Computational-workload based binarization and partition of qualitative Markov trees for belief combination

Weiru Liu, Xin Hong, and Kenny Adamson
School of Computing and Mathematics, University of Ulster at Jordanstown, UK
w.liu@ulster.ac.uk

Abstract. Binary join trees have been a popular structure to compute the impact of multiple belief functions initially assigned to nodes of trees or networks. Shenoy has proposed two alternative methods to transform a qualitative Markov tree into a binary tree. In this paper, we present an alternative algorithm of transforming a qualitative Markov tree into a binary tree based on the computational workload in nodes for an exact implementation of evidence combination. A binary tree is then partitioned into clusters with each cluster being assigned to a processor in a parallel environment. These three types of binary trees are examined to reveal the structural and computational differences.

1 Introduction

Expensive computational cost of Dempster's rule in DS theory led to a stream of study on efficient implementations of the rule over the last two decades. The proposed approaches have emphasized either exact implementations of the rule (e.g., [1], [6], [10], [11], [13], [17], etc.) or its approximations (e.g., [2], [3], [4], [15], [16], etc.), under the assumption that evidence distributions follow certain structures.

Among all these approaches, the method on belief propagation in qualitative Markov trees has been popular. As proved in [11], with this method, the exponential computational complexity in the size of total variables is reduced to the size of the largest node in a tree, a node with the largest number of variables. The major technique supporting the method is *local computation* [14], which was initiated for propagating probabilities in Bayesian causal trees by Pearl [8]. Local computation refers to computation that involves only a small number of nodes in a large tree (or network). The basic idea of local computation is message passing among neighboring nodes in a qualitative Markov tree to compute marginals of the joint belief distribution without actually calculating the joint belief distribution. The sizes of nodes in a qualitative Markov tree determine how efficient the local computation can be. If a node has a large number of neighbors, even local computation can be very inefficient. To solve this problem, concept *binary joint trees* (or simply *binary trees*) was proposed by Shenoy in [12], in which every non-leaf node performs *at most one combination*, and the corresponding algorithm was introduced. Using this algorithm, any qualitative Markov tree can be first transformed into a binary join tree on which local computation can be carried out. Subsequently, Shenoy improved this transformation algorithm in [13]. The improved algorithm constructs a binary tree based on a *one-step lookahead* technique which looks for an optimal order of variables being deleted. Given a qualitative Markov tree, these two algorithms construct rather different join trees, with the second tree bearing little resemblance to the original Markov tree. Furthermore, the *one-step lookahead* algorithm adds many more nodes in the transformation procedure than the first algorithm. These extra nodes will increase the computational costs in terms of calculating marginals for them.

In this paper, we propose a different method to transform a qualitative Markov tree into a binary tree, based on the amount of combinations at each sub-tree. A bi-

nary tree derived in this way is almost balanced in respect to the workload of combining evidence. We then partition a binary tree into a set of clusters with the intention that each cluster will be assigned to a processor in a parallel processing environment. The appearance of our binary tree is similar to the tree obtained from Shenoy's first approach, except that Shenoy's tree may have more added nodes because this tree permits only one combination in each node. However, our study on these two similar types of binary trees shows that Shenoy's tree requires less amount of computation than ours since it eliminates some duplicate combinations. Our study on Shenoy's second approach reveals that a binary tree obtained in this way (it permits one combination per node as well) is very complex and adds much extra computation due to a large number of added nodes, comparing to his first approach.

Other algorithms that binarizing a qualitative Markov tree into a binary tree include a straight-forward binarization procedure for approximate computation in Bayesian networks [2] and a computational workload based algorithm for executing a parallel program using multiple processors [7]. Our algorithm is similar to the procedure in [2] in respect to the structure of the tree, that is, each node has maximum two children. However, our algorithm is more comprehensive because it assesses the amount of computation at each sub-tree before merging two sub-trees together. As a result, our binary tree is a balanced one while a tree from [2] can be extremely unbalanced (see the example in Section 3). If there are many processors available to process some nodes (sub-trees) in parallel [5], a balanced tree provides a good structure to partition it into clusters so as to assign workloads to processors evenly. Our algorithm is also similar to the binarization procedure in [7] in the sense that the latter considers workloads on sub-trees as well when merging two sub-trees (units of a parallel program). The difference between them is that our algorithm needs to consider the amount of computation being carried out in added nodes (which may affect the total workload of a sub-tree with this added node as the root). While the algorithm in [7] does not involve computation in added nodes (only message passing).

The rest of the paper is organized as follows. Section 2 introduces the basics of DS theory and the terminology for belief propagation in qualitative Markov trees. Section 3 provides the algorithm that transforms a qualitative Markov tree into a binary join tree and partitions a binary tree into clusters for parallel processing. Section 4 reviews the two algorithms on binarization proposed by Shenoy, with examples. Section 5 provides a detailed analysis and comparison of our algorithm with that of Shenoy's and concludes the paper.

