ELSEVIER Computational Statistics & Data Analysis 37 (2001) 297–322

COMPUTATIONAL STATISTICS & DATA ANALYSIS

www.elsevier.com/locate/csda

An implementation of the iterative proportional fitting procedure by propagation trees

J.H. Badsberg[a], F.M. Malvestuto[b,*]

[a] *Biometry Research Unit, Dept. Agricultural Systems, Research Centre Foulum, Blichers Alli 1, DK - 8830 Tjcle, Denmark*

[b] *Dip. di Sci. dell'Informazione, Univ. La Sapienza di Roma, via Salaria 113, Roma 00198, Italy*

Received 1 November 1999; received in revised form 1 September 2000; accepted 21 February 2001

Abstract

The space-saving implementation of the iterative proportional fitting procedure proposed by Jirousek and Preucil (Comput. Statist. Data Anal. 19 (1995) 177) on this journal can be improved by applying the tree-computation techniques designed for Markov networks. The optimisation problem raised by the use of Markovian propagation trees is solved. Next, an even better implementation is obtained using certain trees, here introduced and called *fast propagation trees*, which are obtained by "simplifying" optimal Markovian propagation trees.

Keywords: Acyclic hypergraph; Iterative proportional fitting procedure; Markov extension; Probabilistic database; Propagation tree

1. Introduction

The iterative proportional fitting (IPF) procedure (Deming and Stephan, 1940) is employed in probability theory to compute the maximum-entropy extension (MEE) of given discrete probability distributions (Csiszár, 1975); moreover, it is also used in statistics to compute the maximum-likelihood estimate of the parameters of a multinomial sampling distribution under a hierarchical log–linear model (Bishop et al., 1975). The IPF procedure consists in determining better and better approximations

* Corresponding author. Fax: +39-06-8541842.
E-mail address: mal@dsi.uniroma1.it (F.M. Malvestuto).

0167-9473/01/$ - see front matter
PII: S 0167-9473(01)00013-5

of the MEE until the convergence is attained (Bishop et al., 1975). Since the domain of such probability functions may be very large, some techniques have been developed to reduce both storage requirements and computation time (Badsberg, 1995, 1996; Denteneer and Verbeek, 1986; Jirousek, 1991; Jirousek and Preucil, 1995; Larrañaga et al., 1997; Malvestuto, 1988). To save storage, Jirousek (1991) and Jirousek and Preucil (1995) proposed a method (here referred to as the *JP method*) which, instead of storing each approximating distribution of the limit distribution, stores a suitable set of its marginals; furthermore, when the convergence is attained, the limit distribution is directly computed from its stored marginals. An efficient implementation of the JP method can be obtained by thinking of the approximation at the tth step as being the result of an "updating" of the approximation resulting from the $(t-1)$th step and by running an updating procedure based on the technique of propagation in Markov networks (Dawid, 1992; Jensen, 1988; Jensen and Jensen, 1994; Lauritzen, 1992, 1996; Lauritzen and Spiegelhalter, 1988; Shafer, 1996; Spiegelhalter, 1986). Such an implementation makes use of certain tree structures, we call *Markovian propagation trees*, and the efficiency of the implementation depends on the Markovian propagation tree in use. This paper first proves that an optimal Markovian propagation tree for implementing the JP method can be found in a very efficient way and, then, shows how to improve the implementation of the JP based on an optimal Markovian propagation tree using certain directed trees, here called *fast propagation trees*.

The paper is organised as follows. In Section 2, we recall more or less standard definitions and results from hypergraph theory and from the literature on IPF procedure. Section 3 contains the implementation of the JP method based on Markovian propagation trees. Section 4 introduces fast propagation trees and the related implementation of the JP method. Section 5 contains some closing remarks.

2. Background

2.1. Hypergraphs

A *hypergraph* (Berge, 1989) is an arbitrary finite family of distinct finite and nonempty sets, which are called its *edges*; furthermore, a hypergraph is *reduced* (or, equivalently, is a "Sperner system" or a "clutter" or an "antichain") if its edges are pairwise inclusion-incomparable sets. Let $\boldsymbol{H}$ be a hypergraph. The set $\bigcup_{E\in\boldsymbol{H}} E$ is called the *vertex set* of $\boldsymbol{H}$, denoted by $V(\boldsymbol{H})$, and its elements the *vertices* of $\boldsymbol{H}$. A *partial edge* (Beeri et al., 1983) of $\boldsymbol{H}$ is any nonempty (proper or improper) subset of some edge of $\boldsymbol{H}$. A *subhypergraph* of $\boldsymbol{H}$ is a hypergraph whose edges are all partial edges of $\boldsymbol{H}$. A *cover* of $\boldsymbol{H}$ is any hypergraph $\boldsymbol{K}$ such that $V(\boldsymbol{K}) = V(\boldsymbol{H})$ and $\boldsymbol{H}$ is a subhypergraph of $\boldsymbol{K}$. The *subhypergraph* of $\boldsymbol{H}$ *induced* by a nonempty subset X of $V(\boldsymbol{H})$ is the reduced hypergraph whose edges are exactly the maximal edges of the hypergraph $\{E\cap X\colon E \text{ in } \boldsymbol{H}\}$. The subhypergraph of $\boldsymbol{H}$ induced by the complement of X will be denoted by $\boldsymbol{H}$-X. A *path* in hypergraph $\boldsymbol{H}$ is a sequence $\langle E_1,\ldots,E_k\rangle$, $k \geqslant 1$, of edges of $\boldsymbol{H}$ such that, if $k > 1$, then for each $i < k$,

one has $E_i \cap E_{i+1} \neq \emptyset$; then the edges E_1 and E_k are called the *ends* of the path. Two vertices of $\boldsymbol{H}$ are *connected* if they belong to two edges that are the ends of some path in $\boldsymbol{H}$. Hypergraph $\boldsymbol{H}$ is connected if every two vertices of $\boldsymbol{H}$ are connected. The maximal connected induced subhypergraphs of $\boldsymbol{H}$ are called the *components* of $\boldsymbol{H}$.

A nonempty subset X of $V(\boldsymbol{H})$ is a *separator* if there are two vertices of $\boldsymbol{H}$ that lie in two distinct connected components of $\boldsymbol{H}$-X; such vertices are said to be *separated* by X. A *separating partial edge* of $\boldsymbol{H}$ is a partial edge of $\boldsymbol{H}$ that is a separator. A separating partial edge X of $\boldsymbol{H}$ is a *divider* of $\boldsymbol{H}$ (Malvestuto and Moscarini, 1998, 2000) if there exist two connected vertices of $\boldsymbol{H}$ that are separated by X and are not separated by any proper subset of X. The *divider hypergraph* of $\boldsymbol{H}$ is the (possibly nonreduced) subhypergraph of $\boldsymbol{H}$ whose edges are exactly the dividers of $\boldsymbol{H}$. Let X be a divider of $\boldsymbol{H}$. If $\boldsymbol{C}$ is a component of $\boldsymbol{H}$-X, the *boundary* of $\boldsymbol{C}$ with respect to X is the set of vertices in X that are adjacent to some vertex of $\boldsymbol{C}$; moreover, the *degree* of X, denoted by $g(X)$, is the number of components of $\boldsymbol{H}$-X whose boundaries coincide with the whole X.

Example 1. Consider the (reduced) hypergraph $\boldsymbol{H} = \{abc, abd, ace, bcf, cg, ch\}$. The divider hypergraph of $\boldsymbol{H}$ is the nonreduced hypergraph $\{ab, ac, bc, c\}$ and the degrees of the four dividers of $\boldsymbol{H}$ are $g(ab) = 2$, $g(ac) = 2$, $g(bc) = 2$ and $g(c) = 3$.

The 2-*section* of hypergraph $\boldsymbol{H}$, denoted by $[\boldsymbol{H}]_2$, is an ordinary undirected graph on $V(\boldsymbol{H})$ where two vertices are joined by an edge if and only if their pair is a partial edge of $\boldsymbol{H}$. A *clique* of $[\boldsymbol{H}]_2$ is a maximal set of pairwise adjacent vertices of $[\boldsymbol{H}]_2$. The *clique hypergraph* of $[\boldsymbol{H}]_2$ is the (reduced) hypergraph whose edges are exactly the cliques of $[\boldsymbol{H}]_2$; of course, $\boldsymbol{H}$ is a subhypergraph of the clique hypergraph of $[\boldsymbol{H}]_2$. Hypergraph $\boldsymbol{H}$ is *conformal* if every clique of $[\boldsymbol{H}]_2$ is a partial edge of $\boldsymbol{H}$, that is, if $\boldsymbol{H}$ is a cover of the clique hypergraph of $[\boldsymbol{H}]_2$. Note that, if $\boldsymbol{H}$ is reduced, then $\boldsymbol{H}$ is conformal if and only if $\boldsymbol{H}$ coincides with the clique hypergraph of $[\boldsymbol{H}]_2$. Moreover, if $\boldsymbol{H}$ is conformal, then separating partial edges and dividers of $\boldsymbol{H}$ are the same as "clique-separators" (Leimer, 1993; Tarjan, 1985) and "minimal relative clique-separators" of $[\boldsymbol{H}]_2$ (Diestel, 1990), respectively. A conformal hypergraph $\boldsymbol{H}$ is *acyclic* (Beeri et al., 1983) (or "decomposable" in (Lauritzen, 1996; Lauritzen et al., 1984)) if $[\boldsymbol{H}]_2$ is *chordal* (or "triangulated") in the sense that it contains no chordless cycle of length greater than three (Kloks, 1994). Several equivalent definitions of acyclicity exist (Beeri et al., 1983); we wish to mention one of them which will be used later on. A hypergraph $\boldsymbol{H}$ is acyclic if and only if there is an ordering (called a "running intersection ordering") of its edges, say $E_1, \ldots, E_n$, such that if $n > 1$ then, for each i, $1 \leqslant i \leqslant n-1$, the set $(E_1 \cup \cdots \cup E_i) \cap E_{i+1}$ is a partial edge of the hypergraph $\{E_1, \ldots, E_i\}$. It is worth noting that, if $\boldsymbol{H}$ is a connected and acyclic, reduced hypergraph, then the divider hypergraph of $\boldsymbol{H}$ coincides exactly with the hypergraph $\{(E_1 \cup \cdots \cup E_i) \cap E_{i+1} : 1 \leqslant i \leqslant n-1\}$, where $E_1, \ldots, E_n$ is a running intersection ordering of the edges of $\boldsymbol{H}$ (Malvestuto and Moscarini, 1998, 2000), so that the number of dividers of $\boldsymbol{H}$ is never greater than $n-1$.

