Graduate School
University of South Florida
Tampa, Florida

CERTIFICATE OF APPROVAL

Master's Thesis

This is to certify that the Master's Thesis of

NINA SAXENA

with a major in Computer Science has been approved by
the Examining Committee on November 27, 1995
as satisfactory for the thesis requirement
for the Master of Science in Computer Science degree

Examining Committee :

Major Professor: N. Ranganathan, Ph.D.

Co-Major Professor: Sudeep Sarkar, Ph.D.

Member: Peter M. Maurer, Ph.D.

MAPPING AND PARALLEL IMPLEMENTATION OF BAYESIAN BELIEF NETWORKS

by

NINA SAXENA

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science & Engineering
University of South Florida

December 1995

Major Professor: N. Ranganathan, Ph.D.

DEDICATION

This thesis is dedicated to my parents –

Prof. M. K. Saxena and Dr. Bina Saxena.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank my major professor Dr. N. Ranganathan for his valuable guidance and understanding. It was from him that I learned to be patient while facing problems and to be a true researcher. I wish to thank my co-major professor Dr. Sudeep Sarkar for his relentless support and help throughout this work. Without his invaluable suggestions, this work would not have been possible. My thanks to Dr. Peter M. Maurer for his encouragement and support. Finally, I would like to thank Dr. Sartaj Sahni of the Department of Computer Science, University of Florida, Gainesville, for permitting me to work on the nCUBE and James F. Hranicky for helping me time and again with the machine.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

LFC	Lambda From Child
PFP	Pi From Parent
UP. BEL.	Update Belief
IR	Information Retrieval
CPN	Causal Probabilistic Network
PIN	Perceptual Inference Network
CP	Conditional Probability

MAPPING AND PARALLEL IMPLEMENTATION OF BAYESIAN BELIEF NETWORKS

by

NINA SAXENA

An Abstract

Of a thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science & Engineering
University of South Florida

December 1995

Major Professor: N. Ranganathan, Ph.D.

Bayesian belief networks are used for graphically modeling uncertainty and probabilistic dependence. They are useful in representing computations in systems in which the decisions are conditionally dependent on different controlling factors. Bayesian networks are applied in different fields including computer vision, object recognition, feature detection, medicine, CAM and troubleshooting in systems. Since most of the above applications are computationally intensive and require fast response, parallel implementation of Bayesian networks is important. This thesis presents a general technique of mapping tree structured Bayesian belief networks onto the hypercube parallel processing archietecture. The proposed mapping scheme maintains parent-child adjacency and single hop message passing throughout the computation. The scheme is deadlock free since all the messages are received and processed in the order of structural hierarchy of the nodes in a tree. The task allocation is static and is done at the beginning of the computation. The scheme allows efficient mapping of arbitrarily large trees onto a fixed size hypercube. Extensive simulations were performed on a 64-node Silicon Graphics Hypercube machine. It is shown that the overall speed up due to parallel computation depends on the height of the Bayesian tree.

Abstract Approved: _____

Major Professor: N. Ranganathan, Ph.D.
Associate Professor
Department of Computer Science & Engineering

Date Approved: _____

CHAPTER 1

INTRODUCTION

In the field of knowledge based systems, uncertainty plays a crucial role. One of the ways of managing this uncertainty is using the laws of probability theory. However, the complete specification of a general probabilistic system is practically cumbersome. This complexity arises from the need to specify a set of probabilities whose size is exponential with the number of random variables.

Let $P(x_1, ..., x_n)$ represent a knowledge base by a joint probability distribution, on a set of binary random variables, $x_1, ..., x_n$, then to store $P(x_1, ..., x_n)$ would require 2^n entries. Computing $P(x_i|x_j)$ which represents the confidence in x_i given the state of x_j would require computing $P(x_i x_j)/P(x_i)$. This would require the division of one marginal probability by another, where each is individually generated by summing up the probabilities for an exponentially large number of combinations. This computational burden can be significantly reduced if we can express the joint probability density using a set of smaller conditional probabilities. This set depends on the direct dependencies of each node. For example, let probability distribution $P(X_1, ..., X_7)$ be factored as $P(X_7|X_2, X_6)P(X_6|X_4, X_5)P(X_5)P(X_4|X_1)P(X_3|X_1)P(X_2)P(X_1)$.
We can graphically represent this factoring using the graph in Figure 1. Each node in the graph represents a random variable and the links denote direct dependencies as specified in the joint probability factoring. The links denote the conditional probabilities. The graph explicitly represents the underlying dependencies. This network representation of the joint probability density is called a Bayesian network [1]. This is also known as Belief network or causal network. Thus, a Bayesian network is a

directed graph whose nodes denote random variables and the links encode the dependencies between the variables. The network structure also acts as a computational backbone. Each node is a computational unit. The effect of a change in the belief about a random variable $P(x_i)$ can be propagated to other variables using this network structure.

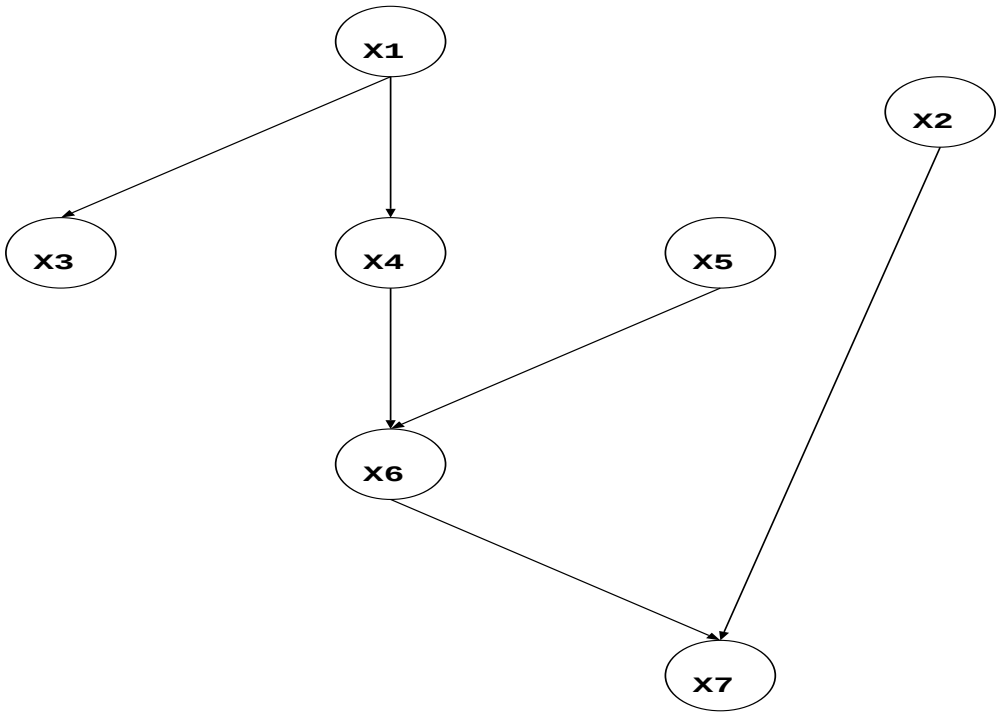

Figure 1. An example Bayesian network with edges directed from causes to effects

Pearl [1] showed that if the Bayesian network is a directed acyclic graph then there exists a very efficient probability updating algorithm which involves message passing among the random variables. The updating algorithm is $O(n)$ in the number of nodes. It may be noted that updating a general Bayesian network with no topological restrictions is an NP-hard problem [2, 3].

Each node in a Bayesian tree is a variable in a Bayesian Network and can acquire any value from a set of mutually exclusive states that together cover the entire set of possible events. There can be two or more states for every variable. A classic example of a variable with 2 states can be the "Yes-No" or the "True-False" type. On the other hand, the status of a process in a machine can acquire more than 2 states such as "Running", "Ready", "Blocked" and "Waiting".

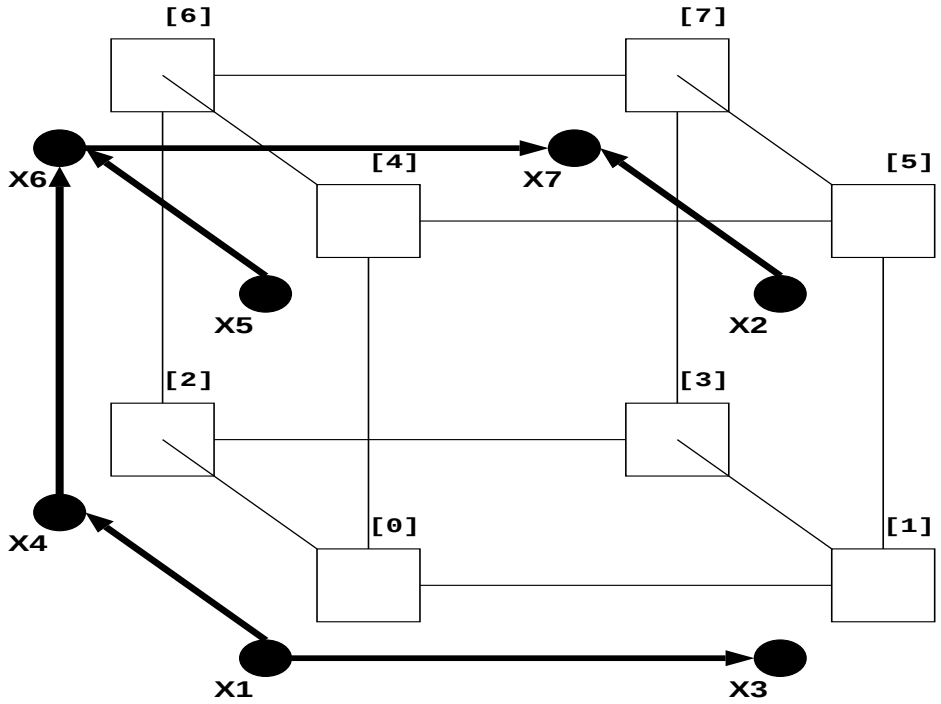

Figure 2. Mapping the Bayesian network of Figure 1 on a hypercube, network shown with filled ovals and dark edges

This thesis describes a new task allocation scheme for mapping a Bayesian tree onto a hypercube architecture. Figure 2 shows the mapping of the Bayesian network shown in Figure 1 onto an eight-node hypercube. The belief network is denoted by thick lines and the shaded nodes are marked as $X1$ to $X6$. The 3-dimensional hy-

percube is marked out by thin lines and the hypercube nodes are specified within square brackets. Issues such as synchronization, preventing deadlock without much busy-wait, load balancing and preserving functional integrity are considered. Depending on the specific application, the computational technique can be made to suit a particular network structure [4, 5] or the nature of inferences to be drawn [6, 7]. This thesis focusses on parallelization of tree structured Bayesian network which is a relatively simpler model than the more general Bayesian networks graphs.

The thesis is organized as follows. A brief literature survey is given in Chapter 2. Some real life applications briefly in this chapter. Chapter 3 discusses Bayesian network updating. Mapping and implementation are discussed in Chapter 4. Finally, Chapter 5 presents the results, comparisons with related schemes and suggestions for further research.

CHAPTER 2

RELATED WORK

This thesis presents a task scheduling algorithm for mapping the computations of a Bayesian network onto a message passing hypercube architecture. The first section of this chapter introduces the Bayesian networks concepts. The second section reviews task mapping schemes for hupercube that have been proposed in the literature. The last part of this chapter discusses different applications in which Bayesian networks are used.

2.1. Belief Networks

Since the first introduction by Wright [8], significant amount of work has appeared on Bayesian networks [9, 10, 11, 12, 1, 13, 14]. Pearl [1] showed that propagating the effect of new evidence and updating of beliefs can be done very efficiently using probabilistic measures on tree structured Bayesian networks. The following section discusses evidential reasoning which involves the outcome of introducing new evidences on different hypothesis within a problem domain.