2 DS theory and qualitative Markov trees

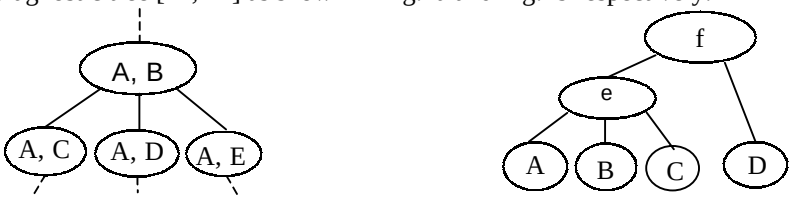
2.1 Basics of DS Theory

In the Dempster-Shafer theory of evidence (DS theory) [9], a piece of information is described as a *mass function* on a set of mutually exclusive and exhaustive elements, known as a *frame of discernment* (or simply a *frame*), denoted as Θ. A mass function m: $2^{\Theta} \rightarrow [0,1]$, represents the distribution of a unit of belief over a frame, Θ, satisfying the following two conditions: $m(\Phi) = 0$ and $\Sigma_{A \subseteq \Theta} m(A)=1$. A *belief function* over Θ is a function Bel: $2^{\Theta} \rightarrow [0,1]$, satisfying $Bel(A)= \Sigma_{B \subseteq A} m(B)$. When several belief functions are obtained through distinct sources based on the same frame of discernment, a

new belief function representing the consensus of them can be produced. Assume that Bel_1 and Bel_2 are two such obtained belief functions on the same frame Θ, the combined impact of them is calculated using the *Dempster's rule of combination*, $Bel = Bel_1 \oplus Bel_2$. The computational complexity of combining two belief functions over a frame is exponential to the size of the initial frame.

2.2 Qualitative Markov trees

Qualitative Markov trees: We use graph-oriented terminology and notation for qualitative Markov trees here. Let a pair $\{V, E\}$ be a graph, with V a finite set of nodes (or variables) and E a set of unordered pairs of distinct nodes in V. A qualitative Markov tree is a graph which has no cycles, and any variable in two nodes should be in any node in the path linking them. Elements in V are denoted using capital letters, such as A, B, S, and subsets of V are denoted with lower cases, such as, x, y, z. A qualitative Markov tree can either be derived from a Bayesian network [2, 13] or from a diagnostic tree [11, 14] as shown in Fig.1a and Fig.1b respectively.



(a) (b)

Figure 1. Two examples of qualitative Markov trees.

When a qualitative Markov tree is constructed from a diagnostic tree, the collection of all leaf nodes defines the overall frame of discernment represented by the root. Any non-leaf node, such as e, contains all the leaf nodes in the sub-tree with this node as the root. The corresponding frame for belief combination is $\{e, \neg e\}=\{A, B, C, \neg e\}$. While the frame for a node in Fig.1a is the Cartesian product of its variable frames. In the rest of the paper, we use a qualitative Markov tree in the form of Fig.1b. Our algorithm and discussions are equally applicable to a tree in the form of Fig.1a.

Variables and Configurations: Let x be a node in a qualitative Markov tree representing a set of *variables* and Θ_x be the frame corresponding to x. Elements of Θ_x are referred to as *configurations* of x, denoted by bold-faced lower cases, such as, $\boldsymbol{g}, \boldsymbol{f}, \boldsymbol{h}$.

Projection and Extension: Let g and h be two sets of variables, $h \subseteq g$, and $\boldsymbol{g}$ is a configuration of g. The *projection* of $\boldsymbol{g}$ to Θ_h, denoted by $\boldsymbol{g}^{\downarrow h}$ is a configuration of h. Let G be a non-empty subset of Θ_g, the projection of G to h, denoted by $G^{\downarrow h}$, is obtained by $G^{\downarrow h} = \{\boldsymbol{g}^{\downarrow h} | \boldsymbol{g} \in G\}$. If g and h are two sets of variables, $h \subset g$, and H is a subset of Θ_h, then the *extension* of H to g, denoted by $H^{\uparrow g}$, is $H \times \Theta_{g-h}$.

Marginalization: If m is a mass function on g, and $h \subseteq g$, $h \neq \Phi$, the marginal of m on h, denoted by $m^{\downarrow h}$, is a mass function on h defined by

$$m^{\downarrow h}(H) = \sum_{H \subseteq \Theta_h} \{m(G) | G \subseteq \Theta_g, G^{\downarrow h} = H\}.$$

On the other hand, if m is a mass function on h, and $h \subseteq g$, $h \neq \Phi$, the marginal of m on g, denoted by $m^{\uparrow g}$, is a mass function on g defined by

$$m^{\uparrow g}(G) = \sum_{G \subseteq \Theta_g} \{m(H) \mid H \subseteq \Theta_h, H^{\uparrow g} = G\}.$$

