International Journal of Approximate Reasoning 52 (2011) 948–959

Contents lists available at SciVerse ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

# Join tree propagation utilizing both arc reversal and variable elimination

C.J. Butz [a,*], K. Konkel [a], P. Lingras [b]

[a] Department of Computer Science, University of Regina, Regina, SK, Canada S4S 0A2
[b] Department of Mathematics and Computing Science, Saint Mary's University, Halifax, NS, Canada B3H 3C3

ARTICLE INFO

*Article history:*
Available online 24 November 2010

*Keywords:*
Bayesian networks
Join tree propagation
Variable elimination
Arc reversal

ABSTRACT

In this paper, we put forth the first join tree propagation algorithm that selectively applies either *arc reversal* (AR) or *variable elimination* (VE) to build the propagated messages. Our approach utilizes a recent method for identifying the propagated join tree messages à priori. When it is determined that a join tree node will construct a single distribution to be sent to a neighbouring node, VE is utilized as it builds a single distribution in the most direct fashion; otherwise, AR is applied as it maintains a factorization of distributions allowing for barren variables to be exploited during propagation later on in the join tree. Experimental results, involving evidence processing in four benchmark Bayesian networks, empirically demonstrate that selectively applying VE and AR is faster than applying one of these methods exclusively on the entire network.

## 1. Introduction

*Bayesian networks* [23,21,27,15] provide a rigorous foundation for uncertainty management by combining probability theory and graph theory, and have been successfully applied in practice to a wide variety of problem domains. A Bayesian network consists of a *directed acyclic graph* (DAG) [23] and a corresponding set of *conditional probability tables* (CPTs) [25]. The vertices in the DAG represent the random variables in the real-world problem, while the arcs in the graph represent probabilistic dependencies amongst the variables. More specifically, the *probabilistic conditional independencies* [26] encoded in the DAG indicate that the product of the CPTs is a joint probability distribution. Therefore, Bayesian networks continue to provide a robust framework for designing expert systems [15]. Although Cooper [11] has shown that the complexity of exact inference in discrete Bayesian networks is NP-hard, various approaches have been developed that seem to work quite well in practice such as [7,16,18,19]. All of these methods center around eliminating variables from the networks to produce posterior probability distributions and can be broadly classified into two categories.

The first category of Bayesian network inference is *direct computation*. The two leading direct computation algorithms are *variable elimination* (VE) [23,13,25,28] and *arc reversal* (AR) [22,24]. VE removes a variable by multiplying together all of the distributions involving the variable and then summing the variable out of the obtained product. AR removes a variable by reversing the arcs between the variable and its children giving a modified DAG and then building the CPTs corresponding to the modified graph. The second category is *join tree propagation*, which Shafer [25] emphasizes is central to the theory and practice of probabilistic expert systems. Join tree propagation first builds a secondary network, called a join tree, from the DAG of the Bayesian network and then performs inference by propagating probabilities in the join tree.

Madsen [18,19] introduced *Lazy-AR* as a variant of its predecessor, *Lazy-VE* [17]. The only difference between Lazy-AR and Lazy-VE is that the former utilizes AR when eliminating variables during join tree propagation, whereas the latter uses VE. The empirical results of [18,19] seem to show that Lazy-AR is sometimes better and usually no worse than Lazy-VE.

* Corresponding author. Tel.: +1 306 585 4856; fax: +1 306 585 4745.
*E-mail addresses:* butz@cs.uregina.ca (C.J. Butz), konkel1k@cs.uregina.ca (K. Konkel), pawan.lingras@stmarys.ca (P. Lingras).

0888-613X/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2010.11.006

Both Lazy-AR and Lazy-VE, however, are too rigid to exploit various kinds of structures found within real-world Bayesian networks. We explicitly demonstrate that during propagation in one join tree, AR can be the best choice for eliminating variables at the one join tree node, yet VE is the most suitable method for eliminating variables at another join tree node. Neither Lazy-AR nor Lazy-VE are flexible enough to take advantage of these situations as they both apply a single technique for eliminating variables throughout the entire join tree.

In this paper, we suggest selectively applying either AR or VE to build the messages propagated in a join tree. A key difference between our system, called *DataBayes*, and all other join tree propagation algorithms are two analytical preprocessing steps. The first step uses the method in [9] to determine à priori those messages that will be propagated in the join tree. The second step uses this information as follows. When it is known that a join tree node will construct a single distribution to be sent to a neighbouring node, VE is applied to build this message. On the other hand, AR is applied when a node needs to construct and to pass more than one distribution to a neighbouring node. Using evidence processing in a real-world Bayesian network, we explicitly show that AR is better suited to construct messages at certain join tree nodes, while VE is better suited to construct other messages at different nodes in the same join tree. The advantage of AR over VE is that AR maintains a factorization of distributions, which allows barren variables to be exploited later on during propagation. In contrast, VE sacrifices this opportunity by computing the product of the factorization as a single distribution. We show this exploitation by AR and VE's lost opportunity with a concrete example. On the contrary, as previously pointed out in [6], VE's advantage over AR is that AR can build intermediate distributions that will neither be passed as messages, nor are they needed in the construction of the distribution to be passed. VE, in stark contrast, constructs a single distribution in the most direct fashion. Again, we show the advantage of VE's direct computation and AR's wasteful indirect computation using an explicit example. The efficiency improvement offered by DataBayes is shown through empirical evaluations involving four benchmark Bayesian networks. DataBayes finished inference faster than Lazy-AR in all the cases without exception. Since Lazy-AR tends not to be slower than Lazy-VE [18,19], our results empirically demonstrate that selectively applying VE and AR is faster than applying one of these methods exclusively.

This paper is organized as follows. In Section 2, background information is reviewed. We then propose a more sophisticated approach to join tree propagation in Section 3. In Sections 4 and 5, advantages of our flexible approach are given. Experimental results are provided in Section 6. In Section 7, related works are discussed. Our conclusions are given in Section 8.

## 2. Preliminaries

Here we review results from discrete Bayesian networks, and three approaches for exact inference therein.

Consider a finite set of discrete random variables $U = \{v_1, v_2, \ldots, v_n\}$. Let $dom(v_i)$ denote the finite domain of values that each variable $v_i \in U$ can assume. For a subset $X \subseteq U$, the Cartesian product of the domains of the individual variables in $X$ is $dom(X)$. An element $x \in dom(X)$ is a *configuration* or *row* of $X$. A *potential* [13] on $dom(X)$ is a function $\phi$ such that $\phi(x) \geq 0$, for each configuration $x \in dom(X)$, and at least one $\phi(x)$ is positive. For simplicity we speak of a potential as defined on $X$ instead of on $dom(X)$, and we call $X$ its domain rather than $dom(X)$ [25]. A *joint probability distribution* [25] on $U$, written $p(U)$, is a function $p$ on $U$ satisfying the following two conditions: (i) $0 \leq p(u) \leq 1$, for each configuration $u \in dom(U)$; (ii) $\sum_{u \in dom(U)} p(u) = 1$. Let $X$ and $Y$ be two disjoint subsets of $U$. A *conditional probability table* (CPT) [25] for $Y$ given $X$, denoted $p(Y|X)$, is a nonnegative function on $X \cup Y$, satisfying the following condition: for each configuration $x \in dom(X)$, $\sum_{y \in dom(Y)} p(Y = y \mid X = x) = 1$. 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 shown in Table 1. Note that the missing probabilities can be obtained by the definition of a CPT. For instance, $p(a = 0) = 0.504$ and $p(b = 0|a = 0) = 0.948$.