2.1.1. Evidential reasoning

The concept of Bayesian network based reasoning and the associated method of propagation can be understood from the following example adopted from [1]. There was a theft in a town X by a masked thief. The police officer, chasing the thief, witnessed the masked thief escaping in a train bound to town Y. The police suspected 3 people, only one of which was the thief. Since the escape was preplanned and the

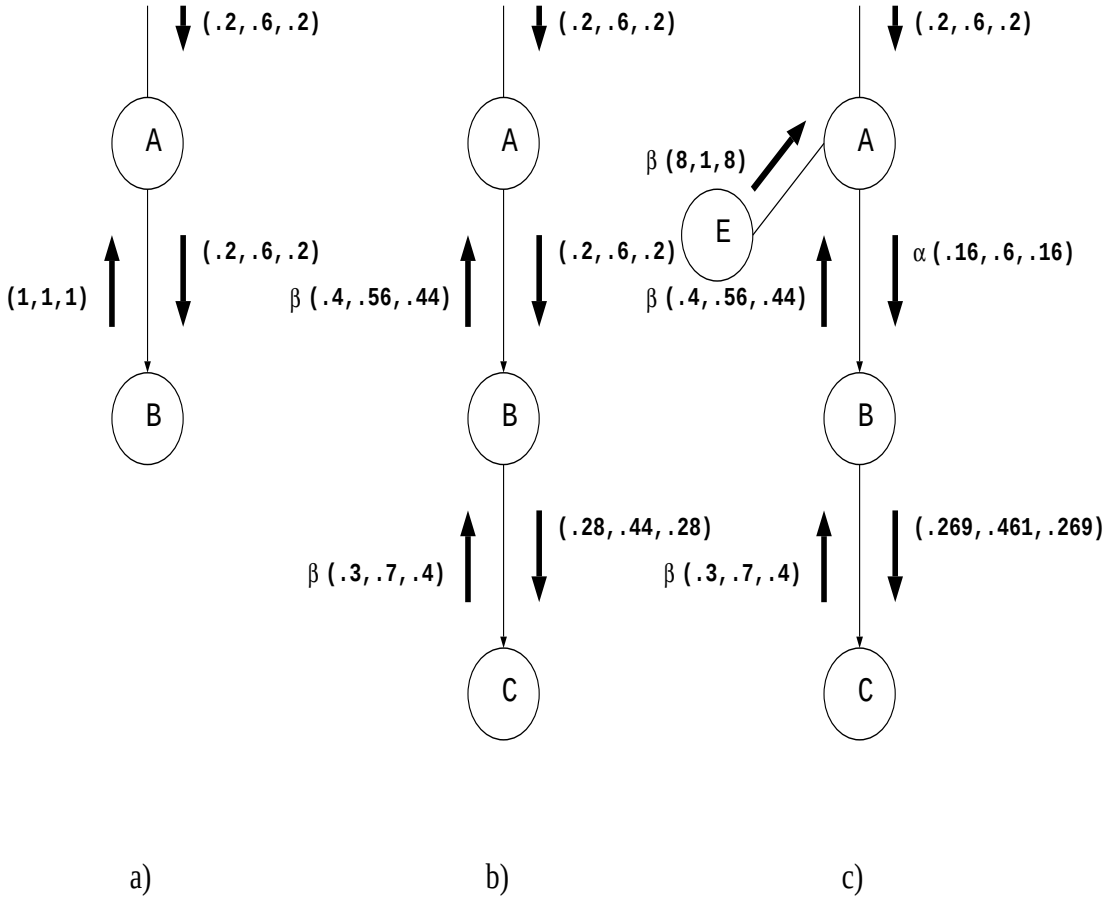

Figure 3. Belief updating and propagation (a) Without check results, (b) With check results C, (c) With evidence E

train was known to have prior reservation for every passenger, the reservation chart was searched to identify the passengers. Let A stand for the identity of the man who boarded the train in a mask, that is, the thief. Let B show the identity of the man who boarded the train from town X and alighted at town Y. Let C be the reliability of the check. The variables a, b and c represent nodes A, B and C respectively.

By keeping in mind that it is very unlikely that 2 or all 3 of the suspects together may have been travelling in the same train, we can construct a 3X3 conditional

probability matrix, $M_{b|a}$, such that

$M_{b|a} = P(b|a) = 0.6$ if $a = b$ for $a, b = 1, 2, 3$ or

$M_{b|a} = P(b|a) = 0.2$ if $a \neq b$ for $a, b = 1, 2, 3$.

Also, considering the check conducted for reservation to be very thorough and reliable, we have

$M_{c|b} = P(c|b)$, such that $\sum_c P(c|b) = 1$ for $c = 1, 2, 3$.

This means that if the suspect B was on the train, then the chances of him being reserved can be expected to be $P(c|b)$.

Let the initial belief in the identity of the thief be

$\pi(a) = (0.2, 0.6, 0.2)$.

Communication of information in Bayesian networks is through two types of messages, called the *lambda* message and the *pi* message. A pi message is the message sent from a parent node to its child node and is defined as

$$\pi_Y(x) \;=\; P\{x, e^+\}$$

A lambda message is sent from the child to its parent. It can be defined as

$$\lambda_Y(x) \;=\; P\{e^-|x\}$$

Here, e^- denotes the evidences of the subtree with its root at node X and e^+ denotes the evidences from the rest of the tree. Hence, in Bayesian networks, lambda messages for any node represent the probability of the existence of evidences for its child nodes. Similarly, pi messages represent the probability of the existence of the node provided the evidences for its parent nodes exist. Since the node B is a leaf node that has not received any evidence, its belief value should be the same as its pi value. Hence, we keep its lambda vector as (1,1,1) as shown in Figure 3a. Now,

$$\pi(b) = \pi_B(a)M_{b|a} = (0.2, 0.6, 0.2).M_{b|a} = (0.28, 0.44, 0.28) = BEL(b).$$

Suppose that the check results now send their information through the lambda message

$\lambda_C(b) = \lambda(b) = \beta(0.3, 0.7, 0.4)$ as shown in Figure 3b.

Then, after belief updation, belief of node B becomes

$$BEL(b) = \alpha\lambda(b)\pi(b) = \alpha(0.3, 0.7, 0.4)(0.28, 0.44, 0.28) = (0.167, 0.611, 0.222),$$

where α is a constant for normalizing, such that the sum of all elements of BEL(b) comes to be 1.

Then, B sends a lambda message to A:

$$\lambda_B(a) = M_{b|a}.\lambda(b) = \beta(0.4, 0.56, 0.44).$$

Hence, A updates its belief to

$$BEL(a) = \alpha\lambda(a)\pi(a) = \alpha(0.4, 0.56, 0.44)(0.2, 0.6, 0.2) = (0.159, 0.667, 0.174)$$

Hence, at this point of time suspect 2 is the most probable thief.

Let us say now that an evidence is found which reduces the chances of suspect 2 being the thief by a factor of 8, as depicted in Figure 3c. This evidence is thus passed as a lambda message to node A of

$$\lambda_E(A) = \beta(8, 1, 8).$$

Hence,

$$\lambda(a) = \lambda_E(a)\lambda_B(a) = \beta(3.2, 0.56, 3.52).$$

Therefore,

$$BEL(a) = \alpha\lambda(a)\pi(a) = \alpha(3.2, 0.56, 3.52)(0.2, 0.6, 0.2) = (0.381, 0.2, 0.419).$$

This causes the message

$$\pi_B(a) = \alpha\lambda_E(a)\pi(a) = \alpha(0.16, 0.6, 0.16).$$

Then,

$$\pi(b) = \pi_B(a).M_{b|a} = \alpha(0.2, 0.6, 0.2).M_{b|a} = (0.269, 0.461, 0.269).$$

So, for node B,

$$BEL(b) = \alpha\lambda(b)\pi(b) = \alpha(0.3, 0.7, 0.4)(0.269, 0.461, 0.269) = (0.158, 0.631, 0.211).$$

This, thereby reduces the probability of suspect 2 being the thief to 0.631. Similarly, a stronger external evidence may even change the probabilities so much as to make someone else the prime suspect.

Thus we can consider an evidence as a perturbation at some point of the network, the effect of which ripples through the entire tree. The exchange of information is through *lambda* and *pi* messages that are from a child node to a parent node or from parent to child respectively. Every time a node gets either a pi or a lambda message, it updates its belief and then passes the revised values ahead to the other neighbours, which follow the same pattern.

In the above scheme, updating for each evidence will involve message passing over the entire network. Peot and Shachter [15] suggested handling multiple observations or evidences in two passes. A third pass is usually added, as the zero-th pass, to probe for changes due to new evidence. The new method required each node to be visited at the most twice, irrespective of the number of evidences. This algorithm has been discussed in greater detail in Chapter 4 since it has been used for network updation in the proposed work.

2.2. Task allocation in communicating processors

The problem of task allocation in any specific application is a very important one and depends on the method of computation. Each parallel algorithm can have several ways of being executed and its efficiency depends much on the way the task allocation is done. It is therefore absolutely essential to have a task mapping strategy that gives maximum benefit in terms of speed and cost. Since the literature on this topic is rich, we only discuss those mapping approaches that are relevent to this work.

Some algorithms are mapped based on the critical path and completion time constraints as in [16]. Since the order and times when each processor is active depends on the flow of data, inter-dependencies and timing requirements, parallelism in any problem can be fully utilized only when these factors are completely known. The task mapping in Bayesian networks will be efficient if the computation behaviour as well as the communication patterns are known to the algorithm.

2.2.1. Mapping on hypercubes

As stated in [17], the hypercube architecture is commonly preferred over other communication networks since it has a small diameter and a large number of redundant paths with very low hardware cost. Hence, it can be used for highly parallel and communicating tasks. There are many works that deal with task mapping on hypercubes, some of which are discussed below.

One of the major problems in task mapping is to reduce the communication overhead. The main issues concerning task mapping in hypercubes are discussed in [18] and [19]. The problem of mapping tasks to get minimum communication overhead in every case is NP-complete [17]. The goal is to get an efficient mapping that leads to less overhead and better run time. In a task graph, the nodes and the links of the graph represent the computations and the communication paths. According to

Horiike [17], the communication cost of an algorithm is the sum of the products of the amount of communication on each path and the path length. So, for an n-dimensional cube, the algorithm proposed by Horiike has n stages. It starts with an initial stage in which the tasks are mapped onto 2^n 0-dimension cubes. Each k-th stage consists of mapping the graph onto 2^{n-k} k-dimension cubes such as to minimize communication time.

In an n-dimensional hypercube, each processor is connected to n other processors. The processors adjacent to a specific processor all differ in just one bit with the processor. Similarly, all processors at a distance d differ in d bits with the processor. The hypercube structure is recursively constructed in that the $n+1$-th cube is constructed by joining the corresponding links of two n-dimension cubes. It consists of 2^n links. The mapping algorithm suggested in [17] is as follows:

```
Begin
    for i = 0 to N-1 do begin
        map task i onto C_{0,i};
    end;
    m = n;
    for k = 1 to n do
    begin
        for i = 0 to 2^m − 2 do begin
            for j = i+1 to 2^m − 1 do begin
               compute a gain of G_{kij};
            end;
```

```
        end;
        determine combinations;
        for i = 0 to N-1 do
        begin
            map task i into C_{k,i};
            map task i onto P_{k,i,j};
        end;
        m = m - 1;
    end;
End.
```

In the above algorithm, n is the dimension of the cube, k is the mapping stage and $m = n - k$. $C_{k,i}$ stands for the 2^m k-dimension cubes and $P_{k,i,j}$ are the 2^k processors. Gain G_{kij} is the difference in communication costs between one stage and the next, that is, the gain obtained on combining $C_{k,i}$ and $C_{k,j}$.

Other criteria for mapping are *Dilation Bound* and *Expansion Ratio* as mentioned in [20]. *Dilation Bound* is defined as the length of any edge in the graph after mapping. *Expansion Ratio* is the ratio of the total number of functional links between processors to the number of edges in the graph. Hence, mapping is done either to minimize the number of extra nodes used in support of intertask communication or to minimize the cost factor of the effective distance between the nodes executing adjacent tasks in the task graph.