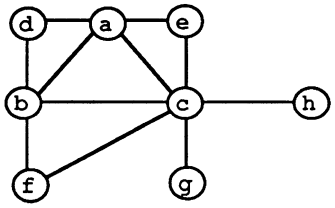

Fig. 1.

Example 1 (*continued*)**.** The hypergraph $\boldsymbol{H}$ is acyclic since it is conformal and its 2-section (see Fig. 1) is chordal. An example of the running intersection ordering of the edges of $\boldsymbol{H}$ is: $abc, abd, ace, bcf, cg, ch$.

A characteristic property of acyclic and connected hypergraphs is that they admit tree representations (Kloks, 1994; Shibata, 1988). More precisely, if $\boldsymbol{H}$ is such a hypergraph then and only then there exists a tree T, with one node for each edge of $\boldsymbol{H}$, such that the following condition, sometimes called *separation property* (Almond and Kong, 1993) or *junction property* in (Lauritzen, 1996), is satisfied:

for each vertex a of $\boldsymbol{H}$, the set of nodes of T corresponding to edges of $\boldsymbol{H}$ containing a induces a subtree of T.

A tree such as T is generally called a *representative tree* of $\boldsymbol{H}$ and, when $\boldsymbol{H}$ is reduced, is called a *junction tree* (Lauritzen, 1996) or a *join tree* of $\boldsymbol{H}$ (Beeri et al., 1983; Maier, 1983; Tarjan and Yannakakis, 1984; Yannakakis, 1981). For an acyclic and connected, reduced hypergraph $\boldsymbol{H}$ we shall make use of a different tree representation by taking a representative tree of the acyclic and nonreduced hypergraph $\boldsymbol{H} \cup \boldsymbol{D}$, where $\boldsymbol{D}$ is the divider hypergraph of $\boldsymbol{H}$. We call such a tree an *edge-divider tree* of $\boldsymbol{H}$; it can be viewed as a spanning tree of the bipartite, undirected graph with node set $\boldsymbol{H} \cup \boldsymbol{D}$ whose arcs join two nodes, one in $\boldsymbol{H}$ and the other in $\boldsymbol{D}$, whenever they are inclusion-comparable. This graph, denoted by $G(\boldsymbol{H})$, will be referred to as the *edge-divider graph* of $\boldsymbol{H}$; moreover, we call $\boldsymbol{H}$*-nodes* the nodes of $G(\boldsymbol{H})$ that are sets in $\boldsymbol{H}$ and $\boldsymbol{D}$*-nodes* the nodes of $G(\boldsymbol{H})$ that are sets in $\boldsymbol{D}$.

Example 1 (*continued*)**.** The edge-divider graph of $\boldsymbol{H} = \{acdh, bd, cde, cdfg\}$ is shown in Fig. 2. An edge-divider tree of $\boldsymbol{H}$ is shown in Fig. 3.

Lemma 1. *Let $\boldsymbol{H}$ be an acyclic and connected, reduced hypergraph, and X a divider of $\boldsymbol{H}$. For every edge-divider tree T of $\boldsymbol{H}$, the degree of X equals the number of neighbours of the node X of T.*

Proof. Let $G(\boldsymbol{H})$ be the edge-divider graph of $\boldsymbol{H}$, and let G' be the subgraph of $G(\boldsymbol{H})$ obtained by removing X and the $\boldsymbol{D}$-nodes of $G(\boldsymbol{H})$ that are proper subsets of X. Since $\boldsymbol{H}$ is reduced, the components of $\boldsymbol{H}$-X correspond one to one to the components of G'; moreover, by the acyclicity of $\boldsymbol{H}$, the boundary of a component

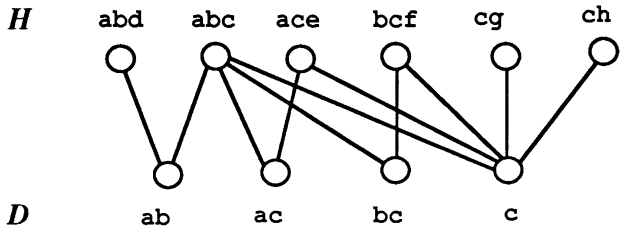

Fig. 2.

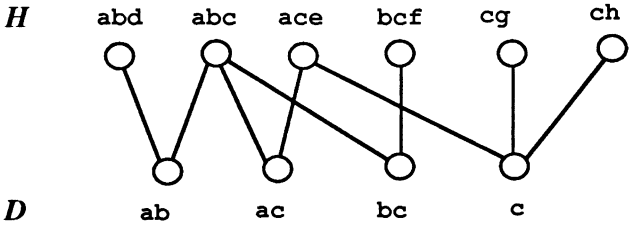

Fig. 3.

$\boldsymbol{C}$ of $\boldsymbol{H}$-X with respect to X is exactly X if and only if the component of G' corresponding to $\boldsymbol{C}$ contains an $\boldsymbol{H}$-node that in $G(\boldsymbol{H})$ is adjacent to X. So, the degree of X equals the number of the components of G' containing an $\boldsymbol{H}$-node that in $G(\boldsymbol{H})$ is adjacent to X. Consider now the subgraph T' of T obtained by removing X and the $\boldsymbol{D}$-nodes of T that are proper subsets of X. Of course, each component of T' is a subgraph of one component of G'. Moreover, by the separation property, if two $\boldsymbol{H}$-nodes are connected in G' then they are also connected in T'; therefore, the number of the components of T' equals the number of the components of G'. Finally, by the acyclicity of T, if a component of G' contains an $\boldsymbol{H}$-node that in $G(\boldsymbol{H})$ is adjacent to X then the corresponding component of T' contains exactly one $\boldsymbol{H}$-node that in T is adjacent to X. So, the degree of X equals the number of neighbours of X in T. □

An edge-divider tree of $\boldsymbol{H}$ can be easily obtained from a running intersection ordering of its edges. However, from a computational point of view, it is more convenient to proceed as follows. Let each arc (E,X) of $G(\boldsymbol{H})$, with E in $\boldsymbol{H}$ and X in $\boldsymbol{D}$, be weighted by $|X|$ so that every spanning subgraph of $G(\boldsymbol{H})$ is implicitly weighted by the sum of the weights of its arcs. Then, it is easily seen that the path optimality conditions of maximum-weight spanning trees (Ahuja et al., 1993; Golumbic, 1980) imply that every edge-divider tree of $\boldsymbol{H}$ is a maximum-weight spanning tree of $G(\boldsymbol{H})$. On the other hand, by the acyclicity of $\boldsymbol{H}$, every maximum-weight spanning tree of $G(\boldsymbol{H})$ is an edge-divider tree of $\boldsymbol{H}$ (for a formal proof see Maier, 1983). Therefore, an edge-divider tree of $\boldsymbol{H}$ can be efficiently found using greedy algorithms

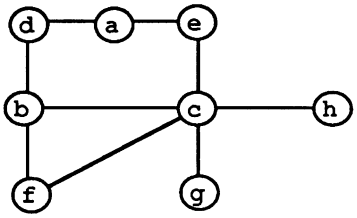

Fig. 4.

such as Kruskal's algorithm or Prim's algorithm or Sollin's algorithm (Ahuja et al., 1993; Golumbic, 1980).

It is worth mentioning that, for an arbitrary connected hypergraph $\boldsymbol{H}$, if $\boldsymbol{K}$ is an acyclic cover of $\boldsymbol{H}$ and T is a representative tree of $\boldsymbol{K}$, then the pair $(\boldsymbol{K}, T)$ defines a *tree-decomposition* of $\boldsymbol{H}$ (Kloks, 1994). Thus, if $\boldsymbol{H}$ is an acyclic and connected, reduced hypergraph and T is an edge-divider tree of $\boldsymbol{H}$, then the pair $(\boldsymbol{H} \cup \boldsymbol{D}, T)$ is a tree-decomposition of $\boldsymbol{H}$. If $\boldsymbol{H}$ is a cyclic and connected, reduced hypergraph then, in order to get a tree-decomposition of $\boldsymbol{H}$, we need an acyclic cover $\boldsymbol{K}$ of $\boldsymbol{H}$ and finding a hypergraph such as $\boldsymbol{K}$ is the same as "triangulating" the graph $[\boldsymbol{H}]_2$, that is, finding a chordal graph that is a cover of $[\boldsymbol{H}]_2$. Though very efficient algorithms exist to triangulate a graph in such a way that no edge is added if it is chordal (Rose et al., 1976; Tarjan and Yannakakis, 1984), the problem of the best triangulation of $[\boldsymbol{H}]_2$ proves to be computationally hard (in the sense that no polynomial-time algorithm is likely to exist in the worst case) when reasonable objective functions are used (Arnborg et al., 1993; Bodlaender, 1996; Kloks, 1994; Sanders, 1995; Wen, 1990; Yannakakis, 1981), and it seems to be unavoidable to resort to some heuristics (Becker and Geiger, 1997; Kjærluff, 1992; Wen, 1989). We now introduce for an arbitrary reduced hypergraph $\boldsymbol{H}$ a special acyclic cover of $\boldsymbol{H}$, called the "compaction" of $\boldsymbol{H}$ (Malvestuto and Moscarini, 1998), which has some useful properties and will be used later on. Some preliminary notions are needed.

A connected set of vertices X of $\boldsymbol{H}$ is *compact* if every two vertices in X are not separated by any partial edge of $\boldsymbol{H}$. The *compact components* of $\boldsymbol{H}$ are the subhypergraphs of $\boldsymbol{H}$ induced by its maximal compact sets or, equivalently, they are the maximal induced subhypergraphs of $\boldsymbol{H}$ where no partial edge is a divider. The vertex sets of compact components of $\boldsymbol{H}$ can be viewed as being the edges of a reduced hypergraph, called the *compaction* of $\boldsymbol{H}$, which is an acyclic cover of $\boldsymbol{H}$ which coincides with $\boldsymbol{H}$ if and only if $\boldsymbol{H}$ is acyclic and reduced; moreover, the divider hypergraph of the compaction of $\boldsymbol{H}$ always coincides with the divider hypergraph of $\boldsymbol{H}$ (Malvestuto and Moscarini, 1998, 2000).

Example 2. Consider the cyclic (reduced) hypergraph $\boldsymbol{H}=\{ad, ae, bcf, bd, ce, cg, ch\}$, whose 2-section is shown in Fig. 4.

