break ties. This is the default.

min-max-neighbour Ties are broken by selecting the node in the conflict set which has the smallest largest neighbour (in terms of the rank given by the search algorithm and the elimination method).

min-fill The node producing the least number of fill edges upon elimination is selected. *Note:* Has no effect when `-T min-fill` has been specified.

min-neighbours The node with the least number of neighbours is selected. *Note:* Has no effect when `-T min-size` has been specified.

min-weight The node which produces the clique with least weight when eliminated is selected. *Note:* Has no effect when `-T min-weight` has been specified.

Example A.4 The command

```
gt -T min-weight -B min-fill <graph_file> <card_file>
```

applies the minimum weight ordering heuristic and break ties by selecting a node producing the least number of fill edges among the candidates. □

Note: In practice it turns out that the choice of tiebreak strategy does not make much difference.

-b **Batch mode — simulated annealing and branch-and-bound** When running `gt` in the background, messages to `stdout` are not appropriate. To inhibit these messages use option `-b`.

-C **Minimise triangulation systematically** Provides a way of forcing the resulting triangulation to be minimal (i.e. no proper subset of the fill edges comprises a triangulation).

Example A.5 The command

```
gt -T max-card -C <graph_file> <card_file>
```

runs the maximum cardinality search ordering algorithm, tests the resulting triangulated graph for minimality, and if not minimal, finds a minimal proper subset of the triangulation (see also option `-D`). □

`-D` **Minimise triangulation randomly** Provides a way of forcing the resulting triangulation to be minimal (i.e. no proper subset of the fill edges comprises a triangulation). As opposed to systematic minimisation (see option `-C`) random minimisation takes no advantage of the order in which the fill edges were created and hence have running times which are significantly longer than those of systematic minimisation. On the other hand, randomness in selecting among several competing redundant edges may significantly improve the quality of the resulting triangulation.

Example A.6 The command

```
gt -T max-card -D <graph_file> <card_file>
```

runs the maximum cardinality search ordering algorithm, tests the resulting triangulated graph for minimality, and if not minimal, finds a minimal proper subset of the triangulation. □

`-E` **Print elimination ordering** Provides a way of having the elimination ordering corresponding to the resulting triangulation printed on a file (maybe for subsequent specification of an initial ordering for the annealing or the branch-and-bound algorithms, see option `-P`). If option `-E` is supplied it must be followed by the name of a (supposedly non-existing) file upon which the elimination ordering will be printed.

Example A.7 The command

```
gt -E ordering.init <graph_file> <card_file>
```

runs the default ordering algorithm and prints the elimination ordering on the file `ordering.init`. □

`-e` **Epsilon — simulated annealing** The stopping criterion for the simulated annealing algorithm can be specified either (indirectly) in terms of a lower temperature limit (i.e. a final temperature) or in terms of the number of iterations (see option `-i`). Specifying the stopping criterion in terms of an ε-value is equivalent to specifying a final temperature, t_f, since $\varepsilon \approx t_f * s$, where s is the logarithm of the size of the search space (i.e. the total number of configurations). An ε-value specifies the acceptable error on the final configuration (elimination ordering), $\#_f$. This error is defined as the difference between the cost of $\#_f$ and the cost of an optimal solution. Normally, however, the stopping criterion is given in terms of the number of iterations.

The default value for ε is determined by a default value of 1.0 for t_f.

`-F` **Print fill edges** Provides a way of having a list of all fill edges produced printed on the report file (see option `-R`).

Note: Option `-F` must be used in conjunction with option `-S`.

Example A.8 The command

```
gt -SF <graph_file> <card_file>
```

runs the default ordering algorithm and prints a summary including (but not restricted to) the fill edges. □

`-G` **Print graph** Provides a facility for printing the resulting triangulated graph on the report file (see option `-R`).

Example A.9 The command

```
gt -GR result <graph_file> <card_file>
```

runs the default ordering algorithm and prints a report including (but not restricted to) the resulting triangulated graph on file `result`. □

`-g` **Gamma — simulated annealing** The rate by which the temperature is lowered is controlled by the alpha and gamma parameters (see also option `-a`). A sensible γ-value is chosen such that the inequality

$$I < \frac{U \log |S|}{\gamma \varepsilon}$$

is satisfied, where I is the total number of iterations (see option `-i`), U is the maximum cost-function increase, S is the size of the search space (i.e. the total number of elimination orderings (the factorial of the number of nodes)), and ε is the acceptable difference between the cost of the final configuration (elimination ordering) and the cost of an optimal configuration (see option `-e`).

`-H` **Help** The command

```
gt -H | more
```

prints a couple of pages of help on `stdout` (standard output).

`-I` **Number of iterations** Provides a way of forcing a selected ordering algorithm to be run a number of times. Option `-I` must be followed by an integer specifying the number of times the algorithm should be run.

Example A.10 The command

```
gt -T min-fill -I 10 <graph_file> <card_file>
```

runs the minimum fill ordering heuristic 10 times. *Note:* This makes only sense when tiebreaking strategy `random` is applied (see option `-B`). □

`-i` **Number of iterations — simulated annealing** The positive integer value associated with the `-i` option determines the number of iterations of the simulated annealing algorithm (i.e. the number of configurations (elimination orderings) visited). Typically, the number of iterations must be several thousand.

Example A.11 The command

```
gt -P min-fill -T anneal -i 10000 -t 0 <graph_file> <card_file>
```

creates an initial elimination ordering for the annealing algorithm using the minimum fill heuristic (see option `-T`) and runs a strict down-hill search (`-t 0`) from that configuration. The search comprises 10000 iterations. □

`-J` **Test for minimality** Provides a way of testing minimality of the resulting triangulated graph.

Example A.12 The command

```
gt -T max-card -J <graph_file> <card_file>
```

runs the maximum cardinality search ordering algorithm and tests the resulting triangulated graph for minimality. □

`-L` **Lookahead depth** Heuristic search is based on selecting the least cost (minimal fill, minimal size, or minimal weight) vertex by looking ahead. The lookahead depth (default is 1) may be specified by option `-L`. Exhaustive heuristic search is given by maximal lookahead (see option `-U`).

Example A.13 The command

```
gt -T min-weight -L 3 <graph_file> <card_file>
```

runs the minimum weight heuristic with a lookahead of 3. □

Note: In practice, a lookahead of more than 3 – –5 is infeasible in most cases.

`-M` **Moral graph (undirected graph)** It is possible to specify an undirected graph to be triangulated. In that case option `-M` must be used to indicate that the graph is undirected (i.e. has already been moralized).

Example A.14 In

```
gt -M <graph_file> <card_file>
```

`graph_file` is treated as a specification of an undirected graph. □

`-N` **HUGIN net format** By applying this option the program assumes the graph specification file obeys the syntax of the HUGIN net language.

Example A.15 By specifying

```
gt -N AuntEmily
```

the underlying graph of the HUGIN net file `AuntEmily` is triangulated using the default triangulation method. □

`-o` **Tau — simulated annealing** The value associated with the `-o` option determines the rate by which the window width is decreased (see options `-w` and `-W`).

`-P` **Pre-ordering** Provides a facility for cascading two ordering algorithms (or to read an initial ordering from a file). The second ordering algorithm given by option `-T` starts (if possible) from the ordering given by the `-P` algorithm (or file). For details see option `-T`.

Example A.16 The command

```
gt -P min-fill -T anneal <graph_file> <card_file>
```

runs the minimum fill ordering heuristic and uses the resulting ordering as the initial configuration for the annealing algorithm. □

`-p` **Report frequency — simulated annealing and branch-and-bound** The value associated with the `-p` option determines how often status reports should be printed to the status file (see option `-r`). The value supplied equals the reciprocal of the frequency (in terms of number of iterations) by which a status report is given. The default value is 100.

`-Q` **Print resulting total state space size** If specified, prints the total state space size of the final triangulated graph on the file the name of which must follow `-Q`.

Example A.17 The command

```
gt -Q result <graph_file> <card_file>
```

runs the default ordering algorithm and prints the total state space size of the resulting triangulated graph on the file `result`. □

`-R` **Report file** The program produces some messages (partly controlled by the user) which are printed on a report file (`stdout` (standard ouput) is default). If supplied, option `-R` must be followed by the name of a (possibly non-existing) file upon which all output messages are printed (with error messages as the only exception).

Example A.18 The command

```
gt -R report <graph_file> <card_file>
```

runs the default ordering algorithm and prints messages to the user on file `report`. □

`-r` **Report file — simulated annealing and branch-and-bound** The simulated annealing and the branch-and-bound algorithms produce periodic messages to inform the user about the current state of affairs. These messages are printed on the file specified by the `-r` option. The default value of the report file is `stdout`.

Example A.19 The command

```
gt -T anneal -r anneal.rpt <graph_file> <card_file>
```

runs simulated annealing on the graph specified on `graph_file` and prints status reports on `anneal.rpt`. □

`-S` **Summary** Provides a way of getting some key figures (e.g. number of fill arcs and total state space size of the triangulated graph) concerning the resulting triangulated graph. Further, option `-S` must be supplied when option `-F` is supplied (but not vice versa).

Example A.20 The command

```
gt -S <graph_file> <card_file>
```

runs the default ordering algorithm and prints key figures on the report file (see option `-R`). □

`-T` **Ordering algorithm** (The following applies for option `-P` as well.) Provides a way of specifying the ordering algorithm (or file) to be used. Option `-T` must be followed by one of the following:

min-fill Minimum fill heuristic (i.e. successively chooses a node producing the least number of fill arcs to be eliminated next).

min-size Minimum size heuristic (i.e. successively chooses a node with the least number of neighbours to be eliminated next).

min-weight Minimum weight heuristic (i.e. successively chooses a node producing a clique with the least state space to be eliminated next).

max-card Maximum cardinality search (Tarjan and Yannakakis 1984).

rmax-card Recursive maximum cardinality search. Run maximum cardinality search on the triangulated graph produced by the most recent call to maximum cardinality search until a minimal triangulation results.

anneal Simulated annealing.

branch-and-bound Branch-and-bound algorithm.

random Random search using a pseudo random number generator.

none Apply ordering in which the nodes appear in the graph specification file.

`<file-name>` Apply the ordering provided by the sequence of node names in `file-name` (the first node is eliminated first, etc.). *Note:* Option `-X` is automatically applied in this case.

Example A.21 The command

```
gt -T my-ordering <graph_file> <card_file>
```

triangulates the graph on `graph_file` using the ordering given by the sequence of node names on file `my-ordering`. □

`-t` **Initial temperature — simulated annealing** The temperature parameter of the simulated annealing algorithm determines the probability by which a neighbouring configuration of higher energy is selected. The temperature is decreased exponentially towards a final temperature determined by e.g. the value supplied via the `-e` option. To run a session of proper simulated annealing (i.e. such that convergence towards a global minimum is guaranteed with a probability close to 1) the initial temperature (to be specified via the `-t` option) should large compared to the maximum cost-function (energy) increase. However, given a sensible start configuration, strict down-hill search (i.e. `-t 0`) often provide elimination orderings with costs close to the minimum cost.

Example A.22 The command

```
gt -P min-fill -T anneal -t 0 <graph_file> <card_file>
```

creates an initial elimination ordering for the annealing algorithm using the minimum fill heuristic (see option `-T`) and runs a strict down-hill search from that configuration. □

`-U` **Elimination strategy** Provides a way of selecting different elimination strategies when applying one of the three heuristic ordering algorithms (see option `-T`). `-U` must be followed by one of the symbols `simple` (default) or `mimimal` which means simple elimination and elimination as defined by Fujisawa and Orino (1974) (i.e. the resulting triangulation is guaranteed to be minimal), respectively.

`-V` **Print graph weight** If specified, prints the weight (base 2 logarithm to the total state space size — see option `-Q`) of the final triangulated graph on the file the name of which must follow `-V`.

Example A.23 The command

```
gt -V www <graph_file> <card_file>
```

triangulates the graph specified on file `graph_file` using the default triangulation algorithm and writes the weight of the resulting triangulated graph on file `www`. □

`-W` **Maximum window width — simulated annealing** The radius of the search space (i.e. the space of configurations (elimination orderings)) is an important parameter for the rate of convergence of the simulated annealing algorithm. If no knowledge is available as to the distance between the initial configuration and an optimal one, then the radius of the search space should be small initially (i.e. such that the distance between any two configurations is relatively small). When the current configuration is believed to be close to an optimal one, then a larger radius is preferable making the search process more fine grained.