2.2.2. Tree mapping

One of the very common structures of task graphs is that of a tree. A tree represents the computations in many applications. Some algorithms suggest "cluster-

mapping" for trees [21] [22]. They are based on partitioning the node set of a task tree into clusters or groups and mapping these groups on to various processors.

However, such a scheme has the following constraints:

a) All computations and communications should be synchronized.

b) Any processor cannot communicate with another processor while computing.

c) Communication between any processor-pair is unidirectional.

d) Each link can transmit no more than one value at a time.

e) Each task is executed only after all its child-tasks are executed.

Therefore the computation and communication times cannot be merged in the above scheme.

2.2.3. Message passing and deadlock free routing

Special techniques are required for mapping tasks that require message-passing. The task allocation should also be deadlock-free and optimal. Line ordering among various processing elements is often used to prevent deadlocks as described in [23]. The paper aims at determining the *Shortest Deadlock-Free Routing* and defines a path to be *shortest deadlock-free* (SDF) with respect to a coding scheme if it is the shortest path between the source and the destination nodes and the sequence of nodes on the path are monotonically increasing or decreasing in order.

There are some dedicated message passing models such as the CRAM or *Communicating Random Access Machine* [24]. Such models are constructed to optimize scheduling, task granularity, synchronization and communication among processors. Message passing between tasks has also been shown in [25] where two tasks with a link from task T1 to task T2, as shown in Figure 4a, share process information by sending and receiving data as shown in Figure 4b.

The scheme suggested by [20] requires multiple message hops between parent and child nodes. It also requires the number of processors to be equal to or greater

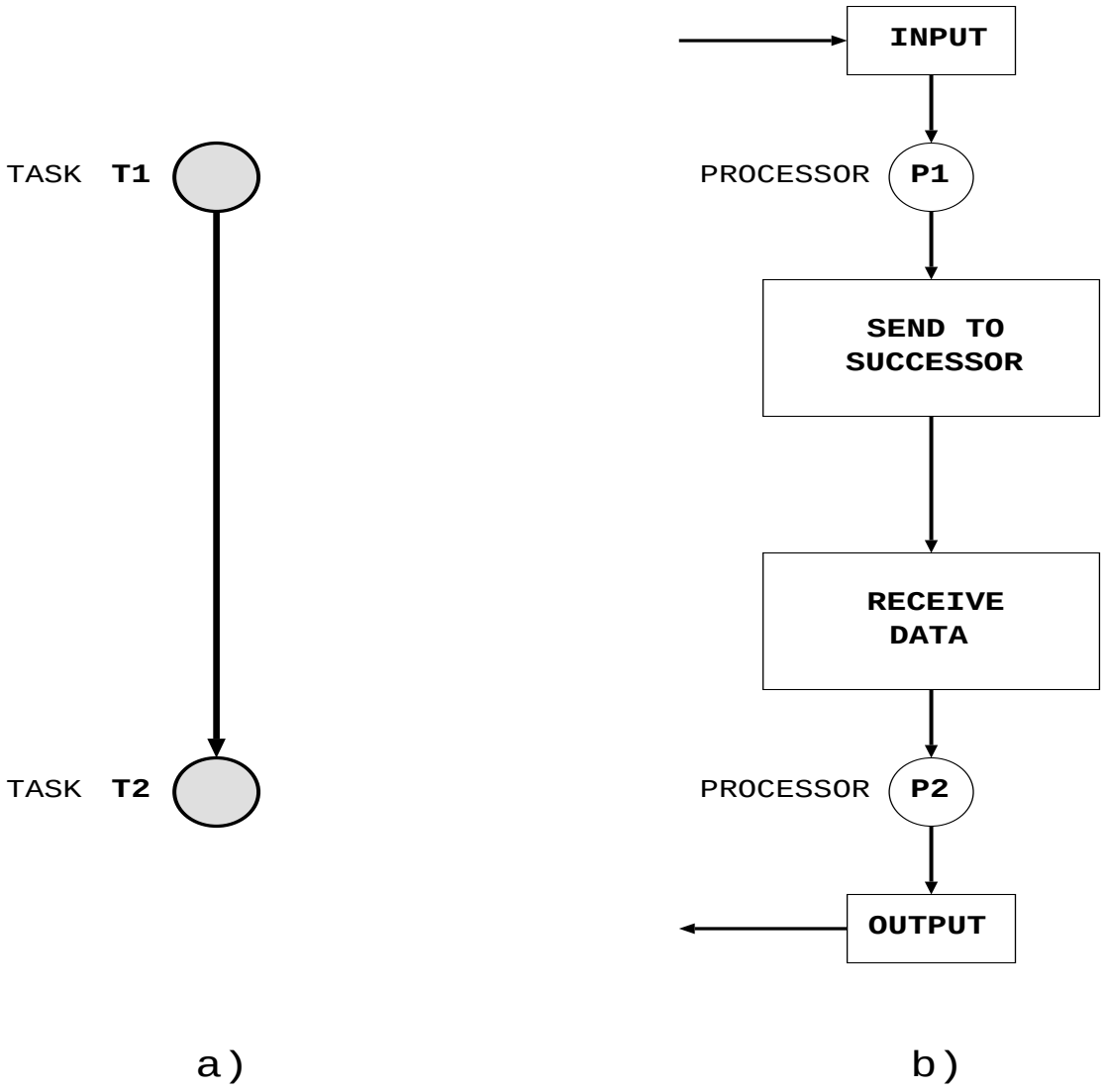

Figure 4. Message passing between connected tasks T1 and T2 which are mapped onto processors P1 and P2

than the number of nodes in the task tree. Therefore, it is not good for Bayesian networks because a large number of processors will be required to map the entire tree and there will be a large communication overhead because of multiple hops for each message. The method proposed in [17] has the same restrictions. The technique

described in [22] is not suitable due to the synchronization criterion required for computation and communication. The proposed scheme, being free of the above mentioned restraints, is thus suitable for Bayesian networks implementation.

2.3. Applications

Bayesian networks are used for various tasks such as price forecasting, manufacturing and processing, medicine, information retrieval and troubleshooting. This section gives a brief overview of some of these applications.

2.3.1. Medicine

Human glucose metabolism knowledge can be encoded using the Bayesian network shown in Figure 5 [26]. A priori insulin sensitivity is the main cause, along with the absorption speed. Carbohydrate intake, blood glucose concentration and insulin injections are also causes. The blood glucose concentration is a function of the present glucose and insulin input as well as the past states, by setting up dependence links from previous state to the outcome in the CPN or causal probabilistic network.

A similar concept is discussed in [27], where a non-uniform distribution of prior probabilities is assumed and the network is initialized with a very high level of glucose concentration and a very low level of insulin. The successive states evolve from the incoming evidences. The rate of convergence of these parameters to normal distributions gives the stability of the system.

As we can see, here fast computation is important for continuous monitoring, and hence parallelization, is very important since the amount of drugs to be administered has to be decided adaptively and continuously for terminal patients. The same

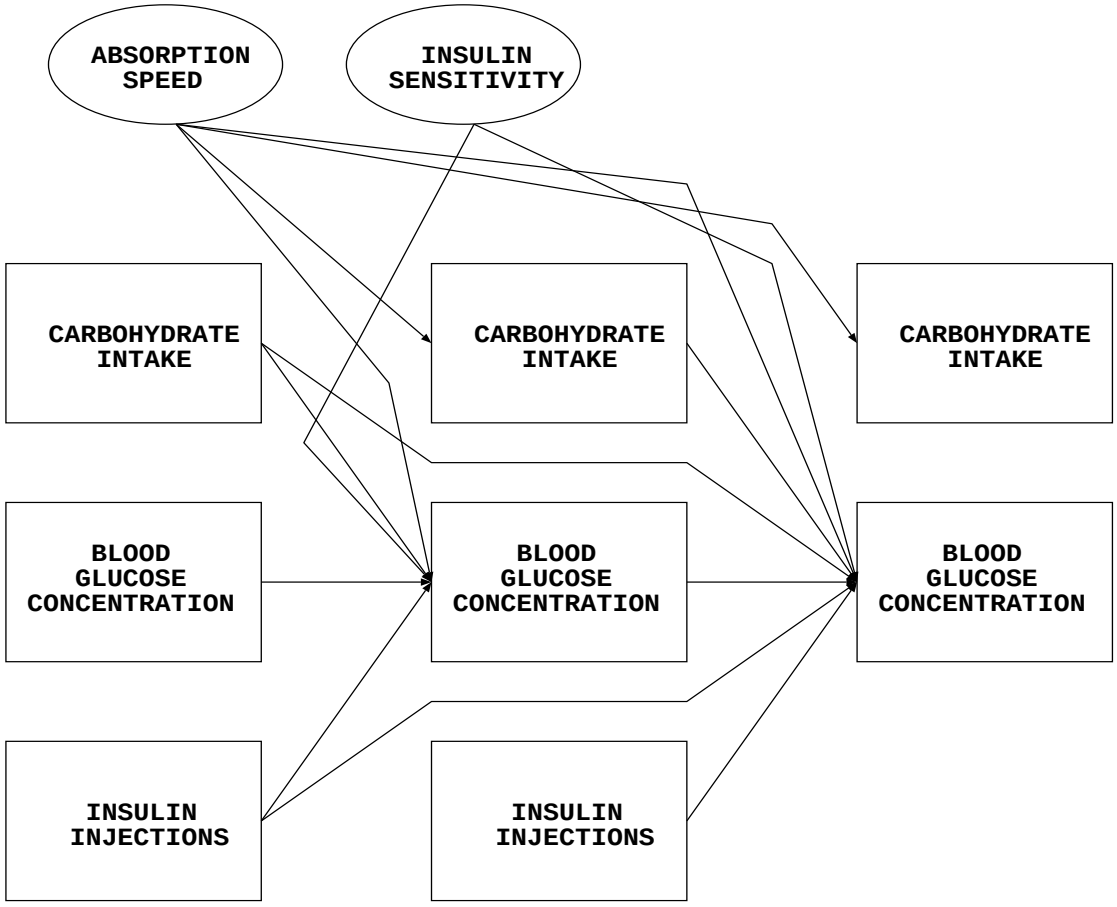

Figure 5. A CPN for blood glucose concentration [26]

holds true for diagnosis during emergencies. The difference between the two situations is that in the former case, the effect is more important and in the latter case, the cause.

2.3.2. Economy

While medical diagnosis is based on observations of symptoms to find the already present disease, that is, it projects its knowledge base backwards to find out the existing reason, forecasting is based on forward projection of the known factors

to presage a future event. Such a prediction is discussed in [28], where the price of gasoline is predicted on the basis of oil import fee, gasoline tax and the tax impact as depicted in Figure 6.

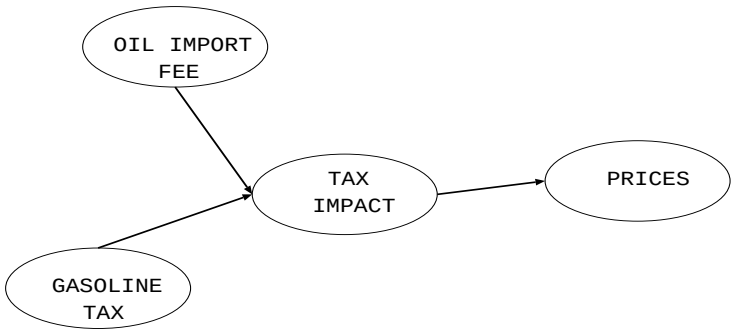

Figure 6. Forecasting gasoline prices [28]

Another example of such a forecasting can be the demand and supply forces in the market, which depend on the following factors:

(i) Prevailing price level at the beginning of the term,

(ii) Duration of time over which the current price has prevailed,

(iii) Amount of increased demand,

(iv) World growth and

(v) Seasonality. The graph can be drawn similar to Figure 6 with link directions depicting causation.

