A Parallel Approach to Evidence Combination on Qualitative Markov trees

Xin Hong, Weiru Liu, Kenny Adamson
School of Computing and Information Engineering
University of Ulster at Coleraine, BT52 1SA, UK
Email: x.hong@ulster.ac.uk

Abstract Dempster's rule of evidence combination is computational expensive. This paper presents a parallel approach to evidence combination on a qualitative Markov tree. Binarization algorithm transforms 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. The parallel implementation improves the computational efficiency of evidence combination.

Keywords: evidence combination, binarization, partition, cluster, parallel processing

I. Introduction

Research work on improving the efficiency of belief combination in the Dempster-Shafer theory of evidence (DS theory) [1] started as earlier as in 1981 when Barnett proposed a linear algorithm for a special type of belief functions in [2]. Since then, many researchers have investigated various algorithms ([3, 4, 5, 6, 7, 8, 9, 10, 11], etc.) in two main categories, exact implementations of Dempster's combination rule or its approximations. Among exact implementations, the Shenoy-Shafer architecture for computing multiple marginals of the joint belief function in qualitative Markov trees using local computation has been popular. The term *Markov trees*, borrowed from probability theory and first used in [12] as *qualitative Markov trees*, means a tree of variables in which a separation implies probabilistic conditional independence given the separation variable [13]. As proved in [8], with this method, the exponential computational complexity in the size of total number of 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*, which was initiated for propagating probabilities in Bayesian causal trees by Pearl [14]. Local computation refers to a computation which involves only a small number of nodes in a large tree (or network). The basic idea of local computation is message passing among neighbouring nodes in a qualitative Markov tree to compute marginals of the joint belief distribution without actually calculating the joint belief distribution.

The efficiency of evidence combination in a qualitative Markov tree with local computation, however, depends on the sizes of nodes. If a node in a qualitative Markov tree has many neighbours, even local computation can become very inefficient.

In this paper, we present a parallel approach to evidence combination on a qualitative Markov tree. The binarization algorithm transforms a qualitative Markov tree into a binary tree, based on the amount of combinations at each sub-tree. The partitioning algorithm then partitions the 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 remainder of the paper is organised as follows. Section 2 introduces the basics of DS theory and belief propagation in qualitative Markov trees using local computation. Section 3 proposes a binarization algorithm for transforming a qualitative Markov tree into a binary join tree and a partitioning algorithm for clustering the binary tree for parallel processing. Section 4 provides a theoretical analysis on how much efficiency can be achieved by parallel implementation. Finally conclusions and discussions are given in section 5.

II. DS Theory and Qualitative Markov Trees

A. Basics of DS Theory

In DS theory [1], 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.

B. 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.

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 case, 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

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

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, $V_x = \{i | (i,x) \in E\}$ 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 neighbours,

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

For a leaf node x with only one neighbour 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 neighbours, the marginal $Bel^{\downarrow y}$ for y is obtained as $Bel^{\downarrow y} = Bel_y \oplus (\oplus\{M^{i \to y} | i \in V_y\})$. 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 $Ch_x = \{k | k \in V_x, k \text{ is a child node of } x\}$ 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 [7]:

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

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.

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

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

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

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.

Fig 1(a) presents 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. $(x{:}\uparrow t_1, \downarrow t_2, 1)$ 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 graph.

III. Binarization and Partitioning Algorithms

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 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.1(a), we get a balanced binarised tree as in Fig.1(b) where bold-faced nodes are added nodes.

Below is a partitioning algorithm that partitions a binary tree into clusters and assigns each cluster to a processor. As an example, the tree in Fig.1(b) is partitioned into four clusters as illustrated with shade.

