J Intell Inf Syst (2009) 33:145–178
DOI 10.1007/s10844-008-0073-4

A join tree probability propagation architecture for semantic modeling

C. J. Butz · H. Yao · S. Hua

Received: 29 September 2006 / Revised: 19 November 2007 /
Accepted: 22 August 2008 / Published online: 25 September 2008

Abstract We propose the first *join tree* (JT) propagation architecture that labels the probability information passed between JT nodes in terms of *conditional probability tables* (CPTs) rather than potentials. By modeling the task of inference involving evidence, we can generate three work schedules that are more time-efficient for LAZY propagation. Our experimental results, involving five real-world or benchmark Bayesian networks (BNs), demonstrate a reasonable improvement over LAZY propagation. Our architecture also models inference not involving evidence. After the CPTs identified by our architecture have been physically constructed, we show that each JT node has a sound, local BN that preserves all conditional independencies of the original BN. Exploiting inference not involving evidence is used to develop an automated procedure for building multiply sectioned BNs. It also allows direct computation techniques to answer localized queries in local BNs, for which the empirical results on a real-world medical BN are promising. Screen shots of our implemented system demonstrate the improvements in semantic knowledge.

Keywords Bayesian networks · Join trees · Probabilistic inference · Conditional independence

1 Introduction

Bayesian networks (BNs) (Castillo et al. 1997; Cowell et al. 1999; Hájek et al. 1992; Jensen 1996; Neapolitan 1990; Pearl 1988; Xiang 2002) are a clear and concise

C. J. Butz (✉) · H. Yao · S. Hua
Department of Computer Science, University of Regina, Regina, S4S 0A2, Canada
e-mail: butz@cs.uregina.ca

H. Yao
e-mail: yao2hong@cs.uregina.ca

S. Hua
e-mail: huash111@cs.uregina.ca

semantic modeling tool for managing uncertainty in complex domains. A BN consists of a *directed acyclic graph* (DAG) and a corresponding set of *conditional probability tables* (CPTs). The *probabilistic conditional independencies* (Wong et al. 2000) encoded in the DAG indicate that the product of the CPTs is a *joint probability distribution* (JPD). Although Cooper (1990) has shown that the complexity of exact inference in BNs is NP-hard, several approaches have been developed that apparently work quite well in practice. One approach, called *multiply sectioned Bayesian networks* (MSBNs) (Xiang 1996, 2002; Xiang and Jensen 1999; Xiang et al. 2006, 2000, 1993), performs inference in sections of a BN. A second approach, called *direct computation* (DC) (Dechter 1996; Li and D'Ambrosio 1994; Zhang 1998), answers queries directly in the original BN. Although we focus on a third approach called *join tree propagation*, in which inference is conducted in a *join tree* (JT) (Pearl 1988; Shafer 1996) constructed from the DAG of a BN, our work has practical applications in all three approaches.

Shafer emphasizes that JT probability propagation is central to the theory and practice of probabilistic expert systems (Shafer 1996). JT propagation passes information, in the form of potentials, between neighbouring nodes in a systematic fashion. Unlike a CPT, a *potential* (Hájek et al. 1992) is not clearly interpretable (Castillo et al. 1997) as the probability values forming the distribution may have no recognizable pattern or structure. In an elegant review of traditional JT propagation (Shafer 1996), it is explicitly written that while independencies play an important role in semantic modeling, they do not play a major role in JT propagation. We argue that they should.

By iterating between semantic modeling and computing the actual probability distributions in computer memory, the JT propagation algorithm suggested by Madsen and Jensen (1999), called *LAZY propagation*, is a significant advancement in the study of JT propagation. Unlike traditional approaches, LAZY propagation maintains structure in the form of a multiplicative factorization of the potentials at each JT node and each JT separator. Thereby, when a node is ready to send its messages to a neighbour, the structure is modeled to remove irrelevant potentials from the multiplicative factorization by exploiting *barren variables* (Shachter 1986) and independencies induced by evidence. Next, physical computation is performed on the relevant potentials. Performing semantic modeling before physical computation improves the efficiency of inference, as the experimental results presented in Madsen and Jensen (1999) explicitly demonstrate. With respect to the computational efficiency at the sending node, any remaining independencies, in the relevant potentials, are immaterial. On the contrary, by ignoring these independencies, LAZY propagation is not able to precisely articulate the probability information being passed from one node to another. Passing potentials not only hinders the identification of barren variables at the receiving node, but also blurs the exact probability information remaining at each JT node when propagation finishes. More importantly, iterating between semantic modeling and physical computation essentially guarantees that LAZY performs JT probability propagation more slowly than necessary.

In this study, we propose the first JT propagation architecture for modeling two tasks of inference, one involving evidence and the other not. Our architecture, itself, does not construct the actual probability distributions stored in computer memory. Instead, it labels the probability information to be propagated more precisely in terms of CPTs. The notions of *parent-set* and *elder-set* are introduced in order to identify independencies that are not utilized in previous architectures. The identified independencies are very useful, since they allow us to maintain a CPT factorization after a variable is eliminated. When a JT node is ready to send its messages to a neighbour, it calls the *IdentifyCPTMessages* (ICM) algorithm to determine the CPT labels corresponding to the probability information to be sent to the neighbour, that is, without building the actual probability tables in computer memory. We prove the correctness of our architecture and also show that each JT node can identify its CPT labels in polynomial time. Screen shots of our implemented system demonstrate the improvements in semantic knowledge.

As discussed in Section 5, modeling the processing of evidence is beneficial as our architecture is able to identify relevant and irrelevant messages faster than it takes other propagation algorithms to construct the actual probability distributions in the memory. In a real-world BN for *coronary heart disease* (CHD) (Hájek et al. 1992), for instance, our architecture can identify all messages in the JT in less time than it takes to physically construct the distribution for one message. We make use of this semantic knowledge to generate three work schedules for LAZY propagation. Our first work schedule identifies irrelevant non-empty messages. This information is very useful, since LAZY's lack of semantic knowledge could force a receiving node to wait for the physical construction of a message that is irrelevant to its subsequent message computation. Our second work schedule indicates the empty messages to be propagated from non-leaf JT nodes. This has merit as LAZY could force a receiving node to wait for the identification of an empty message that will not even be constructed, let alone sent. Our third work schedule also helps LAZY finish sooner by pointing out those variables that can be eliminated at a non-leaf node before the node has received any messages. Thus, our three work schedules save time, which is the measure used to evaluate inference methods in Madsen and Jensen (1999). As reported in Tables 2, 3 and 4, our empirical results, involving processing the evidence in five real-world or benchmark BNs, are encouraging.

Our architecture is also useful for modeling inference not involving evidence. As shown in Section 6, after the CPTs identified by our architecture have been physically constructed, say, by either of the methods in Madsen and Jensen (1999); Zhang (1998), each JT node has a *sound*, *local* BN preserving all *conditional* independencies of the original BN involving variables in this node. These local BNs are useful to the MSBN and to the DC techniques. We show that our JT propagation architecture is instrumental in developing an *automated* procedure for constructing a MSBN from a given BN. This is a worthwhile result, since several problems with the *manual* construction of a MSBN from a BN have recently been acknowledged (Xiang et al. 2000). We also suggest a method for exploiting *localized* queries in DC techniques. Practical experience has demonstrated that queries tend to involve variables in close proximity within a BN (Xiang et al. 1993). Our approach allows DC to process localized queries in local BNs. As a result, the experimental results involving a real-world BN for CHD show promise.

This paper is organized as follows. Section 2 contains definitions. Our semantic architecture for JT probability propagation is given in Section 3. Complexity analysis and the correctness of our architecture are established in Section 4. In Section 5, we model the task of inference involving evidence and show its usefulness in practice. In Section 6, we show the practical benefits of modeling inference not involving evidence. The conclusion is given in Section 7.

2 Definitions

Let $U = \{v_1, v_2, \ldots, v_m\}$ be a finite set of variables. Each variable v_i has a finite domain, denoted $dom(v_i)$, representing the values v_i can assume. For a subset $X \subseteq U$, we write $dom(X)$ for the Cartesian product of the domains of the individual variables in X. Each element $x \in dom(X)$ is called a *configuration* of X.

Definition 1 A *potential* (Hájek et al. 1992) on $dom(X)$ is a function ϕ on $dom(X)$ such that $\phi(x) \geq 0$, for each configuration $x \in dom(X)$, and at least one $\phi(x)$ is positive.

For example, five potentials $\phi(b)$, $\phi(f, g)$, $\phi(g, h)$, $\phi(f)$ and $\phi(g)$ are illustrated in Fig. 1.

Definition 2 For variable v_i, a *unity-potential* $1(v_i)$ is a potential 1 assigning value 1.0 to each configuration of v_i. For a subset $X \subseteq U$, the unity-potential $1(X)$ is defined as the product of the unity-potentials $1(v_i)$, $v_i \in X$.

For brevity, we refer to a potential as a probability distribution on X rather than $dom(X)$, and we call X, not $dom(X)$, its domain (Shafer 1996). Also, for simplified notation, we may write a set $\{v_1, v_2, \ldots, v_k\}$ as $v_1 v_2 \cdots v_k$ and use XY to denote $X \cup Y$.

Definition 3 A *JPD* (Shafer 1996) on U, denoted $p(U)$, is a potential on U that sums to one.

Definition 4 Given $X \subset U$, a *CPT* (Shafer 1996) for a variable $v \notin X$ is a distribution, denoted $p(v|X)$, satisfying the following condition: for each configuration $x \in dom(X)$, $\sum_{c \,\in\, dom(v)} p(\, v = c \mid X = x\,) = 1.0$.

For example, given binary variables $U = \{a, b, \ldots, k\}$, CPTs $p(a)$, $p(b|a)$, $p(c)$, $p(d|c)$, $p(e|c)$, $p(f|d, e)$, $p(g|b, f)$, $p(h|c)$, $p(i|h)$, $p(j|g, h, i)$ and $p(k|g)$ are depicted in Fig. 2. The missing conditional probabilities can be obtained by definition, for instance, $p(a = 0) = 0.504$ and $p(b = 0|a = 0) = 0.943$.

Definition 5 The *label* of a probability distribution is the heading appearing above the probability column.

For example, the five probability distributions in Fig. 1 can be more precisely labeled as $p(b)$, $p(g|f)$, $p(g, h)$, $p(f)$ and $p(g)$, respectively.

b	$\phi(b)$
0	0.910
1	0.090

f	g	$\phi(f, g)$
0	0	0.910
0	1	0.090
1	0	0.000
1	1	1.000

g	h	$\phi(g, h)$
0	0	0.435
0	1	0.325
1	0	0.137
1	1	0.103

f	$\phi(f)$
0	0.835
1	0.165

g	$\phi(g)$
0	0.760
1	0.240

Fig. 1 Five *potentials* $\phi(b)$, $\phi(f, g)$, $\phi(g, h)$, $\phi(f)$ and $\phi(g)$

a	$p(a)$
1	0.496

a	b	$p(b\|a)$
0	1	0.057
1	1	0.123

c	e	$p(e\|c)$
0	1	0.500
1	1	0.500

c	$p(c)$
1	0.423

c	d	$p(d\|c)$
0	1	0.500
1	1	0.500

c	h	$p(h\|c)$
0	1	0.421
1	1	0.437

d	e	f	$p(f\|d,e)$
0	0	1	0.027
0	1	1	0.123
1	0	1	0.101
1	1	1	0.408

h	i	$p(i\|h)$
0	1	0.361
1	1	0.381

b	f	g	$p(g\|b,f)$
0	0	1	0.000
0	1	1	1.000
1	0	1	1.000
1	1	1	1.000

g	k	$p(k\|g)$
0	1	0.498
1	1	0.739

g	h	i	j	$p(j\|g,h,i)$
0	0	0	1	0.353
0	0	1	1	0.421
0	1	0	1	0.406
0	1	1	1	0.562
1	0	0	1	0.353
1	0	1	1	0.421
1	1	0	1	0.406
1	1	1	1	0.562

Fig. 2 CPTs $p(a)$, $p(b|a)$, $p(c)$, $p(d|c)$, $p(e|c)$, $p(f|d,e)$, $p(g|b,f)$, $p(h|c)$, $p(i|h)$, $p(j|g,h,i)$ and $p(k|g)$

Definition 6 Let x, y and z denote arbitrary configurations of pairwise disjoint subsets X, Y, Z of U, respectively. We say X and Z are *conditionally independent* (Wong et al. 2000) given Y under the JPD $p(U)$, denoted $I(X, Y, Z)$, if $p(X = x|Y = y, Z = z) = p(X = x|Y = y)$, whenever $p(Y = y, Z = z) > 0$. If $Y = \emptyset$, then we say X and Z are *unconditionally* independent. The independence $I(X, Y, Z)$ can be equivalently written as $p(X, Y, Z) = \frac{p(X,Y)\cdot p(Y,Z)}{p(Y)}$.