Belief propagation: Let $\{V, E\}$ be a qualitative Markov tree on which a set of belief functions are assigned to its nodes. Given a node x, $V_x = \{i \mid (i,x) \in E\}$ denotes the set of neighbours of x, a set of nodes that are directly linked with x. Bel_x represents the initial belief function assigned to node x. To propagate initial belief functions to obtain the final marginal on a designated node (containing a set of variables), the propagation scheme starts with the leaves of a qualitative Markov tree and moves step by step towards the targeted node. Each time a node x sends a message $M^{x \to i}$, referring to the belief function sent by x to i, to each of its neighbors,

$$M^{x \to i} = ((Bel_x \oplus (\oplus\{M^{k \to x} \mid k \in (V_x - \{i\})\}))^{\downarrow (x \cap i)})^{\uparrow i}. \quad (1)$$

For a leaf node x with only one neighbor i, $M^{x \to i}$ is reduced to $M^{x \to i} = ((Bel_x)^{\downarrow (x \cap i)})^{\uparrow i}$. After the designated node y has received the messages from all of its neighbors, the marginal $Bel^{\downarrow y}$ for y is obtained as $Bel^{\downarrow y} = Bel_y \oplus (\oplus\{M^{i \to y} \mid i \in V_y\})$. As stated in [13], a qualitative Markov tree can always be re-constructed as a rooted one. In this paper, we concentrate on rooted qualitative Markov trees. Let node r be the root of a Markov tree, x be a node. Let $Ch_x = \{k \mid k \in V_x$, k is a child node of $x\}$ be the set of children of x, and $P_x = \{p\}$ be the parent of x. The belief propagation scheme can be carried out in two phases to calculate the combined beliefs on any node [11]:

Phase I. Propagate messages up the tree: starting at leaf nodes, messages are sent up step by step. $M^{x \to p} = ((Bel_x \oplus (\oplus\{M^{k \to x} \mid k \in Ch_x\}))^{\downarrow (x \cap p)})^{\uparrow p}$.

The maximum number of belief functions accumulated in a non-leaf, non-root node in this phase is $1 + |Ch_x|$, if this node has an initial belief function and every of its child node sends a message to it. Therefore the number of combinations is $(1 + |Ch_x|) - 1$, *i.e.* $|Ch_x|$. For a leaf node, no combinations are involved. For the root node, there are maximum $1 + |Ch_r|$ belief functions accumulated. Since the root will not send any messages up, Phase I stops here. After computing the marginal of the root, messages are then sent back down the tree. Therefore, we will count the total number of combinations in the root in the next phase.

Phase II. Propagate messages down the tree: starting at the root node, messages are sent back down step by step.

$$M^{x \to k} = ((Bel_x \oplus M^{p \to x} \oplus (\oplus\{M^{j \to x} \mid j \in Ch_x, j \neq k\}))^{\downarrow (x \cap k)})^{\uparrow k}. \quad (2)$$

For a non-root, non-leaf node x, the maximum number of belief functions accumulated for propagating down to its child node k is $1 + |P_x| + (|Ch_x| - 1)$, so the number of combinations is $|Ch_x|$ (with $|P_x| = 1$), if we have stored every $M^{j \to x}$ in Phase I. The maximum total number of combinations in x is $|Ch_x| \times |Ch_x|$. Its final marginal is

$$Bel^{\downarrow x} = Bel_x \oplus M^{p \to x} \oplus (\oplus\{M^{i \to x} \mid i \in Ch_x\}). \quad (3)$$

If the marginal of the joint for x from Equation (2) is reserved before it is projected to node k, then it can be incorporated into Equation (3) to replace all the messages

except $M^{k\to x}$. Equation (3) can be rewritten as $Bel^{\downarrow x}=(M^{x\to k})^{\uparrow x}\oplus M^{k\to x}$. Therefore, there is only one extra combination to obtain the final marginal for a node.

Because a root has maximum $1+|Ch_r|$ belief functions, the maximum number of combinations for propagating a message down a branch is $|Ch_r|$-1 (the message from a branch to which the message is being sent will not be combined with the rest). The maximum total number of combinations is $(|Ch_r|-1)\times|Ch_r|$. The root needs one combination for its final marginal. A leaf node also needs one combination for its final marginal. The total number of combinations in a qualitative Markov tree is the sum of numbers of combinations of all the nodes.

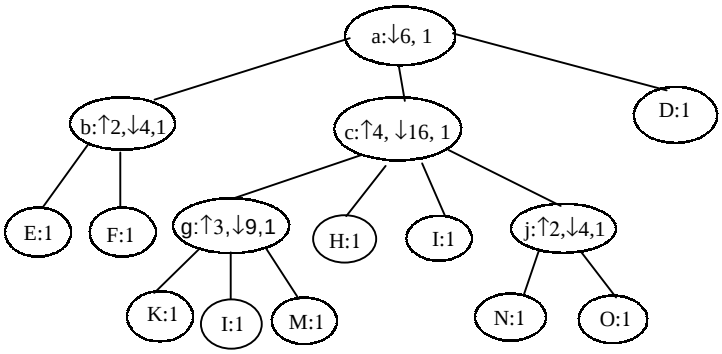


Figure 2. A rooted qualitative Markov tree with maximum number of combinations in each node when an initial belief function is assigned to each node and a final marginal is required for every node.