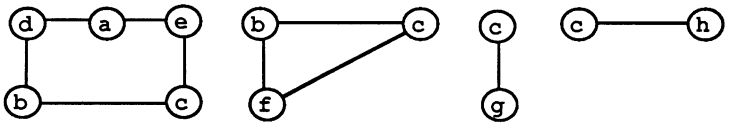

Fig. 5.

The compact components of $\boldsymbol{H}$ are $\boldsymbol{H}_1=\{ad, ae, bc, bd, ce\}$, $\boldsymbol{H}_2=\{bcf\}$, $\boldsymbol{H}_3=\{cg\}$ and $\boldsymbol{H}_4=\{ch\}$, whose 2-sections are shown in Fig. 5. Therefore, the compaction of $\boldsymbol{H}$ is the acyclic hypergraph $\{abcde, bcf, cg, ch\}$.

2.2. Probabilistic databases

Let V be a finite set of random variables that have associated with them finite sets of values. By a *state* of a nonempty subset X of V we mean an assignment of values, one for each variable in X. If $\mathbf{x}$ is a state of X and Y is a subset of X, then by $\mathbf{x}_Y$ we denote the state of Y obtained by projecting $\mathbf{x}$ onto Y. The state space of V is denoted by $\mathbf{V}$.

A *probability distribution* on a nonempty subset X of V is a nonnegative function p defined on the state space of X that adds to 1; the set of states of X to which p assigns a nonzero probability is referred to as the *support* of p. If Y is a subset of X, then by $p^{\downarrow Y}$ we denote the marginal of p on Y.

Let $\boldsymbol{H}$ be a hypergraph with vertex set V. A system $\boldsymbol{P}=\{p_E: E \text{ in } \boldsymbol{H}\}$ of probability distributions on edges of $\boldsymbol{H}$ will be referred to as a *probabilistic database* with *scheme* $\boldsymbol{H}$; moreover, if $\boldsymbol{H}'$ is the reduced version of hypergraph $\boldsymbol{H}$, the *reduced version* of $\boldsymbol{P}$ is the probabilistic database $\{p_E: E \text{ in } \boldsymbol{H}'\}$. An *extension* of $\boldsymbol{P}$ is a probability distribution P on V such that $p_E = P^{\downarrow E}$ for each edge E of $\boldsymbol{H}$. A probabilistic database $\boldsymbol{P}$ is *consistent* if there exists an extension of $\boldsymbol{P}$, and is *pairwise consistent* if, for every two edges E and E' of $\boldsymbol{H}$, the probabilistic database $\{p_E, p_{E'}\}$ with scheme $\{E, E'\}$ is consistent. Of course, if $\boldsymbol{P}$ is a consistent probabilistic database, then the reduced version of $\boldsymbol{P}$ is consistent, too. Finally, it is well known that hypergraph $\boldsymbol{H}$ is acyclic if and only if, for every probabilistic database with scheme $\boldsymbol{H}$, the condition of pairwise consistency is (not only necessary but also) sufficient for consistency (Kellerer, 1964; Malvestuto, 1988; Vorob'ev, 1962).

In the rest of this subsection we limit our considerations to probabilistic databases whose schemes are reduced hypergraphs. Given a consistent probabilistic database $\boldsymbol{P}$ with scheme $\boldsymbol{H}$, the *maximum-entropy extension* (MEE) of $\boldsymbol{P}$ is the extension P of $\boldsymbol{P}$ that maximises the functional

$$-\Sigma P(\mathbf{v}) \log P(\mathbf{v}),$$

the summation being extended over all the states $\mathbf{v}$ of V belonging to the support of P. The MEE P of $\mathbf{P}$ is characterized as being the extension of $\boldsymbol{P}$ for which, for each edge E of $\boldsymbol{H}$, there is a real function f_E defined on the state space of E such

Table 1

ab	p_{ab}	ac	p_{ac}	bc	p_{bc}
00	1/6	00	1/6	00	1/6
01	1/3	01	1/3	01	1/3
10	1/3	10	1/3	10	1/3
11	1/6	11	1/6	11	1/6

Table 2

abc	P
000	0
001	1/6
010	1/6
011	1/6
100	1/6
101	1/6
110	1/6
111	0

that the following factorisation

$$P(\mathbf{v}) = \prod_{E \in \boldsymbol{H}} f_E(\mathbf{v}_E)$$

holds almost everywhere in the state space of V, that is, for every state of V belonging to the support of P (Csiszár, 1975). Moreover since $P(\mathbf{v}) > 0$ implies $p_E(\mathbf{v}_E) > 0$ for each edge E of $\boldsymbol{H}$, the support of P is always a (proper or improper) subset of the set of states of V

$$\mathbf{V}^* = \{\mathbf{v} \in \mathbf{V} : p_E(\mathbf{v}_E) > 0 \text{ for each edge } E \text{ of } \boldsymbol{H}\}.$$

It is worth mentioning that, if the support of P coincides with $\mathbf{V}^*$, then P is said to belong to the extended log–linear model generated by $\boldsymbol{H}$ (Lauritzen, 1996; Malvestuto, 2001).

Example 3. Consider the probabilistic database with scheme $\boldsymbol{H} = \{ab, ac, bc\}$ reported in Table 1. Using standard methods of linear algebra, we find that it admits only one extension, which is reported in Table 2, which hence is also its MEE. Moreover, the support of P is a proper subset of $\mathbf{V}^* = \mathbf{V}$ so that P does not belong to the extended log–linear model generated by $\boldsymbol{H}$.

Consider now the case that $\boldsymbol{H}$ is acyclic. Then the MEE P of $\boldsymbol{P}$ is called the *Markov extension* of $\boldsymbol{P}$ (Vorob'ev, 1963) and has a closed-form expression (Haberman, 1974). More precisely, if $\boldsymbol{D}$ is the divider hypergraph of $\boldsymbol{H}$ and, for each X

in $\boldsymbol{D}$, $g(X)$ is the degree of X, then the following is a factorisation of P that holds for every state $\mathbf{v}$ in the support of P:

$$P(\mathbf{v}) = \prod_{E \in \boldsymbol{H}} p_E(\mathbf{v}_E) \Bigg/ \prod_{X \in \boldsymbol{D}} [p_{E_X}^{\downarrow X}(\mathbf{v}_X)]^{g(X)-1}, \tag{1}$$

where E_X is any edge of $\boldsymbol{H}$ that includes divider X (Bishop et al., 1975; Darroch et al., 1980). On the other hand, the sum of the right-hand side of formula (1) over all the states in $\mathbf{V}^*$ is exactly equal to 1 and, hence, the support of P coincides with $\mathbf{V}^*$; therefore, factorisation (1) holds everywhere in $\mathbf{V}$ with the convention $(0/0)=0$. Finally, every edge-divider tree T of $\boldsymbol{H}$ provides a convenient picture of formula (1) owing to the graphical meaning of the quantities $g(X)$ in T (see Lemma 1).

Example 1 (*continued*)**.** Let $\boldsymbol{P} = \{p_{abc}, p_{abd}, p_{ace}, p_{bcf}, p_{cg}, p_{ch}\}$ be any consistent probabilistic database with scheme $\boldsymbol{H}$. By formula (1), the following is a closed-form expression for the Markov extension of $\boldsymbol{P}$:

$$p_{abc}\, p_{abd}\, p_{ace}\, p_{bcf}\, p_{cg}\, p_{ch} / p_{abc}^{\downarrow ab}\, p_{abc}^{\downarrow ac}\, p_{abc}^{\downarrow bc} (p_{cg}^{\downarrow c})^2,$$

of which the tree of Fig. 3 is the graphical picture.

Consider now the case of a consistent probabilistic database $\boldsymbol{P}$ whose scheme $\boldsymbol{H}$ is a cyclic (reduced) hypergraph. Then, the computation of the MEE P of $\boldsymbol{P}$ based on the IPF procedure consists in computing better and better approximations $P_0, P_1, \ldots$ of P until the convergence is attained. The zero-order approximation P_0 is usually taken to be the uniform distribution on V; however, since the support of P is a subset of $\mathbf{V}^*$, it is convenient to take P_0 to be uniform over $\mathbf{V}^*$ and to vanish elsewhere; that is, for each state $\mathbf{v}$ of V, $P_0(\mathbf{v})$ takes on the value $|\mathbf{V}^*|^{-1}$ if $\mathbf{v}$ belongs to $\mathbf{V}^*$, and 0 otherwise. After computing P_t, $t \geqslant 0$, P_{t+1} is obtained by "fitting" P_t to some probability distribution from $\boldsymbol{P}$. More precisely, given an ordering $E_1, E_2, \ldots, E_n$ of edges of $\boldsymbol{H}$, the IPF procedure cycles through:

$$\text{For } i = 1, \ldots, n, \text{ set } P_{kn+i} := (p_{E_i} / P_{kn+i-1}^{\downarrow E_i}) P_{kn+i-1} \qquad (k = 0, 1, \ldots).$$

Explicitly, the ith assignment statement above consists of the following computation:

(*Iterative step*)
For each state $\mathbf{v}$ of V,
if $\mathbf{v}$ is in $\mathbf{V}^*$ then set $P_{kn+i}(\mathbf{v}) := [p_{E_i}(\mathbf{v}_{E_i}) / P_{kn+i-1}^{\downarrow E_i}(\mathbf{v}_{E_i})] P_{kn+i-1}(\mathbf{v})$; otherwise, set $P_{kn+i}(\mathbf{v}) := 0$.

Note that one can easily prove by induction on t that the support of each P_t, $t \geqslant 0$, does coincide with $\mathbf{V}^*$ so that the divisions above can be carried out without inconvenience whenever data is in absolute precision (i.e., no rounding criterion is used); otherwise, it is sufficient to set $P_{kn+i}(\mathbf{v}) := 0$ if $\mathbf{v}$ is not in $\mathbf{V}^*$ or the rounded value of $P_{kn+i-1}(\mathbf{v})$ is 0.

Table 3

abc	P_0	P_1	P_2	P_3	P_4	P_5	P_6	...
000	1/8	1/12	1/18	1/30	1/36	1/46	31/1658	...
001	1/8	1/12	1/9	1/6	5/36	5/31	23/131	...
010	1/8	1/6	1/9	1/6	5/27	10/69	62/393	...
011	1/8	1/6	2/9	2/15	4/27	16/93	368/2487	...
100	1/8	1/6	2/9	2/15	4/27	16/93	368/2487	...
101	1/8	1/6	1/9	1/6	5/27	10/69	62/393	...
110	1/8	1/12	1/9	1/6	5/36	5/31	23/131	...
111	1/8	1/12	1/18	1/30	1/36	1/46	31/1658	...