The radius of the search space is controlled by constraining the perturbation scheme by imposing a *window* within which the perturbations must take place. The width of the window then indirectly determines the radius of the search space. The width of the window is decreased exponentially as the search process proceeds (see option `-o`).

The initial (maximum) window width is specified by option `-W` (see also option `-w`). The window width defines the maximum distance between the two vertices in the actual configuration (elimination ordering) which are to be interchanged. To get good performance the (maximum) window width should not be too small (i.e. < 3–5).

Example A.24 The command

```
gt -T anneal -W 20 <graph_file> <card_file>
```

runs simulated annealing on the graph specified by `graph_file` and with an initial (maximum) window width of 20. □

`-w` **Minimum window width — simulated annealing** The minimum distance between any two nodes to be interchanged in performing a perturbation of the current configuration (elimination ordering) of a simulated annealing session is specified by the `-w` option. See the description of option `-W` for further details. *Note:* The minimum window width is *not* altered during the course of an annealing session.

Example A.25 The command

```
gt -T anneal -W 20 -w 5 <graph_file> <card_file>
```

runs simulated annealing on the graph specified by `graph_file` and with an initial (maximum) window width of 20 and a minimum width of 5. □

- `-X` **Do not exclude simplicials** Simplicial vertices (recursive definition: vertices for which the adjacency set comprises a complete subgraph of a graph with all simplicial vertices removed) are normally removed before the actual search for a good elimination ordering is initiated. If for some reason the user do not want these vertices removed option `-X` should specified. The order in which simplicial vertices are eliminated might have little influence on the quality of the triangulation, but in most cases this influence is deemed to be insignificant.

- `-Z` **Edge selection strategy** When minimising a triangulation by removing redundant fill edges using a global searching approach (see options `-K` and `-L`) ties are normally broken by selecting an edge which reduces the total state space size the most, but by specifying option `-Z` random tiebreaking will be applied.

Appendix B

dHUGIN API Reference Manual

Preface

The "dHUGIN API 1.9 Reference Manual" provides you with a reference for the Application Program Interface to the dHUGIN inference engine. dHUGIN offers a set of new HUGIN API functions making it possible to implement dynamic Bayesian networks (DBNs). For a detailed description of the notion of DBNs, see Kjærulff (1993b).

This manual is intended as reference for programmers in the use of the dHUGIN API and the semantics of the functions provided. The reader is expected to be familiar with the HUGIN API Reference Manual (Fischer et al. 1992).

dHUGIN has been developed as part of a PhD-study conducted at Aalborg University. The implementation of dHUGIN is based on the HUGIN Expert Systems Shell which is proprietary software of Hugin Expert Ltd. The development of dHUGIN has become possible by the courtesy of Hugin Expert Ltd. who generously lent the author the HUGIN source code.

Section B.1 will provide a brief account of the notion of DBNs and introduce some terminology to be used throughout this manual.

Section B.2 provides a small guide to the inclusion of the dHUGIN API definitions and compilation of an application program. The section is almost entirely a copy of Section 2.2 of the HUGIN API Reference Manual.

Section B.3 gives a description of the domains of a dHUGIN DBN and the naming convention applied to the domain files of a DBN.

Section B.4 describes how a DBN is specified and initiated. Further, this section provides a description of the dHUGIN NET language.

Section B.5 explains how to retrieve information about a DBN.

Section B.6 shows how to expand, reduce and move the time window and how to change the number of time slices preceding and succeeding the time window. (The notion of 'time window' will be introduced in Section B.1.)

Section B.7 introduces functions for exact propagation of evidence in a DBN. Exact propagation takes place within the time window and between time slices not succeeding the time window.

Section B.8 describes methods for propagating evidence from the time window to time slices succeeding it. This operation is referred to as forecasting.

Section B.9 presents miscellaneous functions for e.g. retrieving strings indicating the version and revision of the actual dHUGIN API.

Section B.10 provides an alphabetical list of the functions provided by the dHUGIN API with brief descriptions of their function and references to the more detailed descriptions.

Section B.11 gives a list of the error codes used in the dHUGIN API and their meaning.

B.1 Introduction and terminology

Briefly, a dynamic model can be characterized as a sequence of submodels each of which represents the state of the dynamic system at a particular point or interval in time. In the context of Bayesian networks this means that a dynamic model consists of a series of, most often, structurally identical, subnetworks interconnected by temporal links. Each subnetwork represents a *time slice*. Time slice 0 is called the *initial time slice*. The time slices of a DBN are assumed to be chosen such that the DBN obeys the Markov property: the future is conditionally independent of the past given the present. This means that temporal links are only allowed between pairs of consecutive time slices.

Further, we shall assume that any dynamic system can be described in terms of an initial time slice plus a finite number, p, of submodels each representing a *period*, such that each submodel is applied in every p'th time slice. However, nothing prevents the user from dynamically changing the period length and updating the submodels.

Of course, only a limited number of (consecutive) time slices can be included in the junction tree(s) for the DBN. The most recent time slices of the DBN which are represented in a junction tree is referred to as the *time window* (or simply the *window*). At any point in time, the current (DBN) model consists of three distinct groups of time slices:

- b backward smoothing slices,
- w time window slices, and
- f forecasting slices;

see Figure B.1. In the following we let B denote the maximal number of backward

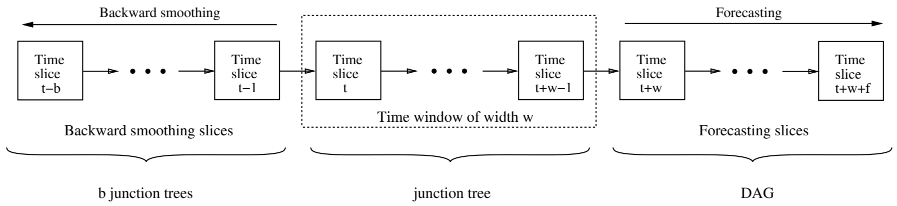

Figure B.1: A DBN consists of three distinct groups of time slices.

smoothing slices and we let b and f denote the actual number of backward smoothing and forecast slices, respectively.

The backward smoothing slices are those (consecutive) time slices of the current model which precedes the time window slices. These slices are represented in separate junction trees (domains) in such a way that information from the time window to the backward smoothing slices (and vice versa) can be transferred very easily. This transfer of information is referred to as *backward propagation* and *forward propagation*, respectively. Time slices $0, \ldots, t-b-1$ have been discarded meaning that we can neither enter observations pertaining to these slices nor perform a backward propagation (backward smoothing) including them.

The time window comprises a series of w consecutive time slices, where observations normally are entered and which is represented by a separate junction tree. Both the junction tree (domain) of the time window and those of the backward smoothing slices can be loaded into the regular HUGIN shell. Moving the time window involves the operations of *window expansion* and *window reduction*. That is, the current model is expanded by one or more new time slices and reduced it by zero or more of the oldest slices.

Window expansion by, say, k new time slices consists of (a) adding k new slices (partially specified Bayesian networks) to the current model, (b) moralizing the hybrid (i.e. both directed and undirected edges) composite graph of the triangulated graph of the window and the DAG of the k consecutive time slices succeeding the current time window, (c) triangulating that graph and identifying the new clique set, (d) constructing the new (expanded) junction tree, and (e) calibrating the new clique potentials (distributions) with appropriate consideration of the old ones. Expanding the window by k new time slices causes the width of the window to be increased by k, while the number of forecasting slices and backward smoothing slices remain unchanged.

Window reduction by, say, k time slices involves creation of k new domains representing the k oldest time slices of the window. That is, the k oldest slices of the window are 'cut off' and converted to backward smoothing slices. Some of the oldest backward smoothing slices may subsequently be discarded, since at most B backward smoothing slices are allowed. Reducing the window by k time slices causes a possible increase in the number of backward smoothing slices (by at most k) and the width of the window to be decreased by k, while the number of forecasting slices remains unchanged.

The forecasting slices are those (consecutive) time slices of the current model which succeed the time window. These slices are represented by a DAG and conditional probabilities, where there are directed (temporal) links from some time window variables to variables of the first forecasting slice. Forecasts may be generated by application of forward sampling techniques in the forecasting slices (see Section B.8).

The terms 'backward smoothing slices' and 'forecast slices' are here defined to mean the slices beyond the current time window for which, respectively, backward smoothing and forecasting are performed, although this is a bit misleading since backward smoothing and forecasting may also take place within the time window.

As described below, the actual numbers b, w and f of backward smoothing slices, time window slices, and forecasting slices, respectively, can be changed 'on the fly' as can the maximal number B of backward smoothing slices.

When running a dynamic model, HUGIN must in some cases behave in a way that is slightly different from its present (standard) behaviour. Thus to notify HUGIN that we want to run a DBN model rather than a BN model, the function

```
int dhugin_init (void)
```

must be called *before* any other dHUGIN API function.

B.2 Including the dHUGIN API definitions

In order to apply the dHUGIN API, the definitions of the dHUGIN functions must be included in the application program. This is done by inserting the following line at the top of the program source code:

```
#include <hugin.h>
```

The include file `hugin.h` contains the definitions of as well the HUGIN as the dHUGIN API functions.

When compiling the application program, the C compiler must be informed about the location of the include file and a compiler flag must be set indicating that the dHUGIN part of the inclusion file should be included as well. Assuming the HUGIN system (including the dHUGIN package) has been installed in the directory `/usr/local/hugin`, the following command is used:

```
cc -DDBN -I/usr/local/hugin/include -c myapp.c
```

where the flag `DBN` makes the dHUGIN API definitions be included and the argument `-I/usr/local/hugin/include` adds the `include` directory of the HUGIN system to the search path for include files.

This will compile the source code file `myapp.c` and store the result in the object code file `myapp.o`, without linking.

If the HUGIN system has been installed somewhere else, the above directory name must be modified as appropriate. If the environment variable `HUGINHOME` has been defined to point to the location of the HUGIN installation, the following command can be used

```
cc -I$HUGINHOME/include -c myapp.c
```

Using the environment variable, `HUGINHOME`, has the advantage that if the HUGIN system is moved, only the environment variable must be changed.

When the source code, possibly stored in several files, has been compiled, the object files must be linked to create an executable file. At this point it is necessary to specify that the object files should be linked with the HUGIN library:

```
cc myapp.o otherfile.o -L$HUGINHOME/lib -lhugin -lgnu -lm
```

The `-L$HUGINHOME/lib` specifies the directory to search for the HUGIN libraries, while the `-lhugin` specifies the library to link with. If the GNU C compiler is used, the `-lgnu` is not needed.

If the source code for the application is a single file, the above commands can be simplified to:

```
cc -I$HUGINHOME/include -c myapp.c
   -L$HUGINHOME/lib -lhugin -lgnu -lm
```

Note that the command should be typed as a single line.

To avoid typing such a complicated command each time compilation is required, the use of a `Makefile` is recommended.

B.3 Domains and files

As mentioned in the Introduction, any dHUGIN DBN consists of $b + 1$ HUGIN domains, where b is the actual number of backward smoothing slices. The first b domains each represents a single time slice, whereas the last one may represent more than one time slice and typically represents the time window.

The domain of the current time window has associated domain files (i.e. `.kb` and `.tables` files)

`<domain name>.kb` and `<domain name>.tables`

and each of the remaining b domains has domain files

`<domain name>.<i>.kb` and `<domain name>.<i>.tables`

if the domain represents a single time slice i and

`<domain name>.<i>-<j>.kb` and `<domain name>.<i>-<j>.tables`

if the domain represents time slices $i, \ldots, j$, where `<domain name>` is read from the net specification files.

Any of the dHUGIN domains can be loaded into the regular HUGIN shell.

B.4 Model specification and initialization

As mentioned above, it is assumed that any DBN can be defined in terms of a finite number, p say, of submodels which in the framework of HUGIN are given as net specification files for BNs. This suggests that the model repeats itself periodically, with p time slices being the length of each period. Even though this may be quite a reasonable assumption in most cases, we do not commit ourselves to strict obedience to this constraint. Suppose, for instance, that during the course of running a dynamic model some inadequacies of the applied model become apparent. In that case, nothing prevents us from modifying the future model quantitatively as well as qualitatively by revising the specification files and/or modifying the period length. Thus the concept of a 'period' should not be interpreted in its most literal meaning. The period length is set by the function

```
int h_period_length (int length)
```

where `length` is any positive integer.