The *heading* of a CPT is the label shown above the probability column. For instance, the heading of CPT $p(a)$ in Table 1 is the label "$p(a)$" appearing above the probability column. It will always be made clear as to whether $p(Y|X)$ refers to the heading or the CPT itself. Moreover, whenever $p(Y|X)$ is written with $X$ and $Y$ not disjoint, we mean $p(Y|X - Y)$ to satisfy the disjointness condition of CPTs.

A discrete *Bayesian network* [23] on $U = \{v_1, v_2, \ldots, v_n\}$ is a pair $(D, C)$. $D$ is a DAG with a vertex set $U$. $C$ is the set of CPTs $\{p(v_i|P_i) \mid i = 1, 2, \ldots, n\}$, where $P_i$ denotes the parents of variable $v_i \in D$. For example, the DAG in Fig. 1(i) together

**Table 1**
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 $p(k|g)$.

| $a$ | $p(a)$ |
|---|---|
| 1 | 0.496 |

| $a$ | $b$ | $p(b\|a)$ |
|---|---|---|
| 0 | 1 | 0.052 |
| 1 | 1 | 0.358 |

| $c$ | $e$ | $p(e\|c)$ |
|---|---|---|
| 0 | 1 | 0.383 |
| 1 | 1 | 0.286 |

| $c$ | $p(c)$ |
|---|---|
| 1 | 0.577 |

| $c$ | $d$ | $p(d\|c)$ |
|---|---|---|
| 0 | 1 | 0.714 |
| 1 | 1 | 0.627 |

| $c$ | $h$ | $p(h\|c)$ |
|---|---|---|
| 0 | 1 | 0.214 |
| 1 | 1 | 0.651 |

| $d$ | $e$ | $f$ | $p(f\|d, e)$ |
|---|---|---|---|
| 0 | 0 | 1 | 0.710 |
| 0 | 1 | 1 | 0.193 |
| 1 | 0 | 1 | 0.485 |
| 1 | 1 | 1 | 0.602 |

| $h$ | $i$ | $p(i\|h)$ |
|---|---|---|
| 0 | 1 | 0.104 |
| 1 | 1 | 0.369 |

| $b$ | $f$ | $g$ | $p(g\|b, f)$ |
|---|---|---|---|
| 0 | 0 | 1 | 0.027 |
| 0 | 1 | 1 | 0.123 |
| 1 | 0 | 1 | 0.898 |
| 1 | 1 | 1 | 0.405 |

| $g$ | $k$ | $p(k\|g)$ |
|---|---|---|
| 0 | 1 | 0.593 |
| 1 | 1 | 0.416 |

| $g$ | $h$ | $i$ | $j$ | $p(j\|g, h, i)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0.178 |
| 0 | 0 | 1 | 1 | 0.565 |
| 0 | 1 | 0 | 1 | 0.446 |
| 0 | 1 | 1 | 1 | 0.729 |
| 1 | 0 | 0 | 1 | 0.931 |
| 1 | 0 | 1 | 1 | 0.582 |
| 1 | 1 | 0 | 1 | 0.403 |
| 1 | 1 | 1 | 1 | 0.222 |

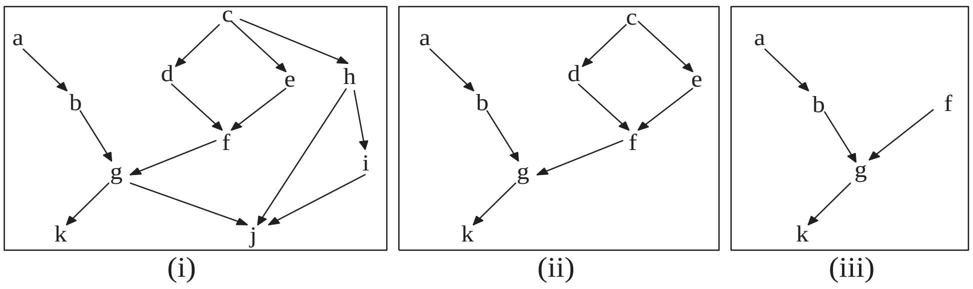

**Fig. 1.** (i) *The coronary heart disease* (CHD) Bayesian network [13]. Given the query $p(k|f=0)$: (ii) barren variables $h$, $i$ and $j$ have been pruned; (iii) variables $c$, $d$ and $e$ are also removed as they are independent of $k$ given evidence $f=0$.

with the corresponding CPTs in Table 1 is based on a real-world Bayesian network for *coronary heart disease* (CHD) [13]. Here, the parents $P_i$ of variable $v_i = g$ are $P_i = \{b, f\}$.

A *topological ordering* [10] of the variables in a Bayesian network is denoted by $\prec$. The *family* $F_i$ of a variable $v_i$ in a Bayesian network is the variable together with its parents, that is $\{v_i\} \cup P_i$. For each child $v_j$ of $v_i$, we use $A_j = F_i F_1 F_2 \cdots F_j$, where $v_1 \prec v_2 \prec \cdots \prec v_j$, and $WZ$ denotes set union $W \cup Z$. In other words, for child $v_j$, $A_j$ denotes the family-sets of its parent $v_i$, itself, and its older siblings. We define $B_j$ to be $A_j - v_i$, and $A_0 = P_i$. A variable without parents is called a *root* variable.

A Bayesian network $D$ graphically encodes *probabilistic conditional independencies* [26], which can be inferred from $D$ using the *d-separation* algorithm [12]. Based on the independencies encoded in $D$, the product of the CPTs in $C$ is a joint distribution $p(U)$ [12], namely, $p(U) = \prod_{v_i \in U} p(v_i|P_i)$. For example, the independencies encoded in the Bayesian network of Fig. 1(i) indicate that the product of the CPTs in Table 1 is a joint distribution on $U = \{a, b, c, d, \ldots, k\}$, namely, $p(U) = p(a) \cdot p(b|a) \cdot p(c) \cdot p(d|c) \cdots p(k|g)$. Thereby, one favourable feature of Bayesian networks is that they provide a compact, graphical representation of a joint distribution modelling a real-world problem domain. For instance, only 30 probabilities are required for the CHD Bayesian network in Fig. 1(i) versus $2^{11} - 1$ probabilities required for specifying the joint distribution $p(U)$ directly.