Example 3 (*continued*). With the edges of $\boldsymbol{H}$ ordered as ab, ac, bc the IPF procedure yields the approximations of P reported in Table 3.

We now evaluate the computational cost of the IPF procedure under the worst-case assumption that $\boldsymbol{P}$ is such that $\mathbf{V}^* = \mathbf{V}$. To achieve this, with each set X of variables we associate its *size*, denoted by $||X||$ and measured by the number of all possible states of X; accordingly, by the size of a hypergraph we mean the sum of sizes of its edges. Then, the $(kn+i)$th iterative step needs

$||V||$ additions to marginalize P_{kn+i-1} to E_i,

$||E_i||$ divisions to compute the ratios $p_{E_i}(\mathbf{v}_{E_i})/P^{\downarrow E_i}_{kn+i-1}(\mathbf{v}_{E_i})$, and

$||V||$ multiplications to compute P_{kn+i}.

Therefore, if α, μ and δ measure the time units required by a single addition, multiplication and division, respectively, then the computational cost of the $(kn+i)$th iterative step amounts to

$$||V||(\alpha+\mu)+||E_i||\delta$$

time units, and the computational cost of each iteration cycle (i.e., n consecutive iterative steps) amounts to

$$n||V||(\alpha+\mu)+||\boldsymbol{H}||\delta$$

time units, where $||\boldsymbol{H}||$ denotes the size of $\boldsymbol{H}$.

Example 2 (*continued*). Let $\boldsymbol{P}$ be a consistent probabilistic database with scheme $\boldsymbol{H}$. We have to resort to the IPF procedure in order to compute the MEE of $\boldsymbol{P}$, and each iteration cycle requires

$$7||abcdefgh||(\alpha+\mu)+(||ad||+||ae||+||bcf||+||bd||+||ce||+||cg||+||ch||)\delta$$

time units.

In order to reduce the computational effort, a good strategy inspired by the divide-and-conquer principle consists in decomposing the problem of computing the MEE P of $\boldsymbol{P}$ into sub problems whose solutions can be easily combined to recover the global solution P (Badsberg, 1995; Malvestuto, 1989). As proven by Malvestuto (1997) in the case that the scheme $\boldsymbol{H}$ of $\boldsymbol{P}$ is a conformal hypergraph, and in (Malvestuto and Moscarini, 2000) for the general case, the best way of decomposing $\boldsymbol{H}$ consists in taking the compact components of $\boldsymbol{H}$, which requires a time polynomial in the dimension of $\boldsymbol{H}$ (Malvestuto and Moscarini, 2000).

Example 2 (*continued*)**.** Instead of applying the IPF procedure to $\boldsymbol{P}$, we first decompose $\boldsymbol{H}$ into its four compact components $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$ and $\boldsymbol{H}_4$ (see Fig. 5), and then determine the MEEs of the following four subdatabases of $\boldsymbol{P}$ with schemes $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$ and $\boldsymbol{H}_4$:

$$\boldsymbol{P}_1 = \{p_{ad}, p_{ae}, p_{bcf}^{\downarrow bc}, p_{bd}, p_{ce}\} \quad \boldsymbol{P}_2 = \{p_{bcf}\} \quad \boldsymbol{P}_3 = \{p_{cg}\} \quad \boldsymbol{P}_4 = \{p_{ch}\}.$$

Let $P^{(j)}$ be the MEE of $\boldsymbol{P}_j, 1 \leqslant 1 \leqslant j \leqslant 4$. Since $P^{(2)} = p_{bcf}$, $P^{(3)} = p_{cg}$ and $P^{(4)} = p_{ch}$ we only need to apply the IPF procedure to $\boldsymbol{P}_1$. Each iteration cycle for computing $P^{(1)}$ requires

$$5||abcde||(\alpha + \mu) + (||ad|| + ||ae|| + ||bc|| + ||bd|| + ||ce||)\delta$$

time units. After computing $P^{(1)}$ the MEE of $\boldsymbol{P}$ is then obtained using the formula

$$P^{(1)} p_{bcf} p_{cg} p_{ch} / p_{bcf}^{\downarrow bc} (p_{cg}^{\downarrow c})^2.$$

By the above-mentioned property of the compact components of a hypergraph, in the next sections without loss of generality we shall restrict our considerations to consistent probabilistic databases whose schemes are cyclic hypergraphs having compact vertex sets. The following will be used as a running example.

Example 4. Let $a, b, \ldots, h, i, j$ be nine random variables, of which i and j are ternary and the remaining ones are all binary. Consider a consistent probabilistic database $\boldsymbol{P}$ which scheme $\boldsymbol{H} = \{ac, ad, ae, af, ag, aj, be, bf, bg, bh, bij, cg, ch, ci, dh, di\}$. Since the 2-section of $\boldsymbol{H}$ (see Fig. 6) is not chordal, $\boldsymbol{H}$ is a cyclic hypergraph. Furthermore, the vertex set of $\boldsymbol{H}$ is compact since $\boldsymbol{H}$ contains no dividers.

3. Markovian propagation trees

Let $\boldsymbol{H}$ be a cyclic and reduced hypergraph whose vertex set is compact, and $\boldsymbol{P}$ a consistent probabilistic database with scheme $\boldsymbol{H}$. In order to compute the MEE P of $\boldsymbol{P}$ we make use the following three-phase procedure which combines the JP method with an ad-hoc tree-computation technique.

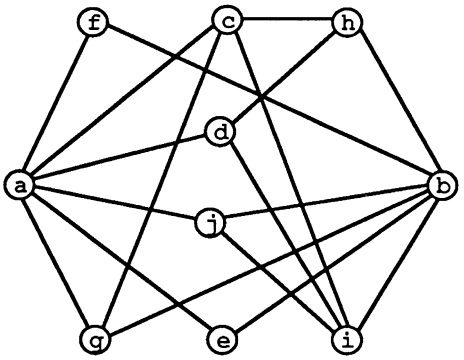

Fig. 6.

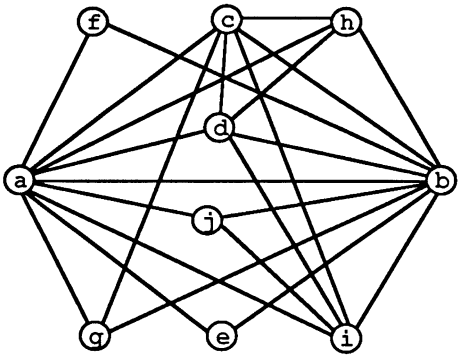

Fig. 7.

Initial phase. An acyclic cover $\boldsymbol{K}$ of $\boldsymbol{H}$ is constructed. For example, using the "fill-in" technique (Tarjan and Yannakakis, 1984), $\boldsymbol{K}$ is taken to be the clique hypergraph of $[\boldsymbol{H}]_2$ if $[\boldsymbol{H}]_2$ is chordal; otherwise, $\boldsymbol{K}$ is taken to be the clique hypergraph of the chordal graph obtained by adding "fill-in edges" to $[\boldsymbol{H}]_2$. Next, given $\boldsymbol{K}$, an edge-divider tree T of $\boldsymbol{K}$ is constructed.

Example 4 (*continued*)**.** Adding the fill-in edges (a,b), (a,i), (a,h), (b,c), (b,d) and (c,d), one obtains a chordal graph (see Fig. 7) whose clique hypergraph is $\boldsymbol{K}=\{abe, abf, abcg, abcdh, abcdi, abij\}$. The divider hypergraph of $\boldsymbol{K}$ is $\boldsymbol{D}=\{ab, abc, abcd, abi\}$. Fig. 8 shows an edge-divider tree of $\boldsymbol{K}$.

Iterative phase. Let $\boldsymbol{D}$ be the divider hypergraph of $\boldsymbol{K}$. Each approximation P_t of P, $t \geqslant 0$, is not explicitly computed but is implicitly determined by a probabilistic database $\boldsymbol{P}_t$ with scheme $\boldsymbol{K} \cup \boldsymbol{D}$ of which P_t is the Markov extension. Precisely, the probabilistic database $\boldsymbol{P}_0$ contains the marginals of the zero-order approximation P_0 on edges of $\boldsymbol{K} \cup \boldsymbol{D}$; next, given the probabilistic database $\boldsymbol{P}_t$, $t \geqslant 0$, the probabilistic database $\boldsymbol{P}_{t+1}$ is constructed in such a way that, if the corresponding iterative step requires fitting P_t to the probability distribution p_E from $\boldsymbol{P}, \boldsymbol{P}_{t+1}$ will contain the

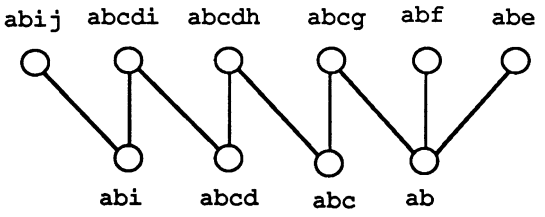

Fig. 8.

marginals of the distribution $P_{t+1} = (p_E/P_t^{\downarrow E})P_t$ on edges of $\boldsymbol{K} \cup \boldsymbol{D}$. Thus, P_{t+1} is the Markov extension of $\boldsymbol{P}_{t+1}$.

In our implementation of the iterative phase, we make use of the probabilistic database $\boldsymbol{P}$ and of two additional databases: a probabilistic database $\boldsymbol{Q}$ with scheme $\boldsymbol{K} \cup \boldsymbol{D}$ and a set of tables $\{r_Y : Y \in \boldsymbol{H} \cup \boldsymbol{D}\}$, where table r_Y reports a real number for each state of Y. Initially, each probability distribution q_X in $\boldsymbol{Q}$ is taken to be the uniform probability distribution on X and, for each table r_Y, the entries in r_Y are set to `nil`. Then, given an ordering $E_1, E_2, \ldots, E_n$ of edges of $\boldsymbol{H}$, at each iteration cycle the probabilistic database $\boldsymbol{Q}$ is updated n times by fitting the probability distributions $p_{E_1}, \ldots, p_{E_n}$ in $\boldsymbol{P}$. In order to fit p_{E_i}, the following algorithm is applied.