2.3.3. Manufacturing

Process modeling and dependence on various process parameters require an adaptive decision-making architecture [29], shown in Figure 7. The same is required for automating the process of setting the control parameters. The description of one such application is given in [29] where "recipe synthesis" for stress, rate and

film thickness in LPCVD or Low Pressure Chemical Vapour Deposition of undoped polysilicon is illustrated.

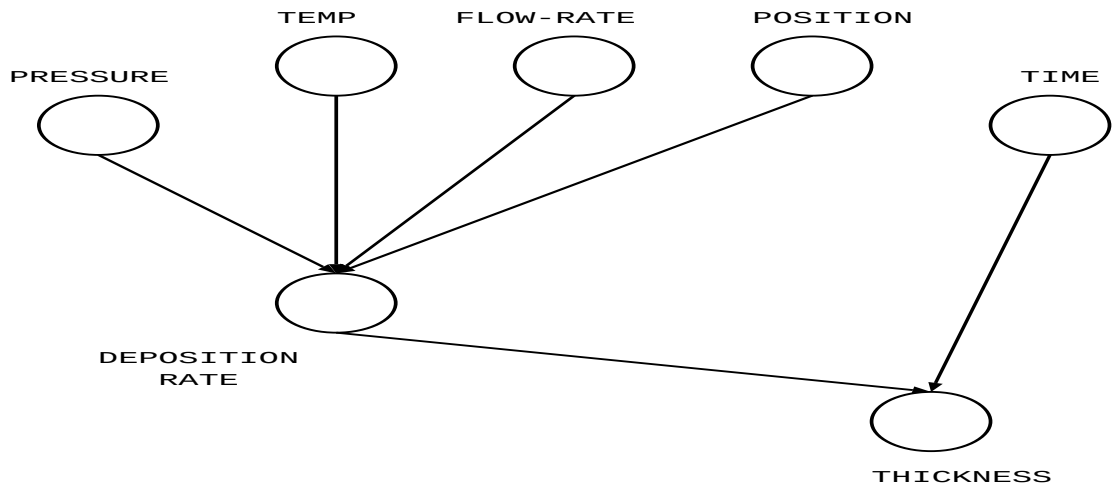

Figure 7. Bayesian network for LPCVD of undoped polysilicon [29]

Hence, a Bayesian graph for the LPCVD process is constructed which shows the dependence of deposition rate on pressure, temperature, flow-rate, position and time. The deposition rate in turn affects the film thickness. The above dependencies are encoded by the links as shown in the figure. Belief networks are chosen to depict such processes since they are able to create a more accurate model with lesser information than other models. Moreover, the inclusion of uncertain and insignificant influences does not degrade the performance of the network. For example, the LPCVD model can be made to include the effects of silane flow rate and the position of the wafer, without any performance degradation, even though these do not have a pronounced effect on the process. Here, again, parallelization could help in speeding up computation since microscopic changes in the deposition rate can corrupt the chip

and render the entire expensive manufacturing process useless. Hence, the decisions required to control the causal parameters should be made quickly.

2.3.4. Information Retrieval

Bayesian Networks have been applied in a multitude of computing tasks, such as information retrieval or IR. IR systems are very popular in electronic systems as an important means of exchanging information. On-line books, papers, news and services are facilities that need information retrieval.

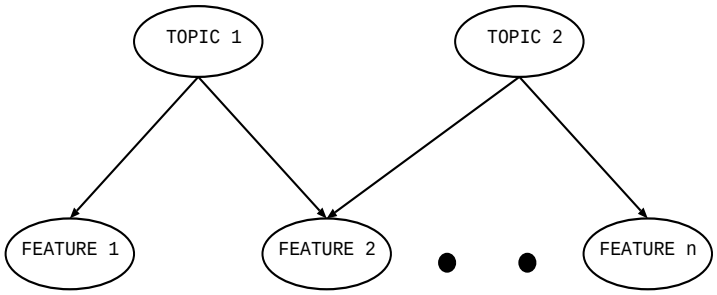

Figure 8. Information Retrieval for different topics using overlapping features [30]

According to [30], retrieving any information involves recognizing it on the basis of specific representations where the features that are characteristic to the desired topic are used to extract the topic. This is illustrated in Figure 8. Some of these features may overlap as shown in the figure. It is similar to searching on the basis of keywords, but requires additional conclusions based on the possibility of the existence of phrases from the request in the relevant document. Bayesian networks are therefore harnessed for handling these probabilities by including uncertainty and inference rules.

Information retrieval based on belief consists of the following steps:

a) Construct the network according to the nature of the request.

b) Extract the features from the document.

c) Provide evidence to those features in the network.

d) Calculate the probability of the feature's relevance to the document.

e) Rank the documents according to the computed probabilities.

The above method can also be used for topics with overlapping features.

Information Retrieval can be further applied to various real-world applications, such as vision. A Perceptual Inference Network or PIN, shown in Figure 9, is used by [31] to infer geometric structures. Each block becomes a node in the Bayesian tree. Also, the entire tree can be hung from any pivotal node, for allotting numbers and levels. In Figure 9, the "closed token" node has been selected as the pivotal node and numbered 0. All other numbers represent the numbers allotted to the rest of the nodes after hanging the tree from the pivot or root. Basic features such as parallel lines, contours or strands act as evidence nodes, in the belief network and circles, ellipses, parallelograms or triangles act as the hypothesis nodes.

2.3.5. General Systems

Debugging in software [32] and troubleshooting in all other general systems [33] are essential for maintenance and repair. Software errors can be handled by analysis of the operating system to create a graph of likelihoods of any feasible execution path being erroneous. Similarly, the cause of these errors and the alternate paths that the system can take also have a probabilistic attribute associated with them and, therefore, can be best handled by Bayesian Networks.

Troubleshooting is an integral part of most systems and some of the most recent products such as Microsoft Windows 95 employ Bayesian Networks for the task as described in [34] and [35]. It can also be applied to day-to-day tasks such as finding out the fault in a car engine that does not start [33]. In such a case, causes such as alternator and fan belt will affect the amount of charge delivered, which will in turn

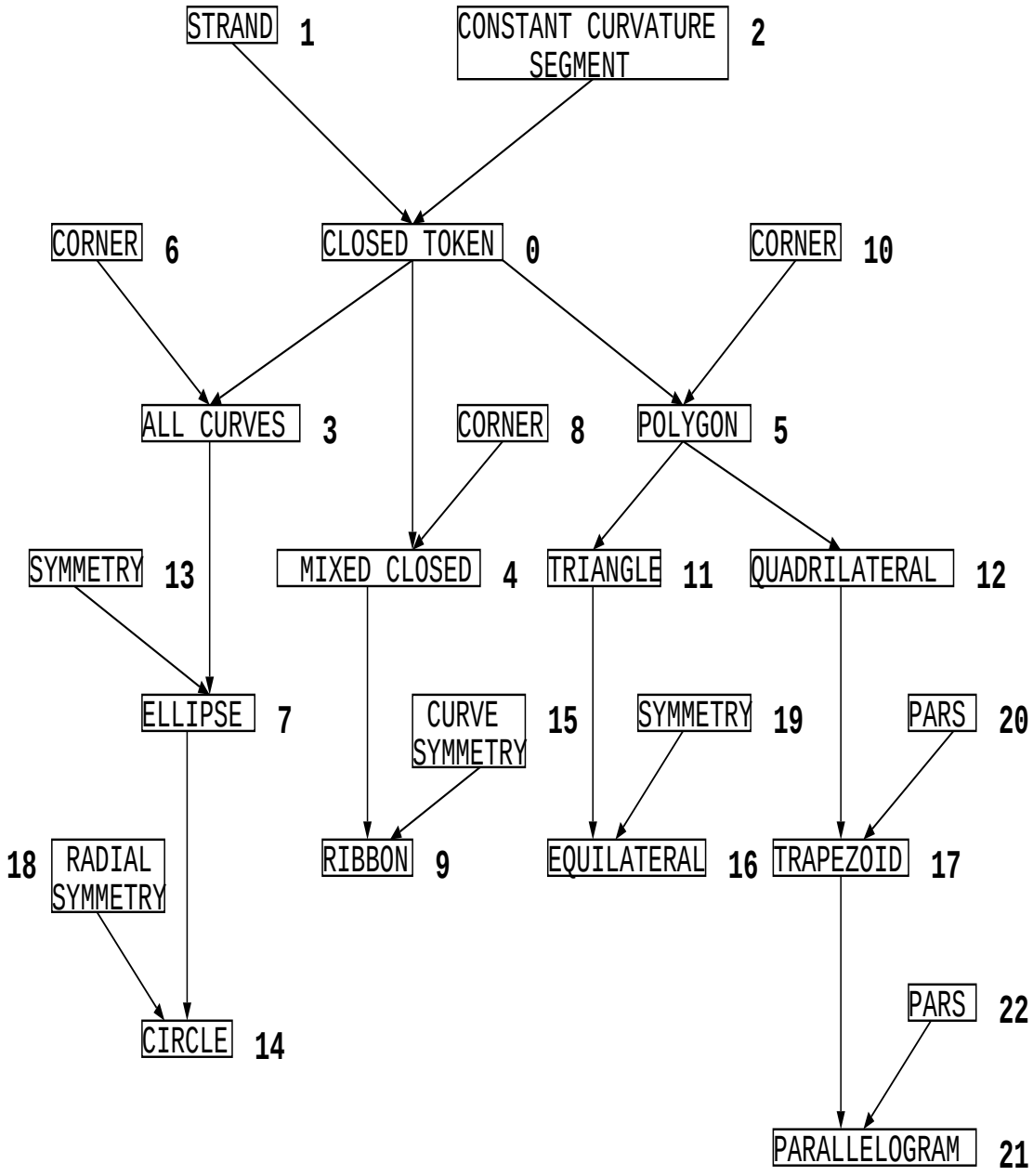

Figure 9. PIN used in visual process management [31]

affect the battery power. The power is responsible for a number of other factors such as the gas gauge and engine turn over, all of which can be causes of the stalling of the engine.

The above serve as examples for Bayesian networks applications. They also indicate how belief networks can be used for backward and forward projections or just determining the state of the probabilistic environment.

CHAPTER 3

BAYESIAN NETWORK UPDATING

The parallel implementation of probabilistic networks is divided into two interlocking phases: (i) *communication*, or the message passing from one node to another to convey the new lambda and pi values of a child or a parent respectively and (ii) *computation*, or the calculation and updation of existing belief, lambda and pi values on receiving a new message. Both phases alternate till all the nodes are updated and the entire network attains steady state again.

In subsequent sections, we will see how these two phases are implemented on a parallel machine without losing the overall integrity of the results obtained.

3.1. Partitioning and separability

The Decomposition Theorem stated in [15] proves the separability of a *singly-connected* Bayesian graph. This concept is similar to the one described in [36] and [37], which became the underlying principle for the HUGIN software. A graph is "singly-connected" if each link in the graph has the property of separating the graph into two separate and independent parts on being removed. In Figure 10, we can see that A and B are two such parts of the same graph, joined by a single link. The evidences have also been separated into e_A and e_B. Communication between the two parts is through two types of messages, called the *lambda* message and the *pi* message. A pi message is the message sent from a parent node to its child node and is defined as

$$\pi_Y(x) \;=\; P\{x, e_A\}$$

A lambda message is sent from the child to its parent. It can be defined as

$\lambda_Y(x) \;=\; P\{e_B|A, e_A, x\} \;=\; P\{e_B|x\}$

Hence, we can see that, given X, B is conditionally independent of A and e_A, by the following derivation.