The function returns 0 if it succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange The function argument is out of range.

herr_nomem Memory allocation failed.

Whenever the function `h_period_length` is activated, a pointer to a file containing a (partial) specification of a BN must be associated with the initial time slice and with each time slice of the repetitive sequence defining a period. Typically this assignment of file pointers are done just once. The assignment is handled by the function

```
int h_slice_net (FILE *net_file,
                 int slice_identifier)
```

where `slice_identifier` must belong to the set $\{0, 1, \ldots, \texttt{length}\}$, where `length` is the period length.

The function returns 0 if it succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange The second function argument is out of range.

Example B.1 Figure B.2 illustrates an example including a period of length five defined in terms of three specification files. It readily appears that there are variables in $\texttt{file}_1$ which have parents in both $\texttt{file}_0$ and $\texttt{file}_2$. Of course, these parents must be given identical names in $\texttt{file}_0$ and $\texttt{file}_2$. Their labels may be different, although this will probably rarely be reasonable since they represent the same set of variables only at different time points. Similarly, $\texttt{file}_2$ contains variables with parents in both $\texttt{file}_1$ and $\texttt{file}_3$, whereas parents of variables in $\texttt{file}_3$ which are not in $\texttt{file}_3$ must be found in $\texttt{file}_1$. □

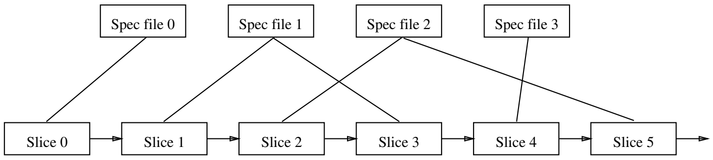

Figure B.2: A period of 5 time slices defined in terms of 3 specification files.

The current implementation of dHUGIN does not provide algorithms supporting automatic construction of elimination sequences for (sub)optimal triangulation of DBNs. Therefore the user is required to provide an elimination ordering for the initial time slice and each time slice of the period. This is done by the function

```
int h_slice_node_ordering (FILE *num_file,
                           int slice_identifier)
```

where `num_file` contains the names of the variables of the associated time slice in the order of elimination.

The function returns 0 if it succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange The second function argument is out of range.

Note: The function `h_period_length` *must* be called *before* calls to `h_slice_net` and `h_slice_node_ordering`.

Because a specification file can represent more than one time slice, time stamps should neither be attached to the node names nor to the node labels in a specification file. dHUGIN has a counter, T say, (initially equal to zero) attached to it, which is incremented by one whenever a new time slice is added. Thus, instead of specifying time stamps in the specification files, dHUGIN appends the string "␣⟨T⟩" to each node label, and the string "_⟨T⟩" to each node name.

Once the period length, net-file pointers, and pointers to node-ordering files have been successfully specified, the initial dynamic model may be created by means of the function

```
h_domain_t h_initiate_dynamic_model
```

```
(FILE *log_file,
 h_parse_error_handler error_handler,
 h_parse_error_handler progress_handler)
```

which reads the specified net files, creates a BN corresponding to the initial time window, compiles this network to obtain a junction tree, and, if the number of forecast slices is positive (see Section B.6.2 below), creates a BN representing the forecast slices in which forecasts can be derived using e.g. Monte-Carlo methods (see Section B.8 below).

The function returns a pointer to the time window domain if it succeeds and the null pointer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_model A dynamic model has already been initiated.

herr_undefmodel A dynamic model has not yet been specified.

herr_nomem Memory allocation failed.

herr_parse Syntax error in a net specification file.

herr_unknown_node_name An undefined node has been detected in a net specification file.

Example B.2 The sample code extract

```
#define eh (h_parse_error_handler) my_error_handler
#define ph (h_parse_error_handler) my_progress_handler
      :
if (h_period_length (n) != 0)
  error;

for (i = 0; i <= n; i++) {
  sprintf (slice_id, "%d", i);
  strcat (strcpy (file_name, domain_name), ".");
  strcat (file_name, slice_id);
  if ((file = fopen (file_name, "r")) == nil)
    fatal ("can't open file %s.", file_name);
  if (h_slice_cpn (file, i) != 0)
    error;
  strcat (strcpy (file_name, domain_name), ".ord-");
  strcat (file_name, slice_id);
  if ((file = fopen (file_name, "r")) == nil)
    fatal ("can't open file %s.", file_name);
  if (h_slice_node_ordering (file, i) != 0)
    error; }
```

```
if (h_initiate_dynamic_model (my_log_file, eh, ph) == nil)
  error;
     :
```

defines a period length of n, opens $n+1$ net specification files `<domain_name>.0`, `<domain_name>.1`, ..., `<domain_name>.n`, where `<domain_name>.0` is the net file of the initial time slice, opens $n+1$ node ordering files `<domain_name>.ord-0`, ..., `<domain_name>.ord-n`, and creates the initial dynamic model with the window width, the number of backward smoothing slices and the number of forecast slices equal to the default values. □

The function

```
int h_dynamic_model_initiated (void)
```

returns 1 if the dynamic model has been initiated; otherwise it returns 0.

B.4.1 The dHUGIN NET language

The dHUGIN NET language for DBNs is almost identical to the HUGIN NET language for BNs (see Chapter 5 in Fischer et al. (1992)). They differ in the following ways:

- A node, say v, of time slice t may have parents belonging to time slice $t-1$ and since time slices t and $t-1$ are specified in two distinct files, a node may be referred to in a file (e.g. as a parent of v) without actually being defined in that file.
- A node of time slice t may have parents in time slice $t-1$ with names identical to some of the nodes of slice t. To distinguish such nodes from one another, the name of the parents belonging to slice $t-1$ can be succeeded by a hyphen.

Example B.3 As an example consider the following DBN with period length $p = 1$ and each time slice, $t > 0$, looking like $X \to Y \to Z$, where Y has parents X and Y of time slice $t-1$. The (incomplete) specification files for this example looks like:

```
test.0:       test "test"     %% BN name and label
                100 40        %% Node width and height
                1.00  %% BN scale factor

              X "X"   %% Node name and label
              0 100   %% Node x and y positions
                      %% No parents
              ( ... )
```

```
            Y "Y"   %% Node name and label
            0 50    %% Node x and y positions
            X       %% Parents
            ( ... )

            Z "Z"   %% Node name and label
            0 0     %% Node x and y positions
            Y       %% Parents
            ( ... )


test.1:     test "test"     %% BN name and label
              100 40        %% Node width and height
              1.00  %% BN scale factor

            X "X"   %% Node name and label
            0 100   %% Node x and y positions
                    %% No parents
            ( ... )

            Y "Y"   %% Node name and label
            0 50    %% Node x and y positions
            X X- Y- %% Parents
            ( ... )

            Z "Z"   %% Node name and label
            0 0     %% Node x and y positions
            Y       %% Parents
            ( ... )
```

The specification file `test.0` defines the slice 0 model and specification file `test.1` defines the slice $1, \ldots$ models. In file `test.1` note that two of Y's parents are not defined in `test.1` and that we have to append a hyphen to the names of these two parents in order to distinguish them from variables X and Y defined in `test.1`. □

B.5 Retrieving model information

This section presents functions which provide information about the status of the current DBN. These functions are divided into three groups. The first group of functions provides general information, the second provides information about the time window, and the third provides domain specific information.

B.5.1 General information

The function

```
int h_number_of_backsmooth_slices (void)
```

returns the number of backward smoothing slices for which there is an associated domain. Thus, this is the *actual* number of backward smoothing slices, not the maximal number.

The function returns -1 if an error is detected. In that case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_number_of_forecast_slices (void)
```

returns the number of forecast time slices. If an error is detected, the function returns -1 and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_newest_time_slice (void)
```

returns the identifier associated with the newest time slice of the dynamic model.

Note: the newest time slice is the last one to which forecasts are made. Thus, the newest time slice is not necessarily identical to the newest time slice considered so far (this situation occurs when the maximal number of forecast slices is decreased; see Section B.6.2 below).

The function returns -1 if an error is detected. In that case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_oldest_time_slice (void)
```

returns the identifier associated with the oldest time slice of the dynamic model. If, however, an error is detected, then -1 is returned and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_node_time_slice (h_node_t node)
```

returns the identifier associated with the time slice containing `node`. If, however, an error is detected, then -1 is returned and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_nonode The argument is not a pointer to a node of an existing domain.

herr_not_time_slice_node The argument is not a pointer to a node belonging to any of the time slices of the current dynamic model.

B.5.2 Time window information

The function

```
int h_window_width (void)
```

returns the width of the time window (i.e. the number of time slices within the window).

Note: the window width is always 1 if the window has been moved backwards (i.e. the window comprises a single backward smoothing slice).

The function returns 0 if an error is detected. In that case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_oldest_slice_in_window (void)
```

returns the identifier associated with oldest time slice of the window. If, however, an error is detected, then -1 is returned and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_newest_slice_in_window (void)
```

returns the identifier associated with newest time slice of the window. If, however, an error is detected, then -1 is returned and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

B.5.3 Domain information

The function

```
h_domain_t h_time_window_domain (void)
```

returns a pointer to the time window domain if a dynamic model has been initiated and the null pointer if an error is detected in which case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The forecast model (i.e. the DAG comprising the forecast slices) is represented by a domain called the *forecast domain*. The function

```
h_domain_t h_forecast_domain (void)
```

returns a pointer to the forecast domain if a dynamic model has been initiated and the null pointer if an error is detected in which case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
h_domain_t h_time_slice_domain (int slice_id)
```

returns a pointer to the domain representing the time slice `slice_id`. If no such domain exists, then the null pointer is returned. If an error is detected, then the null pointer is also returned, but then `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
int h_time_slice_domain_exists (h_domain_t domain)
```

returns a positive integer if `domain` is associated with one of the time slices of the current dynamic model and 0 is this is not the case.

The function returns a negative integer if an error is detected. In that case `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

The function

```
char *h_domain_file_name (h_domain_t domain)
```

returns a pointer to a null-terminated array of characters containing the file name associated with `domain`.

Note: it is the responsibility of the the application programmer to free this array when not used anymore.

The function returns the null pointer if a dynamic model has not been initiated and `hugin_error` returns **herr_nomodel**

(This or a similar function ought to be provided by the regular HUGIN API.)

B.6 Model updating

Having finished the model specification and initialization, proper application of the dynamic model can take place. This involves the usual actions of entering evidence and making propagations, and, in addition, the operations of backward propagation, forward propagation, and forecasting (see Section B.7 below). Further, it involves updatings of the dynamic model in terms of moving the time window, changing the width of it, and changing the number of backward smoothing and forecast slices.

B.6.1 Expanding, reducing and moving the time window

Expansion and reduction of the time window is handled by the function

```
int h_set_time_window_width
    (int width,
     FILE *log_file,
     h_parse_error_handler error_handler,
     h_parse_error_handler progress_handler)
```

which sets the width of the window to `width` if possible. If `width` $> w$, where w is the current window width, then the dynamic model is expanded by $\Delta = |\texttt{width} - w|$ new time slices and the first Δ slices succeeding the current time window are added to it (including moralization, triangulation, etc.). If `width` $< w$, then the Δ oldest time slices of the window are cut off and possibly turned into separate domains. That is, window reduction creates Δ new domains representing the Δ newest backward smoothing slices. If, however, the resulting number of backward smoothing slices becomes greater than the maximal number, B, of backward smoothing slices (see Section B.6.2 below), then a sufficient number of the oldest slices are discarded such that at most B backward smoothing slices are present.

Messages from the HUGIN compiler are written on `log_file` unless it is the null pointer. If the supplied pointer to the error handler differs from the null pointer, any error occuring causes `error_handler` to be called with an error message as argument. Messages indicating the progress made are given as argument to the progress handler unless it is the null pointer.

Note: If the window comprises a single backward smoothing slice (i.e. it has been moved backward and has not yet been moved forward to its original position), the width of the window cannot be changed.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, hugin_error returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange The width argument is out of range. This happens either if width < 1 or the window comprises a backward smoothing slice.

herr_nomem Memory allocation failed.

herr_parse Syntax error in a net specification file.

herr_unknown_node_name An undefined node has been detected in a net specification file.