Markovian propagation (MP) algorithm

Step 1. Select an edge X_i of $\boldsymbol{K} \cup \boldsymbol{D}$ containing E_i.
For each state $\mathbf{e}$ of E_i, update the corresponding entry in the table r_{E_i} as follows

if $q_{X_i}^{\downarrow E_i}(\mathbf{e}) > 0$ then $r_{E_i}(\mathbf{e}) := p_{E_i}(\mathbf{e})/q_{X_i}^{\downarrow E_i}(\mathbf{e})$
else $r_{E_i}(\mathbf{e}) :=$ `nil`.

Step 2. Select an edge A_i of $\boldsymbol{K}$ containing E_i.
For each state $\mathbf{a}$ of A_i, update the corresponding entry in the probability distribution q_{A_i} (in $\boldsymbol{Q}$) by setting

$q_{A_i}(\mathbf{a}) := r_{E_i}(\mathbf{a}_{E_i}) q_{A_i}(\mathbf{a})$

with the convention `nil` $\times\, 0 = 0$.
Start a traversal of T at A_i with a backtrack-style search, e.g., the depth-first search (Golumbic, 1980). During the traversal of T, when an arc (A, B) is traversed, with A in $\boldsymbol{K}$ and B in $\boldsymbol{D}$, carry out the following operations depending on whether the arc is traversed from A to B or from B to A:

Case 1: If (A, B) is traversed from A to B then, for each state $\mathbf{b}$ of B, update the corresponding entries in the table r_B and in the probability distribution q_B (in $\boldsymbol{Q}$) as follows:

if $q_B(\mathbf{b}) > 0$ then $r_B(\mathbf{b}) := q_A^{\downarrow B}(\mathbf{b})/q_B(\mathbf{b})$ else $r_B(\mathbf{b}) :=$ `nil`;
$q_B(\mathbf{b}) := q_A^{\downarrow B}(\mathbf{b})$.

Case 2: If (A,B) is traversed from B to A then, for each state $\mathbf{a}$ of A, update the corresponding entry in the probability distribution q_A (in $\boldsymbol{Q}$) by setting:

$$q_A(\mathbf{a}) := r_B(\mathbf{a}_B) q_A(\mathbf{a})$$

with the convention $\mathtt{nil} \times 0 = 0$.

Remark 1. At Step 2 we adopted the convention $\mathtt{nil} \times 0 = 0$ and now we prove that it is just what we need to update the probability distributions q_A in $\boldsymbol{Q}$ for each A in $\boldsymbol{K}$. For example, consider the updating of q_{A_i} and let q'_{A_i} be its updated version. At the beginning of Step 2, for each state $\mathbf{a}$ of A_i we set

$$q'_{A_i}(\mathbf{a}) = r_{E_i}(\mathbf{a}_{E_i}) q_{A_i}(\mathbf{a}).$$

Now, if $r_{E_i}(\mathbf{a}_{E_i}) = \mathtt{nil}$ then $q_{X_i}^{\downarrow E_i}(\mathbf{a}_{E_i}) = 0$ (see Step 1). By the consistency of $\boldsymbol{Q}$, we also have that $q_{A_i}^{\downarrow E_i}(\mathbf{a}_{E_i}) = 0$ which implies that $q_{A_i}(\mathbf{a}) = 0$. On the other hand, by the proportionality relationship linking the Markov extensions of the updated and current version of $\boldsymbol{Q}$, if $q_{A_i}(\mathbf{a}) = 0$ then $q'_{A_i}(\mathbf{a})$ must be zero, which is exactly what we obtain using the convention $\mathtt{nil} \times 0 = 0$. The same arguments can be applied to each probability distribution q_A in $\boldsymbol{Q}$ with A in $\boldsymbol{K}$.

Final phase. When the convergence is attained, then P is computed from $\boldsymbol{Q}$ using formula (1).

Theorem 1. *The Markovian propagation algorithm is correct.*

Proof. Let $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ be the contents of the probabilistic database before and after executing the ith iteration. We now show that

(i) $\boldsymbol{Q}'$ is consistent, and

(ii) if Q and Q' are respectively the Markov extensions of $\boldsymbol{Q}$ and $\boldsymbol{Q}'$, then $Q' = (p_{E_i}/Q^{\downarrow E_i})Q$.

To prove (i), owing to the acyclicity of $\boldsymbol{K}$, it is sufficient to prove that $\boldsymbol{Q}'$ is pairwise consistent or, more simply, that for every two nondisjoint edges A and A' of $\boldsymbol{K}$ one has $(q'_A)^{\downarrow A \cap A'} = (q'_{A'})^{\downarrow A \cap A'}$. To achieve this, note that, for each a in $A \cap A'$, each node along the undirected path $(A = A_1, B_1, \ldots, A_h, B_h, A_{h+1}, \ldots, A_k = A')$ in T joining A and A' contains a so that, for each B_h, q'_{B_h} always coincides with the marginals of q'_{A_h} and $q'_{A_{h+1}}$ on B_h. It follows that the marginals of q'_A and $q'_{A'}$ on $A \cap A'$ do coincide.

To prove (ii), we begin by writing Q and Q' using formula (1):

$$Q = q_{A_i}\rho \qquad \text{and} \qquad Q' = q'_{A_i}\sigma,$$

where

$$\rho = \prod_{A \in \boldsymbol{K} \setminus \{A_i\}} q_A \Big/ \prod_{B \in \boldsymbol{D}} q_B^{g(B)-1} \tag{2}$$

and

$$\sigma = \prod_{A \in \boldsymbol{K} \setminus \{A_i\}} q'_A \Bigg/ \prod_{B \in \boldsymbol{D}} q'^{g(B)-1}_B. \tag{3}$$

On the other hand, as at the beginning of Step 2 we set

$$q'_{A_i} = r_{E_i} q_{A_i},$$

it is sufficient to prove that $\rho = \sigma$. To achieve this, consider any edge A of $\boldsymbol{K}$ distinct from A_i and suppose that node A is reached from node B. Then

$$q'_A = r_B q_A. \tag{4}$$

By substituting (4) into (3) and on account of (2) one has

$$\begin{aligned}\sigma &= \left[\prod_{B \in \boldsymbol{D}} r_B^{g(B)-1}\right] \left[\prod_{A \in \boldsymbol{K} \setminus \{A_i\}} q_A\right] \Bigg/ \left[\prod_{B \in \boldsymbol{D}} q'^{g(B)-1}_B\right] \\ &= \left[\prod_{A \in \boldsymbol{K} \setminus \{A_i\}} q_A\right] \Bigg/ \left[\prod_{B \in \boldsymbol{D}} q_B^{g(B)-1}\right] = \rho. \qquad \square\end{aligned}$$

From the computational point of view, first of all observe that the complexity of the MP algorithm is sensitive to the acyclic cover $\boldsymbol{K}$ of $\boldsymbol{H}$ that was constructed at the initial phase by triangulating the graph $[\boldsymbol{H}]_2$. In what follows, we focus on the Iterative Phase of the JP method and assume we are given the acyclic hypergraph $\boldsymbol{K}$ constructed at the Initial Phase hopefully in a reasonable way. We now analyse the running time of the MP algorithm (that is, the computational cost of an iterative step of our implementation of the IPF procedure) and discuss its dependence on the input tree T in order to get an edge-divider tree of $\boldsymbol{K}$ that minimises the running time of the MP algorithm.

Step 1 requires $||X_i||$ additions and $||E||$ divisions to update q_{E_i}, that is,

$$||X_i||\alpha + ||E_i||\delta$$

time units. Step 2 requires $||A_i||$ multiplications to update q_{A_i}, that is,

$$||A_i||\mu$$

time units. Next, during the traversal of T, when $\boldsymbol{D}$-node B is reached from $\boldsymbol{K}$-node A, one needs $||A||$ additions and $||B||$ divisions to update r_B and q_B and when $\boldsymbol{K}$-node

A is reached, one needs $||A||$ multiplications to update q_A. To sum up, the time units required by Step 2 amount to

$$\left\{ g(A_i)||A_i|| + \sum_{A \in \boldsymbol{K} \setminus \{A_i\}} [g(A) - 1]||A|| \right\} \alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}||\delta$$

$$= \left\{ ||A_i|| + \sum_{A \in \boldsymbol{K}} [g(A) - 1]||A|| \right\} \alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}||\delta,$$

where, by Lemma 1, $g(A)$ is the number of neighbours of $\boldsymbol{K}$-node A in T. Consequently, the execution of the MP algorithm requires

$$\left\{ ||X_i|| + ||A_i|| + \sum_{A \in \boldsymbol{K}} [g(A) - 1]||A|| \right\} \alpha + ||\boldsymbol{K}||\mu + (||\boldsymbol{D}|| + ||E_i||)\delta$$

time units and, for a fixed T, is minimum if X_i is chosen among minimum-size edges of $\boldsymbol{K} \cup \boldsymbol{D}$ containing E_i, and A_i is chosen among minimum-size edges of $\boldsymbol{K}$ containing E_i.

We now discuss the dependence of the running time of the MP algorithm on the input edge-divider tree T of $\boldsymbol{K}$. It is clear that the running time is minimum if T minimises the quantity

$$\sum_{A \in \boldsymbol{K}} g(A)||A||.$$

This quantity can be re-written as

$$\sum c(A, B),$$

where the summation is extended over all arcs of T, and $c(A, B)$, we call the *cost* of arc (A, B), is taken to be $||A||$. Therefore, the input tree of the MP algorithm should be a minimum-cost edge-divider tree of $\boldsymbol{K}$. One can easily construct such a tree bearing in mind that edge-divider trees of $\boldsymbol{K}$ coincide with maximum-weight spanning trees of the edge-divider graph of $\boldsymbol{K}$; thus, one has an efficient algorithm for computing a minimum-cost edge-divider tree of $\boldsymbol{K}$ by modifying any of the above-mentioned maximum-weight spanning-tree algorithms in such a way that arc cost is used as a second search priority (Jensen and Jensen, 1994).