Definition 7 A *Markov blanket* (Pearl 1988) of a variable $v \in U$ is any subset of variables $X \subset U$ with $v \notin X$ such that the independence $I(v, X, U - X - v)$ holds in $p(U)$.

Definition 8 A *BN* (Pearl 1988) on U is a pair (D, C). D is a DAG on U. C is a set of CPTs defined as: for each variable $v_i \in D$, there is a CPT for v_i given its parents.

Note that, for each CPT in the BN, there is a corresponding CPT labe. The *family* of a variable in the DAG of a BN is the variable and its parents. We may use the terms BN and *DAG* interchangeably, if no confusion arises.

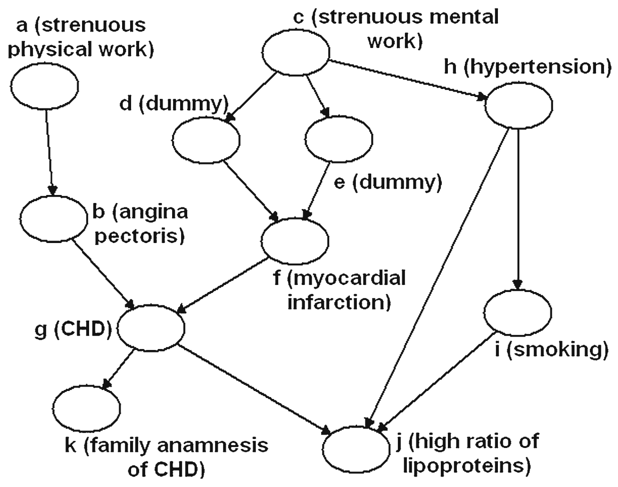
Example 1 One real-world BN for *CHD* (Hájek et al. 1992) is shown in Fig. 3, where the CPTs are given in Fig. 2. For pedagogical reasons, we have made the following minor adjustments: edge (a, f) has been removed; edges (c, f) and (g, i) have been replaced with edges (c, d), (c, e), (d, f), (e, f) and (g, j), respectively, where d and e are dummy variables.

Definition 9 A numbering $\prec$ of the variables in a DAG is called *ancestral* (Castillo et al. 1997), if the number corresponding to any variable v_i is lower than the number corresponding to each of its children v_j, denoted $v_i \prec v_j$.

Definition 10 The *moralization* (Pearl 1988) of a DAG D is the undirected graph obtained by making the family of each variable in D complete.

Given pairwise disjoint subsets of variables X, Y, Z in a DAG D, the independence $I(X, Y, Z)$ holds in D (Lauritzen et al. 1990), if Y separates X and Z in the moralization of D', where D' is the sub-DAG of D restricted to the edges (v_i, v_j) such that v_i, v_j are in $XYZ \cup An(XYZ)$, and where $An(XYZ)$ denotes the ancestors

Fig. 3 The coronary heart disease (*CHD*) BN (Hájek et al. 1992) in Example 1


a (strenuous physical work)
c (strenuous mental work)
h (hypertension)
d (dummy)
e (dummy)
b (angina pectoris)
f (myocardial infarction)
g (CHD)
i (smoking)
k (family anamnesis of CHD)
j (high ratio of lipoproteins)

of XYZ in D. The independencies encoded in D indicate that the product of the CPTs in C is a JPD. For instance, the independencies encoded in the DAG of Fig. 3 indicate that the product of the CPTs in Fig. 2 is a JPD on $U = \{a, b, c, d, \ldots, k\}$, namely, $p(U) = p(a) \cdot p(b|a) \cdot p(c) \cdot p(d|c) \cdot \ldots \cdot p(k|g)$.

Probabilistic inference, also known as query processing, means computing $p(X)$ or $p(X|E = e)$, where $X \cap E = \emptyset$ and $X, E \subseteq U$. The *evidence* in the latter query is that E is instantiated to configuration e, while X contains *target* variables. Barren variables can be exploited in inference (Shachter 1986).

Definition 11 A variable is *barren* (Madsen and Jensen 1999), if it is neither an evidence nor a target variable and either it has no descendants or (recursively) all its descendants are barren.

As discussed in Section 6, probabilistic inference can be conducted directly in an entire BN or in parts of a BN. It can also be conducted in a JT (Jensen et al. 1990; Lauritzen and Spiegelhalter 1988; Madsen and Jensen 1999; Shafer and Shenoy 1990).

Definition 12 A *JT* (Pearl 1988; Shafer 1996) is a tree having sets of variables as nodes, with the property that any variable in two nodes is also in any node on the path between the two. The *separator* (Shafer 1996) S between any two neighbouring nodes N_i and N_j is $S = N_i \cap N_j$. A JT node with only one neighbour is called a *leaf* (Shafer 1996); otherwise it is a *non-leaf* node.

Constructing a minimal JT is NP-complete (Yannakakis 1981). The reader is referred to (Becker and Geiger 2001; Kjaerulff 1990; Olesen and Madsen 2002) for discussions on constructing a JT from a BN. For example, one possible JT for the DAG in Fig. 3 is depicted in Fig. 4 (ignoring the messages for the moment). We label the nodes of this JT as ab, bfg, $cdefgh$, $ghij$ and gk. The separators of this JT are b, fg, gh and g.

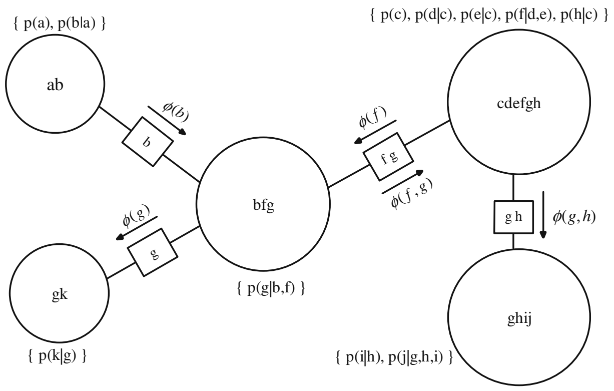


Fig. 4 LAZY propagates *potentials* $\phi(b)$, $\phi(f, g)$, $\phi(g, h)$, $\phi(f)$ and $\phi(g)$

Although JT propagation can be performed serially, our discussion here is based on *parallel* computation (Kozlov and Singh 1999; Madsen and Jensen 1999; Shafer 1996). We provide a quick overview of the *Shafer-Shenoy* (SS) architecture for probability propagation in JTs (Shafer and Shenoy 1990; Shafer 1996). Each JT node N has exactly one potential $\phi(N)$, which is defined by the product of the unity-potential $1(N)$ and all of the BN CPTs assigned to N. The SS architecture allocates two storage registers in each separator, one for a message sent in each direction. Each node sends messages to all its neighbours, according to the following two rules. First, each node waits to send its message to a particular neighbour until it has received messages from all its other neighbours. Second, when a node N_i is ready to send its message to a particular neighbour N_j, it computes the message $\phi(N_i \cap N_j)$ by collecting all its messages from other neighbours, multiplying its potential $\phi(N_i)$ by these messages, and marginalizing the product to $N_i \cap N_j$.

The *LAZY* architecture, proposed by Madsen and Jensen (1999), is more sophisticated than the SS architecture, in that it maintains a multiplicative factorization of potentials at each JT node and each JT separator. Modelling structure in this manner leads to significant computational savings (Madsen and Jensen 1999). For ease of exposition, evidence will not be considered here, but later, in Section 5. When LAZY propagation terminates, the potentials at each JT node N are a factorization of the marginal $p(N)$ of $p(U)$. Example 2 illustrates the messages passed in LAZY and the probability information that remains after propagation.

Example 2 Consider the CHD BN in Fig. 3 and one possible JT with assigned CPTs in Fig. 4. The distributions of the five potentials $\phi(b)$, $\phi(f, g)$, $\phi(g, h)$, $\phi(f)$ and $\phi(g)$ in Fig. 4, passed in LAZY propagation, are illustrated in Fig. 1. After propagation the marginal distribution at each node is:

$$\begin{aligned}
p(a, b) &= p(a) \cdot p(b|a), \\
p(b, f, g) &= p(g|b, f) \cdot \phi(b) \cdot \phi(f), \\
p(c, d, e, f, g, h) &= p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d, e) \cdot p(h|c) \cdot \phi(f, g), \\
p(g, h, i, j) &= p(i|h) \cdot p(j|g, h, i) \cdot \phi(g, h), \\
p(g, k) &= p(k|g) \cdot \phi(g).
\end{aligned}$$

3 Modeling inference not involving evidence

In this paper, we are interested in identifying the probability information passed during propagation and that which remains after propagation terminates. Although LAZY propagation can efficiently compute the five distributions in Fig. 1, it does not clearly articulate the semantics of the messages being propagated between JT nodes. For instance, it can be verified that the potential $\phi(f, g)$ in Fig. 1 is, in fact, the CPT $p(g|f)$ of $p(U)$. To address these problems, instead of developing yet another architecture for probabilistic inference, our architecture *models* probabilistic inference.

Our simple architecture identifies the probability information being passed between JT nodes. It uses five rules, given below, for filling the storage registers in the JT separators with CPT or unity-potential labels. The key to our architecture is the *ICM* algorithm used in Rule 4. Before starting JT propagation, each CPT label $p(v|X)$ in the original BN is assigned to exactly one JT node N, where N contains the variables in $\{v\} \cup X$.

Rule 1. For every variable in every separator, allocate two empty storage registers, one for a label in each direction.
Rule 2. Fix an ancestral numbering $\prec$ of the variables in the BN.
Rule 3. Each node N_i waits to identify its label(s) to a given neighbour until all of N_i's other neighbours have filled their storage registers to N_i.
Rule 4. When a node N_i is able to send the label(s) to a neighbour N_j, it calls the ICM algorithm, passing its assigned CPT labels and all CPT labels received by the storage registers from its other neighbours to N_i, as well as the variables $N_i - N_j$ to be eliminated.
Rule 5. For each CPT label $p(v_k|P_k)$ returned by ICM, fill the storage register for variable v_k from N_i to N_j with label $p(v_k|P_k)$. For any variable v_l with a storage register from N_i to N_j still empty, fill the register with the unity-potential label $1(v_l)$.

We illustrate Rules 1–3 with the following example.

Example 3 Consider the JT with assigned BN CPTs in Fig. 5. By Rule 1, the separator *gh* in Fig. 4, for instance, has four storage registers in Fig. 5, which initially are empty. For Rule 2, we fix the ancestral numbering as $a \prec b \prec \ldots \prec k$. According to Rule 3, node bfg, for example, can only send labels to node ab after the message labels have been received by the storage registers from its other neighbours to bfg, namely, the storage registers from both nodes *cdefgh* and *gk* to bfg. In other words, Rule 3 means leaf JT nodes are ready to identify their messages immediately.

The ICM algorithm is built upon the *FindRelevantCPTs* (FRC) and *MaintainCPT-Labels* (MCL) algorithms.

Definition 13 Given a set C of CPT labels and a variable v_i to be eliminated, the *FindRelevantCPTs* (FRC) algorithm returns the set C' of CPT labels in C involving v_i, where FRC first sorts the CPT labels in C' according to $\prec$ in Rule 2, say $C' = \{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$, where $v_i \prec v_1 \prec \ldots \prec v_k$.

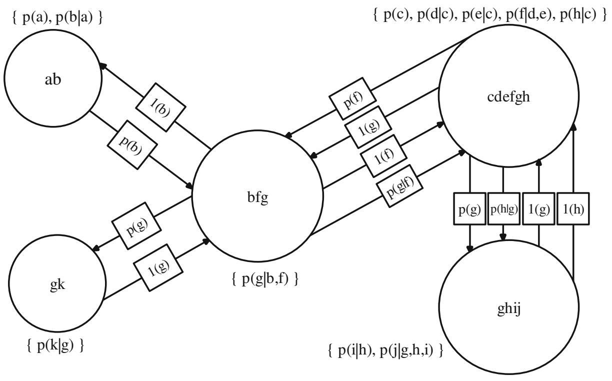


Fig. 5 Unlike Fig. 4, our architecture precisely articulates the probability information being passed between JT nodes

Example 4 Suppose FRC is called with $C = \{p(c), p(d|c), p(e|c), p(f|d,e), p(h|c)\}$ and variable d is to be eliminated. By $\prec$ in Example 3, FRC returns $C' = \{p(d|c), p(f|d,e)\}$ and not $C' = \{p(f|d,e), p(d|c)\}$.

Definition 14 To eliminate v_i, suppose FRC returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$. Consider a variable $v_j \in v_i v_1 \cdots v_k$. The *parent-set* of v_j is P_j.

Example 5 To eliminate c, suppose FRC returns $\{p(c), p(e|c), p(f|c,e), p(h|c)\}$. The parent-sets of variables c, e, f and h are $\{\}$, $\{c\}$, $\{c, e\}$ and $\{c\}$, respectively.

Definition 15 Suppose FRC returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$ for eliminating variable v_i. We call $C_i = \{v_1, \ldots, v_k\}$ the *child-set* of v_i, where $v_1 \prec \ldots \prec v_k$ in Rule 2.