$P\{x, e\} \;=\; P\{x, e_A, e_B\} \;=\; P\{e_B|x, e_A\}P\{x, e_A\} \;=\; \lambda_Y(x)\pi_Y(x)$

Therefore, if the state and the conditional probabilities for the parent node in a singly-connected Bayesian tree are known, we can determine the same for the child without requiring any other information. The statistical properties of such a belief network are, thus, distributable. *This is very important because it guarantees that the Bayesian network has inherent properties that makes it possible to render them parallel.*

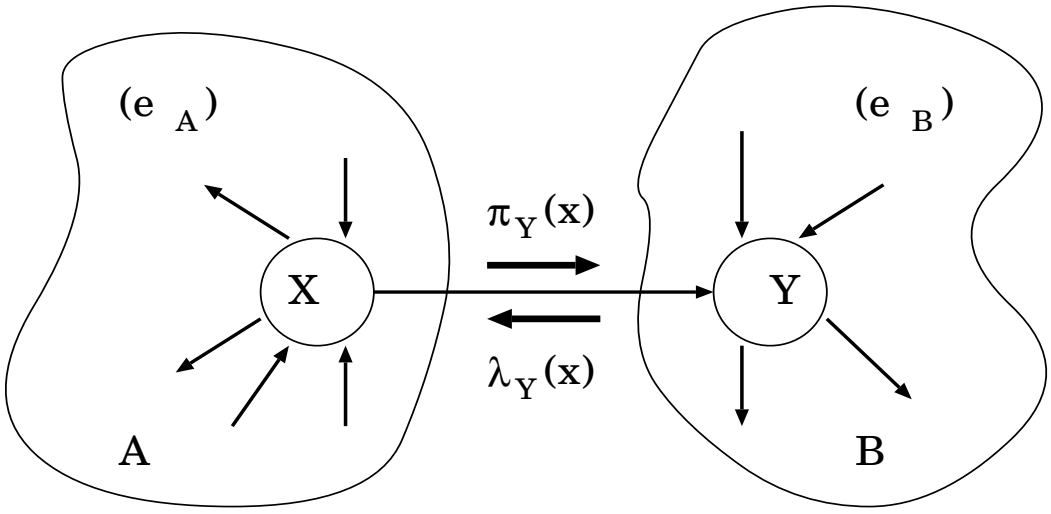

Figure 10. Decomposition theorem conditions between 2 singly connected parts A and B [15]

3.2. Bayesian computations

In the last section, we saw how we can compute the conditional probabilities of any node depending on that of its Bayesian parents. In this section, we will see the types of computations that are actually required to obtain external information and update the entire network accordingly.

3.2.1. Network requirements

In a *directed acyclic graph* (DAG), there can be no clash of hierarchy, for there cannot be a case where the child of any node is its own ancestor. Hence, we can simply propagate all information by sending lambda and pi messages, in two straight passes, *bottom-up* and *top-down* respectively.

A causal tree may be such that the nodes in it have multiple Bayesian parents and multiple Bayesian children. However, for simplicity, we shall consider the case, shown in Figure 11, which is similar to that in [1] where a node has several children but only one parent. The computations can then be extended to accommodate multiple parents.

3.2.2. Belief updation

Let evidence e be divided into two sets of evidences, e_X^- being the set of evidences for the sub-tree with its root at X and e_X^+ being the set of evidences for the rest of the network. Then, $e = e_X^- \cup e_X^+$. Now, for obtaining the *belief* of X, we have

$$\begin{aligned} BEL(x) &= P(x|e_X^+, e_X^-) \\ &= \alpha P(e_X^-|e_X^+, x)P(x|e_X^+) \\ &= \alpha P(e_X^-|x)P(x|e_X^+). \end{aligned}$$

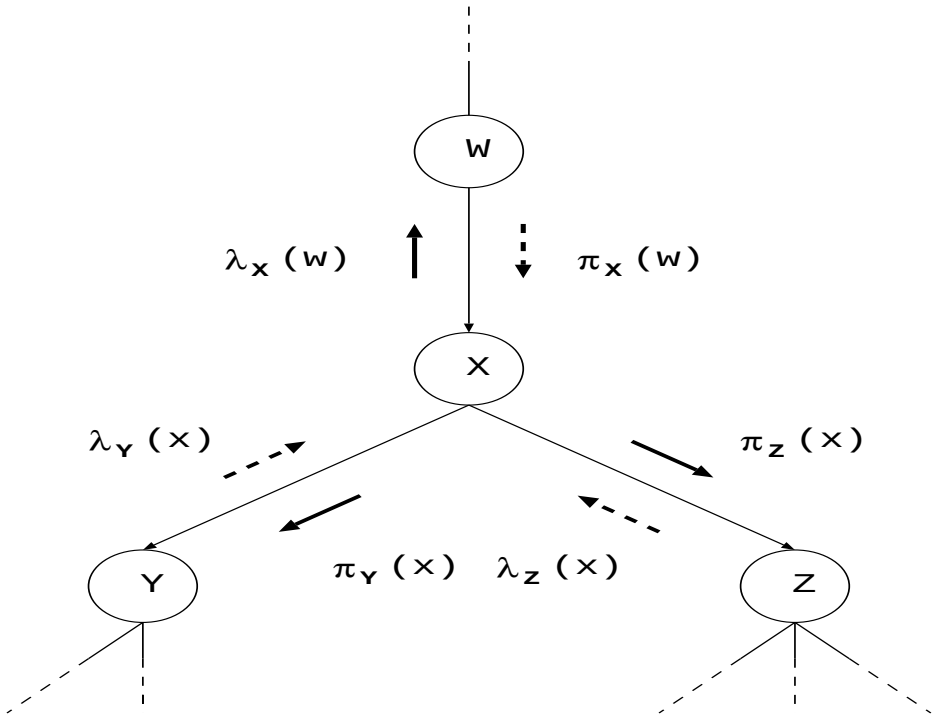

Figure 11. Status of node X, with parent W and children Y and Z, in an inference network

Here, $\alpha = [P(e_X^-, e_X^+)]^{-1}$, and is a constant used for normalizing. By generalizing, we obtain that $P(x|e) = \alpha P(e|x)P(x)$.

3.2.3. Lambda message

The lambda message that node X receives is

$$\begin{aligned} \lambda(x) &= P(e_X^-|x) \\ &= P(e_Y^-, e_Z^-|x) \\ &= P(e_Y^-|x)P(e_Z^-|x). \end{aligned}$$

If we consider $\lambda_Y(x) = P(e_Y^-|x)$ and $\lambda_Z(x) = P(e_Z^-|x)$, then

$\lambda(x) = \lambda_Y(x)\lambda_Z(x)$.

If X is an instantiated node, then it is assumed to have received a lambda message of 1 from an imaginery child on receiving evidence, else it is assumed to have received a lambda message of 0.

3.2.4. Pi message

The pi message that node X receives can be framed as

$$\begin{aligned}\pi(x) &= P(x|e_X^+) \\ &= \sum_w P(x|e_X^+, w)P(w|e_X^+) \\ &= \sum_w P(x|w)P(w|e_X^+).\end{aligned}$$

If $\sum_w P(x|w)$ is stored as a matrix $M_{x|u}$, then the message sent to X from W is just

$\pi_X(w) = P(w|e_X^+)$ giving
$\pi(x) = M_{x|u}pi_X(w)$.

3.2.5. Propagation technique

The *belief* of node X can be defined as

$BEL(x) = \alpha\lambda_Y(x)\lambda_Z(x)\sum_w P(x|w)pi_X(w)$.

This is the updated value obtained at node X after all the messages have been processed. The *belief* for each of the other nodes can be similarly computed.

The general technique, therefore, is to update those nodes that receive direct evidence, and then to update the rest of the network in two passes. The first pass is the *bottom-up* pass, which allows all information to reach the root and merges all new information into a common set of knowledge. The second or the *top-down* pass then carries this common information down from the root to every node of the tree. Both passes consist of a mixture of lambda and pi messages and, since the sense of one pass is exactly the reverse of another, the links that are included in both passes become pathways for both lambda and pi messages, one in each pass.

3.3. Pass reduction

Even though a node in a Bayesian tree can have multiple parents, the tree can be converted into a single-parent, singly-connected tree by selecting a pivot node and "hanging" the entire tree from the pivot node, which then becomes the root. This was described in the previous chapter. The choice of the pivot depends on the user and is usually based on balancing criteria. Thus, a node that is almost in the centre of the largest diameter of the tree would be the choice for the pivot. This reduces tree-updating time, since it ensures that no branch of the tree is unusually longer than the others. On the basis of the structural parent of a node and its Bayesian relation with the structural parent, specified by the user, the tree can be generated in $O(n)$ time, where n is the number of nodes in the Bayesian tree. Since it creates the tree by attaching node by node and deciding the sense of the link to its parent, it takes only $2n$ steps, to be precise. *This is a considerable improvement over Pearl's method that takes $O(n\ log\ n)$ time to construct the entire Bayesian tree.*

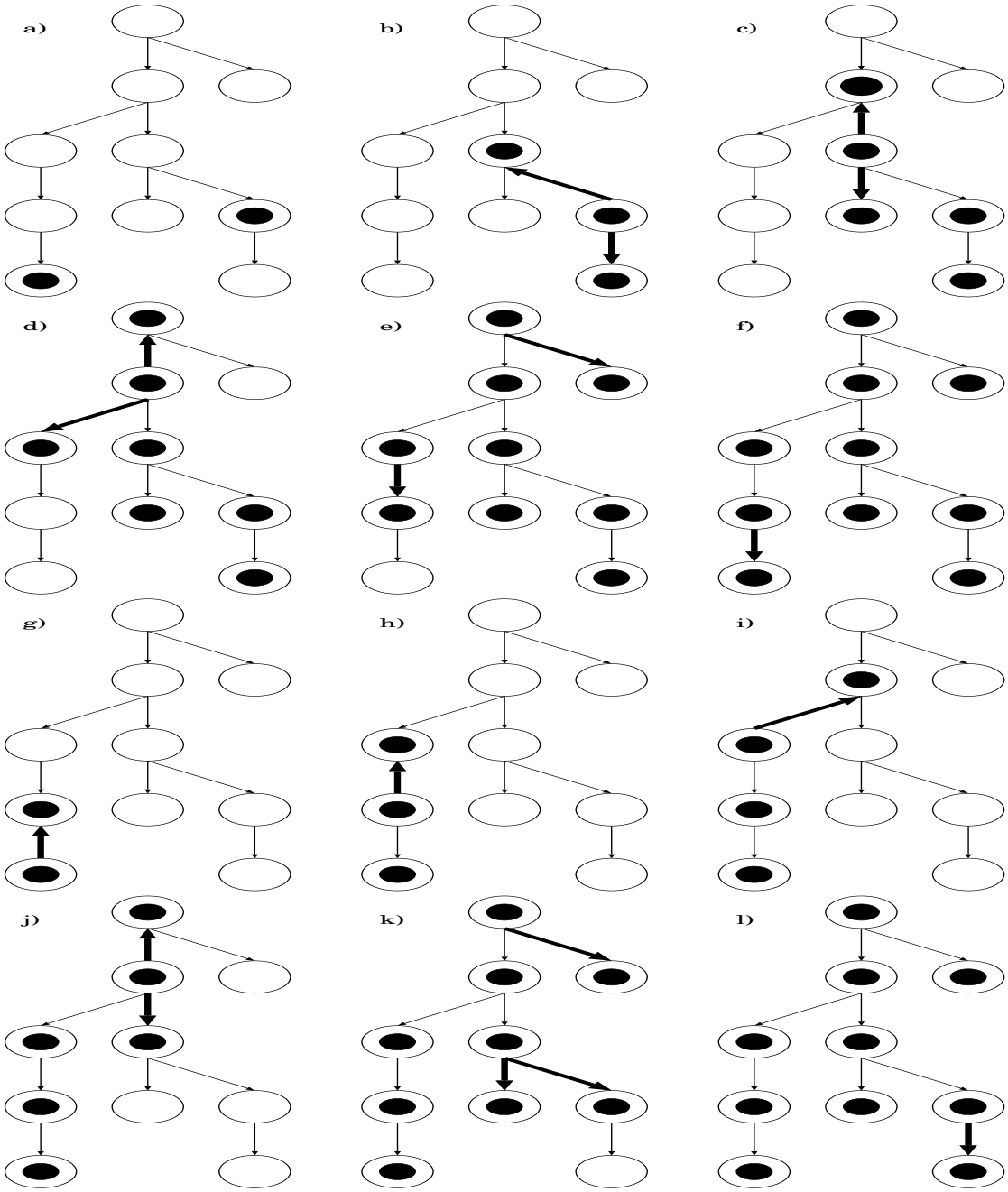

Figure 12. Polytree algorithm : separate phases of network updation for each evidence

3.3.1. Polytree Algorithm

The Polytree algorithm for updating a tree, through propagation of information, was that suggested by Kim [38] and Pearl [39]. In this method, once a node received a message, it updated its own belief and then spread the new information simultaneously to all its neighbours, in all directions, irrespective of the hierarchy of the neighbour in the tree. This is shown in Figure 12. The filled ovals represent updated nodes and the thick arrows represent the current flow of information. Therefore, to avoid clashes, this process had to be made sequential and each evidence was handled separately. Hence, each node in a tree had to be visited once for every evidence.