The function

```
int h_move_time_window
    (int step_size,
     FILE *log_file,
     h_parse_error_handler error_handler,
     h_parse_error_handler progress_handler)
```

causes the window to be moved step_size time slices backward (step_size < 0) or forward (step_size > 0).

Moving the window backward is basically a matter of changing the pointer to the time window domain. Hence, this is a very fast operation.

Moving the window forward may involve the operations of window expansion and window reduction (see above) and may hence be a time consuming operation.

Note: The domain files (i.e. .kb and .tables files) may be renamed to comply with the naming conventions described in Section B.3.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, hugin_error returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_outofrange There is fewer than step_size 'alive' time slices preceding the current time window.

herr_notfound A given domain is not the domain of a time slice of the current DBN.

herr_ioerr An input/output error occurred.

herr_nomem Memory allocation failed.

herr_parse Syntax error in a net specification file.

herr_unknown_node_name An undefined node has been detected in a net specification file.

B.6.2 Changing the number of backward smoothing and forecast slices

The maximal number of backward smoothing and forecast slices may be changed dynamically.

The function

```
int h_set_back_smooth_slices (int number_of_slices)
```

sets the maximal number of backward smoothing slices to `number_of_slices` if possible.

If `number_of_slices` $>$ b, where b is the current actual number of backward smoothing slices, then a simple reassignment of the maximal number, B, takes place.

If `number_of_slices` $<$ b, then the b $-$ `number_of_slices` oldest slices are discarded (i.e. the associated domains are freed and the domain files are deleted). *Note:* If the current time window represents one of the slices to be discarded, then a `h_move_time_window (k, nil, nil, nil)` is implicitly performed before the disposal takes place, where `k` is number of steps needed to reach the 'right-most' position of the window without performing the operations of window expansion and window reduction.

Advancing the time window will eventually result in an actual number of backward smoothing slices of `number_of_slices`.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange `number_of_slices` is negative.

herr_ioerr An input/output error occurred while renaming or deleting a domain file.

herr_nodomain The domain pointer of a time slice object to be deleted is the null pointer.

herr_nomem Memory allocation failed.

The function

```
int h_set_forecast_slices
    (int number_of_slices,
     h_parse_error_handler error_handler,
     h_parse_error_handler progress_handler)
```

sets the number of forecast slices to `number_of_slices` if possible.

If `number_of_slices` $<$ f, where f is the current number of forecast slices, then the number of forecast slices become `number_of_slices`. Note, however, that the f $-$ `number_of_slices` newest slices are not discarded. Thus, when new slices subsequently have to be added to the DBN, these virtually discarded slices re-enter the scene.

If `number_of_slices` $>$ f, `number_of_slices` $-$ f new forecast slices are added to the DBN.

If the supplied pointer to the error handler, `error_handler`, differs from the null pointer, any error occuring causes `error_handler` to be called with an error message as argument. Messages indicating the progress made are given as argument to the progress handler `progress_handler` unless it is the null pointer.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_outofrange `number_of_slices` is negative.

herr_ioerr An input error occurred.

herr_nomem Memory allocation failed.

herr_parse Syntax error in a net specification file.

herr_unknown_node_name An undefined node has been detected in a net specification file.

B.7 Propagation of evidence

This section describes three functions for performing exact propagation of evidence in and between junction trees of a DBN.

B.7.1 Propagation in time window

The function

```
int h_window_propagate (int *did_propagate)
```

performs a (sum-)propagation of evidence in `h_time_window_domain ()` and sets `*did_propagate` to a non-zero value if propagation actually takes place; otherwise it sets it to 0.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_ioerr An input error occurred.

herr_nomem Memory allocation failed.

B.7.2 Backward and forward propagation

The operations of backward and forward propagation deal with propagation of evidence between domains of consecutive time slices.

The function

```
int h_backward_propagate
    (int from_slice,
     int *did_propagate,
     h_parse_error_handler progress_handler)
```

propagates evidence from the domain corresponding to `from_slice` to domains corresponding to time slices `from_slice`$-1, \ldots$. Evidence is first propagated from `from_slice` to `from_slice` $-$ 1, then from `from_slice` $-$ 1 to `from_slice` $-$ 2, etc.

If at some point no new evidence gets transferred between consecutive slices or `h_oldest_time_slice ()` is reached, then the process terminates.

If evidence pertaining to variables of `from_slice` has been entered, then a normal propagation is executed first.

Note: the domains corresponding to time slices preceding `from_slice` are saved when evidence from succeeding slices has been absorbed, since possible subsequent evidence entered into these domains may cause their tables to be reloaded.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_outofrange `from_slice` is less then `h_oldest_time_slice ()` or greater than `h_newest_slice_in_window ()`.

herr_ioerr An input/output error occurred.

herr_nomem Memory allocation failed.

The function

```
int h_forward_propagate
    (int from_slice,
     int *did_propagate,
     h_parse_error_handler progress_handler)
```

propagates evidence from the domain corresponding to `from_slice` to domains corresponding to time slices `from_slice`$+1, \ldots$. Evidence is first propagated from `from_slice` to `from_slice` $+ 1$, then from `from_slice` $+ 1$ to `from_slice` $+ 2$, etc. If at some point no new evidence gets transferred between consecutive slices or `h_oldest_slice_in_window ()` is reached, then the process terminates.

If evidence pertaining to variables of `from_slice` has been entered, then a normal propagation is executed first.

Note: the domains corresponding to time slices succeeding `from_slice` are saved when evidence from preceding slices has been absorbed, since possible subsequent evidence entered into these domains may cause their tables to be reloaded.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_outofrange `from_slice` is less then `h_oldest_time_slice ()` or greater than `h_newest_slice_in_window ()`.

herr_ioerr An input/output error occurred.

herr_nomem Memory allocation failed.

B.8 Making forecasts

In dHUGIN terms, forecasting is the process of computing the distributions of variables pertaining to the forecast slices (i.e. the succeeding the time window when it is in its 'right-most' position).

Three forecast functions are provided. A common trait of these is that propagation is performed in a DAG structure rather than in a compiled junction-tree structure. This has the obvious advantage of reducing the space requirements to a minimum. Also the time complexity may be significantly lower using these forecast functions as opposed to advancing the time window the required number of time slices. Another trait common to these functions, is the assumption that all variables of the forecast slices are unobserved.

B.8.1 Linear forecasting

The function

```
int h_forecast_linear (h_parse_error_handler progress_handler)
```

implements a forecast method that is quite fruitful if (a subset of) the involved conditional probabilities exhibits linearity in the sense that they are (approximately) linear functions in the variables upon which they are given. That is,

$$p(x) \approx \sum_{y \in \mathrm{Sp}(\mathrm{pa}(\alpha))} p(x \mid y) \prod_{\beta \in \mathrm{pa}(\alpha)} p(y_\beta),$$

where $x \in \mathrm{Sp}(\alpha)$ is a state of a variable α and $\mathrm{pa}(\alpha)$ denotes the set of parents of α.

The method is then simply given by calculating all such approximate marginal probability distributions in an appropriate order (i.e. the distributions of all parents of a variable should be calculated before the distribution of the variable itself). Given that the divergence between such approximate distributions and the 'exact' ones are below an acceptable upper bound for the variables of interest, this is a very fast forecast method. The interesting point concerning the exactness of the method is that an upper bound on the error can be computed in advance by application of theorems of linear algebra. Calculation of such upper bounds are not provided by the current dHUGIN implementation.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_cyclic An appropriate forecast order does not exist due to a directed cycle.

herr_nomem Memory allocation failed.

B.8.2 Monte-Carlo forecasting

Concerning non-junction-tree methods (i.e. no triangulation) for forecasting, various Monte-Carlo sampling schemes may be useful. A common trait of these schemes is the fact that the variance of the resulting distributions can be made arbitrarily small. Note, however, that a reduction of the standard error of an estimator by a factor of k requires an increase in the sampling size, n, by around a factor of k^2 due to the ubiquitous $1/\sqrt{n}$ law of statistical variation. Thus, to get forecasts within a small distance from the 'exact' values, we should expect the computing time to be relatively large; in some cases even larger than those required by exact methods. Another important feature of sampling methods is that the time complexity grows only linearly in the dimensionality of the tables involved, whereas it grows exponentially for exact methods.

The following two Monte-Carlo sampling methods are provided for forecasting.

The function

```
int h_forecast_forward_sampling (int iterations)
```

provides forecasts computed by `iterations` iterations of the standard forward sampling method.

The variance of the estimates can be obtained by a call to `h_forecast_variance`; see below.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, hugin_error returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_no_forecast_slices The DBN does not contain any forecast slices.

herr_cyclic An appropriate forecast order does not exist due to a directed cycle.

herr_nomem Memory allocation failed.

The function

```
int h_forecast_fast_forward_sampling (int iterations)
```

is similar to h_forecast_forward_sampling except that the forecasts are given as averages over the sample distributions instead of sample frequencies. This gives a provably smaller variance (Kjærulff 1993b) and, since the complexity of the two algorithms are roughly identical, this algorithm is clearly preferable to plain forward sampling.

The variance of the estimates can be obtained by a call to h_forecast_variance; see below.

The function returns 0 if the operation succeeds and a positive integer if it fails. On failure, hugin_error returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_no_forecast_slices The DBN does not contain any forecast slices.

herr_cyclic An appropriate forecast order does not exist due to a directed cycle.

herr_nomem Memory allocation failed.

The function

```
h_number_t *h_forecast_variance (h_node_t node)
```

returns a k-dimensional vector of h_number_t's representing the variances of the estimated probabilities associated with the various states of node, where k is the number of states of node.

Note: The value of the variance vector depends on which Monte-Carlo forecast algorithm has been applied.

The function returns the null pointer if the operation fails, and `hugin_error` returns one of

herr_notdynmode Dynamic mode has not been selected.

herr_nomodel A dynamic model has not been initiated.

herr_nonode `node` equals the null pointer.

herr_not_time_slice_node `node` does not belong to the current DBN.

herr_not_forecast_node `node` does not belong to any of the forecast slices.

herr_noforecast No forecast has been calculated.

herr_nomem Memory allocation failed.

The function

```
int h_forecast_sample_size (void)
```

returns the number of iteration of the most recently applied forecast algorithm.

The function returns a negative integer if dynamic mode has not been selected. In that case `hugin_error` returns **herr_notdynmode**.

B.9 Miscellaneous

The function

```
char *dhugin_version (void)
```

returns the version number of the current dHUGIN API as a null-terminated string of characters as e.g. "1.9".

The function

```
char *dhugin_revision (void)
```

returns the revision number of the current dHUGIN API as a null-terminated string of characters as e.g. "1.1".

The function

```
char *dhugin_date (void)
```

returns the date of the current dHUGIN API as a null-terminated string of characters in the form yyyy/mm/dd.

The function

```
int dhugin_cleanup (void)
```

deletes all domain files pertaining to the current DBN except those of the current time window.

The function returns 0 if the operation succeeds and a positive integer if an i/o error occurs (`hugin_error` returns **herr_ioerr** in that case).

B.10 Alphabetical list of functions

This appendix provides an alphabetically sorted list of all the dHUGIN API functions. A brief description including a reference to the page containing its full description accompanies each function.

h_oldest_slice_in_window 155

Returns the identifier of the oldest time slice in the window.

h_oldest_time_slice 154

Returns the identifier of the oldest time slice of the dynamic model.

h_period_length 149

Sets the period length of the dynamic model to a specified number of time slices.

h_set_back_smooth_slices 160

Sets the maximal number of backward smoothing slices to a specified value.

h_set_forecast_slices 161

Sets the number of forecast slices to a specified value.

h_set_time_window_width 158

Sets the width of the time window to a specified number of time slices.

h_slice_net 149

Associates pointers to net specification files with the initial time slice and with each time slice of the period.

h_slice_node_ordering 150

Associates pointers to files containing node elimination orderings with the initial time slice and with each time slice of the period.

h_time_slice_domain 157

Returns a pointer to the domain associated with a specified time slice.

B.11 Errors

This section provides a list of all possible dHUGIN error codes returned by the `hugin_error` function. Please note that each function description in the above sections is followed by a list of error codes used by the function and the specific reasons for setting the error codes. For a detailed explanation of a specific error, refer to these descriptions.