Algorithm: Partitioning 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 $|comb(Tc_L)-comb(Tc_R)| < \delta$ (δ is a threshold saying that both branches have almost the same workload)
Then
Disconnect Tc_R from Tp;
$comb(Tp) = comb(Tp) - comb(Tc_R)$;
$comb(Tv) = comb(Tv) - comb(Tc_R)$;
$S \leftarrow S \cup \{v, C_R\}$, $m=m+1$;
Else
Let w be the root of p's bigger child subtree;
Disconnect Tw from Tp;
$comb(Tv) = comb(Tv) - comb(Tw)$;
$comb(Tp) = comb(Tp) - comb(Tw)$;
While $|comb(Tv) - comb(Tw)| > \delta$, do
Reconnect Tw to Tp;
$comb(Tp) = comb(Tp) + comb(Tw)$;
$comb(Tv) = comb(Tv) + comb(Tw)$;
$p \leftarrow w$;
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 $|comb(Tc_L) - comb(Tc_R)| < \delta$, 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 $|comb(Tv) - comb(Tw)| > \delta$, do
If v was w's parent node, Then
Reconnect Tw to Tv;
$comb(Tv) = comb(Tv) + comb(Tw)$;
Else
$p \leftarrow w$'s parent node, reconnect Tw to Tp;
$comb(Tp) = comb(Tp) + comb(Tw)$;
$comb(Tv) = comb(Tv) + comb(Tw)$;
$p \leftarrow w$;
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. $S \leftarrow S \cup S_t$;
5. Each element of S leads a cluster; assign each cluster to a processor.

IV. Theoretical Analysis

The parallel approach presented in this paper is currently being tested in a parallel machine with at most six processors. To illustrate the improvement of evidence combination on a qualitative Markov tree by our parallel technique, we have applied our algorithms to two examples (Fig.1(a) and Fig.2(a)) from [7]. The four-processor schedules produced by our approach are given in Fig.1(c) and Fig.2(c) respectively. The results show a completion time of 31 against 57 by the originally sequential scheme for the first example, and 24 against 55 for the second one. To map the communication cost on sending a message between two processors, we have the assumption as Maheshwari and Shen suggested in [15], which one communication for per data sample takes two time units. The configuration of our parallel machine allows communications for the send processor and receive processor take place simultaneously.

V. Conclusions and Discussions

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 clusers for parallel processing. Our parallel approach especially favors a Markov tree with many nodes that have more than three children nodes.

Other algorithms that binarizing a qualitative Markov tree into a binary tree exist (e.g. [8], [13], [16], etc.). 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. If there are many processors available to process some nodes (sub-trees) in parallel, a balanced tree provides a good structure to partition it into clusters so as to assign workloads to processors evenly. The detailed comparisons may be referred to [17].

The binarization algorithm in our approach is also similar to the binarization procedure in [15] 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), in relation to local computation. While the algorithm in [16] does not support local computation by using added nodes for message passing only.

References

[1] G. Shafer, *A mathematical theory of evidence*, Princeton University Press, 1976.

[2] J. A. Barnett, "Computational methods for a mathematical theory of evidence", *Proc. of IJCAI'81*, Vancouver, BC, 868-875

[3] D. Dubois and H. Prade, "Inference in possibilistic hypergraphs", *Proc. of IPMU'90*, 228-30

[4] J. Gordon and E. H. Shortliffe, "A method for managing evidential reasoning in a hierarchical hypothesis space", *Artificial Intelligence*, Vol.26, 1985, 323-357.

[5] R. Kennes and Ph. Smets, "Computational asspects of the Mobius transform", *Proc. Of UAI* -1990, 401-416.

[6] G. Shafer and R. Logan, "Implementing Dempster's rule for hierarchical evidence", *Artificial Intelligence*, Vol.33, 1987, 271-298.

[7] G. Shafer, P. Shenoy and K. Mellouli, "Propagating Belief Functions in Qualitative Markov Trees", *Int. J. of Approx. Reasoning*, Vol.1, 1987, 349-400.

[8] P. Shenoy, "Binary Joint Trees", *Proc. Of UAI*, 1996, 492-499.

[9] B. Tessem, "Approximations for efficient computation in the theory of evidence", *Artificial Intelligence*, Vol.61, 1993, 315-329.