(x:↑t_1,↓t_2, *1*) indicates that in node x, there are t_1 combinations when x sends a message to its parent, there are t_2 combinations when it sends messages to all of its children, and there is one extra combination to obtain the final marginal for x. When t_1 or t_2 is zero, we have omitted it from the above graph.

3 A Weight-based Binarization Algorithm

When binarizing a qualitative Markov tree, for each non-leaf node x with more than two children, we repeatedly merge two of its children to get a new one with these two children carrying the least amount of computation, until x has only two children left. Such a binary tree should have almost balanced workloads among its branches.

Although a new affiliated node is added whenever two branches are merged, these newly created nodes will only calculate and store some intermediate results of combinations and no computation is required to calculate their own marginals. In the algorithm below, *comb*(x) represents the total number of combinations in node x, and *comb*(T_x) is the total number of combinations in sub-tree T_x with x as the root.

Algorithm: Binarization of a **Q**ualitative **M**arkov **T**ree (**BQMT**)
Input: a qualitative Markov tree with a designated root r.
$x\leftarrow r$.
Procedure Binarization (x):
1. If x is a leaf node and $x=r$ Then *comb*(x) $\leftarrow 0$, *comb*(T_x) $\leftarrow 0$. Terminate the Procedure. (The tree has only one level, the root is also a leaf.)

2. If x is a leaf node and $x \neq r$ Then $comb(x) \leftarrow 1$, $comb(T_x) \leftarrow 1$. Terminate the Procedure. (The tree has more than one level.)
3. For each child node $c_i \in Ch_x$ do **Binarization** (c_i).
4. Sort Ch_x in ascending order, where $Ch_x=\{c_1, \ldots, c_k\}$ satisfying
   $comb(T_{ci}) \leq comb(T_{cj})$ if c_j is after c_i in the ordered set Ch_x.
5. $l \leftarrow 1$.
6. While $|Ch_x| > 2$ do
   6.1 Select c_1, c_2, the first two elements in Ch_x;
   6.2 Create a new node x_l ($x_l=(c_1 \cup c_2) \cap x$ *when we use a tree in the form of Figure 1a;* $x_l=(c_1 \cup c_2)$ *when a tree is of the form Figure 1b*) to connect c_1 and c_2, replace sub-trees T_{c1} and T_{c2} with the new sub-tree T_{xl} with x_l as the root;
   6.3. $comb(x_l) \leftarrow 3$, $comb(T_{xl}) \leftarrow comb(x_l) + comb(T_{c1}) + comb(T_{c2})$;
   6.4. Remove c_1 and c_2 from Ch_x, insert x_l into Ch_x in sorted order;
   6.5. $l=l+1$.
7. If $|Ch_x| = 1$ Then
   7.1 If x is the root Then $comb(x) \leftarrow 1$ Else $comb(x) \leftarrow 3$;
   7.2 $comb(T_x) \leftarrow comb(x) + comb(T_{c1})$.
   Else
   7.3 If x is the root Then $comb(x) \leftarrow 3$ Else $comb(x) \leftarrow 7$;
   7.4 $comb(T_x) \leftarrow comb(x) + comb(T_{c1}) + comb(T_{c2})$.

Return (T_r): A binary tree with the same root

For each newly added node, the maximum number of belief functions accumulated in it is $|Ch_{xl}|$ (it has no initial belief function) instead of $1+|Ch_{xl}|$, so, the maximum number of combinations is $(\uparrow 1, \downarrow 2, 0)=3$. Applying this algorithm to the tree in Fig.2, we get a balanced binarised tree as in Fig.3 where bold-faced nodes are added nodes.

However, if we do not merge the two branches with the lightest workloads each time during the binarization procedure, we could end up with a tree that is totally unbalanced. For example, an alternative binary tree from the qualitative Markov tree in Fig.2 can be in the form as shown in Fig.4. This unbalanced tree will make the parallel processing much less efficient if we were to use multiple processors to process each part simultaneously [5]. Given a multiple processor environment, it is in general not possible to assign each node to a processor, due to the fact that either there are less processors available than the total number of nodes or the communication cost between processors are too expensive comparing to the calculation.

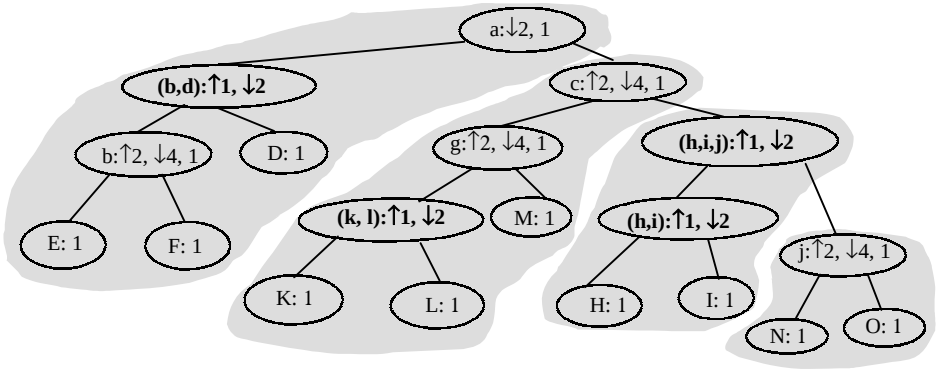


Figure 3. A Binary tree constructed from a qualitative Markov tree in Figure 2.