Error code	Description
`herr_notdynmode`	Dynamic mode has not been selected. Use the function `dhugin_init` to select dynamic mode.
`herr_outofrange`	The value selected is out of the permitted range.
`herr_nomodel`	The dynamic model has not yet been initiated. Use the function `h_initiate_dynamic_model` to read the BN specification(s) and compile the initial time window.

Error code	Description
herr_undefmodel	A dynamic model has not yet been defined. Use the function h_period_length to set the size of a period, and then use the functions h_slice_cpn and h_slice_node_ordering to identify the files containing, respectively, specifications of the time slice models (initial model plus one for each time slice of a period) and corresponding node ordering files (specifying the node elimination order to be used when performing triangulation).
herr_model	A dynamic model has already been initiated.
herr_nonum	No node ordering provided. Use the function h_slice_node_ordering to specify the files containing the node ordering files (specifying the node elimination order to be used when performing triangulation).
herr_not_time_slice_node	The node does not belong to any of the time slices of the current dynamic model.
herr_unknown_node_name	Unknown node name. An undefined node has been detected in a net specification file.
herr_no_forecast_slices	No time slices succeed the current time window. Use the function h_set_forecast_slices to add time slices in which to perform forecasting.
herr_not_forecast_node	The node does not belong to any of the time slices succeeding the time window of the current dynamic model.
herr_noforecast	No forecast has been performed. Use one of the forecasting functions (e.g. h_forecast_fast_forward_sampling) to perform a forecast.
herr_cyclic	The directed graph induced by the time slices succeeding the time window is not acyclic.

Appendix C

User's Guide to `dhugin`

Preface

This document comprises a user's guide to the `dhugin` program which provides a menu-driven user interface to the dHUGIN system. The program is implemented entirely on top of the HUGIN and dHUGIN APIs (i.e. no other functions of the HUGIN system has been applied). The dHUGIN API extends the standard set of HUGIN API functions, making it possible to implement dynamic Bayesian networks (DBNs). A brief introduction to the notion of DBNs is given in Section C.1. For a detailed description of DBNs, see Kjærulff (1993b).

The document is intended as a relatively superficial guide to the use of the `dhugin` program, since large parts of the program is self-explanatory. Only those parts of the program the meaning or function of which are not necessarily evident have been included into this guide. The reader is expected to be familiar with fundamental concepts of Bayesian modelling.

`dhugin` and the dHUGIN packages have been developed as parts of a PhD-study conducted at Aalborg University.

C.1 Introduction and terminology

Briefly, a dynamic model can be characterized as a sequence of submodels each of which represents the state of the dynamic system at a particular point or interval in time. In the context of Bayesian networks this means that a dynamic model consists of a series of, most often, structurally identical, subnetworks interconnected by temporal links. Each subnetwork represents a *time slice*. Time slice 0 is called the *initial time slice*. The time slices of a DBN are assumed to be chosen such that the DBN obeys the Markov property: the future is conditionally independent of the past given the present. This means that temporal links are only allowed between pairs of consecutive time slices.

Further, we shall assume that any dynamic system can be described in terms of an initial time slice plus a finite number, p, of submodels each representing a *period*, such that each submodel is applied in every p'th time slice. However, nothing prevents the user from dynamically changing the period length and updating the submodels.

Of course, only a limited number of (consecutive) time slices can be included in the junction tree(s) for the DBN. The most recent time slices of the DBN which are represented in a junction tree is referred to as the *time window* (or simply the *window*). At any point in time, the current (DBN) model consists of three distinct groups of time slices:

- b backward smoothing slices,
- w time window slices, and
- f forecasting slices;

see Figure C.1. In the following we let B denote the maximal number of backward

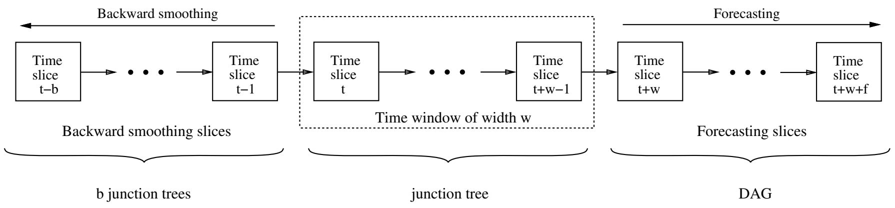

Figure C.1: A DBN consists of three distinct groups of time slices.

smoothing slices and we let b and f denote the actual number of backward smoothing and forecast slices, respectively.

The backward smoothing slices are those (consecutive) time slices of the current model which precedes the time window slices. These slices are represented in separate junction trees (domains) in such a way that information from the time window to the backward smoothing slices (and vice versa) can be transferred very easily. This transfer of information is referred to as *backward propagation* and *forward propagation*, respectively. Time slices $0, \ldots, t-b-1$ have been discarded meaning that we can neither enter observations pertaining to these slices nor perform a backward propagation (backward smoothing) including them.

The time window comprises a series of w consecutive time slices, where observations normally are entered and which is represented by a separate junction tree. Both the junction tree (domain) of the time window and those of the backward smoothing slices can be loaded into the regular HUGIN shell. Moving the time window involves the operations of *window expansion* and *window reduction*. That is, the current model is expanded by one or more new time slices and reduced it by zero or more of the oldest slices.

Window expansion by, say, k new time slices consists of (a) adding k new slices (partially specified Bayesian networks) to the current model, (b) moralizing the hybrid (i.e. both directed and undirected edges) composite graph of the triangulated graph of the window and the DAG of the k consecutive time slices succeeding the current time window, (c) triangulating that graph and identifying the new clique set, (d) constructing the new (expanded) junction tree, and (e) calibrating the new clique potentials (distributions) with appropriate consideration of the old ones. Expanding the window by k new time slices causes the width of the window to be increased by k, while the number of forecasting slices and backward smoothing slices remain unchanged.

Window reduction by, say, k time slices involves creation of k new domains representing the k oldest time slices of the window. That is, the k oldest slices of the window are 'cut off' and converted to backward smoothing slices. Some of the oldest backward smoothing slices may subsequently be discarded, since at most B backward smoothing slices are allowed. Reducing the window by k time slices causes a possible increase in the number of backward smoothing slices (by at most k) and the width of the window to be decreased by k, while the number of forecasting slices remains unchanged.

The forecasting slices are those (consecutive) time slices of the current model which succeed the time window. These slices are represented by a DAG and conditional probabilities, where there are directed (temporal) links from some time window variables to variables of the first forecasting slice. Forecasts may be generated by application of forward sampling techniques in the forecasting slices.

The terms 'backward smoothing slices' and 'forecast slices' are here defined to mean the slices beyond the current time window for which, respectively, backward smoothing and forecasting are performed, although this is a bit misleading since backward smoothing and forecasting may also take place within the time window.

C.2 Getting started

This section describes the various initial steps required to get dhugin running. These include: creation of model specification files, creation of node ordering files, and invocation of dhugin. Further, the section includes a subsection providing a brief explanation of the Main Menu of dhugin which appears right after the invocation.

C.2.1 Creating the specification files

Before being able to activate dhugin on a particular dynamic model, a specification of the dynamic model must be provided. This specification must be provided via two or more specification files.

As mentioned above, it is assumed that any DBN can be defined in terms of a finite number, p say, of submodels. This suggests that the model repeats itself periodically, with p time slices being the length of each period.[1] Each submodel (specifying a time slice) must be given via a specification file. That is, $p+1$ files must be supplied, one for the initial time slice (i.e. time slice 0) and one for each time slice of the repetitive sequence defining a period. (The period length, p, equals 1 in many applications.)

Note: By convention, the dhugin program expects the names of the specification files to obey the syntax

```
<domain_name>.<n>
```

where n identifies the associated time slice.

The specification files must obey the dHUGIN NET language, the syntax of which is described in the following.

[1]Even though this may be quite a reasonable assumption in most cases, we do not commit ourselves to strict obedience to this constraint. Suppose, for instance, that during the course of running a dynamic model some inadequacies of the applied model become apparent. In that case, nothing prevents us from modifying the future model quantitatively as well as qualitatively by revising the specification files and/or modifying the period length. Thus the concept of a 'period' should not be interpreted in its most literal meaning.

The dHUGIN NET language

The dHUGIN NET language for DBNs is almost identical to the HUGIN NET language for BNs (see Chapter 5 in Fischer et al. (1992)). They differ in the following ways:

- A node, say v, of time slice t may have parents belonging to time slice $t-1$ and since time slices t and $t-1$ are specified in two distinct files, a node may be referred to in a file (e.g. as a parent of v) without actually being defined in that file.
- A node of time slice t may have parents in time slice $t-1$ with names identical to some of the nodes of slice t. To distinguish such nodes from one another, the name of the parents belonging to slice $t-1$ can be succeeded by a hyphen.

Example C.1 As an example consider the following DBN with period length $p = 1$ and each time slice, $t > 0$, looking like $X \to Y \to Z$, where Y has parents X and Y of time slice $t-1$. The (incomplete) specification files for this example looks like:

```
test.0:       test "test"     %% BN name and label
                100 40        %% Node width and height
                1.00  %% BN scale factor

              X "X"   %% Node name and label
              0 100   %% Node x and y positions
                      %% No parents
              ( ... )

              Y "Y"   %% Node name and label
              0 50    %% Node x and y positions
              X       %% Parents
              ( ... )

              Z "Z"   %% Node name and label
              0 0     %% Node x and y positions
              Y       %% Parents
              ( ... )


test.1:       test "test"     %% BN name and label
                100 40        %% Node width and height
                1.00  %% BN scale factor

              X "X"   %% Node name and label
              0 100   %% Node x and y positions
                      %% No parents
              ( ... )
```

```
Y "Y"   %% Node name and label
0 50    %% Node x and y positions
X X- Y- %% Parents
( ... )

Z "Z"   %% Node name and label
0 0     %% Node x and y positions
Y       %% Parents
( ... )
```

The specification file `test.0` defines the slice 0 model and specification file `test.1` defines the slice $1, \ldots$ models. In file `test.1` note that two of Y's parents are not defined in `test.1` and that we have to append a hyphen to the names of these two parents in order to distinguish them from variables X and Y defined in `test.1`. □

C.2.2 Creation of node ordering files

The current version of the dHUGIN system does not support automatic methods for determining node elimination orderings which provides the foundation for creation of junction trees; see e.g. Kjærulff (1990) for a description of the notion of elimination orderings and triangulated graphs and Jensen, Olesen and Andersen (1990) or Jensen, Lauritzen and Olesen (1990) for an account on junction trees. Therefore, node elimination orderings must be provided via $p + 1$ files, where p is the period length. That is, each ordering file is associated with a given net specification file. The contents of a node ordering file is simply an ordered list of the names of the nodes (variables) of the associated time slice. The names must be separated by one or more white-space characters (i.e. `space`, `tab`, or `return`).

Note: As for net specification files, the `dhugin` program expects the names of the node ordering files to obey a certain syntax:

`<domain_name>.ord-<n>`

where, as before, n determines the time slice to which the ordering applies. The alternative suffix `num-<n>` is provided for consistency with the naming convention of HUGIN node ordering files.

Example C.2 Node ordering files, `test.ord-0` and `test.ord-1`, for the dynamic model of Example C.1 could be given as

```
test.ord-0:     X Y Z
test.ord-1:     Z X Y
```

implying that the variables will be eliminated in the order X, Y, Z for the initial time slice and in the order Z, X, Y for all the following time slices. □

C.2.3 Starting dhugin

Once the net specification files and node ordering files have been properly established, it is time for invoking `dhugin`. This is done by issuing the command

```
dhugin <domain_name>
```

where `domain_name` is the prefix of the specification and ordering files as described in Sections C.2.1 and C.2.2. The invocation causes nothing else to happen but the display of the Main Menu. That is, the specification and ordering files are not accessed at this point.

C.2.4 The Main Menu

Right after invocation of `dhugin`, the Main Menu appears as

```
                         MAIN MENU

  1) Settings
  2) Create initial window

  3) View beliefs
  4) Enter evidence
  5) Enter likelihood
  6) Retract evidence

  7) Propagate in window
  8) Backward propagate
  9) Forward propagate
 10) Forecast (fast forward sampling, 1000)

 11) Set window width
 12) Move window
 13) Save window

 14) Status
 15) Print
 16) Help

 17) Exit

Enter choice:
```

The menu items have been divided into groups of logically related items.

The first group, comprising items 1 and 2, deals with the definition and creation of the initial model. See Sections C.3 and C.4 for details.