Suppose that the values $e$ of a set $E$ of variables in a Bayesian network have been observed and that the posterior probabilities of set $X$ (disjoint with $E$) are sought. All variables outside of $E \cup X$ must necessarily be eliminated in answering this query, denoted $p(X|E=e)$. A brute-force approach to eliminate these variables, however, can involve unnecessary manipulation of probability distributions in memory. Given a Bayesian network $D$ and a query $p(X|E=e)$, a variable $v_i$ is *barren* [24] if $(\{v_i\} \cup Y) \cap (X \cup E) = \emptyset$, where $Y$ is the set of descendants of $v_i$ in $D$. For example, given the query $p(k|f=0)$ posed to the Bayesian network in Fig. 1(i), variables $h$, $i$ and $j$ are barren. Thus, they can be removed, yielding the network in Fig. 1(ii).

Similarly, independencies induced by evidence can also be taken advantage of to save unnecessary physical computation. Baker and Boult [3] proposed an algorithm, which we will call *Prune*, that prunes all variables from a Bayesian network that are irrelevant to a given query $p(X|E=e)$. Their algorithm removes barren variables as well as those variables rendered immaterial to $X$ given the evidence $E=e$. Note that the time complexity of Prune is $O(|\lambda|)$, where $|\lambda|$ is the number of arcs in the Bayesian network [3]. For example, given evidence $f=0$ in query $p(k|f=0)$ posed to Fig. 1(ii), variable $k$ is conditionally independent of variables $c$, $d$ and $e$. Thus, $c$, $d$ and $e$ can be safely removed to yield the smaller Bayesian network in Fig. 1(iii).

A root variable $v_i$ that is also an evidence variable can have its CPT ignored and, for each child $v_j$ of $v_i$, the CPT $p(v_j|P_j)$ is modified to agree with the observed evidence. In our running example, since $f$ is both an evidence variable and a root variable in Fig. 1(iii), CPT $p(f|d, e)$ is ignored and CPT $p(g|b, f)$ for the child $g$ of $f$ is modified to only contain rows agreeing with the evidence $f=0$ [25]. That is, all rows in $p(g|b, f)$ with $f=1$ are deleted leaving $p(g|b, f=0)$ stored in the computer memory. The query $p(k|f=0)$ can now be answered by eliminating variables $a$, $b$ and $g$ from the distributions stored in the computer memory.

Join tree propagation is central to the theory and practice of probabilistic expert systems [25]. A *join tree* [25] is a tree with sets of variables as nodes, and with the property that any variable in two nodes is also in any node on the path between the two. The DAG $D$ of a Bayesian network is converted into a join tree via the moralization and triangulation procedures. Each maximal *clique* (complete subgraph) [10] of the triangulated graph is represented by a node in the join tree. Finding a minimum triangulation, that is, one where the largest clique in the resulting triangulated graph has minimum size, is NP-hard [14]. Given the collected evidence, messages are systematically passed in a join tree such that each join tree node can compute the posterior probabilities of its variables when propagation finishes. Shafer [25] gives an eloquent discussion on join tree propagation explaining both the inward pass and outward pass in the join tree. In the *Lazy* join tree propagation architecture [17], here called Lazy-VE, messages are computed using the *variable elimination* (VE) algorithm. When a join tree node $N$ is ready to send its messages to a particular neighbour $N'$, the Lazy-VE approach computes the messages from node $N$ to $N'$ using the following three steps: (i) collect all messages from $N$'s other neighbours; (ii) identify the relevant and irrelevant variables; (iii) apply VE to physically eliminate variables in $N - N'$ from the relevant distributions. Madsen's Lazy-AR [18,19] modifies Lazy-VE by applying instead AR to physically eliminate variables in $N - N'$ from the relevant distributions in step (iii) above. Note that the relevant and irrelevant variables in step (ii) can be identified using techniques discussed in [17,7].

## 3. Bayesian inference in DataBayes

In this section, we propose a more flexible join tree propagation algorithm for Bayesian network inference – one that selectively chooses the algorithm for building messages at each node. This algorithm has been implemented in our probabilistic reasoning system, called DataBayes.

While the CHD Bayesian network is useful for illustrating some pertinent concepts, it is not interesting due to its small size. A larger real-world Bayesian network, called *Hailfinder* [1], is used herein instead. Fig. 2 shows the partial depiction of one possible join tree for Hailfinder when evidence $j = 0$ is considered. Some join tree edges have been directed to depict the propagation of those messages pertinent to our forthcoming discussion. Each join tree node name corresponds to the variables in the node. For instance, in Fig. 2, node *abcf* means that the join tree node consists of variables $\{a, b, c, f\}$. The CPT of each remaining variable $v_i$ in the given Bayesian network is assigned to precisely one join tree node containing $v_i$ and its parents $P_i$. For instance, $p(f|a, b, c)$ is assigned to *abcf* in Fig. 2.

### 3.1. Preprocessing before propagation begins

The key difference between DataBayes and all previous join tree propagation algorithms is that DataBayes applies two preprocessing steps before manipulating the probability distributions stored in computer memory during inference.

With or without considering evidence, our first preprocessing step [9] identifies the messages to be passed in a join tree before the propagation begins. The messages we identify are precisely those that will be propagated, provided that AR is chosen as the algorithm for building messages. For example, in Fig. 2, node *ilnqr* will send to node *lmnqr* three messages $p(j = 0)$, $p(l|j = 0)$ and $p(r|j = 0, l, n, q)$. Similarly, two messages $p(j = 0)$ and $p(m|j = 0)$ will be sent from *lmnqr* to node *kmnq*.

Our second preprocessing step, called PickARorVE, is given as Algorithm 1. For each node $N$ in a join tree, and for each neighbour node $N'$ of $N$, Algorithm 1 determines whether DataBayes should apply AR or VE for those messages built at $N$ and passed to $N'$. If a node $N$ does not have to construct any messages for a neighbour $N'$, then $N$ is marked as NA with respect to $N'$.