Example 4 (*continued*)**.** Fig. 9 shows the edge-divider graph $G(\boldsymbol{K})$ of the clique hypergraph $\boldsymbol{K}$ of the graph of Fig. 7. The costs of the edges of $G(\boldsymbol{K})$ are listed below:

$$c(abij, abi) = c(abij, ab) = 36$$

$$c(abcdi, abi) = c(abcdi, abcd) = c(abcdi, abc) = c(abcdi, ab) = 48$$

$$c(abcdh, abcd) = c(abcdh, abc) = c(abcdh, ab) = 32$$

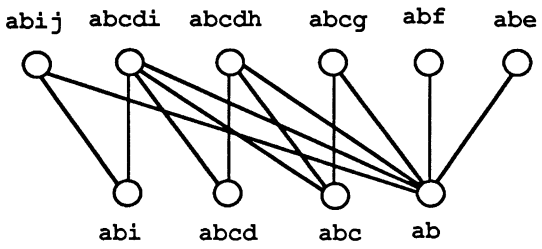

Fig. 9.

$$c(abcg, abc) = c(abcg, ab) = 16$$
$$c(abf, ab) = 8$$
$$c(abe, ab) = 8.$$

Then, a minimum-cost edge-divider tree of $\boldsymbol{K}$ is the tree shown in Fig. 8.

If T is the input tree of the MP algorithm and A_i is the edge of $\boldsymbol{K}$ selected at the beginning of Step 2, then the updating process can be summarised by the directed tree $\mathscr{M}_i$ which is obtained from the traversal tree of T (started at A_i) with the addition of the directed arc $E_i \to A_i$. A tree such as $\mathscr{M}_i$ is referred to as a *Markovian propagation tree* from E_i to $\boldsymbol{K}$ and by the *complexity* of $\mathscr{M}_i$ we mean the amount of the time units needed to execute Step 2, that is,

$$\left\{ ||A_i|| + \sum_{A \in \boldsymbol{K}} [g(A) - 1]||A|| \right\} \alpha + ||\boldsymbol{K}||\mu + (||\boldsymbol{D}||)\delta.$$

Note that, if for each $\boldsymbol{D}$-node B of $\mathscr{M}_i$ we denote by $par(B, \mathscr{M}_i)$ the parent of B in $\mathscr{M}_i$, then the complexity of $\mathscr{M}_i$ can be also written as

$$\left[\sum_{B \in \boldsymbol{D}} ||\, par(B, \mathscr{M}_i)|| \right] \alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}||\delta.$$

Finally, if A_i is a minimum-size edge of $\boldsymbol{K}$ containing E_i, and if T is a minimum-cost edge-divider tree of $\boldsymbol{K}$ then the complexity of $\mathscr{M}_i$ is reduced to a minimum and $\mathscr{M}_i$ is here called an *optimal Markovian propagation tree* from E_i to $\boldsymbol{K}$.

Example 4 (*continued*)**.** Fig. 10 shows optimal Markovian propagation trees from the edges of $\boldsymbol{H}$ to $\boldsymbol{K}$ all of which are supported by the edge-divider tree of Fig. 7.

Finally, by repeatedly applying the MP algorithm, one for each edge of $\boldsymbol{H}$, we obtain an efficient implementation of the iterative phase. Note that the running time of such an implementation does not depend on the ordering the edges of $\boldsymbol{H}$ according to which the probability distributions in $\boldsymbol{P}$ are processed.

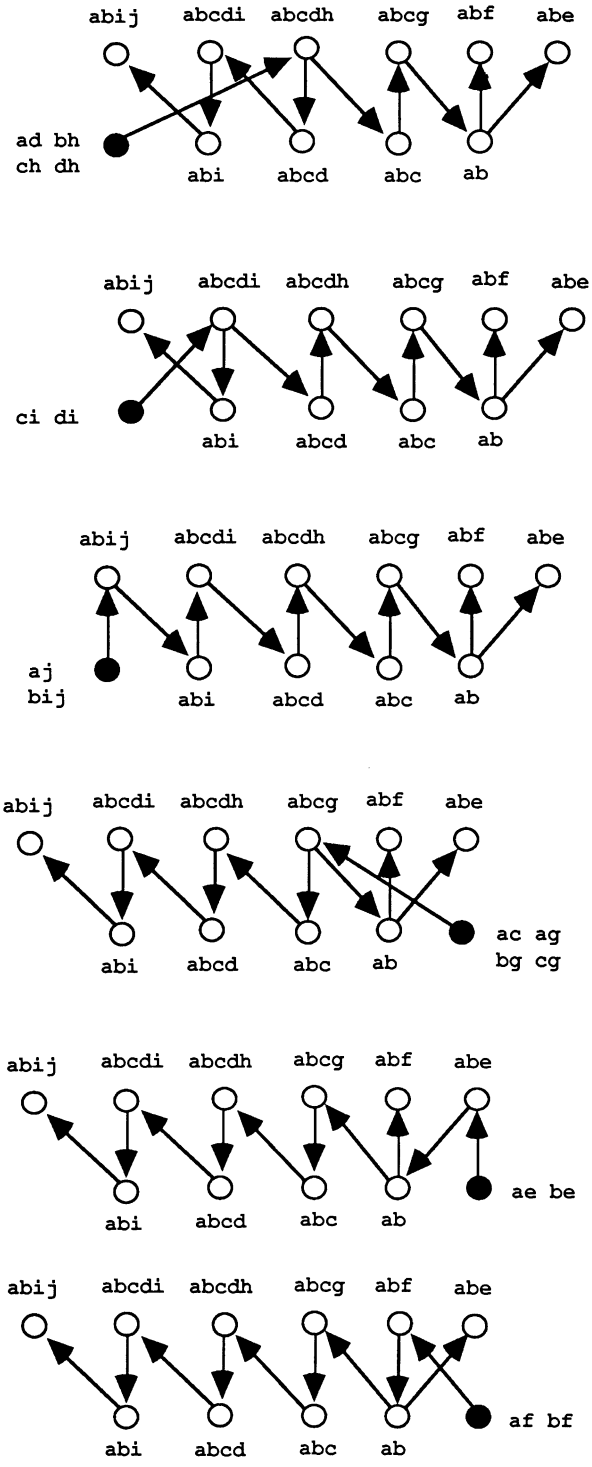
abij abcdi abcdh abcg abf abe
ad bh
ch dh
abi abcd abc ab
abij abcdi abcdh abcg abf abe
ci di
abi abcd abc ab
abij abcdi abcdh abcg abf abe
aj
bij
abi abcd abc ab
abij abcdi abcdh abcg abf abe
ac ag
bg cg
abi abcd abc ab
abij abcdi abcdh abcg abf abe
ae be
abi abcd abc ab
abij abcdi abcdh abcg abf abe
af bf
abi abcd abc ab

Fig. 10.

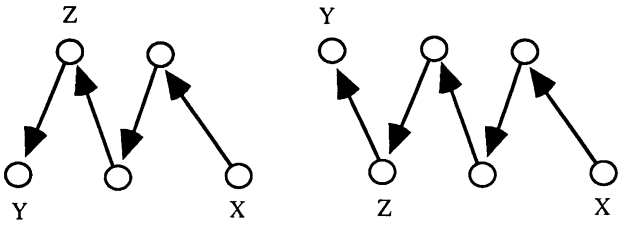

Fig. 11.

4. Fast propagation trees

In this section we present a propagation algorithm which is not less efficient than the MP algorithm. First of all, observe that the execution of the MP algorithm, even if it corresponds to an optimal Markovian propagation tree, may suffer from the following two limitations: for some $\boldsymbol{D}$-node B of $\mathscr{M}_i$, the distribution q_B might be updated in a nonoptimal way (Almond and Kong, 1993; Jensen and Jensen, 1994) and, worst of all, might be avoided. These limitations are apparent in the first Markovian propagation tree of Fig. 10 for $E_i = ad$. For example, the probability distribution q_{ab} can be updated more efficiently by marginalizing q_{abc} and, moreover, given the table r_{ad}, one can soon update not only q_{abcdh} but also q_{abcdi} so that it is useless to update q_{abcd} and r_{abcd}. The following lemma generalises the considerations above to an arbitrary Markovian propagation tree from E_i to $\boldsymbol{K}$.

Lemma 2. *Let $\mathscr{M}_i$ be a Markovian propagation tree from E_i to $\boldsymbol{K}$, and X a node of $\mathscr{M}_i$ from $\{E_i\} \cup \boldsymbol{D}$. If Y is a node of $\mathscr{M}_i$ that contains X and is a proper descendant of X, then the probability distribution q_Z can be updated by setting $q_Y := r_X q_Y$.*

Proof. We prove the statement by induction on the length k of the directed path from X to Y. Let $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ be the contents of the probabilistic database before and after executing the ith iteration.

(*Basis*) For $k = 1$, Y is a $\boldsymbol{K}$-node and $q'_Y = r_X q_Y$ which proves the statement.

(*Induction*) Assume that the statement holds for $k \geqslant 1$; we shall prove that it also holds for $k+1$. Assume that the directed path from X to Y in $\mathscr{M}_i$, say $(X, \ldots, Z, Y)$, has length $k+1$. Since X is a subset of Y, by the separation property X is a subset of Z. The directed path from X to Z has length k so that, by the inductive hypothesis, one has

$$q'_Z = r_X q_Z. \tag{5}$$

Let us now distinguish two cases depending on whether Y is a $\boldsymbol{D}$-node or a $\boldsymbol{K}$-node. Fig. 11 illustrates the two cases.

Case 1. Y is a $\boldsymbol{D}$-node and Z is a $\boldsymbol{K}$-node so that $Y \subseteq Z$ and, hence, q_Y equals the marginal of q_Z on Y. On the other hand, since $X \subseteq Y \subseteq Z$, summing up both sides

of formula (5) over all variables in $Z \setminus Y$, one obtains

$$q'_Y = r_X q_Y,$$

which proves the statement.

Case 2. Y is a $\boldsymbol{K}$-node and Z is a $\boldsymbol{D}$-node so that Z is a subset of Y and, hence,

$$q'_Y = r_Z q_Y.$$

Moreover, by (5) one has $r_Z = r_X$ which proves the statement. □

The next algorithm takes as input an optimal Markovian propagation tree $\mathscr{M}_i$ from E_i to $\boldsymbol{K}$ and produces a directed tree $\mathscr{F}_i$ rooted at E_i which contains all $\boldsymbol{K}$-nodes of $\mathscr{M}_i$ and a subset of dividers of $\boldsymbol{K}$. Tree $\mathscr{F}_i$ will be referred to as a *fast propagation tree* from E_i to $\boldsymbol{K}$.