The second group, comprising items 3–6, provides functions for viewing beliefs, entering evidence and retracting evidence. See Section C.5 for details.

The third group, comprising items 7–10, provides functions for making inference. The text in parentheses following the `Forecast` item reveals the currently selected forecast method and the number of samples if the method is based on a Monte-Carlo sampling scheme. See Section C.6 for details.

The fourth group, comprising items 11–13, provides functions for manipulating the time window. See Section C.7 for details.

The fifth group, comprising items 14–16, offers functions providing information about the current status the dynamic model, print facilities, and an on-line help facility. See Section C.8 for details.

To exit the program select the last item (17) of the Main Menu. Exiting the program causes domain files (i.e. the `.kb` and `.tables` files) not belonging to the current time window to be deleted.

C.3 Settings

Selection of item 1 of the Main Menu will activate the following submenu:

```
 SETTINGS MENU

  0) Exit to Main Menu
  1) Set period length
  2) Set initial window width
  3) Set # backward smoothing slices
  4) Set # forecast slices
  5) Select forecast algorithm
  6) Set sample size
  7) Set confidence level
  8) Select approximate updating

Enter choice [1]:
```

The default choice is 1 (indicated by `[1]` in the prompt). The period length must be specified (i.e. there is no default value of the period length). The specification of the period length causes the net specification and node ordering files to be opened.

Note: Menu item 2 can only be used for specifying the initial width of the time window. That is, once the initial model has been created, item 11 of the Main Menu must be used to change the width of the window.

Menu item 5 provides a submenu of various forecast methods, and if a method based on Monte-Carlo sampling is chosen, the sample size can be selected via menu item 6.

Variances on the beliefs of variables computed by applying one of the Monte-Carlo forecast methods can be computed and — on the basis of the variances, information about the method applied, the sample size, and a user-defined confidence level — a confidence interval can be computed. This interval is displayed when viewing beliefs based on Monte-Carlo forecasting (see Section C.5 below). The default confidence level can be changed through selection of menu item 7.

Menu item 8 provides a handle for toggling between exact and approximate expansion of the time window. Approximate expansion is given by assuming marginal independence among all variables of the newest time slice of the current window which have children belonging to the following time slice, T say. That is, information from the time window contributing to the calculation of belief potentials associated with slice T are given in the form of marginal probabilities of these parent variables.

The advantage of running dHUGIN in the approximate expansion mode is the vast reduction of space and time complexity that might be obtained.

Note: In the approximate expansion mode, the window width is fixed at the value of one and no backward smoothing is possible. The number of backward smoothing slices is set to zero and should be reset (if wanted) when/if normal mode of operation is subsequently chosen.

Warning: No indication of the error made by the approximate expansion scheme is provided, and application of this mode of operation should be made with great care and reflection!

C.4 Creating the initial model

Once the initial model has been specified through specification of the period length, window width (optional), number of forecast slices (optional), etc. (see Section C.3), the initial model can be created by selecting item 2 of the Main Menu. This involves creation of a Bayesian network for the initial time window and, if the number of forecast slices is positive, a Bayesian network for the forecast slices. The networks are created on the basis of the specification files (see

Section C.2.1). The network for the time window is compiled into a junction tree involving moralization, reading the node orderings provided (see Section C.2.2), triangulation, construction of the junction tree, and calculation of an initial set of consistent belief potentials (via propagation).

A progress report is printed to inform the user about the actual status of the compilation process (see Example C.3).

Example C.3 The progress report printed when creating the initial model appears as

```
     :

Enter choice: 2

Reading CPN specification ... Moralising graph ... Reading node ordering .
.. Triangulating graph ... Creating cliques ... Building junction tree ...
 Propagating ... Saving time window model ...  done (0.85 sec)

Type RETURN to continue:
```

where the time of the operation is indicated in parentheses. □

Having sucessfully completed the creation of the initial model, proper application of the dynamic model may take place. This includes the usual processes of entering evidence (see Section C.5), propagation of evidence (within the time window) (see Section C.6), etc. In addition it involves moving the time window forward (or backward), changing the width of the window (see Section C.7), making forecasts, and propagating information backward and forward (i.e. transferring information between time slices) (see Section C.6).

C.5 Viewing beliefs, entering evidence and retracting evidence

C.5.1 Viewing beliefs

To see the marginal distributions of the individual variables, item 3 of the Main Menu should be selected. Right after the selection, you get the prompt

```
Select relevant time slice:
```

As answer, enter the index of the time slice (of the current dynamic model) for which you want to see the beliefs.

Note: If evidence has been entered, you should probably perform a propagation before viewing the beliefs. If the time slice selected is a forecast slice, be sure to perform a forecast operation first (and a propagation of evidence before the forecast operation if evidence has been entered but not propagated). See Section C.6 for details on propagation and forecasting.

Example C.4 Assume that item 3 of the Main Menu and time slice 1 have been selected. Then a submenu of all variables of time slice 1 gets displayed:

```
Select a node:

  0) Exit to Main Menu
  1) H 1
  2) G 1
  3) F 1
  4) E 1
  5) D 1
  6) C 1
  7) B 1
  8) A 1

Enter choice [1]:
Beliefs for node "H 1":
          h0: 0.557072
          h1: 0.442928

Enter choice [2]:
```

The belief vector for each selected variable gets displayed with the name of each state of the variable followed by the probability attached. □

Example C.5 As in Example C.4, assume that item 3 of the Main Menu has been selected. If time slice 2 is a forecast slice (of the current dynamic model), then we get the following when selecting that slice (forecast performed by the 'fast forward sampling' method):

```
NB: The node beliefs printed below are assumed to be produced by
    fast forward sampling with a sample size of 1000.  The first
    number appearing in the parenthesis following an estimated
    belief indicates the confidence interval for the estimate,
    and the second the confidence level assigned to that interval.
    The third number is the sample variance obtained by plain
    forward sampling, and the fourth is the sample variance obtained
    by fast forward sampling.  The ratio between these two variances
    (the fifth number) is the factor by which we should expect to
    increase the sample size in order to achieve a precision similar
    to the one indicated using plain forward sampling.

Select a node:

  0) Exit to Main Menu
```

```
  1) H 2
  2) G 2
  3) F 2
  4) E 2
  5) D 2
  6) C 2
  7) B 2
  8) A 2

Enter choice [1]:
Beliefs for node "H 2":
          h0: 0.519827  (+-0.0075 at level 0.95 -- 0.000250/0.000013 = 18.53)
          h1: 0.480173  (+-0.0075 at level 0.95 -- 0.000250/0.000013 = 18.53)

Enter choice [2]:
```

where the interpretation of the numbers appears from the printed text. □

C.5.2 Entering and retracting evidence

Entering evidence (likelihood) for variables of the time window is done by activating menu item 4 (5).

Right after the selection, you get the prompt

```
Select relevant time slice:
```

As answer, enter the index of the time slice of the current dynamic model for which you want to enter evidence (likelihood).

Example C.6 Assume that item 4 of the Main Menu and time slice 0 have been selected. Then a submenu of all variables of time slice 0 gets displayed:

```
Enter evidence:

  0) Exit to Main Menu
  1) H 0
  2) G 0
  3) F 0
  4) E 0
  5) D 0
  6) C 0
  7) B 0
  8) A 0

Enter choice [0]: 1
Select a state of node "H 0":
  1) h0
  2) h1
```

```
Enter choice: 2
State h1 selected for node "H 0"

Enter choice [0]:
```

We have selected state `h1` of variable `"H 0"`. That is, the belief vector $(0, 1)$ has been imposed on variable `"H 0"`. □

Retraction of evidence (selected via item 6 of the Main Menu) appears with an interface similar to that of entering evidence.

Note: Evidence can only be entered into variables included in the current time window.

C.6 Making inference

We distinguish between four different inference procedures.

C.6.1 Propagation the in time window

To get beliefs (probabilities on variables) consistent with the evidence (likelihood) entered (see Section C.5), propagation in the time window must be performed. This is done by selecting item 7 of the Main Menu.

C.6.2 Backward propagation

If there is a positive number of backward smoothing slices (see Section C.3 for details on how to set this number), information can be propagated backward from the time window to time slices preceding it (if any). The operation of backward propagation is also sometimes referred to as 'backward smoothing'. The operation is invoked by selecting item 8 of the Main Menu.

Note: Make sure that possible evidence pertaining to time slices preceding the time window has been propagated forward before backward propagation is performed. Otherwise, inconsistency may result.

C.6.3 Forward propagation

If the time window has been moved farther to the 'right' (time increasing from 'left to right') than its current position, then information can be propagated forward from the time window to the time slices succeeding it. The operation is invoked by selecting item 9 of the Main Menu.

Note: Only time slices, which have at some point been part of the time window, can participate in forward propagation. That is, propagation to the forecast slices cannot be performed in this way (see Section C.6.4).

Note: Make sure that possible evidence pertaining to time slices succeeding the time window has been propagated backward before forward propagation is performed. Otherwise, inconsistency may result.

C.6.4 Forecasting

The dHUGIN system provides three methods for making forecasts (i.e. calculation of beliefs for variables of the forecast slices) all of which can be selected via the `dhugin` interface. The actual method is selected via item 1 of the Main Menu and the text in parentheses following the text `Forecast` of item 10 reveals the currently selected method and the number of samples if the method is based on a Monte-Carlo sampling scheme. See Kjærulff (1993b) for a detailed description of the Monte-Carlo methods.

Forecasting is performed in a DAG structure rather than in the a junction tree. This has the obvious advantage of reducing the space requirements to a minimum. Also the time complexity may be significantly lower using these forecast functions as opposed to moving the time window the required number of time slices. Another trait common to these functions, is the assumption that all variables of the forecast slices are unobserved.

Select item 10 of the Main Menu to perform forecasting.

Linear forecasting

This function implements a forecast method that is quite fruitful if (a subset of) the conditional probabilities of a probabilistic model exhibits linearity in the sense that they are (approximately) linear functions in the variables upon which they are given. That is, if

$$p(x_a) \approx \sum_{y \in Sp(pa(a))} p(x_a \mid y) \prod p(y)$$

where $x_a \in Sp(a)$ and $pa(a)$ denotes the set of parents of a. The method is then simply given by calculating all such approximate marginal probability distributions in an appropriate order (i.e. the distributions of all parents of a variable should be calculated before the distribution of the variable itself). Given that the deviation between such approximate distributions and the 'exact' ones are below an acceptable upper bound for the variables of interest, this is a very fast forecast method.

Warning: No indication of the precision of the forecasts provided by the 'linear approximation' method is provided, and application of this method should be done with great care and reflection!

Monte-Carlo forecasting

Concerning non-junction-tree methods for forecasting, various Monte-Carlo sampling schemes may be useful. A common trait of these schemes is the fact that the variance of the resulting distributions can be made arbitrarily small. Note, however, that a reduction of the standard error of an estimator by a factor of k requires an increase in the sample size, n, by around a factor of k^2 due to the ubiquitous $1/\sqrt{n}$ law of statistical variation. Thus, to get forecasts within a small distance from the 'exact' values, we should expect the computing time to be relatively large; in some cases even larger than those required by exact methods.

The following two Monte-Carlo sampling methods are provided for forecasting.

Forward sampling In this method a state x_i of a variable x of a forecast slice is assigned the probability $p(x_i)$ which is proportional to the number of times the random sampling from the state space of X resulted in x_i. In sampling from the state space of X in sample number k, x_i is selected with probability $p(x_i \mid y, \ldots, z)$, where $y, \ldots, z$ are the states selected (in sample number k) for the parents of X.

Fast forward sampling This method is similar to forward sampling except that the forecasted probability distributions are derived by averaging over the vectors $p(x \mid y, \ldots, z)$, where the above notation applies.

Note: The use of the 'fast forward sampling' method is recommended. (The plain 'forward sampling' method will probably not be offered in the next version of the dHUGIN system, since it has no advantages over the 'fast forward sampling' method whatsoever.)

C.7 Manipulating the time window

The time window defines an interval of consecutive time slices for which evidence (or likelihoods) can be entered, retracted, and propagated. The width of the window can be changed if the window is in its 'right-most position'. That is, when the time slice immediately succeeding the window is a forecast slice. When the window is not in its right-most position, its width is fixed to one. As already indicated, the time window can be moved both forward and backward.