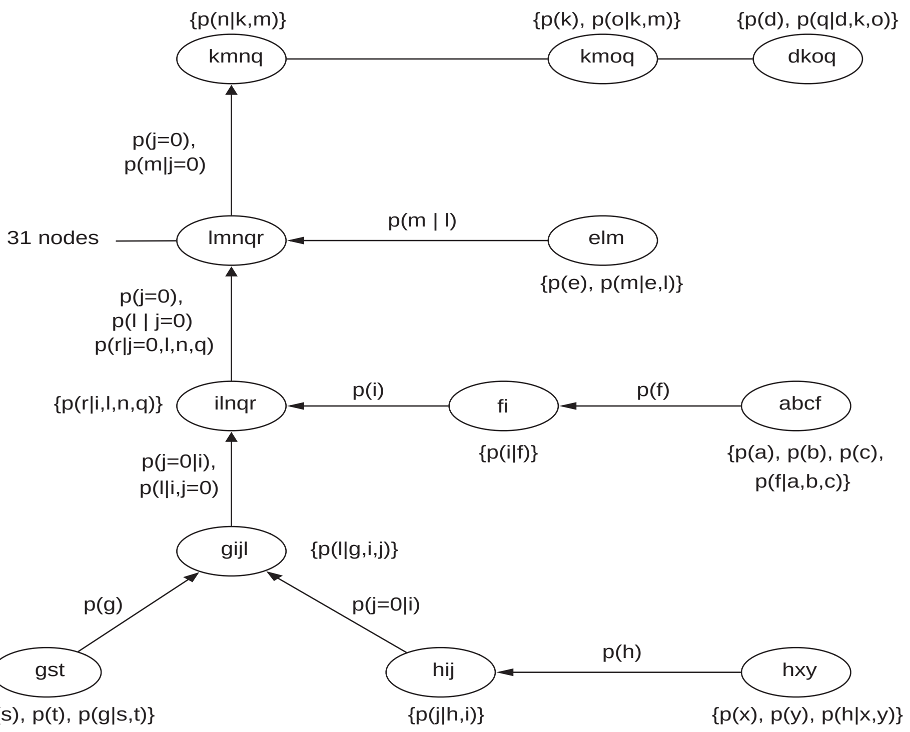

**Fig. 2.** The Hailfinder join tree, where only the pertinent nodes and messages are shown, when evidence $j = 0$ is considered.

```
Algorithm 1. PickARorVE ($J$)
Input: a join tree $J$ with messages to be propagated identified.
Output: the choice $\omega$ of AR or VE for messages built at $N$ and passed to $N'$.
begin
for each join tree node $N$
  for each neighbour node $N'$ of $N$
    Count the number $n$ of messages to be built at $N$ and passed to $N'$.
    if $n > 1$
      $\omega(N \rightarrow N') = \text{AR}$
    else if $n == 1$
      $\omega(N \rightarrow N') = \text{VE}$
    else
      $\omega(N \rightarrow N') = \text{NA}$
return($\omega$)
end
```

For example, let us apply the algorithm PickARorVE on the join tree with identified messages in Fig. 2. Node *ilnqr* is marked as AR as it will construct three messages for node *lmnqr*. On the other hand, node *abcf* is marked as VE, since it will construct just one distribution $p(f)$ for node *fi*. We will discuss a node marked as NA later.

There are two important points about selectively applying VE during join tree propagation. Firstly, it must be emphasized that VE is not simply applied when a single distribution is passed between neighbouring nodes. It is entirely possible that more than one distribution is passed from a node $N$ to a neighbour but only one of these distributions need to be constructed at $N$. In Fig. 2, node *lmnqr* is sending two distributions $p(j = 0)$ and $p(m|j = 0)$ to its neighbour *kmnq*, but $p(j = 0)$ does not need to be built at *lmnqr* as it is being forwarded. Only one message needs to be constructed at *lmnqr*, namely, $p(m|j = 0)$. Therefore, node *lmnqr* is labelled by PickARorVE as VE. Secondly, suppose a join tree node will construct a single distribution. Since VE and AR are sound, the same distribution is the output regardless of whether AR or VE is utilized. Therefore, once VE is applied at a join tree node, it does not prohibit the application of AR at a subsequent node later on during propagation. For example, applying VE at node *abcf* in Fig. 2 does not ruin the opportunity for applying AR at node *ilnqr*.

The important point is that the PickARorVE labelling indicates, which technique, if any, should be applied to eliminate variables at each join tree node. In Fig. 2, DataBayes will be applying, for instance, AR for message construction at node *ilnqr*, while applying VE at nodes *lmnqr* and *abcf*. Thus, within the same join tree, the message construction algorithm can vary from node to node.

## 3.2. Applying AR in DataBayes

*Arc reversal* (AR) [22,24] eliminates a variable $v_i$ by reversing the arcs $(v_i, v_j)$ for each child $v_j$ of $v_i$, where $j = 1, 2, \ldots, k$. With respect to multiplication and addition, AR reverses one arc $(v_i, v_j)$ as a three-step process:

$$p(v_i, v_j|A_j) = p(v_i|A_{j-1}) \cdot p(v_j|P_j), \tag{1}$$

$$p(v_j|B_j) = \sum_{v_i} p(v_i, v_j|A_j), \tag{2}$$

$$p(v_i|A_j) = \frac{p(v_i, v_j|A_j)}{p(v_j|B_j)}. \tag{3}$$

Suppose the variable $v_i$ to be removed has $k$ children. The distributions defined in Eqs. (1)–(3) are built for the first $k - 1$ children. For the last child, however, only the distributions in Eqs. (1) and (2) are built. When considering the last child, there is no need to build the final distribution for $v_i$ in Eq. (3), since $v_i$ will be removed as a barren variable. Therefore, AR removes a variable $v_i$ with $k$ children by building $3k - 1$ distributions. However, AR only outputs the $k$ distributions built in Eq. (2).

In Fig. 2, consider the construction of the three messages $p(j = 0)$, $p(l|j = 0)$ and $p(r|j = 0, l, n, q)$ sent from node *ilnqr* to node *lmnqr* in the Hailfinder join tree given evidence $j = 0$. Herein let us refer to the three steps used in Lazy-AR. By step (i), node *ilnqr* has collected messages $p(i)$, $p(j = 0|i)$ and $p(l|i, j = 0)$ from its other neighbours. In step (ii), all variables are relevant. For step (iii), variable $i$ needs to be eliminated from these three messages together with the CPT $p(r|i, l, n, q)$ assigned to *ilnqr*, namely,

$$\sum_{i} p(i) \cdot p(j = 0|i) \cdot p(l|i, j = 0) \cdot p(r|i, l, n, q). \tag{4}$$

Suppose AR reverses arc $(i, j)$, then $(i, l)$, and finally $(i, r)$. Arc $(i, j)$ is reversed as $(j, i)$ using

$$\begin{aligned} &p(i, j = 0) = p(i) \cdot p(j = 0|i), \\ &p(j = 0) = \sum_i p(i, j = 0), \\ &p(i|j = 0) = p(i, j = 0)/p(j = 0). \end{aligned} \tag{5}$$

To create $(l, i)$, AR builds

$$\begin{aligned} &p(i, l|j = 0) = p(i|j = 0) \cdot p(l|i, j = 0), \\ &p(l|j = 0) = \sum_i p(i, l|j = 0), \\ &p(i|j = 0, l) = p(i, l|j = 0)/p(l|j = 0). \end{aligned} \tag{6}$$

Lastly, reversing $(i, r)$ is accomplished by physically building

$$\begin{aligned} &p(i, r|j = 0, l, n, q) = p(i|j = 0, l) \cdot p(r|i, l, n, q), \\ &p(r|j = 0, l, n, q) = \sum_i p(i, r|j = 0, l, n, q). \end{aligned} \tag{7}$$

The distributions output by AR are those in Eqs. (5)–(7), namely,

$$p(j = 0) \cdot p(l|j = 0) \cdot p(r|j = 0, l, n, q). \tag{8}$$

Thus, by applying AR for message construction at node *ilnqr* in Fig. 2, DataBayes sends messages $p(j = 0)$, $p(l|j = 0)$ and $p(r|j = 0, l, n, q)$ to node *lmnqr*.

### 3.3. Applying VE in DataBayes

Given a set of variables to be eliminated from a set $S$ of potentials, *variable elimination* (VE) [25,28] recursively eliminates the variables $v_i$ one-by-one using the following four steps: (i) remove from $S$ the set of potentials containing $v_i$; (ii) multiply together the distributions removed from $S$; (iii) sum $v_i$ out of the potential obtained in (ii); and (iv) add the resulting potential to $S$. Note that VE outputs a single distribution – the one output from step (iii).