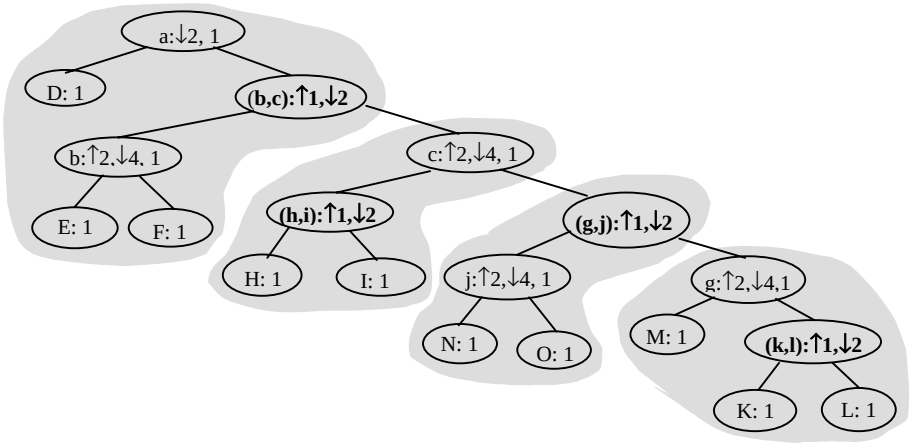


Figure 4. An unbalanced binary tree from the qualitative Markow tree in Figure 2.

Below is a clustering algorithm that partitions a binary tree into clusters and assigns each cluster to a processor. In [5], we have been testing the algorithm in a four-processor environment where a tree in Fig.3 is partitioned into four clusters as illustrated with shade. These four processors perform combinations simultaneously starting from leaves. Each sends its results to the processor that contains its parent for further combination before it calculates final marginals for its nodes. This algorithm also partitions the tree in Fig.4 into four clusters as shown. However, the processor that contains node (b,c) has to wait for the result of other three processors which deal with clusters from its right branch before it can go further. Therefore, the intended parallel process is reduced to almost a linear one, in addition to the extra cost of communication between processors.

Algorithm: Clustering a binary tree

Input: r – the binary tree with the root r, N – the number of processors provided

1. Create two empty queues S and S_t (S is the working queue, S_t is the temporary queue);
2. $S \leftarrow \{r\}$, counter $m \leftarrow 1$;
3. While $m < N$ and queue S is not empty, do
   - 3.1 Select the first element v in S and let $S \leftarrow S/\{v\}$;
   - 3.2 If v has no children, Then $S_t \leftarrow S_t \cup \{v\}$; $m=m+1$;
     Else
     - 3.2.1 If v has one child, Then
       $p \leftarrow v$
       While p has one child, do $p \leftarrow$ the child of p
       Let C_L and C_R be the children of node p;
       If $|comb(Tc_L)\text{-}comb(Tc_R)| < \delta$ (δ is a threshold saying that both branches have almost the same workload)
       Then
         Disconnect Tc_R from Tp;
         $comb(Tp) = comb(Tp)\text{-}\ comb(Tc_R), comb(Tv) = comb(Tv)\text{-}\ comb(Tc_R)$;
         $S \leftarrow S \cup \{v, C_R\}$, $m=m+1$;
       Else
         Let w be the root of p's bigger child subtree;
         Disconnect Tw from $Tp, comb(Tv) = comb(Tv)\text{-}\ comb(Tw)$;

comb(*Tp*) = *comb*(*Tp*)- *comb*(*Tw*);
While |*comb*(*Tv*) - *comb*(*Tw*)| > δ, do
Reconnect *Tw* to *Tp*;
comb(*Tp*) = *comb*(*Tp*) + *comb*(*Tw*);
comb(*Tv*) = *comb*(*Tv*) + *comb*(*Tw*);
p ← *w*;
Let *w* be the root of *p*'s bigger child subtree;
Disconnect *Tw* from *Tp*;
comb(*Tp*) = *comb*(*Tp*)- *comb*(*Tw*);
comb(*Tv*) = *comb*(*Tv*)- *comb*(*Tw*);
$S \leftarrow S \cup \{v, w\}$, *m=m+1*;