[10] F. Voorbraak, "A computationally efficient approximation of Dempster-Shafer theory", *International Journal of Man-Machine Studies*, Vol.30, 1989, 525-536.

[11] N. Wilson, "A Monte-Carlo algorithm for Dempster-Shafer belief", *Proc. of UAI'*91, 414-17.

[12] P. Shenoy and G. Shafer, "Propagating belief functions with local computations", *IEEE Expert*, Vol. 1, 1986, 43-52.

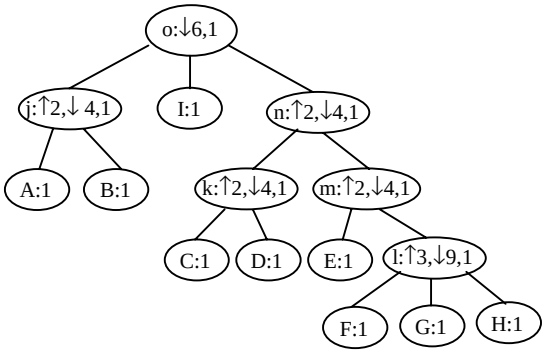
[13] P. Shenoy, "Binary Join Trees for Computing Marginals in the Shenoy-Shafer Architecture", *Int. J. of Approx. Reasoning*, Vol.17, 1997, 239-263.

[14] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann, San Mateo, 1988.

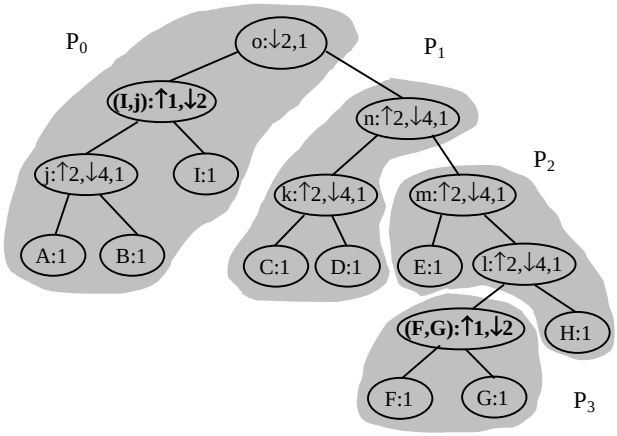
[15] P. Maheshwari and H. Shen, "An Efficient Clustering Algorithm for Partitioning Parallel Programs", *Parallel Computing*, Vol.24, 1998, 893-909.

[16] A. Cano, S. Moral and A. Salmeron, "Penniless propagation in join trees", *International Journal of Intelligent Systems*, Vol.15, 2000, 1027-1059.

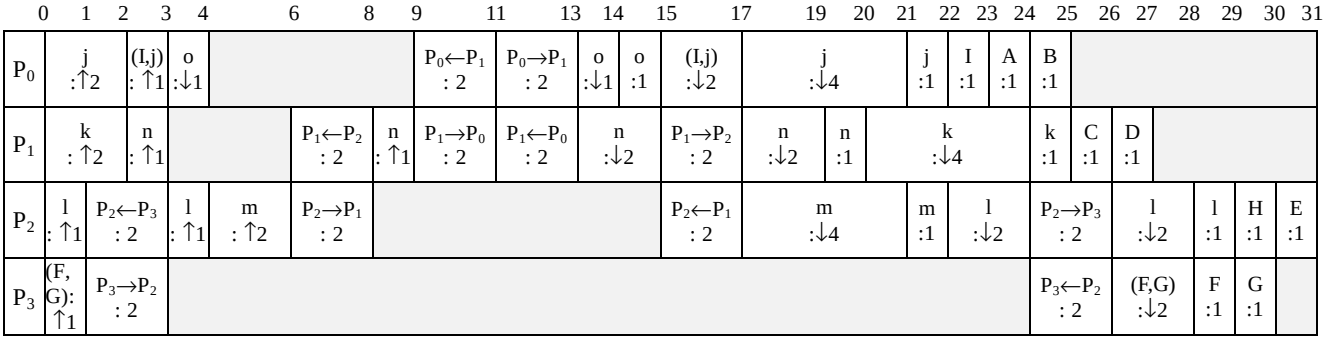
[17] W. Liu, X. Hong and K. Adamson, "Computational-workload based binarization and partition of qualitative Markov trees for belief combination", technical report, University of Ulster.