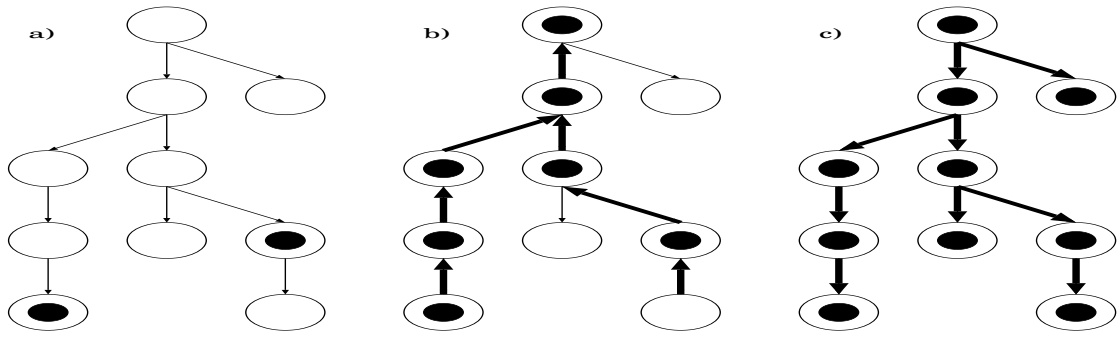

Figure 13. Revised Polytree algorithm : a) Direct evidences, b) Up pass, c) Down pass

3.3.2. Revised Polytree Algorithm

According to Peot and Shachter's revised method [15], the entire process of updating the network can be completed in just two passes, one of which is downwards and one upwards. When a node receives an evidence, it communicates the same to the root, via the link through its parent. In this way, multiple evidences are handled

together and the combined information from different nodes reaches the root togather. This eliminates the need for separately propagating the effect of each evidence. Once the message reaches the root, the down pass begins in which all the nodes of the tree are updated starting from the root and travelling down to the leaves. A "zeroth pass" can sometimes be included, which is a downward pass, originating at the pivot or the root, requesting all nodes for updated messages.

This is a revised form of the Polytree algorithm of simultaneous message propagation in all directions of [38] and [39]. Here, since in the upward pass, all messages have the same *equi-sense*, that is, they have the same direction of propagation, multiple evidences and observations can be combined in the same pass. The two passes are depicted in Figure 13. The first part shows the evidence nodes, the second shows the state of the tree after the upward pass and the third shows the state after the downward pass. In Figures 12 and 13, only the structural hierarchy has been shown and the Bayesian link directions are not specified since the passes are independent of the Bayesian relations and depend only on the topological arrangement of nodes. *Therefore, multiple evidences can be introduced and processed simultaneously and each node in the tree needs to be visited only twice, regardless of the number of evidences.*

CHAPTER 4

MAPPING AND IMPLEMENTATION

For implementing Bayesian Networks on parallel machines, the mapping scheme not only has to be efficient with respect to task scheduling and load balancing but also with respect to the inter-node communication problems required in the network. An intelligent mapping is necessary since each node in the parallel machine has to be actively involved in message passing in order to update other nodes and spread new information throughout the network whenever there is any change in probabilities or states anywhere in the entire Bayesian setup. Therefore, the mapping should be such as to minimize the distance that each message has to traverse before it reaches its destination. Load balancing is desirable to maintain uniform utilization of each processor and thereby speedup the computation through uniform task allocation. Also, when the number of nodes in the Bayesian network is much larger than the number of processors in the parallel machine, the task allocation must still maintain all requirements for efficient processing.

The nodes A and B, in Figure 14, are Bayesian parents of node C. The nodes D, E and F are Bayesian children of the node C. The tree is hung from a pivot or root, as shown in Figure 15, so that each node has only 1 parent. Each node is then named in breadth first order and the levels are assigned. Hence, for any node, although the adjacency and neighbour information is preserved while transforming a Bayesian tree into a single-parent tree, the parents and children of the Bayesian tree may not necessarily correspond to the ones in the single-parent tree, requiring that this information be stored separately.

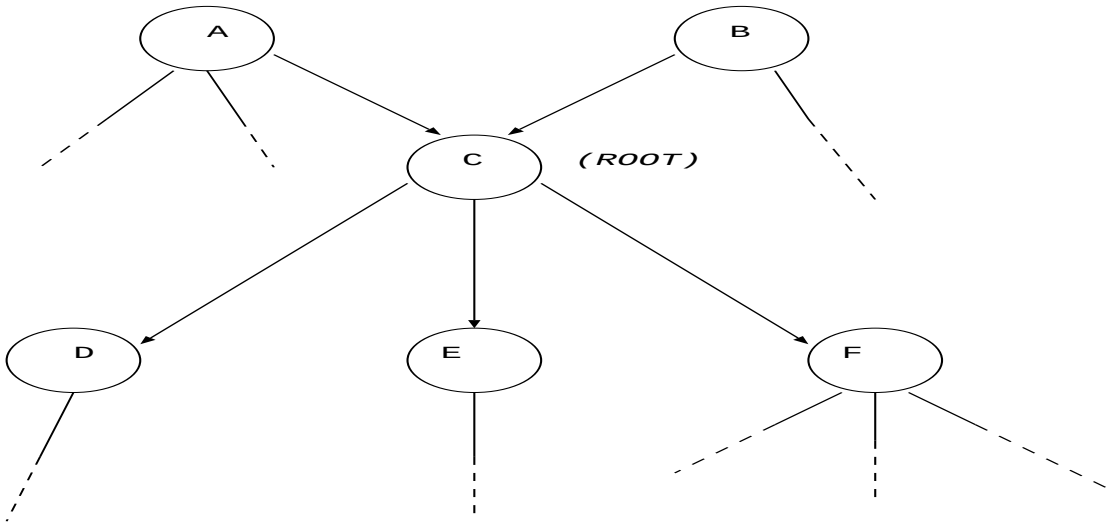

Figure 14. A Bayesian tree with multiple parents as in node C

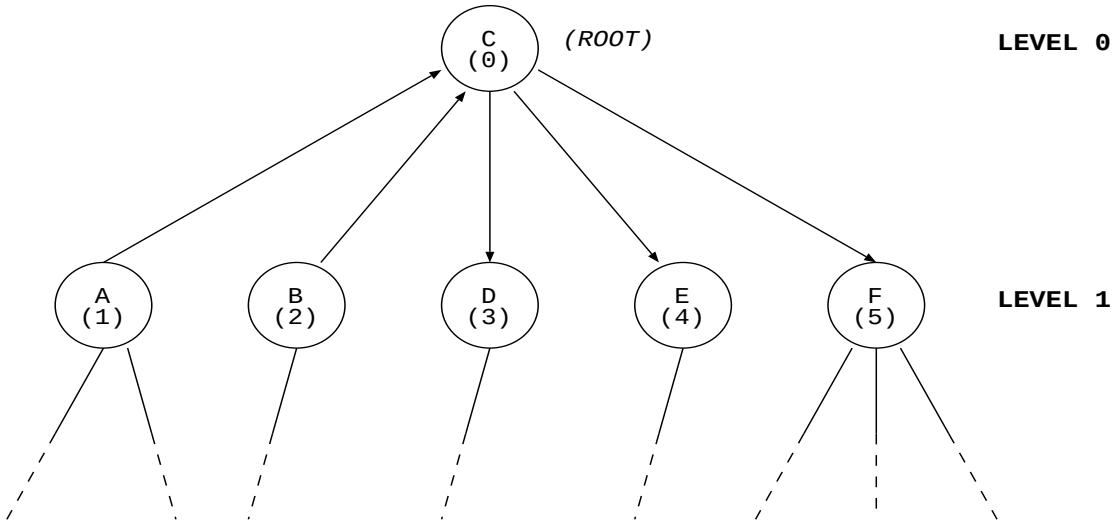

Figure 15. Conversion of the Bayesian tree to a levelled single-parent tree, with node C as the pivot

In the subsequent sections, we will see how a tree is mapped onto the processor. Since the mapping is independent of Bayesian relations and depends only on the topological order of the tree nodes, the figures in the following sections of this chapter show only the topological hierarchy and the link directions in each tree. In each figure, the small circles represent tree nodes, the numbers near them, that are not within brackets, represent the corresponding node number in the tree itself. The numbers within the square brackets give the binary encoding of the hypercube node in which the specific tree node is stored.

4.1. Breadth First Mapping

The most uniform task allocation scheme would be a breadth first mapping of the nodes of the Bayesian tree onto the nodes of the parallel machine, in sequence, as shown in Figure 16. This would ensure that the number of tree-nodes per processor differs at the most by one. Since in a probabilistic network, all the tree nodes are actively involved in updating and message passing, a breadth first allocation of nodes would also imply a fair division of storage and computation. However, this scheme would not support fast communication since multiple hops would be required for each message between one tree node and the next. It would, therefore, not only slow down the entire network but also put extra load on each processor, by involving more processors than required, for each instance of message passing. Most of these processors would then be busy just relaying messages.

4.2. Adjacent Parent-Child Mapping

To reduce communication time and the number of *relay processors*, it is essential to maintain parent-child adjacency when mapping the tree, as much as possible, on

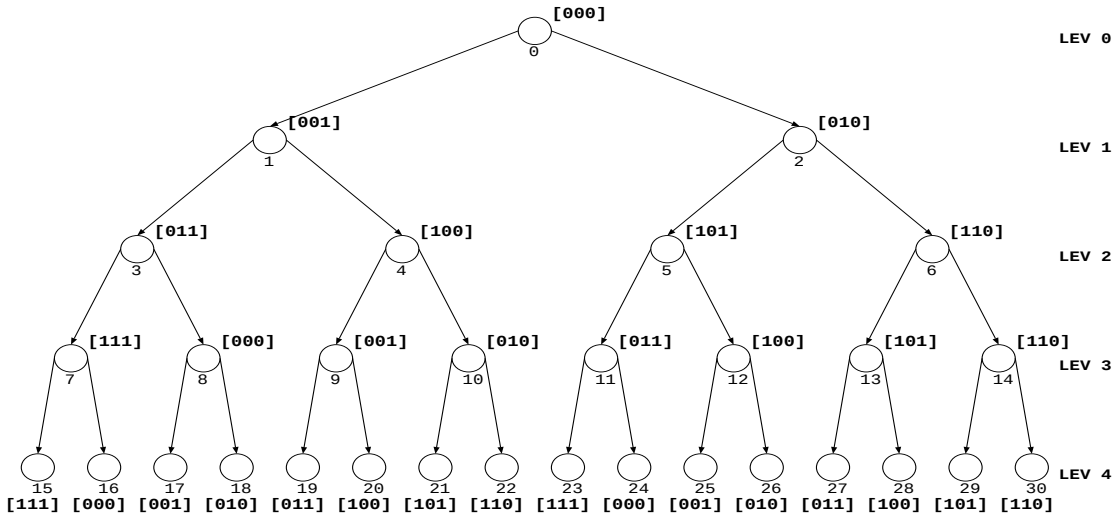

Figure 16. Breadth first mapping; allocated hypercube node numbers are shown within square brackets

to the parallel machine. For a hypercube, therefore, it would be best to map the tree such that the parent and children of any node are allocated to its immediate neighbours. A good way of achieving this would be to assign the root (*level* 0) of the tree always to *Node* 0 of the hypercube and then assign its children (*level* 1) to its neighbours in a *round robin* fashion. This means that if the number of children is less than the degree of the hypercube, then each neighbour stores at the most one child. The first child is mapped on to the neighbour which differs from the root in only its lowest bit. Subsequent children are then mapped in increasing order of *differing bit*, as in Figure 17. When the number of children exceeds the degree of the hypercube, then the rest of the children are *wrapped up* around the same neighbours, so each neighbouring processor stores more than one child node. Hence, if m is the degree of the hypercube and n be the number of children of a node, then each neighbouring processor stores a maximum of $n \ mod \ m$ and a minimum of $(n \ mod \ m) \ - \ 1$ child

nodes. This method is continued to map the rest of the levels in a similar fashion till the entire tree is mapped. The steps are illustrated in Figure 18.