3.2.2 Else

Let C_L and C_R be the children of node *v*;
If |*comb*(Tc_L) - *comb*(Tc_R)| < δ, Then
Disconnect Tc_R from *Tv*;
comb(*Tv*) = *comb*(*Tv*)- *comb*(Tc_R);
$S \leftarrow S \cup \{v, C_R\}$, *m=m+1*;
Else
Let *w* be the root of *v*'s bigger child subtree;
Disconnect *Tw* from *Tv*, *comb*(*Tv*) = *comb*(*Tv*)- *comb*(*Tw*);
While |*comb*(*Tv*) - *comb*(*Tw*)| > δ, do
If *v* was *w*'s parent node, Then
Reconnect *Tw* to *Tv*, *comb*(*Tv*) = *comb*(*Tv*)+ *comb*(*Tw*);
Else
p ← *w*'s parent node, reconnect *Tw* to *Tp*;
comb(*Tp*) = *comb*(*Tp*) + *comb*(*Tw*);
comb(*Tv*) = *comb*(*Tv*) + *comb*(*Tw*);
p ← *w*;
Let *w* be the root of *p*'s bigger child subtree;
Disconnect *Tw* from *Tp*;
comb(*Tp*)=*comb*(*Tp*)-*comb*(*Tw*), *comb*(*Tv*)=*comb*(*Tv*)-*comb*(*Tw*);
$S \leftarrow S \cup \{v, w\}$, *m=m+1*;

4. $S \leftarrow S \cup S_i$;
5. Each element of S leads a cluster; assign each cluster to a processor.

4 Shenoy's Binary Trees

Shenoy described an approach to constructing a binary join tree in [12] where each node accumulates at most two pieces of evidence. This approach was further developed based on a one-step lookahead heuristic search to construct binary trees [13].

Approach in [12]: For a given qualitative Markov tree (that could be derived from a valuation network), a designated node is chosen as the root. Starting from the root, the total number of belief functions that are to be combined in the root is counted, in order to obtain the marginal for the root. If the root has *n* children, each sending it a belief function, in addition to its own, there will be *(n+1)-1=n* combinations. When *n>2*, multiple replicates (*n-1* replicates) of the root are created. These multiple copies and the children of the root are re-organized so that there is only *one combination* in each replicate node and in the root. For each tree in the forest obtained by ignoring the root and all its multiple copies (and the links from them), repeat the above proce-

dure until every node has only one combination. It should be pointed out that this *one* combination in a node only contributes to the final marginal of the joint for the root. That is, this *one* combination happens in Phase I in Section 2. If the marginals for some non-root nodes are required, there will be *at least another (and at most two)* combination in such a node when messages are propagated down the tree, depending on whether the node has one child or two. The total number of combinations in each node is shown in Fig.5, if we assume that a marginal for every node is of interest.

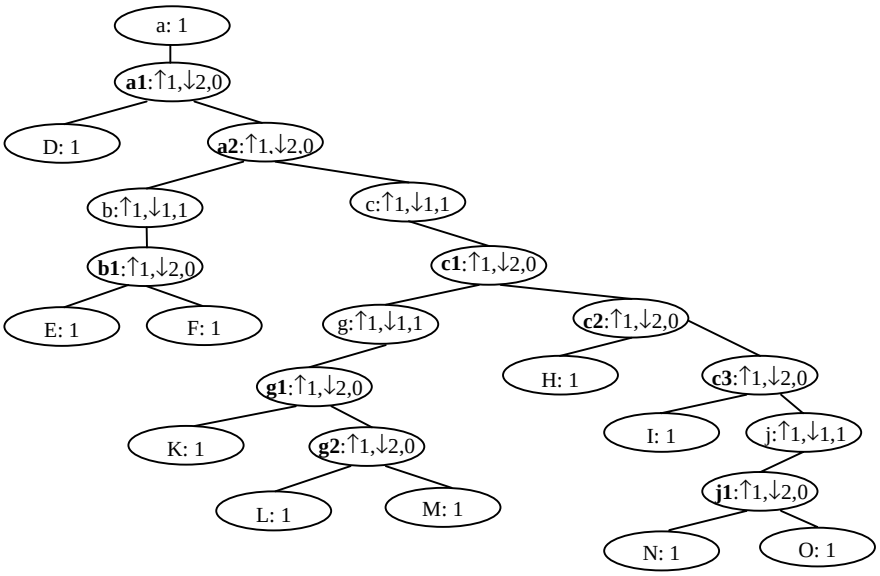


Figure 5. A binary join tree created using Shenoy's first method based on the original Markov tree in Figure 2. Bold-faced nodes are added ones. The multiple replicates of a node are numbered following the name of the original node.

One-step lookahead heuristic approach in [13]: Given a qualitative Markov tree, to compute the marginal for a node, alternative sequences of combinations of belief functions with different amount of computation, can be carried out. The heuristics in the one-step lookahead approach schedules combinations by sequencing deletion of variables from a qualitative Markov tree. The variable to be deleted next is the one that leads to a combination over the smallest set of configurations [13]. Starting from the nodes containing variables that should be deleted first, this method constructs a binary tree with these nodes as initial leaves, and build the binary tree as more variables to be deleted. Let Ω be the set containing all the nodes in a Markov tree (network), we summarize Shenoy's algorithm as follows.

Step 1. Selecting the variable(s) that should be deleted first, a subset Φ of Ω is formed with each element in Φ containing this (or these) variable(s). Let $\Omega=\Omega\backslash\Phi$.

Step 2. A pair of elements in Φ is chosen with the union set of the pair containing the minimum number of variables among all possible unions of pairs in set Φ. The elements in the pair are two leaves and removed from Φ. The union set of the pair is created and inserted into Φ. This new node acts as the parent of the two leaves.