Fast propagation tree (FPT) algorithm

Input: An optimal Markovian propagation tree $\mathscr{M}_i$ from E_i to $\boldsymbol{K}$.
Output: A directed tree $\mathscr{F}_i$.
Step 1. Create a list named *Top* containing the topological sort of the $\boldsymbol{D}$-nodes of $\mathscr{M}_i$, that is, if B precedes B' in *Top* then B is not a descendant of B'.
Step 2. $\mathscr{F}_i := \mathscr{M}_i$; mark "unvisited" all $\boldsymbol{K}$-nodes of $\mathscr{F}_i$; $X{:=}E_i$.
Step 3. For each unvisited $\boldsymbol{K}$-node A of $\mathscr{F}_i$ that contains X and is a descendant of X, do:
  mark A "visited";
  delete the incoming arc of A from $\mathscr{F}_i$;
  add arc $X \to A$ to $\mathscr{F}_i$.
Step 4. If *Top* is empty then go to (6).
Step 5. Set X to the first $\boldsymbol{D}$-node in *Top*. Delete X from *Top*. If X is a leaf of $\mathscr{F}_i$ then delete X from $\mathscr{F}_i$ and go to (4). Otherwise, go to (3).
Step 6. Let $B_1, \ldots, B_k$ be the $\boldsymbol{D}$-nodes of $\mathscr{F}_i$ arranged in non decreasing order of their cardinalities.
  For $h = 1, \ldots, k$ do:
    Among all the nodes of $\mathscr{F}_i$ that are not descendants of B_h, find a minimum-size superset X of B_h;
    delete the incoming arc of B_h from $\mathscr{F}_i$;
    add arc $X \to B_h$ to $\mathscr{F}_i$.

Example 4 (*continued*)**.** Let us apply the FPT algorithm to the (optimal) Markovian propagation tree from $E = ad$ to $\boldsymbol{K}$ shown in Fig. 10. At step 1, the list $Top = (abcd, abc, ab, abi)$ is created. After executing Steps 2 and 3, the current content of $\mathscr{F}$ is shown in Fig. 12. The $\boldsymbol{D}$-node *abcd* is removed from *Top* and $\mathscr{F}$ (Step 5). The current content of $\mathscr{F}$ is shown in Fig. 13.

The removal of the remaining $\boldsymbol{D}$-nodes from *Top* does not change the content of $\mathscr{F}$. Finally, Step 6 produces the tree shown in Fig. 14. Fig. 15 shows the outputs

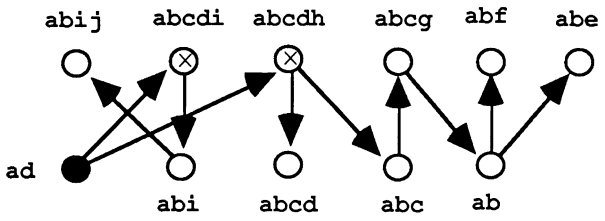

Fig. 12.

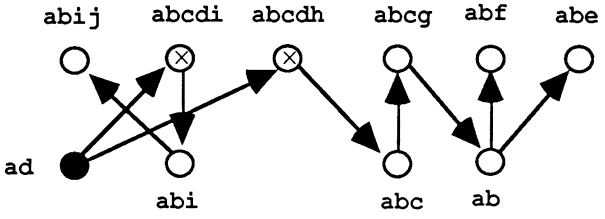

Fig. 13.

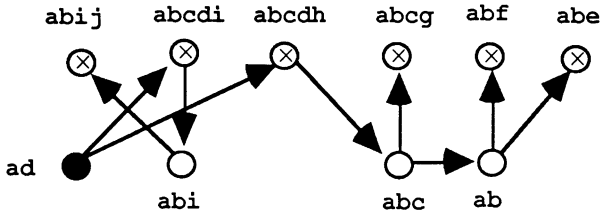

Fig. 14.

of the FPT algorithm when it is applied to the (optimal) Markovian propagation trees of Fig. 10. They are arranged in non increasing order of the number of their ***D***-nodes.

The following algorithm is alternative to the MP algorithm of Section 3.

Fast propagation (FP) algorithm

Step 1. Find a node X_i of $\mathscr{F}_i$ containing E_i, and set $r_{E_i} := p_{E_i}/(q_{X_i})^{\downarrow E_i}$.

Step 2. Perform a traversal of $\mathscr{F}_i$ with start point E_i. During the traversal of $\mathscr{F}_i$, when arc $Y \to Z$ is traversed, carry out the following operations depending on whether Y is a subset or a superset of X.
Case 1: If Z is a subset of Y, then set $r_Z := (q_Y)^{\downarrow Z}/q_Z$ and $q_Z := (q_Y)^{\downarrow Z}$.
Case 2: If Z is a superset of Y, then set $q_Z := r_Y q_Z$.

We can define the complexity of a fast propagation tree $\mathscr{F}_i$ in the same way as for $\mathscr{M}_i$; that is, the complexity of $\mathscr{F}_i$ measures the time units needed to execute Step 2

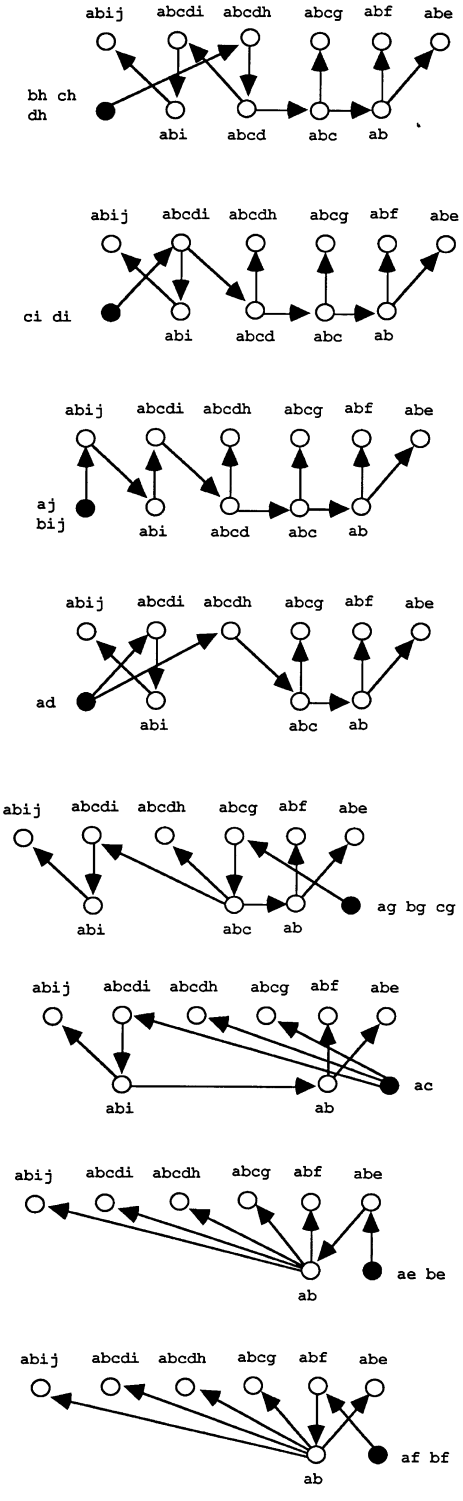
abij abcdi abcdh abcg abf abe
bh ch dh
abi abcd abc ab
abij abcdi abcdh abcg abf abe
ci di
abi abcd abc ab
abij abcdi abcdh abcg abf abe
aj bij
abi abcd abc ab
abij abcdi abcdh abcg abf abe
ad
abi abc ab
abij abcdi abcdh abcg abf abe
abi abc ab
ag bg cg
abij abcdi abcdh abcg abf abe
abi ab
ac
abij abcdi abcdh abcg abf abe
ab
ae be
abij abcdi abcdh abcg abf abe
ab
af bf

Fig. 15.

of the FP algorithm. Let $\boldsymbol{D}_i$ be the set of $\boldsymbol{D}$-nodes of $\mathscr{F}_i$. If for each B in $\boldsymbol{D}_i$, we denote by $par(B, \mathscr{F}_i)$ the parent of B in $\mathscr{F}_i$, then the complexity of $\mathscr{F}_i$ is given by the quantity

$$\left[\sum_{B\in\boldsymbol{D}_i} ||\,par(B,\mathscr{F}_i)||\right]\alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}_i||\delta.$$

Theorem 2. *The fast propagation algorithm is correct and its running time is not greater than the running time of the MP algorithm.*

Proof. The correctness of the FP algorithm easily follows from Theorem 1 and Lemma 2. We now show that the FP algorithm is not less efficient than the MP algorithm. Let m and f be the complexities of $\mathscr{M}_i$ and $\mathscr{F}_i$, respectively; that is,

$$m = \left[\sum_{B\in\boldsymbol{D}} ||\,par(B,\mathscr{M}_i)||\right]\alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}||\delta$$

and

$$f = \left[\sum_{B\in\boldsymbol{D}_i} ||\,par(B,\mathscr{F}_i)||\right]\alpha + ||\boldsymbol{K}||\mu + ||\boldsymbol{D}_i||\delta.$$

Consider the quantity

$$m - f = \left\{\sum_{B\in\boldsymbol{D}\setminus\boldsymbol{D}_i} ||\,par(B,\mathscr{M}_i)|| + \sum_{B\in\boldsymbol{D}_i}[||\,par(B,\mathscr{M}_i)|| - ||\,par(B,\mathscr{F}_i)||]\right\}\alpha + (||\boldsymbol{D}|| - ||\boldsymbol{D}_i||)\delta.$$

Since the terms $\sum_{B\in\boldsymbol{D}\setminus\boldsymbol{D}_i} ||\,par(B,\mathscr{M}_i)||$ and $(||\boldsymbol{D}|| - ||\boldsymbol{D}_i||)$ are never negative, it is sufficient to prove that

$$\sum_{B\in\boldsymbol{D}_i}[||\,par(B,\mathscr{M}_i)|| - ||\,par(B,\mathscr{F}_i)||]$$

is never negative. But this follows from the fact that, for each B in $\boldsymbol{D}_i$, if A is the parent of B in $\mathscr{M}_i$ then A is also a node of $\mathscr{F}_i$ and A is not a descendant of B in $\mathscr{F}_i$ so that the size of A is not greater than the size of $par(B, \mathscr{F}_i)$, which is a minimum-size superset of B among the nodes of $\mathscr{F}_i$. □

Example 4 (*continued*)**.** Let m and f be the complexities of the optimal Markovian propagation tree from ad to $\boldsymbol{K}$ and of the corresponding fast propagation tree. Then, one has $m - f = 40\alpha + 16\delta$.