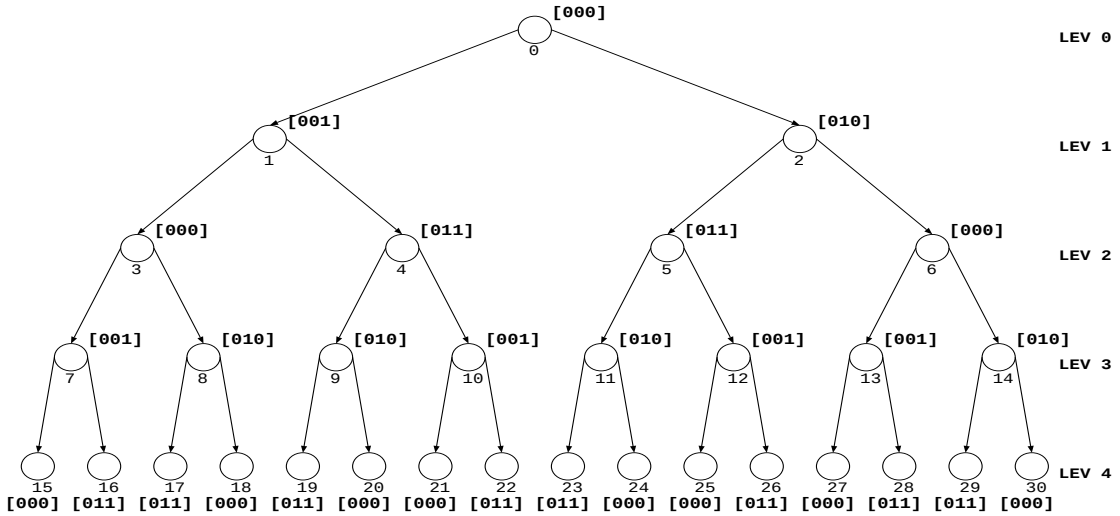

Figure 17. Adjacent parent-child mapping; allocated hypercube node numbers are shown within square brackets

The main advantage of this kind of mapping is that there would be no *relay processors* involved since each message would traverse from its source to destination via a single link. The scheme is simple and straight-forward and utilizes the structure of the hypercube to implement the communication pattern in a Bayesian network. However, in this approach, the *load balancing* may not be effective. There is a tendency among the tree-nodes to be mapped on to the first few nodes of the hypercube, due to the policy of mapping nodes on to neighbours obtained by varying one bit each time, starting with the LSB or the smallest bit. Hence, the first child of $Node$ 0 would be mapped on to $Node$ 1 and the first child of $Node$ 1 would again be mapped on to $Node$ 0 and so on. Ultimately, the higher number nodes of the hypercube would

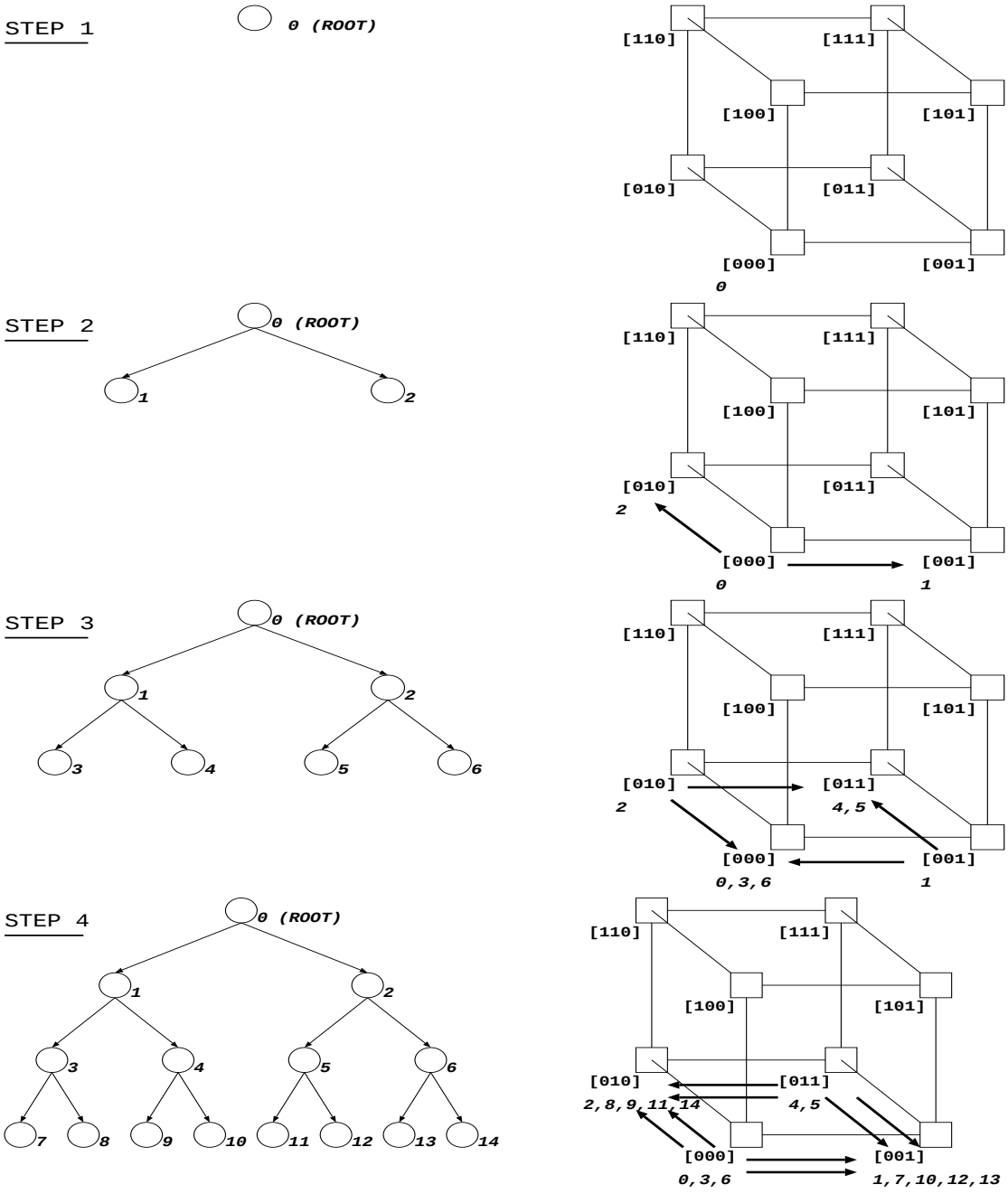

Figure 18. Step by step illustration of parent-child mapping

have very few tree-nodes mapped on to them, while the lower number nodes have an unfairly high share of computation and communication to do.

4.3. Adjacent Parent-Child Mapping with Load Balancing

To maintain *load balancing*, the scheme should be adjusted to for a uniform variation of tree nodes over hypercube nodes. One way would be to determine the next neighbour by varying bits alternately from the LSB or MSB at each level of the tree. Thus by alternating at each level, the upper part of the hypercube can be made to participate as much as the lower part. A more efficient way, though, would be to change the starting bit also in a *round robin* fashion at every level. This would mean that for the k-th level of a tree on an m degree hypercube, the starting bit would be the (k *mod* m)-th bit instead of always being either the LSB or the MSB. The two mapping methods are shown in Figures 19 and 20 respectively. Figure 21 illustrates load-balanced, round robin mapping and Figure 22 shows the status of a three-dimensional hypercube at the end of a binary tree mapped using adjacent parent-child mapping with round robin rotation.

4.3.1. Optimal load balancing

From the given figures, we can deduce two very important pieces of information. One is that there cannot be a situation where consecutive levels of a tree get mapped on to hypercube nodes belonging to the same group, except when the dimension is 0 as in sequential case. This follows from the fact that in any hypercube of dimension greater than 0, there are two neat groups of equal number of nodes, each of which has all its neighbour-nodes in the other group. The second is that half the nodes of the hypercube will be repeated almost r times as frequently as the other half, in the round robin method, r being the average number of children of a node. In the case of a

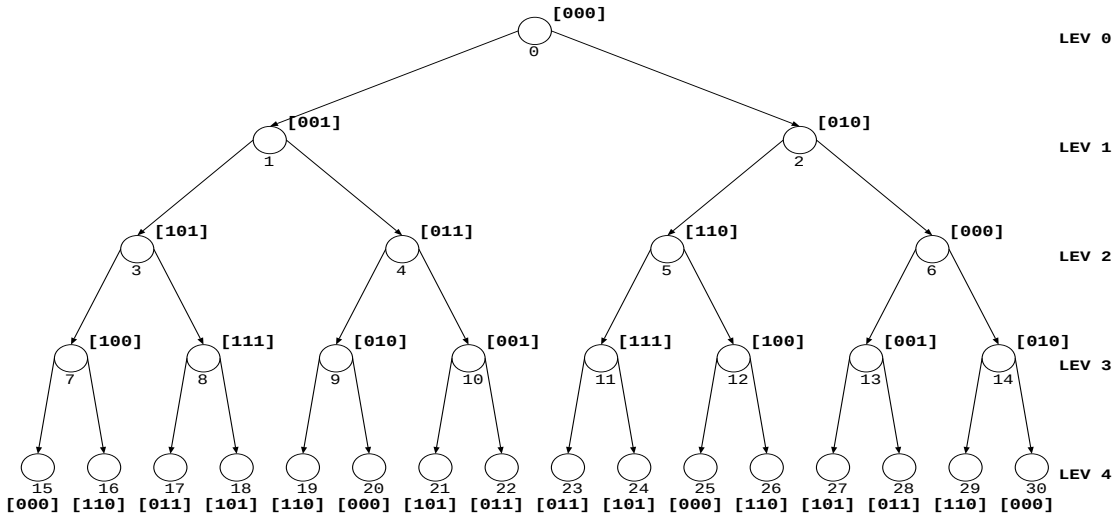

Figure 19. Adjacent parent-child mapping with alternate bit reversal; allocated hypercube node numbers are shown within square brackets

complete tree, say tertiary, each node has 3 children, i.e., r is 3. The number of nodes occuring at each level l would therefore be 3^l. Hence, the frequency of occurance of an odd level node is $1 + 3 + 3^3 + 3^5 + 3^7$ ... and so on, which is approximately $3(1 + 3^2 + 3^4 + 3^6 ...)$. Similarly, the frequency of occurance of an even level node is $1 + 1 + 3^2 + 3^4 + 3^6$ ... and so on. Hence, one will be 3 times the other depending on whether there is an odd number of levels in the tree or even.

Now we can derive a superior scheme by incorporating the advantages of above schemes, that is, the *alternate bit reversal* scheme and the *round robin* scheme.