For example, VE is applied for message construction at node *lmnqr* in Fig. 2. The task is:

$$\sum_l \sum_r p(j = 0) \cdot p(l|j = 0) \cdot p(r|j = 0, l, n, q) \cdot p(m|l). \tag{9}$$

In Eq. (9), variable $r$ is barren, namely,

$$\sum_l p(j = 0) \cdot p(l|j = 0) \cdot p(m|l) \cdot \sum_r p(r|j = 0, l, n, q) \tag{10}$$

$$= \sum_l p(j = 0) \cdot p(l|j = 0) \cdot p(m|l) \cdot 1, \tag{11}$$

meaning that no physical manipulation of the distributions in memory is required to eliminate $r$ at node *lmnqr*. To eliminate variable $l$, by step (i) of VE we collect the relevant distributions as

$$p(j = 0) \cdot \sum_l p(l|j = 0) \cdot p(m|l). \tag{12}$$

By step (ii), we compute the product of the relevant distributions, namely,

$$p(j = 0) \cdot \sum_l p(l, m|j = 0). \tag{13}$$

We marginalize $l$ out in step (iii),

$$p(j = 0) \cdot p(m|j = 0). \tag{14}$$

This gives the remaining factorization in step (iv) of VE and corresponds precisely to the messages passed from node *lmnqr* to node *kmnq* in Fig. 2. As previously mentioned, observe that $p(j = 0)$ is simply received and forwarded by node *lmnqr*.

### 3.4. Message forwarding in DataBayes

When PickARorVE marks a join tree node $N$ as NA with respect to a neighbouring node $N'$, it means that $N$ does not have to construct any messages for $N'$. Instead, $N$ will simply be forwarding messages to $N'$.

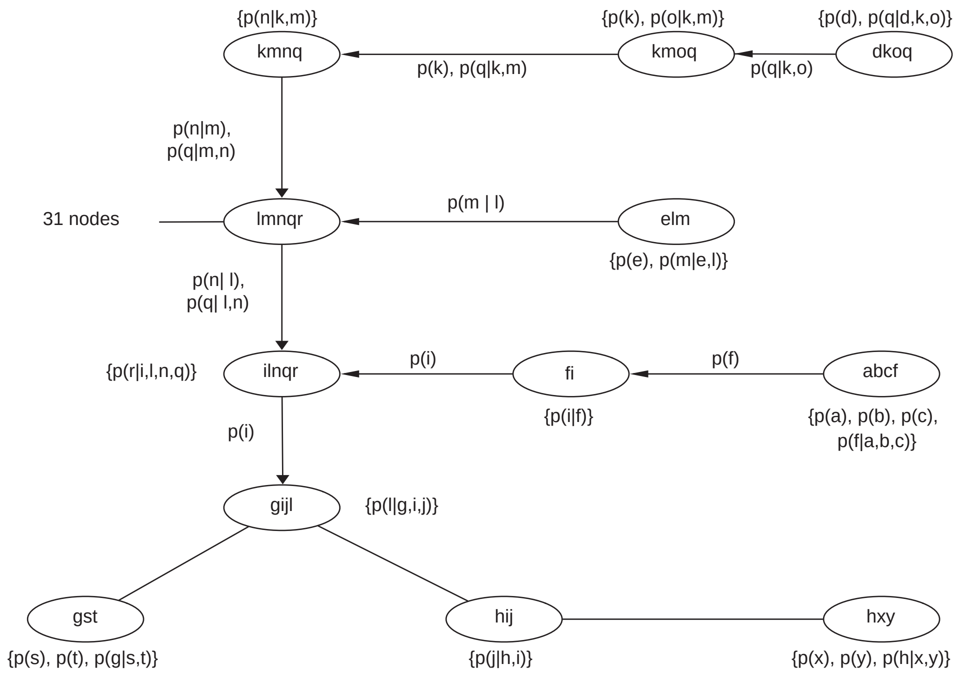

**Fig. 3**. Both messages $p(n|l)$ and $p(q|l, n)$ from *lmnqr* are *irrelevant* to the forwarding of message $p(i)$ at node *ilnqr*.

For example, in Fig. 3, consider node *ilnqr*. Here message $p(i)$ is to be passed from *ilnqr* to node *gijl*. In this case, *ilnqr* simply forwards to *gijl* the incoming message $p(i)$ from node *fi*. Thus, PickARorVE marks node *ilnqr* as *NA*, since *ilnqr* does not have to build a message to be sent to *gijl*.

It is worth repeating that our labelling process is always done with respect to the messages constructed at a node $N$ and passed to a neighbouring node $N'$. For instance, PickARorVE labels node *ilnqr* in Fig. 2 as AR with respect to its neighbour *lmnqr*, yet labels *ilnqr* in Fig. 3 as NA with respect to its neighbour *gijl*.

Using the processing of evidence in a real-world Bayesian network, in the next two sections we highlight the advantages of having a flexible message construction algorithm. Section 4 shows that AR is sometimes better than VE for constructing messages, while Section 5 shows the opposite.

## 4. Advantages of AR over VE

We first consider the message construction from node *ilnqr* to *lmnqr*, and then the subsequent message construction at the receiving node *lmnqr*, in the join tree of Fig. 2 when evidence $j = 0$ is considered.

### 4.1. Message construction at node ilnqr

In Fig. 2, message construction at node *ilnqr* means to compute Eq. (4). The question is whether it is better to apply VE or AR.

We have previously shown in Section 3.2 how AR eliminates variable $i$ yielding Eq. (8). If VE is applied at node *ilnqr*, on the other hand, VE would compute:

$$\begin{aligned}
&\sum_i p(i) \cdot p(j=0|i) \cdot p(l|i, j=0) \cdot p(r|i, l, n, q) \\
&\quad = \sum_i p(i, j=0) \cdot p(l|i, j=0) \cdot p(r|i, l, n, q) \\
&\quad = \sum_i p(i, j=0, l) \cdot p(r|i, l, n, q) \\
&\quad = \sum_i p(i, j=0, l, r|n, q) \\
&\quad = p(j=0, l, r|n, q). \qquad (15)
\end{aligned}$$

Therefore, in Fig. 2, if VE was applied for message construction at node *ilnqr*, then the single distribution $p(j = 0, l, r|n, q)$ in Eq. (15) would be passed to node *lmnqr*.

### 4.2. Subsequent message construction at node *lmnqr*

Now consider the subsequent message construction at the receiving node *lmnqr* in Fig. 2. The message passed from *elm* to *lmnqr* is $p(m|l)$ regardless of the choice of VE or AR at node *ilnqr*.