Step 3. The above step is repeated until Φ has only one node, *m*, left. Create a new node, *n*, containing the remaining variables after deleting the chosen variable(s) from *m* and let *n* be its parent. Let $\Omega=\Omega\cup\{n\}$.

Step 4. Repeating Steps 1 to 3 for the next chosen variable(s) that should be deleted, until Ω has one element left which will be the root of the created binary tree.

When using Shenoy's second approach to transforming the qualitative Markov tree in Fig.2, we assume that each leaf node contains one variable, such as *e={E}*, and every non-leaf node contains the collection of variables in the leaves below it, such as *b={E,F}*. Based on the deleting sequence *{E}, {F}, {K}, {L}, {M}, {H}, {I}, {N}, {O},* and with *{D}* as the final remaining variable, the binary tree is built as shown in Fig.6. All the leaf nodes are the initial subsets of variables with original belief functions. All the other nodes are inserted later in order to either merge two subsets or to delete a variable(s) from a merged node, with latter being denoted with bold-faced font. If the final marginal is required for every original subset, the total number of combinations in each node is shown in Fig.6. In summary, the total numbers of combinations in the original Markov tree (Fig.2), in the binary tree in Fig.3, and in the binary trees in Fig.5 and Fig.6 are 64, 53, 50 and 55 respectively.

5 Comparison among these three types of binary trees and Conclusion

Where does binary occur? The binarization in our structure means that each node in the final tree should contain no more than two child nodes. If a node has its initial belief function in addition to that from its two children, then there will be three belief functions requiring *two combinations.* However, this is *not allowed* in Shenoy's approaches. Since in his structures, there can only be one combination in each node. Therefore, a replicate of this node will separate the node with its two children. For example, node *a1,* a replicate of *a*, separates nodes *a2* (another replicate of *a*) and *D*, in Fig.5. Such a replicate node *does not exist* in our binary tree.

Structural differences. In Shenoy's first approach, the number of nodes being added depends on how initial belief functions are assigned. For instance, if *a* does not have its initial belief function, then there will only be one replicate of *a*, since there is only one combination. Therefore, the structure (in terms of total number of added nodes) of a binary tree changes when different sets of belief functions are given initially. On the contrary, in our algorithm, the number of nodes being added is fixed. The only difference that alternative sets of initial belief functions make is the arrangement of sub-branches, due to the change of calculations in different branches.

Computational cost. In terms of computational cost, our algorithm has some extra combinations than Shenoy's first algorithm. The extra combinations occur when a node has a belief function from its parent and one of its own. These two belief functions are combined twice before it being combined with that from one of its children for the purpose of sending the combined belief function to another.

Why there are so many added nodes? The major step in Shenoy's second approach is to arrange leaves (all the nodes in the original Markov tree) based on the deletion sequence, and to construct sub-trees. Each sub-tree results in at least two newly added nodes, one is the union of variables in the two nodes being merged and another (the root of the sub-tree) contains variables after deleting selected variables

from the former. All the non-leaf nodes are added ones. In Fig.6, there are 23 added nodes in comparison to 4 and 9 added nodes in Fig.3 and Fig.5 respectively. Moreover, these added nodes cause more calculations on marginals. For example, when propagating up the tree, each bold-faced node requires its marginal from the node below it after deleting a variable. A marginal is again required the other way round. Therefore, this method is expensive in both the total number of combinations and in preparation for combinations.

Conclusion. In this paper, we proposed a computational workload-based algorithm to transform a qualitative Markov tree into a binary tree and an algorithm for partitioning the tree into clusters for parallel process. We also compared our binary tree with that constructed from Shenoy's two algorithms in [12] and [13] respectively. The study shows that Shenoy's first algorithm requires the least amount of combination and it is the most efficient one if only one processor is used. Our binary trees require some extra combination than Shenoy's first type of trees, but should perform well when multiple processors are available. Shenoy's second type of trees is expensive both in a single or a multiple processor environment due to large number of added nodes and additional calculations on marginals for added nodes.