Example 6 To eliminate c, suppose FRC returns $\{p(c), p(d|c), p(e|c), p(h|c)\}$. Besides $p(c)$, variable c appears in the CPT labels for $\{h, d, e\}$. By $\prec$ in Example 3, $e \prec h$, while $d \prec e$. By definition, the child-set of $v_i = c$ is $C_i = \{v_1 = d, v_2 = e, v_3 = h\}$.

To eliminate v_i, given that FRC returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$, those CPT labels $p(v_1|P_1), \ldots, p(v_k|P_k)$ of the variables $v_j \in C_i$ are modified. While variable v_i is deleted from P_j, only certain variables preceding v_j in $\prec$ of Rule 2 may be added to P_j.

Definition 16 To eliminate v_i, suppose FRC returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$. Consider a variable $v_j \in v_i v_1 \cdots v_k$. The *family-set*, denoted F_j, of v_j is $v_j P_j$.

Example 7 To eliminate c, suppose FRC returns $\{p(c), p(e|c), p(f|c,e), p(h|c)\}$. The family-sets of variables c, e, f and h are $\{c\}$, $\{c, e\}$, $\{c, e, f\}$ and $\{c, h\}$, respectively.

Definition 17 Given a set of CPT labels $\{p(v_1|P_1), \ldots, p(v_m|P_m)\}$, the directed graph defined by these CPT labels, called the *CPT-graph*, has variables $F_1 \cdots F_m$, and a directed edge from each variable in P_k to v_k, $k = 1, \ldots, m$.

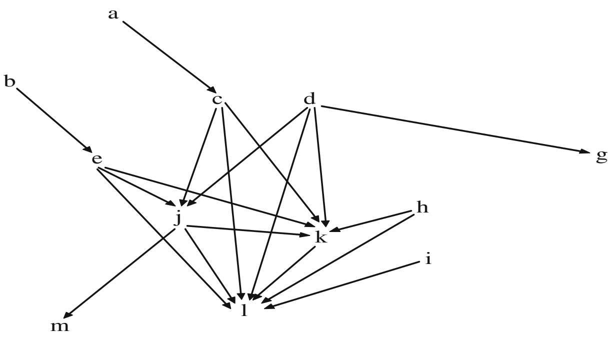


Fig. 6 The CPT-graph defined by the CPT labels in Example 8

Example 8 Consider the set of CPT labels $\{p(a), p(c|a), p(d), p(e|b), p(g|d), p(h), p(j|c, d, e), p(k|c, d, e, h, j), p(l|c, d, e, h, i, j, k), p(m|j)\}$. The CPT-graph is shown in Fig. 6. Note that variables b and i do not have CPT labels in the given set.

Definition 18 To eliminate v_i, suppose FRC returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$. The child-set $C_i = \{v_1, \ldots, v_k\}$ is defined by $\prec$ in Rule 2. For each variable $v_j \in C_i$, the *elder-set* E_j is defined as $E_1 = F_i - v_i$, $E_2 = (E_1 F_1) - v_i$, $E_3 = (E_2 F_2) - v_i, \ldots, E_k = (E_{k-1} F_{k-1}) - v_i$.

For simplified notation, we will write $(E_j F_j) - v_i$ as $E_j F_j - v_i$. Definition 18 says that the elder-set E_j of variable v_j in C_i is F_i together with the family-sets of the variables in C_i preceding v_j, with v_i subsequently removed.

Example 9 Consider the set of CPT labels $\{p(a), p(c|a), p(d), p(e|b), p(f|c, d), p(g|d), p(h), p(j|e, f), p(k|d, f, h), p(l|f, i), p(m|j)\}$. The CPT-graph defined by these CPT labels is shown in Fig. 7. Let us assume that $\prec$ in Rule 2 orders this subset of variables in alphabetical order. Consider eliminating variable $v_i = f$. The call to FRC returns $\{p(f|c, d), p(j|e, f), p(k|d, f, h), p(l|f, i)\}$. With respect to $\prec$ in Rule 2, the elder-set of each variable in the child-set $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$ is $E_1 = \{c, d\}$, $E_2 = \{c, d, e, j\}$ and $E_3 = \{c, d, e, h, j, k\}$, as depicted in Fig. 7.

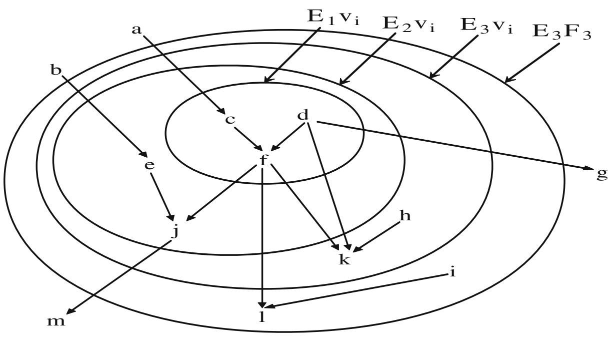


Fig. 7 With respect to variable $v_i = f$, illustrating the elder-set E_1, E_2 and E_3 for each variable in the child-set $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$ in Example 9. Note that one Markov blanket of f is $E_3 F_3$

The MCL algorithm now can be introduced.

Algorithm 1 MCL(C')

```
Input: $C' = \{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$ from FRC for eliminating $v_i$
Output: the modified CPT labels for all $k$ variables $v_j$ in $C_i = \{v_1, \ldots, v_k\}$
begin
Determine the elder-set $E_j$ and parent-set $P_j$ for all $k$ variables $v_j \in C_i$
for $j = k, \ldots, 1$
    $P_j = (E_j P_j) - v_i$
return($\{p(v_1|P_1), \ldots, p(v_k|P_k)\}$)
end
```

For simplified notation, we will write $(E_j P_j) - v_i$ as $E_j P_j - v_i$.

Example 10 To eliminate variable $v_i = f$, suppose the set of CPT labels obtained by FRC in Example 9 is passed to MCL. For $\prec$ in Example 9, consider variable $v_3 = l$. By definition, $E_3 = \{c, d, e, h, j, k\}$ and $P_3 = \{f, i\}$. Then $p(l|f, i)$ is adjusted to $p(l|c, d, e, h, i, j, k)$, as $E_3 P_3 - v_i$ is $\{c, d, e, h, i, j, k\}$. Similarly, $p(k|d, f, h)$ is changed to $p(k|c, d, e, h, j)$, and $p(j|e, f)$ is modified to $p(j|c, d, e)$. By Definition 17, the CPT-graph defined by the CPT labels remaining after the elimination of variable f is shown in Fig. 6.

Lemma 1 *According to our architecture, let $C'' = \{p(v_1|P_1), \ldots, p(v_k|P_k)\}$ be the set of CPT labels returned by the MCL algorithm. Then the CPT-graph defined by C'' is a DAG.*

Proof For any BN CPT $p(v_j|P_j)$, by Rule 2, $v \prec v_j$, where $v \in P_j$. Moreover, by the definition of elder-set, $v \prec v_j$, where $v \in E_j$. Since the parent set P_j of v_j is modified as $P_j E_j - v_i$, for any CPT label $p(v_j|P_j)$ returned by the MCL algorithm, it is still the case that $v \prec v_j$, where $v \in P_j$. Therefore, the CPT-graph defined by C'' is acyclic, namely, it is a DAG. □

We now present the ICM algorithm.

Algorithm 2 ICM(C, X)

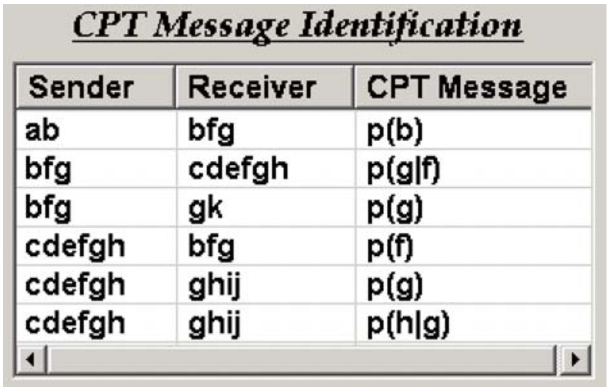
```
Input: a set $C$ of CPT labels,
        the set $X$ of variables to be eliminated from $C$
Output: the set $C$ of CPT labels sent from a node to a neighbour
begin
for each variable $v$ in $X$
{
    $C' = FRC(C, v)$
    $C'' = MCL(C')$
    $C = (C - C') \cup C''$
}
return($C$)
end
```

We use two examples to illustrate the subtle points of all five rules in our architecture.

Example 11 Let us demonstrate how our architecture determines the CPT labels $p(g)$ and $p(h|g)$ in the storage registers for g and h from $cdefgh$ to $ghij$, shown in Fig. 5. By Rule 2, we use $\prec$ in Example 3. By Rule 4, $cdefgh$ has collected the CPT label $p(g|f)$ in the storage register of g from bfg to $cdefgh$, but not $1(f)$ as this is not a CPT label, and calls ICM with $C = \{p(c), p(d|c), p(e|c), p(f|d, e), p(g|f), p(h|c)\}$ and $X = \{c, d, e, f\}$. For pedagogical purposes, let us eliminate the variables in the order d, c, e, f. ICM calls FRC, passing it C and variable d. FRC returns $\{p(d|c), p(f|d, e)\}$, which ICM initially assigns to C' and subsequently passes to MCL. Here $v_i = d, C_i = \{v_1 = f\}, P_1 = \{d, e\}$ and $E_1 = \{c\}$. As $E_1 P_1 - v_i$ is $\{c, e\}$, MCL returns $\{p(f|c, e)\}$, which ICM assigns to C''. Next, the set C of CPT labels under consideration in ICM is adjusted to be $C = \{p(c), p(e|c), p(f|c, e), p(g|f), p(h|c)\}$. For variable c, the call to FRC results in $C' = \{p(c), p(e|c), p(f|c, e), p(h|c)\}$. To eliminate $v_i = c$, the elder-set of each variable in the child-set $C_i = \{v_1 = e, v_2 = f, v_3 = h\}$ is $E_1 = \{\}$, $E_2 = \{e\}$ and $E_3 = \{e, f\}$. Moreover, the parent-set of each variable in the child-set is $P_1 = \{c\}$, $P_2 = \{c, e\}$ and $P_3 = \{c\}$. In MCL, then $p(h|c)$ is adjusted to $p(h|e, f)$, as $E_3 P_3 - v_i$ is $\{e, f\}$. Similarly, $p(f|c, e)$ is changed to $p(f|e)$ and $p(e|c)$ is modified to $p(e)$. Therefore, MCL returns the set $\{p(e), p(f|e), p(h|e, f)\}$ of CPT labels, which ICM assigns to C''. Then C is updated to be $C = \{p(e), p(f|e), p(g|f), p(h|e, f)\}$. After eliminating variable e, the set of labels under consideration is modified to $C = \{p(f), p(g|f), p(h|f)\}$. Moreover, after considering the last variable f, the set of labels under consideration is $C = \{p(g), p(h|g)\}$. ICM returns C to $cdefgh$. By Rule 5, $cdefgh$ places the CPT labels $p(g)$ and $p(h|g)$ in the storage registers of g and h from $cdefgh$ to $ghij$, respectively.

Example 12 Now let us show how our architecture determines the labels $p(f)$ and $1(g)$ in the storage registers for f and g from $cdefgh$ to bfg, shown in Fig. 5. By Rule 2, we use $\prec$ in Example 3. By Rule 4, $cdefgh$ does not collect the unity-potential labels $1(g)$ and $1(h)$ in the storage registers of g and h from $ghij$ and calls ICM with $C = \{p(c), p(d|c), p(e|c), p(f|d, e), p(h|c)\}$ and $X = \{c, d, e, h\}$. For simplicity, let us eliminate the variables in the order d, c, e, h. Similar to Example 11, the set of CPT labels under consideration, after the elimination of variables c, d, e, is $C = \{p(f), p(h|f)\}$. After eliminating the last variable h, the set of labels under consideration is $C = \{p(f)\}$. ICM returns C to $cdefgh$. By Rule 5, $cdefgh$ places the CPT label $p(f)$ in the storage register of f from $cdefgh$ to bfg and places the unity-potential label $1(g)$ in the empty storage register for variable g from $cdefgh$ to bfg.

All of the identified CPT messages for Fig. 5 are illustrated in Fig. 8, which is a screen shot of our implemented system. For simplified discussion, we will henceforth speak of messages passed between JT nodes with the separators understood.