If AR was used at *ilnqr*, then the passed distributions are those in Eq. (8). Hence, message construction at *lmnqr* involves:

$$\sum_l \sum_r p(j=0) \cdot p(l|j=0) \cdot p(r|j=0,l,n,q) \cdot p(m|l). \tag{16}$$

If, however, VE was used at *ilnqr*, then the passed distribution is in Eq. (15). Hence, message construction at *lmnqr* requires:

$$\sum_l \sum_r p(j=0,l,r|n,q) \cdot p(m|l). \tag{17}$$

There is now a crucial difference between Eqs. (16) and (17). The important distinction is that $r$ is a barren variable in Eq. (16) but $r$ is not a barren variable in Eq. (17). As previously shown in Eqs. (9)–(11), by applying AR at node *ilnqr*, $r$ can be eliminated at node *lmnqr* without modifying the distributions in the computer memory. On the contrary, by applying VE at node *ilnqr*, $r$ must be eliminated at node *lmnqr* by modifying the distribution $p(j=0,l,r|n,q)$ in computer memory as:

$$\sum_l p(m|l) \cdot \sum_r p(j=0,l,r|n,q) \tag{18}$$

$$= \sum_l p(m|l) \cdot p(j=0,l|n,q). \tag{19}$$

Moreover, the product to be taken with $p(m|l)$ in Eq. (19) involves a larger distribution $p(j=0,l|n,q)$ than in Eq. (12), resulting in

$$\sum_l p(j=0,l,m|n,q). \tag{20}$$

And the marginalization of variable $l$ takes place over $p(j=0,l,m|n,q)$ instead of $p(l,m|j=0)$ in Eq. (13).

#### 4.2.1. Analysis of why AR is better than VE

The advantage of AR over VE is that AR maintains a valid DAG structure during evidence processing, which implies that more barren variables can be identified.

For example, by applying AR at node *ilnqr*, the messages passed to node *lmnqr* are the three distributions in Eq. (8). For the subsequent message computation at node *lmnqr*, $r$ was exploited as a barren variable as shown in Eqs. (9)–(11). The important point is that $r$ was eliminated at *lmnqr* without disturbing the probability distributions stored in the computer memory.

On the contrary, if VE were applied at *ilnqr*, node *lmnqr* would only receive a single distribution – the one in Eq. (15). Unfortunately for *lmnqr*, $r$ is no longer barren and requires physical manipulation of the probability distribution stored in computer memory as shown in Eq. (18).

More specifically, assuming binary variables, the following computation is required at the table-entry level. Applying AR at node *ilnqr* and VE at node *lmnqr* requires 38 multiplications, 19 additions and 6 divisions, and 4 multiplications and 2 additions, respectively. Therefore, the combined work at both nodes is 42 multiplications, 21 additions and 6 divisions. On the other hand, exclusively applying VE at *ilnqr* and *lmnqr* involves 38 multiplications and 16 additions, and 16 multiplications and 16 additions, respectively. Thus, the combined work for VE at both nodes is 54 multiplications and 32 additions. The above analysis reveals that selectively applying AR and VE can be better than exclusively applying VE.

It should be noted, however, that VE and AR will perform exactly the same computation if the variable being eliminated only appears in two distributions at the time that it is removed. In other words, applying AR to eliminate a variable with precisely one child involves the exact same computation as applying VE to eliminate the variable. For example, provided that AR is applied at *ilnqr* in Fig. 2, there is no computational difference between VE or AR at node *lmnqr*. Either approach will simplify Eq. (9) as Eq. (12), from which AR and VE perform the same operations.

## 5. Advantages of VE over AR

In this section, we examine possible negative consequences of choosing AR instead of VE for message construction.

### 5.1. Message construction at node *abcf*

Recall node *abcf* in Fig. 3. For pedagogical reasons, suppose the CPTs assigned to *abcf* are $p(a)$, $p(b|a)$, $p(c|b)$ and $p(f|a,b,c)$, and that the elimination ordering is $a$, followed by $b$, followed by $c$. Let us examine the choice of algorithm for eliminating these three variables.

If VE is used for message construction at node *abcf*, it will perform the following:

$$\begin{aligned}
&\sum_c \sum_b \sum_a p(a) \cdot p(b|a) \cdot p(c|b) \cdot p(f|a,b,c) \\
&= \sum_c \sum_b p(c|b) \cdot \sum_a p(a) \cdot p(b|a) \cdot p(f|a,b,c) \qquad (21)\\
&= \sum_c \sum_b p(c|b) \cdot \sum_a p(a,b) \cdot p(f|a,b,c) \qquad (22)\\
&= \sum_c \sum_b p(c|b) \cdot \sum_a p(a,b,f|c) \qquad (23)\\
&= \sum_c \sum_b p(c|b) \cdot p(b,f|c) \qquad (24)\\
&= \sum_c \sum_b p(b,c,f) \\
&= \sum_c p(c,f) \\
&= p(f).
\end{aligned}$$

If AR is used for message construction at node *abcf*, it will perform the following. To eliminate variable $a$, arcs $(a, b)$ and $(a, f)$ need to be reversed. Regarding the first child $b$ of $a$, AR computes:

$$p(a,b) = p(a) \cdot p(b|a), \qquad (25)$$

$$p(b) = \sum_a p(a,b), \qquad (26)$$

$$p(a|b) = p(a,b)/p(b). \qquad (27)$$

For the second child $f$ of $a$, AR performs:

$$p(a,f|b,c) = p(a|b) \cdot p(f|a,b,c), \qquad (28)$$

$$p(f|b,c) = \sum_a p(a,f|b,c). \qquad (29)$$

At this point variable $a$ is barren and can be removed. Next, AR removes the variable $b$, that has two children $c$ and $f$. Regarding child $c$ of $b$, AR computes:

$$\begin{aligned}
p(b,c) &= p(b) \cdot p(c|b),\\
p(c) &= \sum_b p(b,c),\\
p(b|c) &= p(b,c)/p(c).
\end{aligned}$$

AR performs the following operations for the second child $f$:

$$\begin{aligned}
p(b,f|c) &= p(b|c) \cdot p(f|b,c),\\
p(f|c) &= \sum_b p(b,f|c).
\end{aligned}$$

Variable $b$ has been made barren and can be removed. Finally, AR eliminates the last variable $c$, that has a single child $f$, as follows:

$$\begin{aligned}
p(c,f) &= p(c) \cdot p(f|c),\\
p(f) &= \sum_c p(c,f).
\end{aligned}$$

## 5.2. Analysis of why VE is better than AR

To the best of our knowledge, [5] is the first comprehensive comparison of VE and AR in probabilistic reasoning literature. The key findings are briefly reviewed here. We introduced the notion of *row-equivalent* to indicate that the rows (configurations) in one distribution are precisely the same as those in another distribution.

Within AR itself, let the variable to be eliminated $v_i$ have $k$ children, $v_1, v_2, \ldots, v_k$. We established that the rows appearing in the first constructed distribution are exactly the same as those rows appearing in the third constructed distribution, for each child (of the variable being eliminated) except the last. That is, the variable configurations appearing in first distribution built in Eq. (1) are exactly variable configurations appearing in the third distribution built in Eq. (3) for each child except the last.