C.7.1 Changing the width of the window

The width of the time window is altered via selection of item 11 of the Main Menu.

Increasing the window width

Expanding the time window by, say, k new time slices consists of (a) adding k new slices to the current window, (b) moralizing the hybrid (i.e. both directed and undirected edges) composite graph of the triangulated graph of the window and the DAG of the k consecutive time slices succeeding the current time window, (c) triangulating that graph and identifying the new clique set, (d) constructing the new (expanded) junction tree, and (e) calculating the new clique potentials with appropriate consideration of the old ones. Expanding the current model by k new time slices causes the width of the window to be increased by k, while the number of forecasting slices remains unchanged.

Example C.7 Assume we have a model including a window of three time slices and want to expand the window to include five slices instead, we illustrate this as follows:

```
Before expansion:      2 3 [4 5 6] 7 8 9 10

After expansion:       2 3 [4 5 6 7 8] 9 10 11 12
```

Note that the number of forecast slices remain unchanged. □

Reducing the window width

Reducing the window width by, say, k time slices creates k new HUGIN domains each representing a backward smoothing slice. Backward smoothing slices are disposed of as appropriate when new slices get added to the collection such that no more than the specified maximal number of backward smoothing slices exist. Reducing the width of the window by k time slices causes the number of backward smoothing slices to be increased by at most k and the width of the window to be decreased by k, while the number of forecasting slices remains unchanged.

Example C.8 Assume we have a model including a window of five time slices and want to reduce the window by two slices we illustrate this as follows:

```
Before reduction:      2 3 [4 5 6 7 8] 9 10 11 12

After reduction:           4 5 [6 7 8] 9 10 11 12
```

Here the maximum number of backward smoothing slices has been set to two. Thus, time slices 2 and 3 get discarded. □

C.7.2 Moving the window

The time window may be moved either forward or backward. Moving it backward implies that the new window will comprise a single backward smoothing slice and that the width of the window is changed to one, since there is only one time slice associated each of the backward smoothing domains. Please note that moving the window backward does not imply that information gets transferred backward accordingly; thus, if a backward smoothing operation should be performed, backward propagation must be done explicitly (see Section C.6).

Moving the window forward is similar to moving it backward except when it involves addition of new slices to the model.

The time window is moved using the item 12 of the Main Menu.

C.7.3 Saving the window

Whenever a new time window is created (i.e. when the window is moved or resized), a `.kb` file and a `.tables` file is saved in the directory where the specification and node ordering files are located. These files are named <domain_name>.kb and <domain_name>.tables and can be loaded into the HUGIN shell as an ordinary (static) domain, where <domain_name> is the command-line argument given to `dhugin` upon invocation (see Section C.2.3).

Moreover, when performing backward (forward) propagation (see Section C.6) the domains corresponding to time slices preceding (succeeding) the time slice from which backward (forward) propagation is performed need to be saved once evidence has been absorbed, since possible subsequent evidence (or likelihoods) entered into these domains may cause their `.tables` file to be reloaded.

Apart from these automatic saving procedures, the user is free to save the time window domain at any point. This can be done by selecting item 13 of the Main Menu.

C.8 Miscellaneous

C.8.1 Status of the current dynamic model

Once the initial model has been created (see Section C.4), the appearance of item 14 of the Main Menu alters. It then provides information about the actual number of backward smoothing slices, the width and location of the time window, and the number of forecast slices.

Example C.9 Assume an initial model, comprising a time window of width 2 and four forecast slices, has been created. Then item 14 of the Main Menu appears as

```
14) Status: [0 1] 2 3 4 5
```

where the square brackets indicate the time window. Assume now that 3 has been selected as the maximal number of backward smoothing slices and that the time window is moved two time slices forward. The status item of the Main Menu then appears as

```
14) Status: 0 1 [2 3] 4 5 6 7
```

Moving the window two time slices forward once more makes the status item appear as

```
14) Status: 1 2 3 [4 5] 6 7 8 9
```

We observe that the initial time slice has now been discarded, since 3 has been selected as the maximal number of backward smoothing slices. Now, moving the window two time slices backward makes the status item appear as

```
14) Status: 1 [2] 3 (4 5) 6 7 8 9
```

showing that the time window now comprises time slice 2 only and that the 'right-most' position of the time window (without compiling new time slices) is given by time slices 4 and 5. □

By selecting item 14 of the Main Menu, a more detailed status report is provided. This include information about the name of the actual domain, the period length, the window width, the number of backward smoothing and forecast slices, etc.

C.8.2 Printing facilities

By selecting item 15 of the Main Menu, the following submenu appears:

```
                              PRINT MENU

  0) Exit to Main Menu
  1) Print window beliefs
  2) Print forecasted beliefs
  3) Print window domain
  4) Print forecast domain

Enter choice [0]:
```

These subitems offer the following services:

1. The beliefs of all variables of the time window are printed on a specified file in the same format as that of the viewing facility provided through item 3 of the Main Menu.
2. The beliefs of all variables of the forecast slices are printed on a specified file in the same format as in 1.
3. The values of the node attributes (e.g. states, neighbours, associated tables), the clique attributes (e.g. table size, members, neighbours), and the table attributes (e.g. kind, rank, associated nodes) of the current window domain are printed on a specified file. This facility is for debugging purposes.
4. The values of the node attributes (e.g. states, neighbours, associated tables) and the table attributes (e.g. kind, rank, associated nodes) of the current forecast domain are printed on a specified file. This facility is for debugging purposes.

C.8.3 On-line help

The `dhugin` program offers an on-line help facility which is invoked by selecting item 16 of the Main Menu. The help facility offers a submenu of 10 different issues. Try it.

Appendix D

Dansk resumé

Dette resumé af nærværende afhandling er den danske oversættelse af indledningen til afhandling.

Anvendelse of sandsynlighedsteori til kvantitativ modellering af videndomæner omfattende komplekse stokastiske fænomener, blev indtil starten af 1980'erne, betragtet som værende uhensigtsmæssig. Specielt eksisterede der ingen beregningsmæssigt egnede inferensmetoder. Metoder til håndtering af usikkerhed, som ikke var baseret på sandsynlighedsteori, blev derfor udviklet. To af de mest velkendte eksempler på sådanne metoder er 'certainty factors' i regelbaserede systemer (Shortliffe and Buchanan 1975) og fuzzy logik (Zadeh 1983).

Modellering af usikkerhed baseret på sandsynlighedsteori har imidlertid opnået en bemærkelsesværdig renæssance, som påbegyndtes i starten af 1980'erne. To faktorer spiller en central rolle for dette succesfulde comeback. For det første, har den gren af statistisk modellering, hvor fokusering på den kvalitative struktur er det fremherskende træk, bidraget med grafiske metoder til at udtrykke betingede uafhængighedsegenskaber (Darroch et al. 1980, Wermuth and Lauritzen 1983). I sådanne grafiske modeller er de (stokastiske) variable repræsenteret som knuder i en graf, og de statistiske afhængigheder er repræsenteret som kanter i grafen. For det andet bliver løsningen af inferensproblemet beregningsmæssigt muligt for en stor klasse af ikke-trivielle modeller. Dette sker som en konsekvens af udnyttelsen af de betingede uafhængighedsegenskaber ved de grafiske modeller.

Et antal forskellige metoder til udførelse af inferens i grafiske modeller er blevet foreslået og realiseret (implementeret), og nogle af disse er med succes blevet anvendt til løsning af realistiske problemer, som involverede flere hundrede variable. Den kendsgerning, at de fleste grafiske modeller relateret til disciplinen 'knowledge engineering' (konstruktion af videnbaserede systemer), er baseret på orienterede, acykliske grafer, er reflekteret i udbudet af metoder. Metoderne omfatter propageringsteknikker (beregning af opdaterede fordelinger på baggrund af observation af en eller flere variable) baseret på den oprindelige orienterede graf (Kim and Pearl 1983), propageringsteknikker baseret på en relateret, ikke-orienteret graf (Cannings et al. 1976, Cannings et al. 1978, Lauritzen and Spiegelhalter 1988, Jensen, Olesen and Andersen 1990, Jensen, Lauritzen and Olesen 1990, Shafer and Shenoy 1990), propagering i en anden, men relateret, orienteret graf (Cooper 1990a, D'Ambrosio 1989, Shachter et al. 1990), betingning (conditioning) (Pearl 1986a), og knudereduktion (Shachter 1988, Shachter 1990). Bortset fra propageringsteknikken af Kim and Pearl (1983), er disse metoder fundamentalt set ækvivalente, idet de udfører ensartede operationer på ensartede ikke-orienterede grafer (Shachter et al. 1991).

Blandt statistikere kaldes grafiske modeller, baseret på orienterede, acykliske grafer, ofte rekursive grafiske modeller (Wermuth and Lauritzen 1983). Med tilvejebringelsen af effektive inferensmetoder, er dette sprog til videnrepræsentation blevet stadig mere populær indenfor konstruktion af videnbaserede systemer (ekspertsystemer). I denne sammenhæng benævnes denne type modeller dog anderledes. De mest velkendte benævnelser er (med de engelske benævnelser i parentes) influensdiagram (influence diagram) (Howard and Matheson 1984), kausalt sandsynlighedsnet (probabilistic causal network / causal probabilistic network) (Cooper 1984, Andreassen et al. 1987), Bayesiansk 'belief' net (Bayesian belief network) (Pearl 1986b), kausalt net (causal network) (Lauritzen and Spiegelhalter 1988). I nærværende afhandling benyttes benævnelsen Bayesiansk net (Bayesian network).

Den underliggende graf for et Bayesiansk net udtrykker alle (betingede) uafhængighedsegenskaber for den til nettet hørende sandsynlighedsmodel, fx. p. Det vil sige, egenskaber af typerne: 'mængderne af variable A og B er betinget uafhængige givet værdierne af en tredje mængde C. Denne graf kaldes derfor ofte uafhængighedsgrafen for p.

Som antydet ovenfor, er den beregningsmæssige kompleksitet, for alle eksakte metoder til inferens i Bayesianske net, bestemt af en underliggende graf. Denne graf tilhører en klasse af grafer kaldet triangulerede grafer eller dekomposable grafer, som begge er synonymer for grafer, der ikke indeholder cykler af længde større end tre uden en korde. Valget af trianguleret graf er kun begrænset derved, at uafhængighedsgrafen skal være en delgraf af den triangulerede graf. Det vil sige, at den triangulerede graf skal definere en dekomposabel (hypergraf) indhylning (cover) af uafhængighedsgrafen. Da der kan være en enorm forskel på kompleksiteten tilknyttet forskellige dekomposable indhylninger, er valget af indhylning essentielt for effektiviteten af inferensprocessen. Del I af afhandlingen behandler problemet vedrørende valget af en optimal eller næsten-optimal dekomposabel indhylning.

Selvom det lykkes at bestemme en optimal dekomposabel indhylning, kan inferensproblemet i nogle tilfælde stadig være uløseligt, idet inferensproblemet generelt er NP-fuldstændigt for Bayesianske net (Cooper 1990b). Derfor opstår der ofte et behov for at gå på kompromis med nøjagtigheden af repræsentationen og inferensmetoden. I sådanne tilfælde kan der enten anvendes mindre krævende repræsentationsformer og inferensmetoder (om nogen) eller (yderligere) approksimationer af repræsentationen kan finde sted. I anden del af afhandlingen præsenteres en metode af sidstnævnte type. Metoden er baseret på approksimation under kantfjernelse, hvorved modellens betingede uafhængighedsegenskaber ændres.

En karakteristisk egenskab ved mange succesfulde anvendelser af det Bayesianske paradigme, er den statiske natur af den resulterende model. Det vil sige, at enhver observerbar størrelse observeres højst én gang, og at værdien og indflydelsen af en sådan observation ikke ændres med tiden. Der findes imidlertid, indenfor mange grene af videnskaben (fx. medicinsk, økonomisk, biologisk), mange systemer, som involverer gentagne observationer af en samling af stokastiske størrelser. For sådanne systemer er en statisk model ikke særlig anvendelig, idet beregningen af sandsynlighedsfordelinger for systemvariable på basis af forhåndsviden og observationer af andre systemvariable, kun er gyldige i begrænsede tidsintervaller, og derudover må både nye og gamle observationer af identiske systemvariable kunne tages i betragtning under beregningerne. For at kunne håndtere sådanne dynamiske systemer vha. Bayesianske net, kræves sammenkobling af multiple statiske net. Som tiden går, må der naturligvis tilføjes nye 'tidsskiver' til modellen og gamle må skæres væk. Dette introducerer begrebet dynamiske Bayesianske net, som er emnet for tredje og sidste del af afhandlingen.