CPT Message Identification

Sender	Receiver	CPT Message
ab	bfg	p(b)
bfg	cdefgh	p(g|f)
bfg	gk	p(g)
cdefgh	bfg	p(f)
cdefgh	ghij	p(g)
cdefgh	ghij	p(h|g)

Fig. 8 Identification of the CPT messages in Fig. 5

Example 13 Let us review Example 11 in terms of equations:

$$\sum_{c,d,e,f} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d,e) \cdot p(g|f) \cdot p(h|c) \tag{1}$$

$$= \sum_{f} p(g|f) \cdot \sum_{e} \sum_{c} p(c) \cdot p(e|c) \cdot p(h|c) \cdot \sum_{d} p(d|c) \cdot p(f|d,e) \tag{2}$$

$$= \sum_{f} p(g|f) \cdot \sum_{e} \sum_{c} p(c) \cdot p(e|c) \cdot p(h|c) \cdot p(f|c,e) \tag{3}$$

$$= \sum_{f} p(g|f) \cdot \sum_{e} p(e) \cdot p(f|e) \cdot p(h|e,\ f)$$

$$= \sum_{f} p(g|f) \cdot p(f) \cdot p(h|f)$$

$$= p(g) \cdot p(h|g).$$

The derivation of $p(g)$ and $p(h|g)$ is not correct without independencies. For instance, the independencies $I(d, c, e)$ and $I(f, de, c)$ are necessary when moving from (2) to (3). In fact, without the unconditional independence $I(b, \emptyset, f)$, the message $p(g|f)$, from bfg to $cdefgh$, used in (1), is not correct either:

$$\sum_{b} p(b) \cdot p(g|b,\ f) \ = \ \sum_{b} p(b) \cdot \frac{p(b,\ f, g)}{p(b,\ f)} \ = \ \sum_{b} p(b) \cdot \frac{p(b,\ f, g)}{p(b) \cdot p(f)} \ = \ p(g|f).$$

Thus, the importance of showing that we can identify these independencies and utilize them, as shown above, is made obvious.

4 Complexity and correctness

Here we establish the time complexity of the ICM algorithm and the correctness of our architecture for modeling inference not involving evidence.

Lemma 2 *Let n be the number of CPT labels in C' given as input to the MCL algorithm for eliminating variable v_i. The time complexity of MCL is $O(n)$.*

Proof Let $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_{n-1}|P_{n-1})\}$ be the input set C' of CPT labels given to MCL. The elder-sets $E_1, \ldots, E_{n-1}$ and parent-sets $P_1, \ldots, P_{n-1}$ can be obtained in one pass over C'. Given that C' has n labels, the time complexity to determine the required parent-sets and elder-sets is $O(n)$. Similarly, for each label in $\{p(v_1|P_1), \ldots, p(v_{n-1}|P_{n-1})\}$, the parent-set is modified exactly once. Hence, this *for-loop* executes $n-1$ times. Therefore, the time complexity of MCL is $O(n)$. □

Our main complexity result, given in Theorem 1, is that the CPT labels sent from a JT node can be identified in polynomial time.

Theorem 1 *In the input to the ICM algorithm, let n be the number of CPT labels in C, and let X be the set of variables to be eliminated. The time complexity of ICM is $O(n^2 \log n)$.*

Proof As the number of CPT labels in C is n, the maximum number of variables to be eliminated in X is n. The loop body is executed n times, once for each variable in X. Recall that the loop body calls the FRC and MCL algorithms. Clearly, FRC takes $O(n)$ time to find those CPT labels involving v_i. According to $\prec$ in Rule 2, these CPT labels can be sorted using the merge sort algorithm in $O(n \log n)$ time (Cormen et al. 2001). Thus, FRC has time complexity $O(n \log n)$. By Lemma 2, MCL has time complexity $O(n)$. Thus, the loop body takes $O(n \log n)$ time. Therefore, the time complexity of the ICM algorithm is $O(n^2 \log n)$. □

We now turn to the correctness of our architecture. The next result shows that certain independencies involving elder-sets hold in any BN.

Lemma 3 *Let (D, C) be a BN defining a JPD $p(U)$. Given the set C of CPT labels and any variable $v_i \in D$, the FRC algorithm returns $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$, where the variables in the child-set $C_i = \{v_1, \ldots, v_k\}$ are written according to $\prec$ in Rule 2. For each $v_j \in C_i$ with elder-set E_j, the independencies $I(v_i, E_j, P_j - v_i)$ and $I(v_j, P_j, E_j)$ hold in the JPD $p(U)$.*

Proof Observe that the parents, children, and family of any variable v in D are precisely the parent-set, child-set and family-set of v, which are defined by C', respectively. Let us first show that $I(v_j, P_j, E_j)$ holds for $j = 1, \ldots, k$. Pearl (1988) has shown that $I(v, P_v, N_v)$ holds for every variable v in a BN, where N_v denotes the set of all non-descendants of v in D. By definition, no variable in the elder-set E_j can be a descendant of v_j in D. Thereby, $E_j \subseteq N_j$. By the *decomposition axiom* of probabilistic independence (Pearl 1988; Wong et al. 2000), the JPD $p(U)$ satisfying $I(v_j, P_j, N_j)$ logically implies that $p(U)$ satisfies $I(v_j, P_j, E_j)$.

We now show that $I(v_i, E_j, P_j - v_i)$ holds, for $j = 1, \ldots, k$. According to the method in Section 2 for testing independencies, we write $I(v_i, E_j, P_j - v_i)$ as $I(v_i, E_j, P_j - v_i - E_j)$ and determine the ancestral set of the variables in $I(v_i, E_j, P_j - v_i - E_j)$ as $An(v_i) \cup An(E_j) \cup An(P_j - v_i - E_j)$, which is the same as $An(E_j P_j)$, since $v_i \in P_j$. Note that v_j, and any child of v_i in D succeeding v_j in $\prec$ of Rule 2, are not members in the set $An(E_j P_j)$. Hence, in the constructed sub-DAG D' of DAG D onto $An(E_j P_j)$, the only directed edges involving v_i are those from each variable in P_i to v_i and from v_i to every child preceding v_j. By Corollary 6 in Pearl (1988), E_j is a Markov blanket of v_i in D'. By definition, $I(v_i, E_j, U - E_j - v_i)$ holds in $p(U)$. By the decomposition axiom, $p(U)$ satisfies $I(v_i, E_j, P_j - v_i)$. □

Let us first focus on the elimination of just one variable. We can maintain a CPT factorization after $v_i \in X$ is eliminated, provided that the corresponding elder-set independencies are satisfied by the JPD $p(U)$. Note that, as done previously, we write $(E_j P_j) - v_i$, $(E_j F_k) - v_i$ and $(E_j F_j) - v_i v_j$ more simply as $E_j P_j - v_i$, $E_j F_k - v_i$ and $E_j F_j - v_i v_j$, respectively.

Lemma 4 *Given the output $\{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$ of FRC to eliminate variable v_i in our architecture. By Rule 2, $v_1 \prec \ldots \prec v_k$ in the child-set $C_i = \{v_1, \ldots, v_k\}$ of v_i. For each variable $v_j \in C_i$ with elder-set E_j, if the independencies $I(v_i, E_j, P_j - v_i)$ and $I(v_j, P_j, E_j)$ hold in the JPD $p(U)$, then*

$$\sum_{v_i} p(v_i|P_i) \cdot p(v_1|P_1) \cdot \ldots \cdot p(v_k|P_k) \;=\; \prod_{j=1}^{k} p(v_j|E_j P_j - v_i).$$

Proof We first show, by mathematical induction on the number k of variables in C_i, that the product of these relevant CPTs can be rewritten as follows:

$$p(v_i|P_i) \cdot p(v_1|P_1) \cdot \ldots \cdot p(v_k|P_k) \;=\; p(v_i|E_k F_k - v_i) \cdot \prod_{j=1}^{k} p(v_j|E_j P_j - v_i).$$

(Basic step: $j = 1$). By definition, $E_1 = P_i$. Thus, $p(v_i|P_i) \cdot p(v_1|P_1) = p(v_i|E_1) \cdot p(v_1|P_1)$. By definition,

$$p(v_i|E_1) \cdot p(v_1|P_1) \;=\; \frac{p(v_i E_1)}{p(E_1)} \cdot \frac{p(v_1 P_1)}{p(P_1)}. \tag{4}$$

When $j = 1$, by assumption, $I(v_i, E_1, P_1 - v_i)$ and $I(v_1, P_1, E_1)$ hold. We can apply these independencies consecutively by multiplying the numerator and denominator of the right side of Eq. 4 by $p(E_1 P_1 - v_i)$, namely,

$$\begin{aligned}\frac{p(v_i E_1)}{p(E_1)} \cdot \frac{p(v_1 P_1)}{p(P_1)} &= \frac{p(v_i E_1) \cdot p(E_1 P_1 - v_i)}{p(E_1) \cdot p(E_1 P_1 - v_i)} \cdot \frac{p(v_1 P_1)}{p(P_1)} \\ &= \frac{p(E_1 P_1)}{p(E_1 P_1 - v_i)} \cdot \frac{p(v_1 P_1)}{p(P_1)} \\ &= \frac{p(v_1 P_1 E_1)}{p(E_1 P_1 - v_i)}. \end{aligned} \tag{5}$$

By definition,

$$\frac{p(v_1 P_1 E_1)}{p(E_1 P_1 - v_i)} = \frac{p(E_1 F_1)}{p(E_1 F_1 - v_i v_1)} = p(v_i v_1 | E_1 F_1 - v_i v_1). \tag{6}$$

By the *product rule* (Xiang 2002) of probability, which states that $p(X, Y|Z) = p(X|Y, Z) \cdot p(Y|Z)$ for pairwise disjoint subsets X, Y and Z, Eq. 6 is expressed as:

$$\begin{aligned} p(v_i v_1 | E_1 F_1 - v_i v_1) &= p(v_i | E_1 F_1 - v_i) \cdot p(v_1 | E_1 F_1 - v_i v_1) \\ &= p(v_i | E_1 F_1 - v_i) \cdot p(v_1 | E_1 P_1 - v_i). \end{aligned} \tag{7}$$

By (4)–(7),

$$p(v_i|P_i) \cdot p(v_1|P_1) = p(v_i|E_1 F_1 - v_i) \cdot p(v_1|E_1 P_1 - v_i). \tag{8}$$

(Inductive hypothesis: $j = k-1, k \geq 2$). Suppose

$$p(v_i|P_i) p(v_1|P_1) \cdots p(v_{k-1}|P_{k-1}) = p(v_i|E_{k-1} F_{k-1} - v_i) \prod_{j=1}^{k-1} p(v_j|E_j P_j - v_i).$$

(Inductive step: $j = k$). Consider the following product

$$p(v_i|P_i) p(v_1|P_1) \ldots p(v_k|P_k) = p(v_i|P_i) p(v_1|P_1) \ldots p(v_{k-1}|P_{k-1}) p(v_k|P_k). \tag{9}$$

By the inductive hypothesis,

$$\begin{aligned} &(p(v_i|P_i) p(v_1|P_1) \cdot \ldots \cdot p(v_{k-1}|P_{k-1})) \cdot p(v_k|P_k) \\ &= \left(p(v_i|E_{k-1} F_{k-1} - v_i) \prod_{j=1}^{k-1} p(v_j|E_j P_j - v_i) \right) \cdot p(v_k|P_k) \\ &= p\,(v_i|E_{k-1} F_{k-1} - v_i) \cdot p(v_k|P_k) \cdot \prod_{j=1}^{k-1} p(v_j|E_j P_j - v_i). \end{aligned} \tag{10}$$

Since $E_k = E_{k-1} F_{k-1} - v_i$, (10) can be rewritten as

$$p(v_i|P_i) p(v_1|P_1) \cdot \ldots \cdot p(v_k|P_k) = p(v_i|E_k) p(v_k|P_k) \prod_{j=1}^{k-1} p(v_j|E_j P_j - v_i). \tag{11}$$

It can easily be shown that

$$p(v_i|E_k) \cdot p(v_k|P_k) = p(v_i|E_k F_k - v_i) \cdot p(v_k|E_k P_k - v_i), \tag{12}$$

by following (4)–(7) replacing $j = 1$ with $j = k$. Substituting (12) into (11), the desired result follows:

$$\begin{aligned} & p(v_i|P_i) \cdot p(v_1|P_1) \cdot \ldots \cdot p(v_k|P_k) \\ &= p(v_i|E_k F_k - v_i) \cdot p(v_k|E_k P_k - v_i) \cdot \prod_{j=1}^{k-1} p(v_j|E_j P_j - v_i) \\ &= p(v_i|E_k F_k - v_i) \cdot \prod_{j=1}^{k} p(v_j|E_j P_j - v_i). \qquad (13) \end{aligned}$$

Having rewritten the factorization of the relevant CPTs, we now consider the elimination of variable v_i as follows:

$$\begin{aligned} & \sum_{v_i} p(v_i|E_k F_k - v_i) \cdot \prod_{j=1}^{k} p(v_j|E_j P_j - v_i) \\ &= \prod_{j=1}^{k} p(v_j|E_j P_j - v_i) \cdot \sum_{v_i} p(v_i|E_k F_k - v_i) \\ &= \prod_{j=1}^{k} p(v_j|E_j P_j - v_i) \cdot 1.0 \\ &= \prod_{j=1}^{k} p(v_j|E_j P_j - v_i). \qquad (14) \end{aligned}$$

By (13) and (14), we obtain the desired result

$$\sum_{v_i} p(v_i|P_i) \cdot p(v_1|P_1) \cdot \ldots \cdot p(v_k|P_k) = \prod_{j=1}^{k} p(v_j|E_j P_j - v_i). \qquad (15)$$

□

Equation (15) explicitly demonstrates the form of the factorization in terms of CPTs after variable v_i is eliminated. Hence, the right-side of (15) is used in the MCL algorithm to adjust the label of the CPT for each variable in v_i's child-set. That is, $\prod_{j=1}^{k}$ corresponds to the *for-loop* construct running from $j = k, \ldots, 1$, while $p(v_j|E_j P_j - v_i)$ corresponds to the statement $P_j = E_j P_j - v_i$ for one iteration of the *for-loop*.