Note that row-equivalence is not saying that the two distributions have the same number of rows. It is a stronger result stating that both distributions have the exact same rows. Within AR, the rows in $p(v_i, v_j|A_j)$ of the first equation in AR have been shown to be precisely the same as those in $p(v_i|A_j)$ of the third equation in AR. It was then shown that the CPT built for the last child $v_k$ is row-equivalent to the only distribution built by VE. This means that number of multiplications and number of additions needed by AR to build the new CPT of the last child $v_k$ are precisely the same as those needed by VE. However, AR necessarily requires more additions (in the second equation of AR), and more divisions (in the third equation of AR), for the children $j = 1, \ldots, k-1$. The consequence of the above results is that AR can build intermediate distributions that will not be passed as messages, nor are they required in the construction of the passed message [6].

For example, consider the message construction at node *abcf* in the last subsection. To eliminate variable $a$, VE directly computes Eqs. (21)–(24). On the contrary, AR eliminates variable $a$ using Eqs. (25)–(29). Since $p(a, b)$ in Eq. (25) has the same variable configurations as $p(a|b)$ in Eq. (27), the number of table-entry multiplications performed by AR to reach (29) is exactly the same number as those of table-entry multiplications performed by VE to reach Eq. (23). It follows that the number of additions at the table-level entry required by AR in Eq. (29) are those done by VE to yield Eq. (24). However, AR also performed two more table-level additions to build the intermediate distribution $p(b)$ in Eq. (26), and four table-entry level divisions to build the intermediate distribution $p(a|b)$ in Eq. (27). It can then be seen that AR performed more work than VE to eliminate variable $a$. Worse yet, AR built a new CPT for variable $b$ only to subsequently eliminate $b$. And when eliminating variable $b$, AR built a new CPT for variable $c$ only to subsequently eliminate $c$.

It can be verified that, at the table-entry level, VE eliminates variables $a$, $b$ and $c$, with 28 multiplications and 14 additions, while AR eliminates variables $a$, $b$ and $c$, with 36 multiplications, 18 additions and 8 divisions.

The analysis here shows that selectively applying AR and VE to build messages can be wiser than exclusively applying AR at every join tree node.

## 6. Experimental results

In this section, we report an empirical comparison of our DataBayes join tree propagation approach and Lazy-AR. Both methods were implemented in the C++ programming language. The experiments were conducted on a 2.80 GHz Intel Core 2 Duo P9700 with 8 GB RAM. The evaluation was carried out on four benchmark Bayesian networks taken from the 2006 UAI probabilistic inference competition [4]. The elimination ordering is determined using the min-fill criteria [14], while the ordering of the children of the variable being eliminated when using AR is determined by a fixed topological ordering of the variables in the Bayesian network [8]. Table 2 describes the Bayesian network name and number from the 2006 UAI competition, the number of variables in each Bayesian network, the number of evidence variables in each Bayesian network, the number of rows in the CPTs of the Bayesian network, the number of nodes in each join tree, and the number of messages passed in the join tree when no evidence is involved.

Our experiments not involving evidence are conducted as follows. Load the Bayesian network ignoring the given evidence variables and build a join tree. The inward and outward phases of join tree propagation are performed to compute a factorization of $p(N)$ for every join tree node $N$. For experiments involving collected evidence, the evidence $E = e$ is stated in the description of the Bayesian network and was determined by the competition organizers. In this case, the DAG of the Bayesian network is pruned based on the given evidence. Next, a join tree is constructed from the pruned DAG. Finally, inward and outward phases of join tree propagation are performed to compute a factorization of $p(N - E, E = e)$ for every join tree node $N$.

Table 3 reports on Bayesian inference not involving evidence processing. Running times for the PickARorVE algorithm are negligible [8] and are not shown separately, although they are included in the running times for DataBayes. Running times for Lazy-AR and for DataBayes are listed in milliseconds and are the average of three runs. The last column shows the speed-up percentage of DataBayes over Lazy-AR. The average percentage gain is 46%.

Next, we measure the runtime of inference involving evidence. Table 2 indicates the number of evidence variables as specified in the 2006 UAI probabilistic inference competition. The times reported in Table 4 are given in milliseconds and are the average of three runs. Note that, once again, DataBayes is always faster than Lazy-AR. The average percentage gain is 30%.

The results in Tables 3 and 4 empirically demonstrate that by selectively applying VE and AR, join tree propagation can be performed faster.

**Table 2**
Description of four benchmark Bayesian networks and the constructed join trees (JTs).

| Bayesian network | # variables | # evidence variables | # CPT rows | # Join tree nodes | # Join tree messages |
|---|---|---|---|---|---|
| Alarm (BN0) | 100 | 26 | 974 | 68 | 798 |
| Water (BN28) | 24 | 8 | 18,029,184 | 12 | 27 |
| ISCAS 85 (BN43) | 880 | 10 | 5096 | 326 | 1742 |
| CPCS (BN78) | 54 | 10 | 1658 | 30 | 193 |

**Table 3**
The performance of Lazy-AR and DataBayes not involving evidence processing in four benchmark Bayesian networks.

| Bayesian network | Lazy-AR inward | Lazy-AR outward | DataBayes inward | DataBayes outward | Net inward (%) | Net outward (%) | Net total (%) |
|---|---|---|---|---|---|---|---|
| Alarm | 1866 | 24,602 | 1696 | 19,610 | 9 | 20 | 20 |
| Water | 355 | 1653 | 15 | 111 | 96 | 93 | 94 |
| ISCAS 85 | 5845 | 24,420 | 5380 | 13,212 | 8 | 46 | 39 |
| CPCS | 1129 | 1976 | 1039 | 1133 | 8 | 43 | 30 |

**Table 4**
The performance of Lazy-AR and DataBayes involving evidence processing in four benchmark Bayesian networks.

| Bayesian network | Lazy-AR inward | Lazy-AR outward | DataBayes inward | DataBayes outward | Net inward (%) | Net outward (%) | Net total (%) |
|---|---|---|---|---|---|---|---|
| Alarm | 2387 | 12,694 | 1310 | 9392 | 45 | 26 | 29 |
| Water | 2131 | 7857 | 1766 | 2598 | 17 | 67 | 56 |
| ISCAS 85 | 3700 | 13,494 | 3419 | 13,103 | 8 | 3 | 4 |
| CPCS | 1372 | 3034 | 1213 | 1918 | 12 | 37 | 29 |

## 7. Related works

Here we discuss two related works, one focusing on how distributions are built and the other on when distributions are built.

### 7.1. Message construction with AR and posteriors with VE

Very recently, Madsen [20] has also examined the utilization of applying both AR and VE during join tree propagation. His suggestion, which he writes was inspired by Butz and Hua [6], is as follows:

(i) exclusively apply AR for message construction, and
(ii) exclusively apply VE for computing posterior probabilitites.

The output of step (i) is a factorization of $p(N - E, E = e)$ for each join tree node $N$. Step (ii) computes $p(v|E = e)$ for each variable $v \in U$.

The experimental results in [20] show that there is essentially no gain in this kind of hybrid usage of AR and VE. While VE is the obvious choice for step (ii), there is no time savings overall as the time required for step (i) dominates that for step (ii).

Our experimental results show that step (i), the dominating step in join tree propagation, can be performed faster by selectively choosing AR or VE.