(a) A diagnostic tree

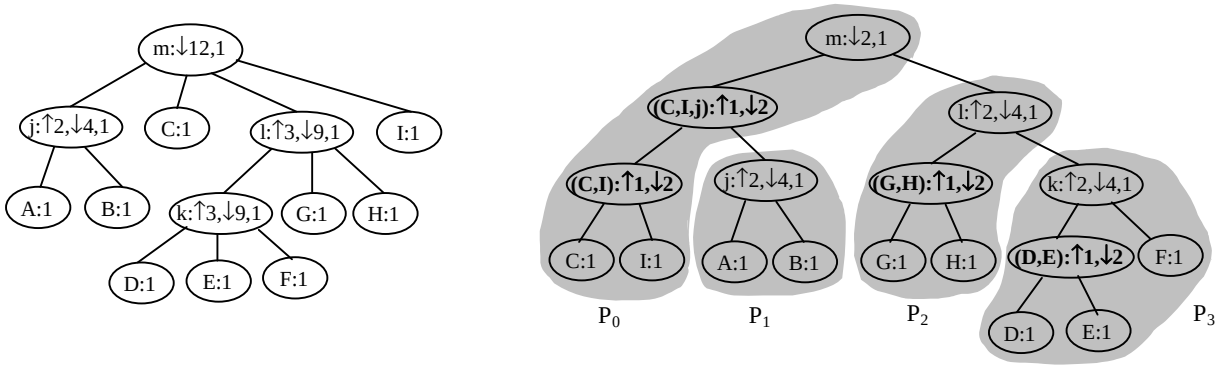


(b) Binarized tree with four clusters



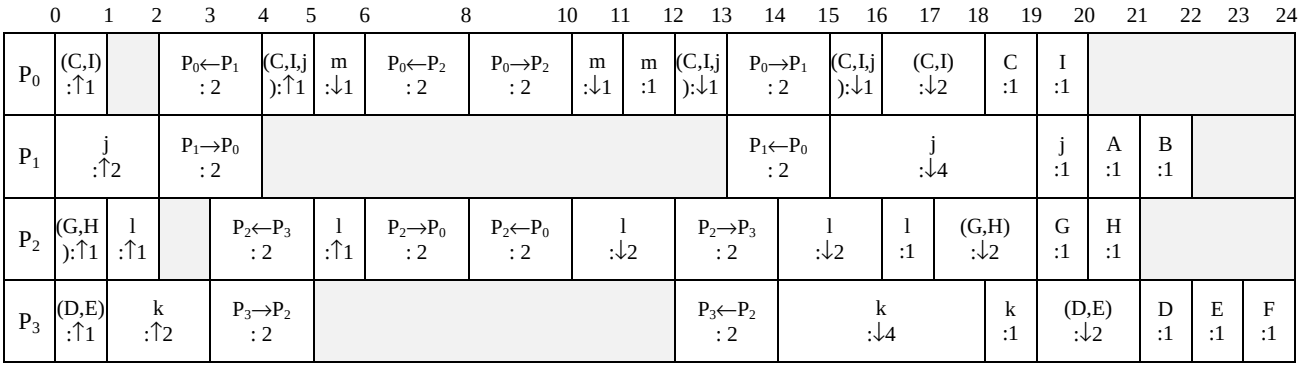
(c) Four processor schedule

Fig 1. Example 1 from [7]



(a) A tree

(b) Binarized tree with four clusters



(c) Four processor schedule

Fig. 2 Example 2 from [7]