Reference

1. Barnett, J. A.: Computational methods for a mathematical theory of evidence. *Proc. of IJCAI'81*, Vancouver, BC, 868-875
2. Cano, A., Moral, S., and Salmeron, A.: Penniless propagation in join trees. *International Journal of Intelligent Systems*, Vol. 15, 2000, 1027-1059
3. Dubois, D. and Prade, H.: Inference in possibilistic hypergraphs. *Proc. of IPMU'90*, 228-30
4. Gordon, J. and Shortliffe, E.H.: A method for managing evidential reasoning in a hierarchical hypothesis space. *Artificial Intelligence*, Vol.26, 1985, 323-357
5. Liu, W., Hong, X., and Adamson, K.: Parallel implementation of evidence combination in qualitative Markov tress. Technical Report, University of Ulster
6. Madsen, A. and Jensen, F.: LAZY propagation: A junction tree inference algorithm based on lazy propagation. *Artificial Intelligence*, Vol.113, 1999, 203-245
7. Maheshwari, P. and Shen, H.: An efficient clustering algorithm for partitioning parallel programs. *Parallel Computing*, Vol. 24, 1998, 893-909
8. Pearl J.: *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, San Mateo, 1988
9. Shafer, G.: *A mathematical theory of evidence*. Princeton University Press, 1976
10. Shafer, G. and Logan, R.: Implementing Dempster's rule for hierarchical evidence. *Artificial Intelligence*, Vol.33, 1987, 271-298
11. Shafer, G., Shenoy, P. and Mellouli, K.: Propagating belief functions in qualitative Markov trees. *Int. J. of Approx. Reasoning*. Vol. 1, 1987, 349-400
12. Shenoy, P.: Binary join trees. *Proc. of UAI*'96, 492-499
13. Shenoy, P.: Binary join trees for computing marginals in the Shenoy-Shafer architecture. *Int. J. of Approx. Reasoning*, Vol. 17, 1997, 239-263
14. Shenoy, P. and Shafer, G.: Propagating belief functions with local computations. *IEEE Expert*, Vol. 1, 1986, 43-52
15. Tessem, B.: Approximations for efficient computation in the theory of evidence. *Artificial Intelligence*, Vol.61, 1993, 315-329
16. Voorbraak, F.: A computationally efficient approximation of Dempster-Shafer theory. *International Journal of Man-Machine Studies*, Vol.30, 1989, 525-536
17. Wilson, N.: A Monte-Carlo algorithm for Dempster-Shafer belief. *Proc. of UAI*'91, 414-17

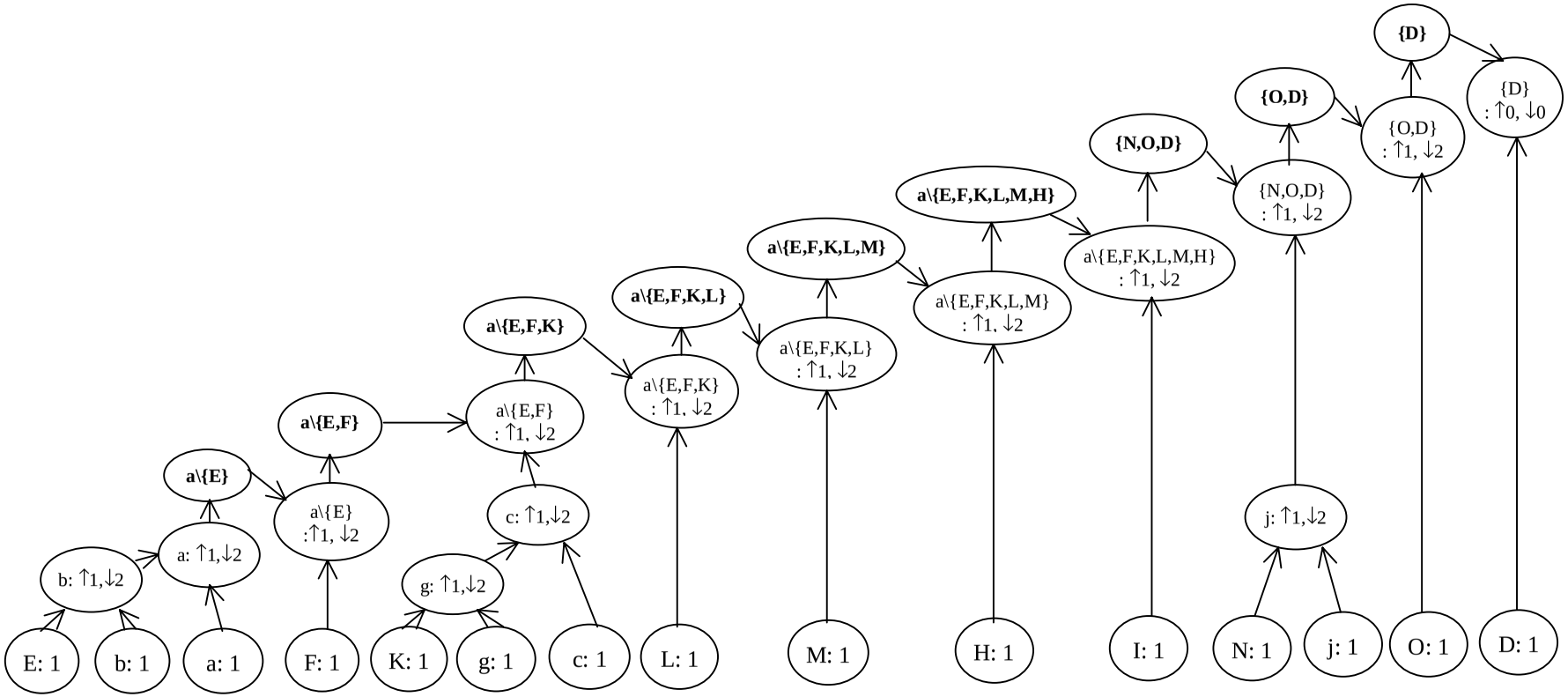


Figure 6. A binary tree created from Shenoy's second approach based on Figure 2