Remark 2. Using the FP algorithm, the probability distributions q_B in $\boldsymbol{Q}$ for each $\boldsymbol{B}$ in $\boldsymbol{D}\setminus\boldsymbol{D}_i$ are not updated. Thus, before starting the next iteration, a preliminary computation is needed to update the probability distributions q_B for each B from $\boldsymbol{D}'_i$ where $\boldsymbol{D}'_i = \boldsymbol{D}_{i+1}\setminus\boldsymbol{D}_i$ if $i < n$ and $\boldsymbol{D}'_i = \boldsymbol{D}_1\setminus\boldsymbol{D}_i$ if $i = n$. To achieve this, it is sufficient to add the following statement to the FTP algorithm:

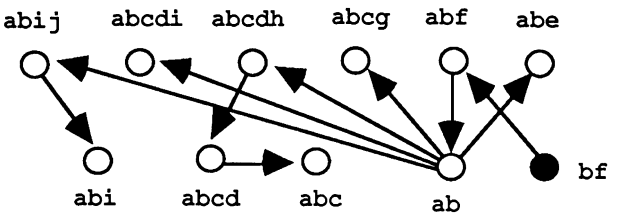

Fig. 16.

Step 7. Let $B_1,\ldots,B_k$ be the sets in $\boldsymbol{D}'_i$ arranged in non decreasing order of their cardinalities. For $h=1,\ldots,k$ do:
Among all the nodes of $\mathscr{F}_i$,
find a minimum-size superset X of B_h and add arc $X \to B_h$ to $\mathscr{F}_i$.

Example 4 (*continued*). Let $E_n = bf$ and $E_1 = bh$. Then, $\boldsymbol{D}_n = \{ab\}$ and $\boldsymbol{D}_1 = \{ab, abc, abcd, abi\}$ (see Fig. 15). After executing Step 7, the fast propagation tree from E_n to $\boldsymbol{K}$ changes to the tree shown in Fig. 16.

In order to reduce the additional computation due to Step 7 in each iteration cycle, it is reasonable to order the edges of $\boldsymbol{H}$ in the same order as the fast propagation trees appear in Fig. 15. Thus, Step 7 is actually executed only for the last edge (bf) of $\boldsymbol{H}$ since the set of $\boldsymbol{D}$-nodes in the corresponding fast propagation tree does not contain the $\boldsymbol{D}$-nodes *abc*, *abcd*, *abi* of the first propagation tree which corresponds to the edge *bh* of $\boldsymbol{H}$.

5. Closing remarks

We proposed an implementation of the IPF procedure based on Markovian propagation trees, which has the following two nice properties:

(1) given a distribution from the input probabilistic database, an optimal Markovian propagation tree can be constructed in an efficient way, and
(2) the implementation of the IPF procedure is independent of the order in which the distributions in the input probabilistic database are processed.

We also showed that an optimal Markovian propagation tree $\mathscr{M}$ may still suffer from some undesirable computational aspects, which can be removed by "simplifying" $\mathscr{M}$ with an efficient algorithm that still produces a tree $\mathscr{F}$, we called the fast propagation tree associated to $\mathscr{M}$. We then provided a propagation algorithm with input $\mathscr{F}$ which has a computational cost not greater than the propagation algorithm with input $\mathscr{M}$. To sum up, the implementation of the IPF procedure by fast propagation trees is never less efficient than the implementation by Markovian propagation trees. Before closing, we wish to point out that the implementation of the IPF procedure by fast propagation trees turns out to depend on the order in which the probability distributions in the input database are processed (see Remark 2 at the end of Section 4), and we conjecture that the problem of finding an optimal ordering of the input distributions is hard from a complexity-theoretic point of view.

References

Ahuja, R.K., Magnanti, M.L., Orlin, J.B., 1993. Network Flows. Prentice-Hall, Englewood Cliffs, NJ.

Almond, R., Kong, A., 1993. Optimality issues in constructing a Markov tree from graphical models. Unpublished manuscript.

Arnborg, S., Corneil, D.G., Proskurowski, A., 1993. Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Math. 9, 277–284.

Badsberg, J.H., 1995. An environment for graphical models. Part I: efficient algorithms for estimation in contingency tables. Ph.D. Thesis, Department of Mathematics and Computer Science, Ålborg University, Denmark.

Badsberg, J.H., 1996. Decomposition of graphs and hypergraphs with identification of conformal hypergraphs. Research Report R 96-2032. Department of Mathematics & Computer Science, Ålborg University, Denmark.

Becker, A., Geiger, D., 1997. A sufficiently fast algorithm for finding close to optimal junction trees. Abstracts of the II IASC World Conference, Pasadena.

Beeri, C., Fagin, R., Maier, D., Yannakakis, M., 1983. On the desirability of acyclic database schemes. J. Assoc. Comput. Mach. 30, 479–513.

Berge, C., 1989. Hypergraphs. North-Holland, Amsterdam.

Bishop, Y.M.M., Fienberg, S.E., Holland, P.W., 1975. Discrete Multivariate Analysis, Theory and Practice. MIT Press, Cambridge, MA.

Bodlaender, H.L., 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317.

Csiszár, I., 1975. I-divergence geometry of probability distributions and minimization problems. Ann. Probab. 3, 146–158.

Darroch, J.N., Lauritzen, S.L., Speed, M.P., 1980. Markov fields and log–linear interaction models for contingency tables. Ann. Statist. 8, 522–539.

Dawid, A.P., 1992. Applications of a general propagation algorithm for probabilistic expert systems. Statist. Comput. 2, 1992.

Deming, W.E., Stephan, F.F., 1940. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. Ann. of Math. Statist. 11, 427–444.

Denteneer, D., Verbeek, A., 1986. A fast algorithm for iterative proportional fitting in log–linear models. Comput. Statist. Data Anal. 3, 251–264.

Diestel, R., 1990. Graph Decompositions. Clarendon Press, Oxford.

Golumbic, M.C., 1980. Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York.

Haberman, S.J., 1974. Log–linear Models for Contingency Tables. University of Chicago Press, Chicago.

Jensen, F.V., 1988. Junction trees and decomposable hypergraphs. Research Report, JUDEX DATA SYSTEMER, Ålborg, Denmark.

Jensen, F.V., Jensen, F., 1994. Optimal junction trees, Proceedings of the X Conference on Uncertainty in Artificial Intelligence, pp. 400–406.

Jirousek, R., 1991. Solution of the marginal problem and decomposable distributions. Kybernetika 27, 403–412.

Jirousek, R., Preucil, S., 1995. On the effective implementation of the iterative proportional fitting procedure. Comput. Statist. Data Anal. 19, 177–189.

Kellerer, H.G., 1964. Verteilungsfunktionen mit gegebenen Marginalverteilungen. Z. Wahrscheinlichkeitstheorie Verw. Gebiete 3, 247–270.

Kjærluff, U., 1992. Optimal decomposition of probabilistic networks by simulated annealing. Comput. Statist. Data Anal. 2, 7–17.

Kloks, M., 1994. Treewidth: Computations and Approximations. Lecture Notes in Computer Science, Vol. 842. Springer, Berlin.

Larrañaga, P., Kuijpers, C.M.H., Poza, M., Murga, R.H., 1997. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. Statist. Comput. 7, 19–34.

Lauritzen, S.L., 1992. Propagation of probabilities. J. Amer. Statist. Assoc. 87, 420.

Lauritzen, S.L., 1996. Graphical Models. Oxford Science Publications, Clarendon Press, Oxford.
Lauritzen, S.L., Speed, M.P., Vijayan, K., 1984. Decomposable graphs and hypergraphs. J. Austral. Math. Soc. Ser. E 36, 12–29.
Lauritzen, S.L., Spiegelhalter, D.J., 1988. Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statist. Soc. Ser. A 50, 157–224.
Leimer, H.-G., 1993. Optimal decompositions by clique-separators. Discrete Math. 113, 99–123.
Maier, D., 1983. The Theory of Relational Databases. Computer Science Press, Rockville, MD.
Malvestuto, F.M., 1988. Existence of extensions and product extensions for discrete probability distributions. Discrete Math. 69, 61–77.
Malvestuto, F.M., 1989. Computing the maximum-entropy extension of given discrete probability distributions. Comput. Statist. Data Anal. 8, 299–311.
Malvestuto, F.M., 1997. Computing marginals in graphical log–linear models. In: Wegman, E.J., Azen, S.P. (Eds.), Proceedings of the II World Congress of the IASC, Computing Science and Statistics, Vol. 29(2). Pasadena, CA, pp. 52–58.
Malvestuto, F.M., 2001. A hypergraph-theoretic analysis of collapsibility and decomposability for extended log-linear models. Statistics and computing 11, 155–169.
Malvestuto, F.M., Moscarini, M., 1998. A fast algorithm for query optimization in universal-relation databases. J. Comput. System Sci. 56, 299–309.
Malvestuto, F.M., Moscarini, M., 2000. Decomposition of a hypergraph by partial-edge separators. Theoret. Comput. Sci. 237 (1–2), 57–79.
Rose, D.J., Tarjan, R.E., Lueker, G.S., 1976. Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5, 266–283.
Sanders, D.P., 1995. On linear recognition of tree-width at most four. SIAM J. Discrete Math. 9, 101–117.
Shafer, G., 1996. Probabilistic Expert Systems. CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 67. SIAM, Philadelphia, PA.
Shibata, Y., 1988. On the tree representation of chordal graphs. J. Graph Theory 12, 421–428.
Spiegelhalter, D.J., 1986. Probabilistic reasoning in predictive expert systems. In: Kanal, L.N., Lemmer, J.F. (Eds.), Proceedings of the Conference on Uncertainty in Artificial Intelligence. North-Holland, Amsterdam.
Tarjan, R.E., 1985. Decomposition by clique-separators. Discrete Math. 55, 221–232.
Tarjan, R.E., Yannakakis, M., 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13, 566–579.
Vorob'ev, N.N., 1962. Consistent families of measures and their extensions. Theory Probab. Appl. 7, 147–163.
Vorob'ev, N.N., 1963. Markov measures and Markov extensions. Theor. Prob. Appl. 8, 420–429.
Wen, W.X., 1989. Optimal decomposition of belief networks. Technical Report, Dept. of CS, The University of Melbourne.
Wen, W.X., 1990. Optimal decomposition of belief networks. Proceedings of the VI Workshop on Uncertainty in Artificial Intelligence. pp. 245–256.
Yannakakis, M., 1981. Computing the minimum fill-in is NP-complete. SIAM J. Algebraic Discrete Math. 2, 77–79.