Example 14 Recall eliminating variable d in (2). By Example 11, $v_i = d$, $C_i = \{v_1 = f\}$, $P_1 = \{d, e\}$ and $E_1 = \{c\}$. By Lemma 3, $I(v_i, E_j, P_j - v_i)$ and $I(v_j, P_j, E_j)$ hold, namely, $I(d, c, e)$ and $I(f, de, c)$. By Lemma 4, $\sum_d p(d|c) \cdot p(f|d, e)$ is equal to $p(f|c, e)$, as shown in (3).

We now establish the correctness of our architecture by showing that our messages are equivalent to those of the SS architecture (Shafer and Shenoy 1990; Shafer 1996). Since their architecture computes the probability distributions stored in computer memory, while our architecture determines labels, let us assume that the label in each storage register of our architecture is replaced with its corresponding probability distribution.

Theorem 2 *Given a BN D and a JT for D. Apply our architecture and also the SS architecture. For any two neighbouring JT nodes N_i and N_j, the product of the distributions in the storage registers from N_i to N_j in our architecture is the message in the storage register from N_i to N_j in the SS architecture.*

Proof When a JT node N_i receives a message from a neighbour N_j, it is also receiving, indirectly, information from the nodes on the other side of N_j (Shafer 1996). Thus, without a loss of generality, let the JT for D consist of two nodes, N_1 and N_2. Consider the message from N_2 to N_1. Let Z be those variables v_m of N_2 such that the BN CPT $p(v_m|P_m)$ is assigned to N_2. The variables to be eliminated are $X = N_2 - N_1$. Let $Y = N_2 - (XZ)$. Following the discussion in Section 2, the SS message $\phi(N_2 \cap N_1)$ from N_2 to N_1 is:

$$\begin{aligned}
\phi(N_2 \cap N_1) &= \sum_X \phi(N_2) \\
&= \sum_X 1(N_2) \cdot \prod_{v_m \in Z} p(v_m|P_m) \\
&= \sum_X 1(Y) \cdot 1(XZ) \cdot \prod_{v_m \in Z} p(v_m|P_m) \\
&= 1(Y) \cdot \sum_X \prod_{v_m \in Z} p(v_m|P_m).
\end{aligned}$$

Consider the first variable v_i of X eliminated from $\prod_{v_m \in Z} p(v_m|P_m)$. By Lemma 3, for each $v_j \in C_i$, the independencies $I(v_i, E_j, P_j - v_i)$ and $I(v_j, P_j, E_j)$ hold in $p(U)$. By Lemma 4, the exact form of the CPTs is known after the elimination of v_i. By (Shafer 1996), the product of all remaining probability tables in the entire JT is the marginal distribution $p(U - v_i)$ of the original joint distribution $p(U)$. Let us more carefully examine the remaining CPTs in the entire JT. First, each variable in $U - v_i$ has exactly one CPT. It follows from Lemma 1 that the CPT-graph defined by all CPT labels remaining in the JT is a DAG. Therefore, the CPTs for the remaining variables $U - v_i$ are a BN defining the marginal distribution $p(U - v_i)$ of the original JPD $p(U)$. By the definition of probabilistic conditional independence, an independence holding in the marginal $p(U - v_i)$ necessarily means that it holds in the joint distribution $p(U)$. Thereby, we can recursively apply Lemmas 1, 3 and 4 to eliminate the other variables in X. The CPTs remaining from the marginalization of X from $\prod_{v_m \in Z} p(v_m|P_m)$ are exactly the distributions of the CPT labels output by the ICM algorithm when called by N_2 to eliminate X from $C = \{p(v_m|P_m) \mid v_m \in Z\}$. In our architecture, the storage registers from N_2 to N_1 for those variables in Y are still empty. Filling these empty storage registers with unity-potentials $1(v_l)$, $v_l \in Y$, follows directly from the definition of the unity-potential $1(Y)$. Therefore, the product of the distributions in the storage registers from N_2 to N_1 in our architecture is the message in the storage register from N_2 to N_1 in the SS architecture. □

Example 15 It can be verified that the identified CPTs shown in Fig. 5 are correct. In particular, the SS message $\phi(f, g)$ from $cdefgh$ to bfg is $p(f) \cdot 1(g)$.

Corollary 1 *The marginal distribution $p(N)$ for any JT node N can be computed by collecting all CPTs sent to N by N's neighbours and multiplying them with those CPTs assigned to N.*

5 Modeling inference involving evidence

Pearl (1988) emphasizes the importance of structure by opening his chapter on Markov and BNs with the following quote:

> Probability is not really about numbers; it is about the structure of reasoning.
> - G. Shafer

In this section, we extend our architecture to *model* the processing of evidence and show that it can still identify CPT messages. Modeling the processing of evidence is faster than the physical computation needed for evidence processing. By allowing our architecture to take full responsibility for modeling structure, we empirically demonstrate that LAZY can finish its work sooner.

It is important to realize that the processing of evidence $E = e$ can be viewed as computing marginal distributions (Schmidt and Shenoy 1998; Shafer 1996; Xu 1995). For each JT node N, compute the marginal $p(NE)$, from whence $p(N - E, E = e)$ can be obtained. The desired distribution $p(N - E|E = e)$ can then be determined via normalization.

Rules 6 and 7 extend our architecture to *model* the processing of evidence.

Rule 6. Given evidence $E = e$. For each node N in the JT, set $N = N \cup E$. On this augmented JT, apply Rules 1–5 of our architecture for modeling inference not involving evidence.

Rule 7. For each evidence variable $v \in E$, change each occurrence of v in an assigned or propagated CPT label from v to $v = \varepsilon$, where ε is the observed value of v.

We now show the correctness of our architecture for modeling inference involving evidence.

Theorem 3 *Given a BN D, a JT for D, and observed evidence $E = e$. After applying our architecture, extended by Rules 6 and 7 for modeling inference involving evidence, the probability information at each JT node N defines $p(N - E, E = e)$.*

Proof Apply Rule 6. By Corollary 1, the probability information at each node N is $p(NE)$. By selecting those configurations agreeing with $E = e$ in Rule 7, the probability information at each node is $p(N - E, E = e)$. □

Example 16 Consider evidence $b = 0$ in the real-world BN for CHD in Fig. 3. With respect to the CHD JT in Fig. 5, the CPT messages to be propagated are depicted in Fig. 9.

The next example involves three evidence variables.

Example 17 Consider the JT with assigned BN CPTs in Fig. 10 (top). Given evidence $a = 0, c = 0, f = 0$, the LAZY potentials propagated towards node de are shown (Madsen and Jensen 1999). In comparison, all CPT labels identified by our architecture are depicted in Fig. 10 (bottom).

We can identify the labels of the messages faster than the probability distributions themselves can be built in computer memory.

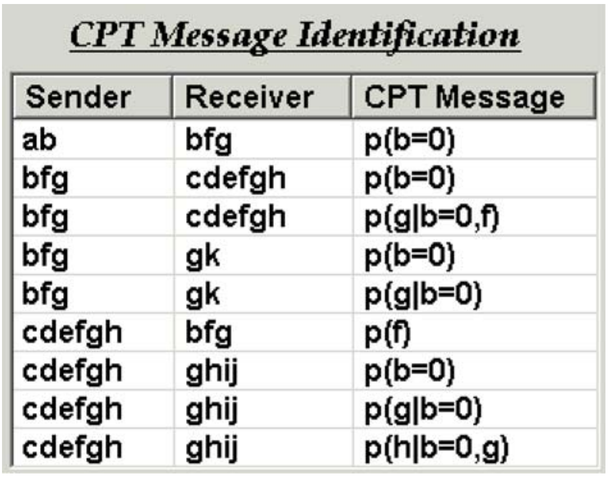
Example 18 Given evidence $b = 0$ in Example 16, constructing the distribution $p(b = 0)$ in memory for the message from node ab to bfg required 1.813 ms. Identifying all CPT messages in Fig. 9 required only 0.954 ms.

Although semantic modeling can be done significantly faster than physical computation, the LAZY architecture applies these two tasks iteratively. The consequence, as the next two examples show, is that semantic modeling must wait while the physical computation catches-up.

Example 19 Madsen and Jensen (1999) Consider the BN (left) and JT with assigned CPTs (right) in Fig. 11. Suppose evidence $d = 0$ is collected. Before node $bcdef$ can send its messages to node efg, it must first wait for the message $\phi(b, c)$ to be physically constructed at node abc as:

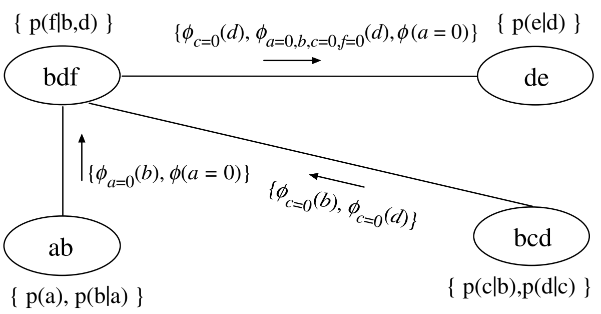
$$\phi(b, c) = \sum_{a} p(a) \cdot p(b|a) \cdot p(c|a). \tag{16}$$

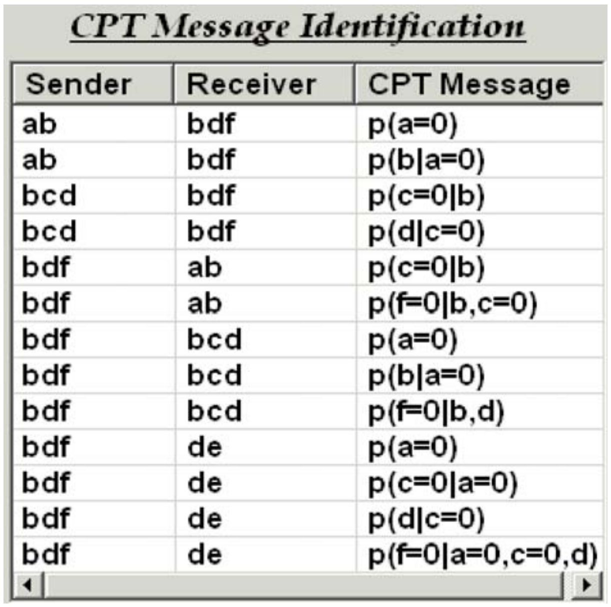
Upon receiving distribution $\phi(b, c)$ at node $bcdef$, LAZY exploits the independence $I(bc, d, ef)$ induced by the evidence $d = 0$ to identify that $\phi(b, c)$ is irrelevant to the computation for the messages to be sent from $bcdef$ to efg.


CPT Message Identification

Sender	Receiver	CPT Message
ab	bfg	p(b=0)
bfg	cdefgh	p(b=0)
bfg	cdefgh	p(g|b=0,f)
bfg	gk	p(b=0)
bfg	gk	p(g|b=0)
cdefgh	bfg	p(f)
cdefgh	ghij	p(b=0)
cdefgh	ghij	p(g|b=0)
cdefgh	ghij	p(h|b=0,g)

Fig. 9 Recall the real-world BN for CHD in Fig. 3 and the JT in Fig. 5. Given evidence $b = 0$, the propagated CPT labels are shown here


{ p(f|b,d) }
bdf
$\{\phi_{c=0}(d), \phi_{a=0,b,c=0,f=0}(d), \phi(a=0)\}$
{ p(e|d) }
de
$\{\phi_{a=0}(b), \phi(a=0)\}$
$\{\phi_{c=0}(b), \phi_{c=0}(d)\}$
ab
bcd
{ p(a), p(b|a) }
{ p(c|b),p(d|c) }



CPT Message Identification

Sender	Receiver	CPT Message
ab	bdf	p(a=0)
ab	bdf	p(b|a=0)
bcd	bdf	p(c=0|b)
bcd	bdf	p(d|c=0)
bdf	ab	p(c=0|b)
bdf	ab	p(f=0|b,c=0)
bdf	bcd	p(a=0)
bdf	bcd	p(b|a=0)
bdf	bcd	p(f=0|b,d)
bdf	de	p(a=0)
bdf	de	p(c=0|a=0)
bdf	de	p(d|c=0)
bdf	de	p(f=0|a=0,c=0,d)

Fig. 10 Madsen and Jensen (1999) A JT with assigned BN CPTs (*top*). Given evidence $a = 0, c = 0, f = 0$, this shows the LAZY potentials propagated towards node *de*. In contrast to Fig. 10 (*top*), this screen shot shows all identified CPT labels given evidence $a = 0, c = 0, f = 0$ (*bottom*)

In the exploitation of independencies induced by evidence, Example 19 explicitly demonstrates that LAZY forces node *bcdef* to wait for the construction of the irrelevant message $\phi(b, c)$.