Studiet har været dels teoretisk og dels praktisk. Som resultat af den praktiske del, er der blevet produceret et større antal edb-programmer og programpakker. Tre af disse er blevet dokumenteret i form af manualer og brugervejledninger, som kan findes i appendiks A–C.

Bibliography

Andreassen, S. A., Woldbye, M., Falck, B. and Andersen, S. K. (1987). MUNIN – A causal probabilistic network for interpretation of electromyographic findings, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 366–372.

Andreassen, S., Hovorka, R., Benn, J., Olesen, K. G. and Carson, E. R. (1991). A model-based approach to insulin adjustment, *in* M. Stefanelli, A. Hasman, M. Fieschi and J. Talmon (eds), *Proceedings of the Third Conference on Artificial Intelligence in Medicine*, Springer-Verlag, pp. 239–248.

Andreassen, S., Jensen, F. V., Andersen, S. K., Falck, B., Kjærulff, U., Woldbye, M., Sørensen, A. R., Rosenfalck, A. and Jensen, F. (1989). MUNIN — an expert EMG assistant, *in* J. E. Desmedt (ed.), *Computer-Aided Electromyography and Expert Systems*, Elsevier Science Publishers, Amsterdam, chapter 21.

Bohachevsky, I. O., Johnson, M. E. and Stein, M. L. (1986). Generalized simulated annealing for function optimization, *Technometrics* **28**(3): 209–217.

Box, G. E. P. and Jenkins, G. M. (1976). *TIME SERIES ANALYSIS: Forecasting and Control*, Holden-Day Series in Time Series Analysis and Digital Processing, Holden-Day Inc.

Cannings, C., Thompson, E. A. and Skolnick, H. H. (1976). The recursive derivation of likelihoods on complex pedigrees, *Advances in Applied Probability* **8**: 622–625.

Cannings, C., Thompson, E. A. and Skolnick, H. H. (1978). Probability functions on complex pedigrees, *Advances in Applied Probability* **10**: 26–61.

Cooper, G. F. (1984). NESTOR: A computer based medical diagnostic aid that integrates causal and probabilistic knowledge, *Research Report HHP-84-*

48, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, California.

Cooper, G. F. (1987). Probabilistic inference using belief networks is NP-hard, *Research Report KSL-87-37*, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, California.

Cooper, G. F. (1990a). Bayesian belief-network inference using recursive decomposition, *Research Report KSL-90-05*, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, California.

Cooper, G. F. (1990b). The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence* **42**(2–3): 393–405.

Csiszár, I. (1967). Information-type measures of difference of probability distributions and indirect observations, *Studia Scientiarum Mathematicarum Hungarica* **2**: 299–318.

Dagum, P., Galper, A. and Horvitz, E. (1992). Dynamic network models for forecasting, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*.

D'Ambrosio, B. (1989). Symbolic probabilistic inference in belief nets, *Technical report*, Department of Computer Science, Oregon State University.

Darroch, J. N., Lauritzen, S. L. and Speed, T. P. (1980). Markov-fields and log-linear models for contingency tables, *The Annals of Statistics* **8**: 522–539.

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation, *Computational Intelligence* **5**: 142–150.

Dean, T., Basye, K. and Lejter, M. (1990). Planning and active perception, *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pp. 271–276.

Dirac, G. A. (1961). On rigid circuit graphs, *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* **25**: 71–76.

Fischer, L. P., Abrahamsen, P. and Nielsen, L. B. (1992). *Hugin API Reference Manual*, 1.1.1.3 edn, Hugin Expert A/S.

Frydenberg, M. (1989). The chain graph Markov property, *Scandinavian Journal of Statistics* **17**: 333–353.

Fujisawa, T. and Orino, H. (1974). An efficient algorithm of finding a minimal triangulation of a graph, *IEEE International Symposium on Circuits and Systems*, San Francisco, pp. 172–175.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(6): 721–741.

Geva, R. Y. and Paz, A. (1992). Representation of irrelevance relations by annotated graphs, *Research Report 603*, Technion — Israel Institute of Technology, Department of Computer Science, Haifa.

Hilden, J. (1988). Journal of the Royal Statistical Society, Series B, 50(2): 198–199 (Discussion of the paper by Lauritzen and Spiegelhalter).

Højsgaard, S. and Thiesson, B. (1992). BIFROST — Block recursive models Induced From Relevant knowledge, Observations, and Statistical Techniques, *Research report R 92-2010*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Howard, R. A. and Matheson, J. E. (1984). Influence diagrams, *Principles and Application of Decision Analysis*, Strategic Decisions Group, Menlo Park, California.

Jensen, F. and Andersen, S. K. (1990). Approximations in Bayesian belief universes for knowledge based systems, *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, Cambridge, MA.

Jensen, F. V. (1988). Junction trees and decomposable hypergraphs, *Research report*, Judex Datasystemer A/S, Aalborg, Denmark.

Jensen, F. V., Andersen, S. K., Kjærulff, U. and Andreassen, S. (1987). MUNIN – on the case for probabilities in medical expert systems – a practical exercise, *in* J. Fox, M. Fieschi and R. Engelbrecht (eds), *Proceedings of First Conference of the European Society for Artificial Intelligence in Medicine (AIME)*, Springer-Verlag, Heidelberg, Marseilles, France, pp. 149–160.

Jensen, F. V., Kjærulff, U., Olesen, K. G. and Pedersen, J. (1989). Et forprojekt til et ekspertsystem for drift af spildevandsrensning, *Technical report*, Judex Datasystemer A/S, Aalborg, Danmark.

Jensen, F. V., Lauritzen, S. L. and Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations, *Computational Statistics Quarterly* **4**: 269–282.

Jensen, F. V., Olesen, K. G. and Andersen, S. K. (1990). An algebra of Bayesian belief universes for knowledge based systems, *Networks* **20**(5): 637–659.

Kemperman, J. H. B. (1967). On the optimum rate of transmitting information, *Probability and Information Theory* pp. 126–169. Lecture notes in mathematics, Springer-Verlag.

Kim, J. H. and Pearl, J. (1983). A computational model for causal and diagnostic reasoning in inference systems, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 190–193.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing, *Science* **220**: 671–680.

Kjærulff, U. (1990). Triangulation of graphs — algorithms giving small total state space, *Research Report R 90-09*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1992a). A computational scheme for reasoning in dynamic probabilistic networks, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo, California, pp. 121–129.

Kjærulff, U. (1992b). Optimal decomposition of probabilistic networks by simulated annealing, *Statistics and Computing* **2**: 7–17.

Kjærulff, U. (1993a). Approximation of bayesian networks through edge removals, *Research report*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1993b). A computational scheme for dynamic Bayesian networks, *Research report*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1993c). *dHUGIN API Reference Manual*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1993d). *User's Guide to **dhugin***, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1993e). *User's Manual to gt*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kullback, S. (1959). *Information Theory and Statistics*, John Wiley, New York.

Kullback, S. (1967). A lower bound for discrimination information in terms of variation, *IEEE Transactions on Information Theory* **IT-13**: 126–127.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society, Series B* **50**(2): 157–224.

Lauritzen, S. L. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative, *The Annals of Statistics* **17**: 31–57.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N. and Leimer, H.-G. (1990). Independence properties of directed Markov fields, *Networks* **20**(5): 491–505. Special Issue on Influence Diagrams.

Lauritzen, S. L., Speed, T. P. and Vijayan, K. (1984). Decomposable graphs and hypergraphs, *Journal of The Australian Mathematical Society, A* **36**: 12–29.

Lundy, M. (1985). Applications of the annealing algorithm to combinatorial problems in statistics, *Biometrika* **72**: 191–198.

Lundy, M. and Mees, A. (1986). Convergence of an annealing algorithm, *Mathematical Programming* **34**: 111–124.

Nicholson, A. E. and Brady, J. M. (1992). Sensor validation using dynamic belief networks, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*.

Olesen, K. G., Kjærulff, U., Jensen, F., Jensen, F. V., Falck, B., Andreassen, S. and Andersen, S. K. (1989). A MUNIN network for the median nerve — a case study on loops, *Applied Artificial Intelligence*. Special issue: Towards Causal AI Models in Practice.

Pearl, J. (1986a). A constraint-propagation approach to probabilistic reasoning, *in* L. M. Kanal and J. Lemmer (eds), *Uncertainty in Artificial Intelligence*, North-Holland, Amsterdam, pp. 357–370.

Pearl, J. (1986b). Fusion, propagation, and structuring in belief networks, *Artificial Intelligence* **29**: 241–288.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Series in Representation and Reasoning, Morgan Kaufmann Publishers, Inc.

Read, T. R. C. and Cressie, N. A. C. (1988). *Goodness-of-Fit Statistics for Discrete Multivariate Data*, Springer Series in Statistics, Springer-Verlag, New York, Inc.

Rose, D. J. (1970). Triangulated graphs and the elimination process, *Journal of Mathematical Analysis and Applications* **32**: 597–609.

Rose, D. J. (1973). A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, *in* R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, New York, pp. 183–217.

Rose, D. J., Tarjan, R. E. and Lueker, G. S. (1976). Algorithmic aspects of vertex elimination on graphs, *SIAM Journal on Computing* **5**: 266–283.

Sato, N. and Tinney, W. F. (1963). Techniques for exploiting the sparsity of the network admittance matrix, *IEEE, PAS* pp. 944–950.

Shachter, R. D. (1988). Probabilistic inference and influence diagrams, *Operations Research* **36**: 589–604.

Shachter, R. D. (1990). Evidence absorption and propagation through evidence reversals, *in* M. Henrion, R. D. Shachter, J. Lemmer and L. M. Kanal (eds), *Uncertainty in Artificial Intelligence*, North-Holland, Amsterdam, pp. 173–190.

Shachter, R. D., Andersen, S. K. and Szolovits, P. (1991). The equivalence of exact methods for probabilistic inference on belief networks, *Technical report*, Department of Engineering-Economic Systems, Stanford University.

Shachter, R. D., D'Ambrosio, B. and Favero, B. A. D. (1990). Symbolic probabilistic inference in belief networks, *Eighth National Conference on Artificial Intelligence*, Boston, pp. 126–131.

Shafer, G. and Shenoy, P. P. (1988). Bayesian and belief-function propagation, *Working Paper 121*, School of Business, University of Kansas, Lawrence, Kansas.

Shafer, G. and Shenoy, P. P. (1990). Probability propagation, *Annals of Mathematics and Artificial Intelligence*.

Shortliffe, E. H. and Buchanan, B. (1975). A model for inexact reasoning in medicine, *Mathematical Biosciences* **23**: 351–379.

Spiegelhalter, D. J. (1986). Probabilistic reasoning in predictive expert systems, *in* J. F. Lemmer and L. N. Kanal (eds), *Uncertainty in Artificial Intelligence*, Elsevier Science Publishers B. V. (North-Holland).

Suermondt, H. J. and Cooper, G. F. (1988). Updating probabilities in multiply-connected belief networks, *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, Minneapolis, pp. 335–343.

Tarjan, R. E. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* **13**(3): 566–579.

Thomas, A. (1986). Optimal computation of probability functions for pedigree analysis, *IMA Journal of Mathematics Applied in Medicine & Biology* **3**: 167–178.

Tinney, W. F. and Walker, J. W. (1967). Direct solutions of sparse network equations by optimally ordered triangular factorization, *Proc. IEEE* **55**: 1801–1809.

Wen, W. X. (1990). Optimal decomposition of belief networks, *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, Cambridge, MA, pp. 245–256.

Wermuth, N. and Lauritzen, S. L. (1983). Graphical and recursive models for contingency tables, *Biometrika* **70**(3): 537–552.

West, M. and Harrison, J. (1989). *Bayesian Forecasting and Dynamic Models*, Series in Statistics, Springer-Verlag, New York Inc.

Whittaker, J. (1989). *Graphical Models in Applied Multivariate Statistics*, Wiley & Sons, Inc., Chichester.

Zadeh, L. A. (1983). The role of fuzzy logic in the management of uncertainty in expert systems, *Fuzzy Sets and Systems* **11**: 199–228.