### 7.2. Prioritized join tree propagation

As opposed to focusing on *how* join tree messages are built, we recently suggested the concept of prioritized join tree propagation [7], which focuses on *when* individual join tree messages can be built. The motivation for [7] was based on the observation that during inference in real-world Bayesian networks, it is often the case that only some of the messages passed to a join tree node are actually needed in the physical construction of the subsequent probability distributions (messages) sent out from the node. For example, consider the message $p(i)$ to be passed from node *ilnqr* to node *gijl* in Fig. 3. Clearly, *ilnqr* simply forwards to *gijl* the incoming message $p(i)$ from node *fi*. Therefore, all of the physical computation done at node *lmnqr* to construct messages $p(n|l)$ and $p(q|l, n)$ is irrelevant to the subsequent message construction at *ilnqr*. This shows that Lazy-AR and Lazy-VE will force node *ilnqr* to wait for node *lmnqr* to build messages $p(n|l)$ and $p(q|l, n)$, even though these messages are irrelevant to the forwarding of the subsequent message $p(i)$ from *ilnqr*. As a more complicated example, in Fig. 2, only some of the messages sent from *ilnqr* and *elm* are relevant to subsequent message computation at *lmnqr*. That is, $p(j = 0)$ is the only relevant message regarding the forwarding of $p(j = 0)$ at node *lmnqr*. Moreover, as shown in Eq. (12), only distributions $p(l|j = 0)$ and $p(m|l)$ are relevant to the message construction of $p(m|j = 0)$ at node *lmnqr*. However, Lazy-AR forces *lmnqr* to wait for all four messages to be received. These two examples motivate the development of a new approach to Bayesian inference – one that conducts inference in a join tree at a "message-to-message" level rather than at a "node-to-node" level. In experimental results using four real-world Bayesian networks and one benchmark Bayesian network, and with varying amounts of evidence, prioritized join tree propagation finished faster than Lazy-AR without exception.

## 8. Conclusions

Current join tree propagation algorithms, such as Lazy-AR [18] and Lazy-VE [17], utilize a single inference technique for constructing the messages throughout the entire join tree of a given Bayesian network. In this paper, we have proposed a new join tree propagation approach to probabilistic inference in Bayesian networks. In our probabilistic expert system, called *DataBayes*, we selectively apply either VE or AR to build the messages at each node in the JT. As was demonstrated, the motivation for our approach is that, during inference in real-world Bayesian networks, it is often the case that one of VE and AR is best suited to construct the messages at a particular node. In these cases, selecting the most appropriate algorithm will provide a better performance than exclusively applying a single method. Therefore, DataBayes is an improvement over Lazy-VE and Lazy-AR. Our empirical comparison of DataBayes and Lazy-AR was conducted on four benchmark Bayesian networks taken from the 2006 UAI probabilistic inference competition [4]. Our experiments involved no evidence as well as cases where evidence was specified by the competition organizers. As reported in Tables 3 and 4, selectively choosing AR or VE as the message construction algorithm at each join tree node allowed DataBayes to exhibit a reasonable improvement over Lazy-AR.

Deriving a more precise measure of determining when to use AR or VE for message construction, rather than simply counting the number of distributions to be constructed, remains as future work.

## Acknowledgments

This research is supported by the NSERC Discovery Grant 238880. The authors would like to thank anonymous reviewers for their contribution in making a more clear and focused paper.

## References

[1] B. Abramson, J. Brown, W. Edwards, A. Murphy, R.L. Winkler, Hailfinder: a Bayesian system for forecasting severe weather, International Journal of Forecasting 12 (1996) 57–71.
[2] S. Arnborg, D. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM Journal on Algebraic and Discrete Methods 8 (2) (1987) 277–284.
[3] M. Baker, T.E. Boult, Pruning Bayesian networks for efficient computation, in: Proceedings of the UAI, 1990, pp. 225–232.
[4] J. Blimes, R. Dechter, A report from the 1st Evaluation of Probabilistic Inference, presented at UAI, 2006.
[5] C.J. Butz, J. Chen, K. Konkel, P. Lingras, A formal comparison of variable elimination and arc reversal in Bayesian network inference, Intelligent Decision Technologies 3 (3) (2009) 173–180.
[6] C.J. Butz, S. Hua, An improved Lazy-AR approach to Bayesian network inference, in: Proceedings of the Canadian Conference on AI, 2006, pp. 183–194.
[7] C.J. Butz, S. Hua, K. Konkel, H. Yao, Join tree propagation with prioritized messages, Networks 55 (4) (2010) 350–359.
[8] C.J. Butz, K. Konkel, P. Lingras, Join tree propagation utilizing both arc reversal and variable elimination, in: Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference, 2009, pp. 523–528.
[9] C.J. Butz, H. Yao, S. Hua, A join tree probability propagation architecture for semantic modelling, Journal of Intelligent Information Systems 33 (2) (2009) 145–178.
[10] E. Castillo, J. Gutiérrez, A. Hadi, Expert Systems and Probabilistic Network Models, Springer, 1997.
[11] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (2–3) (1990) 393–405.
[12] D. Geiger, T. Verma, J. Pearl, Identifying independence in Bayesian networks, Networks 20 (1990) 507–534.
[13] P. Hájek, T. Havránek, R. Jiroušek, Uncertain Information Processing in Expert Systems, CRC Press, 1992.
[14] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[15] U.B. Kjaerulff, A.L. Madsen, Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis, Springer, 2008.
[16] Z. Li, B. D'Ambrosio, Efficient inference in Bayes networks as a combinatorial optimization problem, International Journal of Approximate Reasoning 11 (1) (1994) 55–81.
[17] A.L. Madsen, F.V. Jensen, Lazy propagation: a junction tree inference algorithm based on lazy evaluation, Artificial Intelligence 113 (1–2) (1999) 203–245.
[18] A.L. Madsen, An empirical evaluation of possible variations of lazy propagation, in: Proceedings of the UAI, 2004, pp. 366–373.
[19] A.L. Madsen, Variations over the message computation algorithm of lazy propagation, IEEE Transactions on Systems, Man, and Cybernetics, B 36 (2006) 636–648.
[20] A.L. Madsen, Improvements to message computation in lazy propagation, International Journal of Approximate Reasoning 51 (5) (2010) 499–514.
[21] R.E. Neapolitan, Probabilistic Reasoning in Expert Systems, John Wiley & Sons, 1990.
[22] S. Olmsted, On representing and solving decision problems, Ph.D. Thesis, Department of Engineering Economic Systems, Stanford University, Stanford, California, 1983.
[23] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[24] R. Shachter, Evaluating influence diagrams, Operations Research 34 (1986) 871–882.
[25] G. Shafer, Probabilistic Expert Systems, SIAM, 1996.
[26] S.K.M. Wong, C.J. Butz, D. Wu, On the implication problem for probabilistic conditional independency, IEEE Transactions on Systems, Man, and Cybernetics, A 30 (6) (2000) 785–805.
[27] Y. Xiang, Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach, Cambridge University Press, 2002.
[28] N.L. Zhang, D. Poole, A simple approach to Bayesian network computations, in: Proceedings of the Canadian Conference on AI, 1994, pp. 171–178.