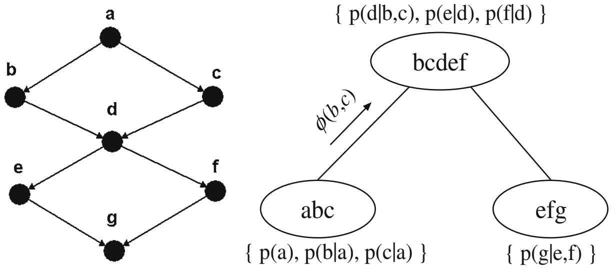


Fig. 11 Madsen and Jensen (1999) A BN (*left*) and a JT with assigned CPTs (*right*). LAZY exploits the independence $I(bc, d, ef)$ induced by evidence $d = 0$ only after the irrelevant potential $\phi(b, c)$ has been physically constructed at *abc* and sent to *bcdef*

Example 20 Madsen and Jensen (1999) Consider the BN (left) and JT with assigned CPTs (right) in Fig. 12. Before node *acde* can send its messages to node *ef*, it must first wait for the message $\phi(c, d|a)$ to be constructed in computer memory at node *abcd* as:

$$\phi(c, d|a) \quad = \quad \sum_{b} p(b\,|a) \cdot p(c|a, b) \cdot p(d|a, b). \tag{17}$$

Upon receiving distribution $\phi(c, d|a)$ at *acde*, LAZY exploits barren variables c and d to identify that $\phi(c, d|a)$ is irrelevant to the computation for the messages to be sent from *acde* to *ef*.

In the exploitation of barren variables, Example 20 explicitly demonstrates that LAZY forces node *acde* to wait for the physical construction of the irrelevant message $\phi(c, d|a)$. These unnecessary delays in Examples 19 and 20 are inherently built into the main philosophy of LAZY propagation (Madsen and Jensen 1999):

> The bulk of LAZY propagation is to maintain a multiplicative [factorization of the CPTs] and to postpone combination of [CPTs]. This gives opportunities for exploiting barren variables . . . during inference. . . . Thereby, when a message is to be computed only the required [CPTs] are combined.

The notion of a barren variable, however, is relative. For instance, in Example 20, variables c and d are not barren at the sending node *abcd*, but are barren at the receiving node *acde*. Thus, the interweaving of modeling structure and physical computation underlay these unnecessary delays.

We advocate the uncoupling of these two independent tasks. More specifically, we argue that modeling structure and computation of the actual probability distributions in computer memory should be performed separately. As our architecture can model structure faster than the physical computation can be performed, it can scout the structure in the JT and organize the collected information in three kinds of work schedules.

Our first work schedule pertains to non-empty messages that are irrelevant to subsequent message computation at the receiving node, as evident in Examples 19 and 20. More specifically, for three distinct nodes N_i, N_j and N_k such that N_i and N_j are neighbours, as are N_j and N_k, our first work schedule indicates that the messages from N_i to N_j are irrelevant in the physical construction of the messages to be sent from N_j to N_k.

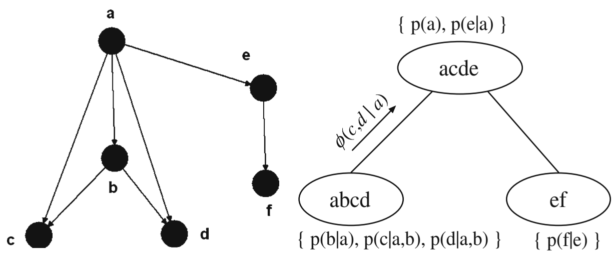
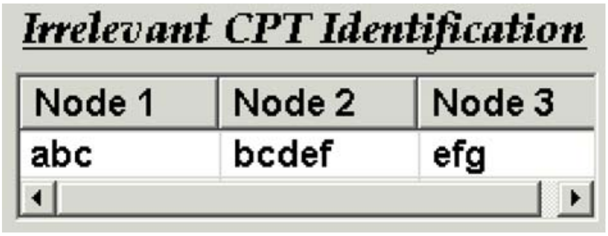


Fig. 12 Madsen and Jensen (1999) A BN (*left*) and a JT with assigned CPTs (*right*). LAZY exploits barren variables c and d only after the irrelevant potential $\phi(c, d|a)$ has been physically constructed at *abcd* and sent to *acde*



Irrelevant CPT Identification

Node 1	Node 2	Node 3
abc	bcdef	efg

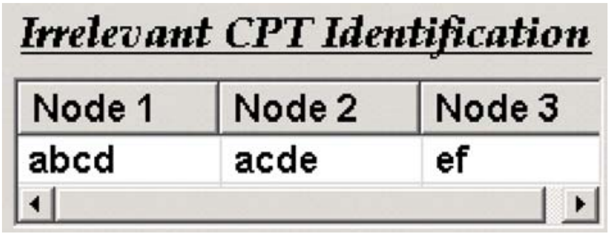
Fig. 13 Given evidence $d = 0$ in Example 19, our architecture can identify the irrelevant message from *abc* to *bcdef*

Example 21 Given evidence $d = 0$ in Example 19, the work schedule in Fig. 13 indicates that the messages from node abc are irrelevant to node $bcdef$ in the physical construction of the messages to be sent from $bcdef$ to node efg. Now reconsider Example 20. The work schedule in Fig. 14 indicates that the messages from node *abcd* are irrelevant to node *acde* in the physical construction of the messages to be sent from *acde* to node ef.

Our second work schedule indicates that an empty message will be propagated from a non-leaf node to a neighbour. (As LAZY could begin physical computation at the leaf JT nodes, we focus on the non-leaf JT nodes.) For instance, given evidence $b = 0$ in the extended CHD BN and JT with assigned CPTs in Fig. 15, the work schedule in Fig. 16 indicates that bfg will send *ab* an empty message. Our second work schedule saves LAZY time.

Example 22 Recall the extended CHD BN and JT in Fig. 15. Given evidence $b = 0$, LAZY determines that bfg will send *ab* an empty message only after bfg receives the probability distributions $\phi(f)$ from $cdefgh$. Physically constructing $\phi(f)$ involves eliminating the four variables c, d, e, h from the five potentials $p(c)$, $p(d|c)$, $p(e|c)$, $p(f|d, e)$, $p(h|c)$. In less time than is required for this physical computation, our architecture has generated the work schedule in Fig. 16. Without waiting for LAZY to eventually consider bfg, node *ab* can immediately send its messages $\{p(a|b = 0), p(b = 0)\}$to node *al*, which, in turn, can send its respective messages $\{p(b = 0), p(l|b = 0)\}$ to node *lm*.

Last, but not least, our architecture can generate another type of beneficial work schedule. This third work schedule indicates the variables that can be eliminated at a non-leaf node with respect to the messages for a particular neighbour, before the sending node has received any messages. Given evidence $b = 0$ in the CHD JT of Fig. 5, the work schedule in Fig. 17 indicates that, for instance, non-leaf node $cdefgh$



Irrelevant CPT Identification

Node 1	Node 2	Node 3
abcd	acde	ef

Fig. 14 For Example 20, our architecture can identify the irrelevant message from *abcd* to *acde*

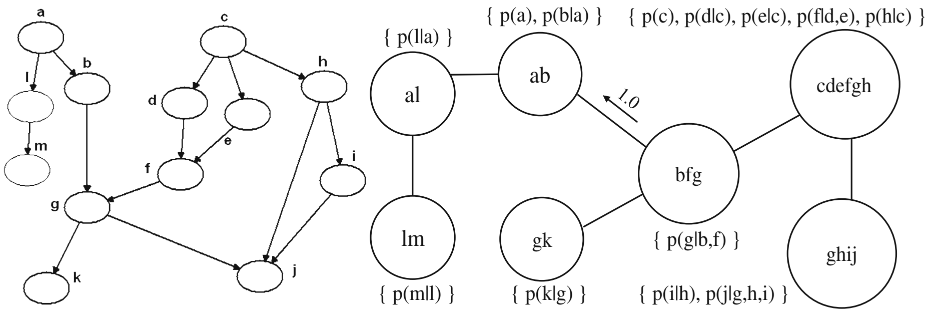


Fig. 15 An extended CHD BN (*left*) and a JT with assigned CPTs (*right*). Given evidence $b = 0$, LAZY forces node *ab* to wait for the identification of an irrelevant empty message from *bfg*

can immediately eliminate variables c, d, e in its construction of the message to *ghij*. Assisting LAZY to eliminate variables early saves time, as the next example clearly demonstrates.

Example 23 Given evidence $b = 0$ in the CHD JT of Fig. 5, consider the message from *cdefgh* to *ghij*. LAZY waits to eliminate variables c, d, e, f at *cdefgh* until *cdefgh* receives its messages $\{\phi(b = 0), \phi_{b=0}(f, g)\}$ from *bfg*, which, in turn, has to wait for message $\phi(b = 0)$ from *ab*. Thus, LAZY computes the message from *cdefgh* to *ghij* as:

$$\sum_{cdef} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d,e) \cdot p(h|c) \cdot \phi(b = 0) \cdot \phi_{b=0}(f, g). \tag{18}$$

In contrast, the work schedule of Fig. 17 allows LAZY to immediately eliminate variables c, d, e as:

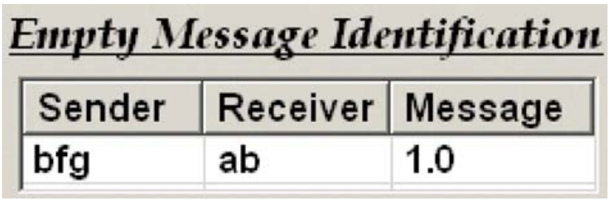
$$\phi(f, h) = \sum_{c,d,e} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d,e) \cdot p(h|c).$$

The benefit is that when the LAZY messages $\{\phi(b = 0), \phi_{b=0}(f, g)\}$ from *bfg* are received, only variable f remains to be eliminated. That is to say, LAZY's (18) is simplified to

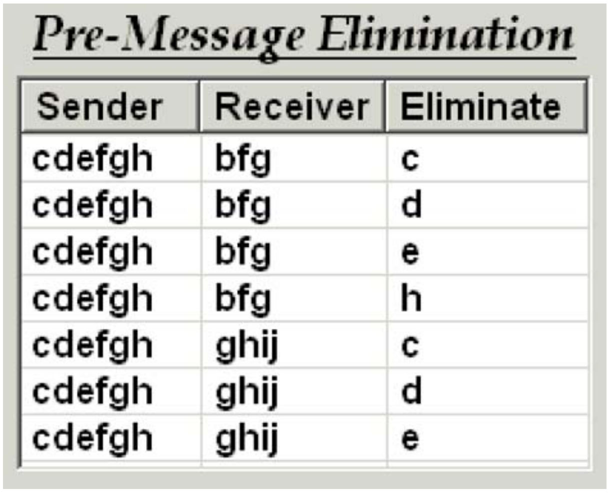
$$\sum_{f} \phi(f, h) \cdot \phi(b = 0) \cdot \phi_{b=0}(f, g). \tag{19}$$

In our CHD example, LAZY will eliminate c, d, e twice at node *cdefgh*, once for each message to *bfg* and *ghij*. This duplication of effort is a well-known undesirable

Fig. 16 Our architecture can identify all empty messages sent by non-leaf nodes given evidence $b = 0$ in the extended CHD JT of Fig. 15 (right)


Empty Message Identification

Sender	Receiver	Message
bfg	ab	1.0



Pre-Message Elimination

Sender	Receiver	Eliminate
cdefgh	bfg	c
cdefgh	bfg	d
cdefgh	bfg	e
cdefgh	bfg	h
cdefgh	ghij	c
cdefgh	ghij	d
cdefgh	ghij	e

Fig. 17 Given evidence $b = 0$ in the CHD JT of Fig. 5, our architecture identifies those variables that can be eliminated at non-leaf nodes before any messages are received

property in JT propagation (Shafer 1996). Utilizing our third work schedule, such as in Fig. 17, to remove this duplication remains for future work.

We conclude this section by providing some empirical results illustrating the usefulness of our JT architecture. In particular, we provide the time taken for LAZY propagation and the time saved by allowing our architecture to guide LAZY. The source code for LAZY propagation was obtained from (Consortium 2002), as was the code for generating a JT from a BN. Table 1 describes five real-world or benchmark BNs and their corresponding JTs used in the experiments.

We first measure the time cost of performing inference not involving evidence. Table 2 shows the time taken for : (i) LAZY propagation by itself, (ii) LAZY propagation guided by our architecture, (iii) time saved by using our approach, and (iv) time saved as a percentage. All propagation was timed in seconds using a SGI R12000 processor. Note that our architecture lowered the time taken for propagation in all five real-world or benchmark BNs. The time percentage saved ranged from 11.76% to 69.26% with an average percentage of 37.02%.

Next, we measure the time cost of performing inference involving evidence. As shown in Tables 3 and 4, approximately nine percent and eighteen percent of the variables in each BN are randomly instantiated as evidence variables, respectively. Note that once again our architecture lowered the time taken for propagation in all five real-world or benchmark BNs. In Table 3, the time percentage saved ranged from 0.42% to 15.48% with an average percentage of 6.42%. In Table 4,

Table 1 Five real-world or benchmark BNs used in our experiments

BN	Number of variables in the BN	Number of JT Nodes	Maximum number of variables in a JT node
Alarm	37	85	5
CHD	11	24	4
Hailfinder	56	127	5
Insurance	27	64	9
Mildew	35	73	5

Table 2 Experimental results on five BNs not involving evidence

BN	LAZY propagation time	LAZY propagation time guided by our architecture	Time saved	Percentage of time saved
Alarm	0.758	0.233	0.525	69.26%
CHD	0.083	0.054	0.029	34.94%
Hailfinder	2.56	2.259	0.301	11.76%
Insurance	8.932	7.223	1.709	19.13%
Mildew	882.276	441.032	441.244	50.01%

the time percentage saved ranged from 1.86% to 14.89% with an average percentage of 6.12%.

It is worth mentioning that our architecture is more useful with fewer evidence variables. One reason for this is that the cost of physical computation is lowered with collected evidence, since the probability tables to be propagated are much smaller. For instance, LAZY takes over 882 s to perform propagation not involving evidence on the Mildew BN, yet only takes about 15 s to perform propagation if 6 variables are instantiated as evidence variables. Therefore, LAZY needs more help in the case of processing no or fewer evidence variables and this is precisely when our architecture is most beneficial.

6 Local BNs and practical applications

After applying our architecture, as described in Section 3, to identify the probability information being passed in a JT, let us apply either method from (Madsen and Jensen 1999; Zhang 1998) to physically construct the identified CPT messages. That is, we assume that the label in each storage register of our architecture is replaced with its corresponding probability distributions. Unlike all previous JT architectures (Jensen et al. 1990; Lauritzen and Spiegelhalter 1988; Madsen and Jensen 1999; Shafer and Shenoy 1990), in our architecture, after propagation not involving evidence, each JT node N has a *sound*, *local* BN preserving all *conditional* independencies of the original BN involving variables in N. Practical applications of local BNs include an automated modeling procedure for MSBNs and a method for exploiting localized queries in DC techniques.

Table 3 Experimental results on five BNs with 9% of the variables randomly instantiated as evidence variables

BN	LAZY propagation time	LAZY propagation time guided by our architecture	Time saved	Percentage of time saved
Alarm	0.556	0.496	0.06	10.79%
CHD	0.084	0.071	0.013	15.48%
Hailfinder	0.98	0.948	0.021	3.36%
Insurance	1.76	1.724	0.036	2.05%
Mildew	692.61	689.68	2.93	0.42%

Table 4 Experimental results on five BNs with 18% of the variables randomly instantiated as evidence variables

BN	LAZY propagation time	LAZY propagation time guided by our architecture	Time saved	Percentage of time saved
Alarm	0.579	0.545	0.034	5.87%
CHD	0.047	0.040	0.007	14.89%
Hailfinder	0.783	0.762	0.021	2.68%
Insurance	0.453	0.429	0.024	5.29%
Mildew	15.03	15.00	0.028	1.86%

Lemma 5 *Given a BN D and a JT for D, apply Rules* 1–5 *in our architecture. For any JT node N, the CPTs assigned to N, together with all CPTs sent to N from N's neighbours, define a local BN D_N.*

Proof As previously mentioned, when a JT node N_i receives a message from a neighbour N_j, it is also receiving, indirectly, information from the nodes on the other side of N_j (Shafer 1996). Thus, without a loss of generality, let the JT for D consist of two nodes, N_1 and N_2. We only need focus on variables in the separator $N_1 \cap N_2$. Consider a variable v_m in $N_1 \cap N_2$ such that the BN CPT $p(v_m|P_m)$ is assigned to N_1. Thus, v_m is without a CPT label with respect to N_2. In N_1's call to ICM for its messages to N_2, variable v_m is not in $X = N_1 - N_2$, the set of variables to be eliminated, Thus, ICM will return a CPT label for v_m to N_1. By Rule 5, N_1 will place this CPT label in the empty storage register for v_m from N_1 to N_2. Therefore, after propagation, variable v_m has a CPT label at node N_2. Conversely, consider N_2's call of ICM for its messages to N_1. Since the BN CPT $p(v_m|P_m)$ is assigned to N_1, there is no CPT label for v_m passed to ICM. Thus, ICM does not return a CPT label for v_m to N_2. By Rule 5, N_2 fills the empty storage register of v_m from N_2 to N_1 with the unity-potential label $1(v_m)$. Therefore, after propagation, both N_1 and N_2 have precisely one CPT label for v_m. Moreover, for each node, it follows from Lemma 1 that the CPT-graph defined by the assigned CPTs and the message CPTs is a DAG. By definition, each JT node N has a local BN D_N. □

Example 24 Recall the JT with assigned CPTs in Fig. 5. Each JT node has a local BN after propagation not involving evidence, as depicted by a screen shot of our implemented system in Fig. 18.

Theorem 4 *Given Lemma 5. If an independence $I(X, Y, Z)$ holds in a local BN D_N for a JT node N, then $I(X, Y, Z)$ holds in the original BN D.*

Proof We will show the claim by removing variables in the JT towards the node N. Let v_i be the first variable eliminated in the JT for D. After calling MCL, let D' be the CPT-graph defined by the CPTs remaining at this node, together with the CPTs assigned to all other nodes. It follows from Lemma 1 that D' is a DAG. We now establish that $I(X, Y, Z)$ holding in D' implies that $I(X, Y, Z)$ holds in D. Note that since v_i is eliminated, $v_i \notin XYZ$. By contraposition, suppose $I(X, Y, Z)$ does not hold in D. By the method for testing independencies in Section 2, let D_m be the moralization of the sub-DAG of D onto $XYZ \cup An(XYZ)$. There are two cases

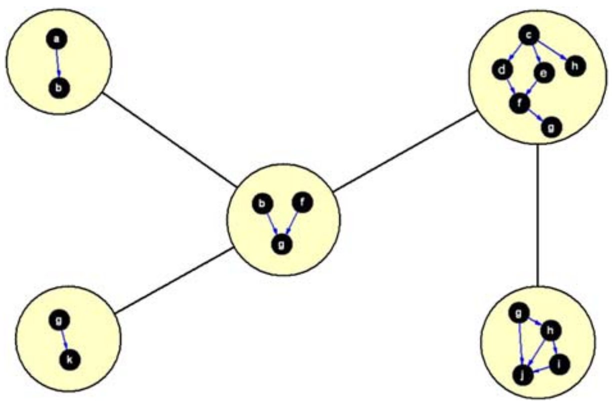


Fig. 18 *Local* BNs after propagation not involving evidence for the CHD BN

to consider. Suppose $v_i \notin An(XYZ)$. In this case, $I(X, Y, Z)$ does not hold in D' as D_m is also the moralization of the sub-DAG of D' onto $XYZ \cup An(XYZ)$. Now, suppose $v_i \in An(XYZ)$. Let D'_m be the moralization of the sub-DAG of D' onto $XYZ \cup An(XYZ)$. By assumption, there is an undirected path in D_m from X to Z, which does not involve Y. If this path does not involve v_i, this same path must exist in D'_m as the MCL algorithm only removes those edges involving v_i. Otherwise, as $v_i \notin XYZ$, this path must include edges (v, v_i) and (v_i, v'), where $v, v' \in P_iC_i$. Since the moralization process will add an undirected edge between every pair of variables in P_i, and since the MCL algorithm will add a directed edge from every variable in P_i to every variable in C_i and also from every variable v_j in C_i to every other variable v_k in C_i such that $v_j \prec v_k$ in Rule 2, it is necessarily the case that (v, v') is an undirected edge in the moralization D'_m of D'. As there is an undirected path in D'_m from X to Z not involving Y, by definition, $I(X, Y, Z)$ does not hold in D'. Thus, every independence $I(X, Y, Z)$ encoded in D' is encoded in the original BN D. Therefore, by recursively eliminating all variables $v \notin N$, all conditional independencies encoded in D_N are also encoded in D. □

Example 25 In Fig. 18, it is particularly illuminating that our architecture correctly models $I(g, c, h)$, the conditional independence of g and h given c, in the local BN for *cdefgh*, yet at the same time correctly models $I(g, h, i)$, the conditional independence of g and i given h, in the local BN for *ghij*.

Although unconditional independencies of the original BN might not be saved in the local BNs, Theorem 5 shows that conditional independencies are.

Theorem 5 *Given Lemma 5. If an independence $I(X, Y, Z)$ holds in the original BN D, where $Y \neq \emptyset$ and XYZ is a subset of a JT node N, then $I(X, Y, Z)$ holds in the local BN D_N.*

Proof Let $v_i \in U$ be the first variable eliminated by our architecture. Suppose FRC returns the set of CPT labels $C' = \{p(v_i|P_i), p(v_1|P_1), \ldots, p(v_k|P_k)\}$. By $\prec$ in Rule 2, the child-set of v_i is $C_i = \{v_1, \ldots, v_k\}$. The set of all variables appearing in any label of C' is E_kF_k. Clearly, the CPT-graph D' defined by C' is a DAG. By Corollary 6 in Pearl (1988), variable v_i is independent of all other variables in U given $E_k F_k - v_i$. Thus, we only need to show that any conditional independence $I(X, Y, Z)$ holding in D' with $v_i \notin XYZ$ is preserved in D'', where D'' is the CPT-graph defined by the set C'' of CPT labels output by MCL given C' as input. By Lemma 1, D'' is a DAG. By contraposition, suppose a conditional independence $I(X, Y, Z)$ does not hold in D''. According to the method for testing independencies in Section 2, let D'_m be the moralization of the sub-DAG of D' onto $XYZ \cup An(XYZ)$. There are two cases to consider. Suppose $v_i \notin An(XYZ)$. In this case, $I(X, Y, Z)$ does not hold in D' as D'_m is also the moralization of the sub-DAG of D'' onto $XYZ \cup An(XYZ)$. Now, suppose $v_i \in An(XYZ)$. By definition, the moralization process makes families complete. Observe that the family of each variable v_m in D' is, by definition, the family-set of v_m in the CPT label $p(v_m|P_m)$ of C'. For every variable v_m in the sub-DAG of D' onto $XYZ \cup An(XYZ)$, v_i is a member of the family-set F_m. Therefore, there is an edge (v_i, v) in D'_m between v_i and every other variable v in D'_m. Then, in particular, there is a path $(x, v_i), (v_i, z)$ from every variable $x \in X$ to every variable $z \in Z$. Since $v_i \notin Y$, by definition, $I(X, Y, Z)$ does not hold in D'. Hence, all conditional independencies on $U - v_i$ are kept. Therefore, by recursively eliminating all variables $v \notin N$, all conditional independencies on N are kept. That is, by Lemma 5, all conditional independencies $I(X, Y, Z)$ with $XYZ \subseteq N$ are preserved in the local BN D_N. □

Example 26 Recall the CPT-graphs in Figs. 7 and 6 defined by the CPT labels before and after the elimination of variable $v_i = f$, respectively, where $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$. It may seem as though some conditional independencies are lost due to the additional directed edges like (c, l), (e, l) and (j, l), where $c \in P_i$, $e \in P_1$ and $j, l \in C_i$. On the contrary, although $I(c, ejkhi, l)$, $I(e, bcdhjk, l)$ and $I(j, cde, l)$ do not hold in Fig. 6, these independencies do not hold in Fig. 7 either. Some unconditional independencies were lost, however, such as $I(be, \emptyset, kl)$.

Our first practical application of local BNs concerns *MSBNs* (Xiang 1996, 2002; Xiang and Jensen 1999; Xiang et al. 2006, 2000, 1993), which are formally defined as follows.

Definition 19 A *MSBN* is a finite set $\{B_1, B_2, \ldots, B_n\}$ of BNs satisfying the following condition: $N_1, N_2, \ldots, N_n$ can be organized as a join tee, where N_i is the set of variables in the local BN B_i, $i = 1, 2, \ldots, n$.

Before any inference takes place, a given BN needs first be modeled as a MSBN. Several problems, however, with the manual construction of a MSBN from a BN have recently been acknowledged (Xiang et al. 2000). We resolve this modeling problem as follows.

Recently, Olesen and Madsen (2002) gave a simple method for constructing a special JT based on the maximal prime decomposition (MPD) of a BN. One desirable property of a MPD JT is that the JT nodes are unique for a given BN. We favour MPD JTs over conventional JTs, since they facilitate inference in the LAZY architecture while still only requiring polynomial time for construction (Olesen and Madsen 2002). For example, given the CHD BN in Fig. 3, the JT shown in Fig. 5 is the unique MPD JT.

We now propose Algorithm 3 to introduce an *automated* procedure for constructing a MSBN from a given BN.

Algorithm 3 Construct-MSBN(D)

```
Input: A BN $D$.
Output: A MSBN $\{B_1, B_2, \ldots, B_n\}$.
begin
1. Construct a MPD JT with nodes $\{N_1, N_2, \ldots, N_n\}$ for the BN $D$.
2. Assign the CPTs of the BN $D$ to the MPD JT nodes $N_1, N_2, \ldots, N_n$ as usual.
3. Apply our architecture to label the messages to be propagated during JT
   propagation.
4. Compute the distributions of the identified CPTs in Step 3 using any of the
   available inference algorithms.
5. Define the local BN $B_i$ for each $N_i$ to be the CPTs assigned
   and passed to $N_i$.
6. Return the MSBN $\{B_1, B_2, \ldots, B_n\}$.
```

Example 27 We illustrate Algorithm 3 using the real-world CHD BN in Fig. 3. The MPD JT with assigned BN CPTs is shown in Fig. 5. The CPTs identified by our architecture are listed in Fig. 8. Apply any probabilistic inference algorithm to physically construct these CPTs. By definition, Fig. 18 is a MSBN for the CHD BN.

We now establish the correctness of Algorithm 3.

Lemma 6 *Given as input a BN D, the output $\{B_1, B_2, \ldots, B_n\}$ of Algorithm 3 is a MSBN.*

Proof The claim holds immediately by Lemma 5. □

It is worth emphasizing that the local BNs have been shown in Theorems 4 and 5 to possess two favourable features, namely, the local BNs are sound and they preserve all conditional independencies of the original BN onto the context of the variables in each JT node. Note that *recursive conditioning* (Allen and Darwiche 2003) can also be used to build the CPTs in step 4 of Algorithm 3. Recursive conditioning was recently introduced as the first any-space algorithm for exact inference in BNs. Recursive conditioning finds an optimal cache factor to store the probability distributions under different memory constraints. The experimental results in Allen and Darwiche (2003) show that the memory requirements for inference in many large real-world BNs can be significantly reduced by recursive conditioning. Therefore, recursive conditioning allows for performing exact inference in the situations previously considered impractical (Allen and Darwiche 2003).

The significance of our suggestion is not aimed at an improvement in MSBN computational efficiency. Instead, an automated procedure for the semantic modeling of MSBNs overcomes the recently acknowledged problems with manually constructing MSBNs (Xiang et al. 2000).

Our second practical application of local BNs concerns *DC* (Dechter 1996; Li and D'Ambrosio 1994; Zhang 1998). MSBNs are well established in the probabilistic

Table 5 The computation needed in DC to process five localized queries in the original CHD BN in Fig. 3 versus the local BNs in Fig. 18

Localized	Original BN			Local BNs		
query	$+$	$\times$	$\div$	$+$	$\times$	$\div$
$p(a\|b=0)$	1	2	2	1	2	2
$p(b\|f=0, g=0)$	3	8	2	1	4	2
$p(d\|h=0)$	3	6	2	3	6	2
$p(g\|h=0, i=0, j=0)$	19	42	2	1	6	2
$p(g\|k=0)$	19	38	2	1	2	2

reasoning community due, in large part, to the presence of *localized* queries (Xiang 1996, 2002; Xiang and Jensen 1999). That is, practical experience has previously demonstrated that queries tend to involve variables in close proximity within the BN (Xiang et al. 1993). We conclude our discussion by showing how our semantic architecture allows DC to exploit localized queries.

DC is usually better than JT propagation, if one only is interested in updating a small set of non-evidence variables (Madsen and Jensen 1999), where small is shown empirically to be twenty or fewer variables in Zhang (1998). However, DC processes every query using the original BN. Therefore, it is not exploiting localized queries.

Our semantic architecture models the original BN as a set of *local* BNs, once the actual probability distributions corresponding to the identified CPT labels have been constructed in computer memory. Hence, DC techniques can process localized queries in local BNs. The following empirical evaluation is styled after the one reported by Schmidt and Shenoy (1998).

Example 28 We suggest that the CHD BN in Fig. 3 be represented as the smaller local BNs in Fig. 18, after the physical construction of CPTs $p(b)$, $p(f)$, $p(g)$, $p(g|f)$ and $p(h|g)$. Table 5 shows the work needed by DC to answer five localized queries using the original CHD BN of Fig. 3 in comparison to using the local BNs of Fig. 18.

While it is acknowledged that our suggestion here is beneficial only for localized queries, practical experience with BNs, such as in neuromuscular diagnosis (Xiang et al. 1993), has long established that localized queries are a reality.

7 Conclusion

Unlike all previous JT architectures (Jensen et al. 1990; Lauritzen and Spiegelhalter 1988; Madsen and Jensen 1999; Shafer and Shenoy 1990), we have proposed the first architecture to precisely *model* the processing of evidence in terms of CPTs. The key advantage is that we can identify the labels of the messages significantly faster than the probability distributions themselves can be built in computer memory. For instance, in the medical BN for CHD, our architecture can identify all messages to be propagated in the JT in less time than it takes to physically construct one message (see Example 18). We can assist LAZY propagation (Madsen and Jensen 1999), which interlaces semantic modeling with physical computation, by uncoupling these two independent tasks. Treating semantic modeling and physical computation as being dependent practically ensures that LAZY will perform probability propagation unnecessarily slowly. When exploiting barren variables and independencies induced by evidence, Examples 19 and 20 explicitly demonstrate that LAZY forced a node to wait for the physical construction of a non-empty message that was irrelevant to its subsequent message computation. These irrelevant non-empty messages are identified by our first work schedule, as depicted in Figs. 13 and 14. Another advantage of allowing semantic modeling to scout the structure in the JT is our second work schedule, which presents the empty messages propagated from non-leaf JT nodes, such as shown in Fig. 16. This second work schedule is beneficial as was demonstrated in Example 22, where LAZY forced a receiving node to wait for the identification of an empty message that would be neither constructed nor sent. Finally, to send a message from one node to a neighbour, LAZY does not eliminate any variables at the sending node until all messages have been received from its other neighbours (see Example 23). Our third work schedule lists those variables that can be eliminated before any messages are received, as the screen shot in Fig. 17 indicates. Besides the real-world BN for CHD, we also evaluated our architecture on four benchmark BNs, called Alarm, Insurance, Hailfinder and Mildew. The experimental results reported in Tables 2, 3 and 4 are very encouraging. The important point, with respect to JT probability propagation, is that our architecture can assist LAZY by saving time, which is the measure used to compare inference methods in Madsen and Jensen (1999).

Even when modeling inference not involving evidence, our architecture still is useful to the MSBN technique and to the DC techniques. We have shown that our JT propagation architecture is instrumental in developing an automated procedure for constructing a MSBN from a given BN. This is a worthwhile result, since several problems with the manual construction of a MSBN from a BN have recently been acknowledged (Xiang et al. 2000). We also have suggested a method for exploiting localized queries in DC techniques. Practical experience, such as that gained from neuromuscular diagnosis (Xiang et al. 1993), has demonstrated that queries tend to involve variables in close proximity within a BN. Our approach allows DC to process localized queries in local BNs. The experimental results in Table 5 involving a real-world BN for CHD show promise.

In his eloquent review of three traditional JT architectures (Jensen et al. 1990; Lauritzen and Spiegelhalter 1988; Shafer and Shenoy 1990), Shafer (1996) writes that the notion of *probabilistic conditional independence* (Wong et al. 2000) does not play a major role in inference. More recently, the LAZY architecture has demonstrated a remarkable improvement in efficiency over the traditional methods by actively exploiting independencies to remove irrelevant potentials before variable elimination. However, LAZY propagation does not utilize the independencies holding in the relevant potentials. In our architecture, we introduce the notions of *parent-set* and *elder-set* in order to take advantage of these valuable independencies. Based on this exploitation of independency information, we believe that the computationally efficient LAZY method, and the semantically rich architecture proposed here, serve as complementary examples of second-generation JT probability propagation architectures.

Acknowledgements This research is supported by NSERC Discovery Grant 238880. The authors would like to thank F.V. Jensen, Q. Hu, H. Geng, C.A. Maguire and anonymous reviewers, for insightful suggestions.

References

Allen, D., & Darwiche, A. (2003). Optimal time-space tradeoff in probabilistic inference, In *Proc. 18th international joint conference on artificial intelligence* (pp. 969-975). Acapulco, Mexico.

Becker, A., & Geiger, D. (2001). A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence, 125*(1–2) 3–17.

Castillo, E., Gutiérrez, J., & Hadi, A. (1997). *Expert systems and probabilistic network models.* New York: Springer.

Consortium, E. (2002). Elvira: An environment for probabilistic graphical models. In *Proceedings of the 1st European workshop on probabilistic graphical models* (pp. 222–230). Cuenca, Espana.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence, 42*(2–3), 393–405.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms.* Toronto: MIT.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., & Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems.* New York: Springer.

Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In *Proc. 12th conference on uncertainty in artificial intelligence* (pp. 211–219). Portland, OR.

Hájek, P., Havránek, T., & Jiroušek, R. (1992). *Uncertain information processing in expert systems.* Ann Arbor: CRC.

Jensen, F. V. (1996). *An introduction to Bayesian networks.* London: UCL.

Jensen, F. V., Lauritzen, S. L., & Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly, 4*, 269–282.

Kjaerulff, U. (1990). *Triangulation of graphs—algorithms giving small total state space, Research Report R-90-09.* Dept. of Math. and Comp. Sci., Aalborg University, Denmark.

Kozlov, A. V., & Singh, J. P. (1999). Parallel implementations of probabilistic inference. *IEEE Computer, 29*(12), 33–40.

Lauritzen, S. L., & 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), 157–244.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., & Leimer, H. G. (1990). Independence properties of directed Markov fields. *Networks, 20*(5), 491–505.

Li, Z., & D'Ambrosio, B. (1994). Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning, 11*(1), 55–81.

Madsen, A. L., & Jensen, F. V. (1999). LAZY propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence, 113*(1–2), 203–245.

Madsen, A. L., & Jensen, F. V. (1999). *Parallelization of inference in Bayesian networks.* Research Report R-99-5002, Dept. of Comp. Sci., Aalborg University, Denmark

Neapolitan, R. E. (1990). *Probabilistic reasoning in expert systems.* Toronto: Wiley.

Olesen, K. G., & Madsen, A. L. (2002). Maximal prime subgraph decomposition of Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics B, 32*(1), 21–31.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Francisco: Morgan Kaufmann.

Schmidt, T., & Shenoy, P. P. (1998). Some improvements to the Shenoy-Shafer and Hugin architectures for computing marginals. *Artificial Intelligence, 102*, 323–333.

Shachter, R. (1986). Evaluating influence diagrams. *Operational Research, 34*(6), 871–882.

Shafer, G., & Shenoy, P. P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence, 2*, 327–352.

Shafer, G. (1996). *Probabilistic expert systems.* Philadelphia: SIAM.

Wong, S. K. M., Butz, C. J., & Wu, D. (2000). On the implication problem for probabilistic conditional independency. *IEEE Transactions on Systems, Man and Cybernetics A, 30*(6), 785–805.

Xiang, Y. (1996). A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence, 87*(1–2), 295–342.

Xiang, Y. (2002). *Probabilistic reasoning in multiagent systems: A graphical models approach*. New York: Cambridge University Press.

Xiang, Y., & Jensen, F. V. (1999). Inference in multiply sectioned Bayesian networks with extended Shafer-Shenoy and Lazy propagation, In *Proc. 15th conference on uncertainty in artificial intelligence* (pp. 680–687). Stockholm, Sweden.

Xiang, Y., Jensen, F. V., & Chen, X. (2006). Inference in multiply sectioned Bayesian networks: Methods and performance comparison. *IEEE Transactions on Systems, Man and Cybernetics B, 36*(3), 546–558.

Xiang, Y., Olesen, K. G., & Jensen, F. V. (2000). Practical issues in modeling large diagnostic systems with multiply sectioned Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence, 14*(1), 59–71.

Xiang, Y., Pant, B., Eisen, A., Beddoes, M. P., & Poole, D. (1993). Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine, 5*, 293–314.

Xu, H. (1995). Computing marginals for arbitrary subsets from marginal representation in Markov trees. *Artificial Intelligence, 74*(1), 177–189.

Yannakakis, M. (1981). Computing the minimal fill-in is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods, 2*, 77–79.

Zhang, N. L. (1998). Computational properties of two exact algorithms for Bayesian networks. *Applied Intelligence, 9*(2), 173–184.