Here, we can see that the chances of repitition are lesser in the *round robin* scheme, shown in Figure 23. This is so because in the *alternate bit reversal* scheme, shown in Figure 24, changing the end bits often leads to a loop of the same nodes. For example, nodes 0 and 5 lead to nodes 1 and 4 on changing the end bits (LSB or MSB). Nodes 1 and 4, in turn, lead back to nodes 0 and 5 on changing the end bits.

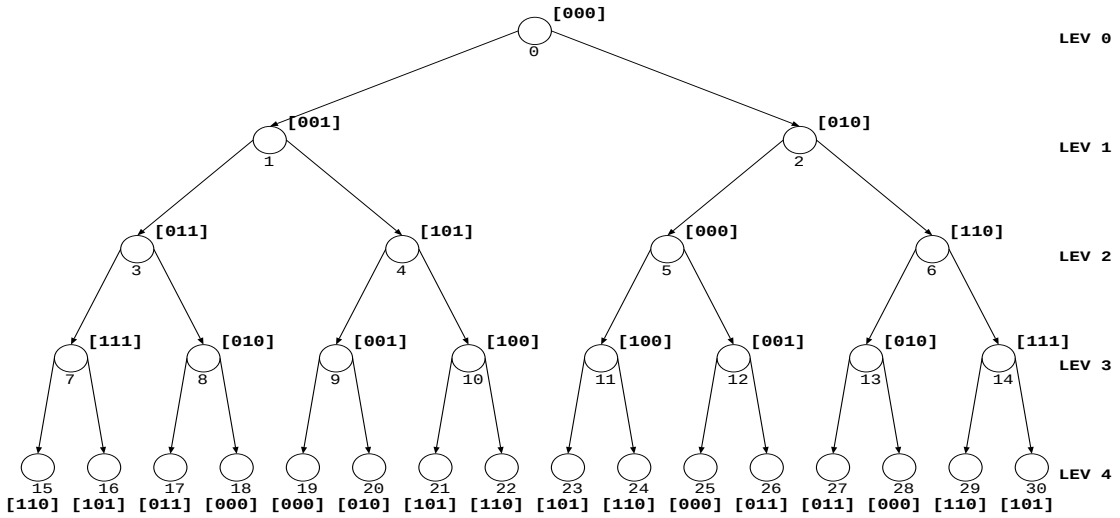

Figure 20. Adjacent parent-child mapping with round robin rotation; allocated hypercube node numbers are shown within square brackets

The same happens with nodes 3,6 and 2,7 too.

Hence, the last mapping scheme of round robin rotation at each level results in an optimal combination of load balancing and communication. It is therefore the most suitable for mapping Bayesian Networks onto hypercubes.

4.3.2. Basic algorithms for mapping

For mapping the tree nodes onto the hypercube nodes, we first need to have an algorithm for assigning levels to the tree. This is due to the fact that for round robin allocation over height, we need to change the order of allocation at each level. The tree levelling algorithm is simple and needs just the parent-name of each node to assign levels. It is as follows.

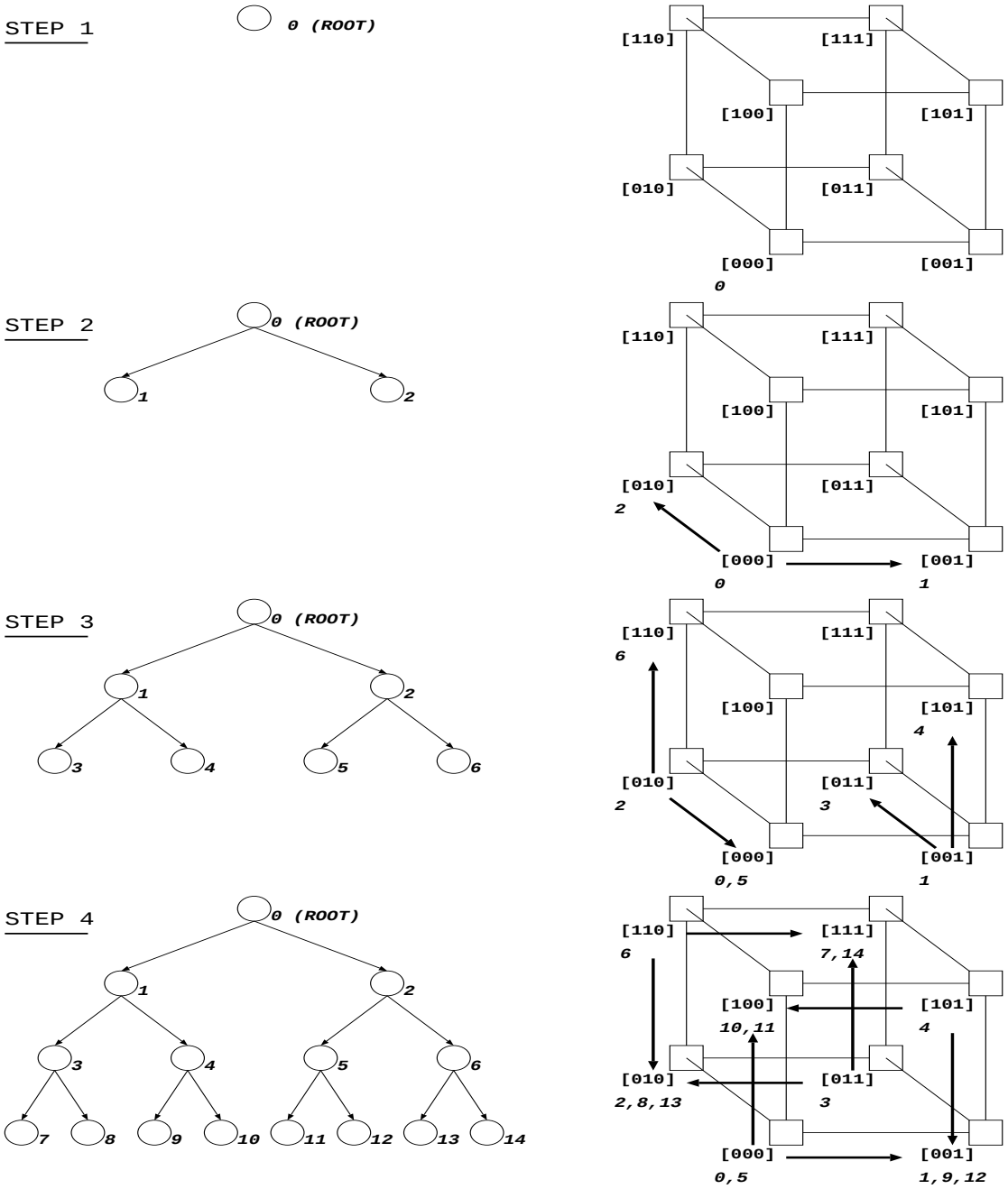

Figure 21. Step by step illustration of load balanced parent-child mapping

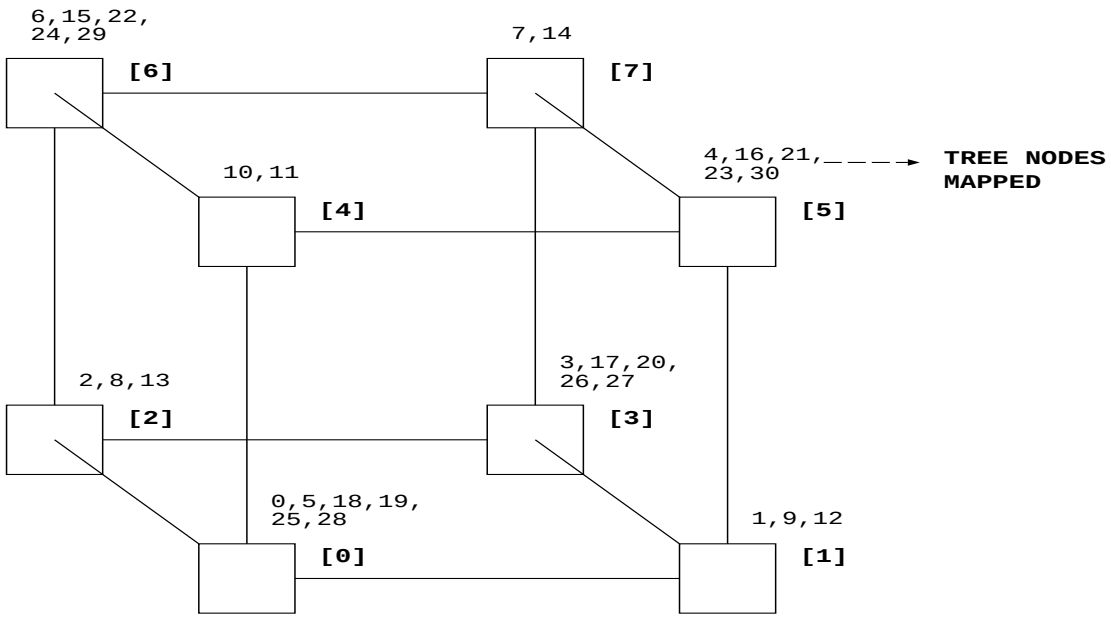

Figure 22. Hypercube showing round robin mapping of a complete binary tree of height 4

ALGORITHM : ASSIGN LEVEL

Begin

root.level = 0;

for (node = first to last) do

begin

node.level = node.parent.level + 1;

end;

End.

The mapping algorithm requires 2 masks; *WidthMask* is responsible for the round robin rotation among the children of a node and *HeightMask* does the same for

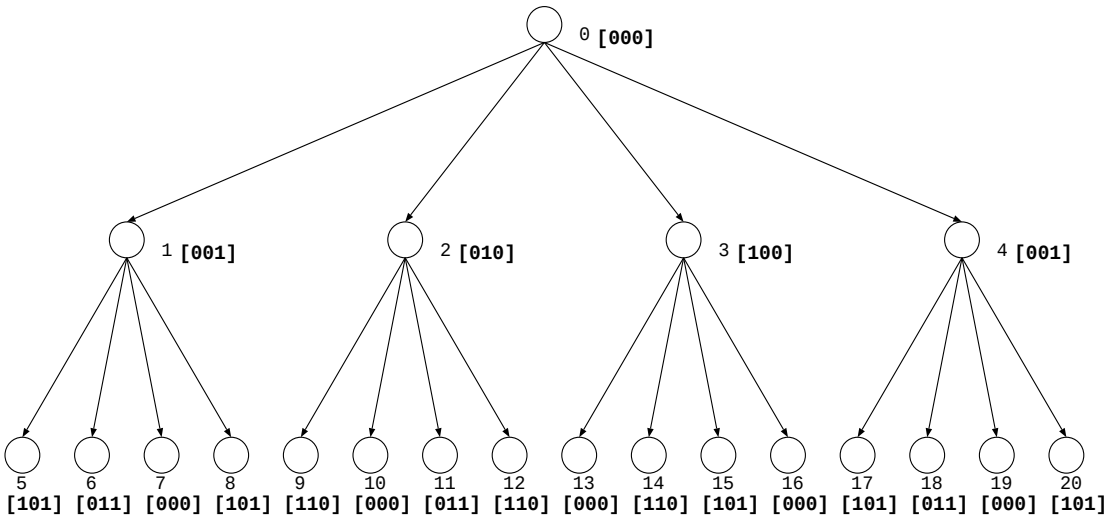

Figure 23. Alternate bit reversal in a quarternary tree; allocated hypercube node numbers are shown within square brackets

levels. The variable *hnode* corresponds to the hypercube-node number that will store the specific tree-node. Following is the algorithm for load balanced Bayesian network mapping.

ALGORITHM : LOAD BALANCED MAP

```
Begin

    root.hnode = 0;
    HeightMask = 1;
    for (ParentNode = first to last) do
    begin

        WidthMask = HeightMask;
        for (node = first to last) do
        begin
```

```
            if (node.parent == ParentNode)
            begin
               node.hnode = (ParentNode->hnode) EXOR
               (WidthMask);
               LeftRotate (WidthMask);
            end;
         end;
         LeftRotate (HeightMask);
      end;
   End.
```

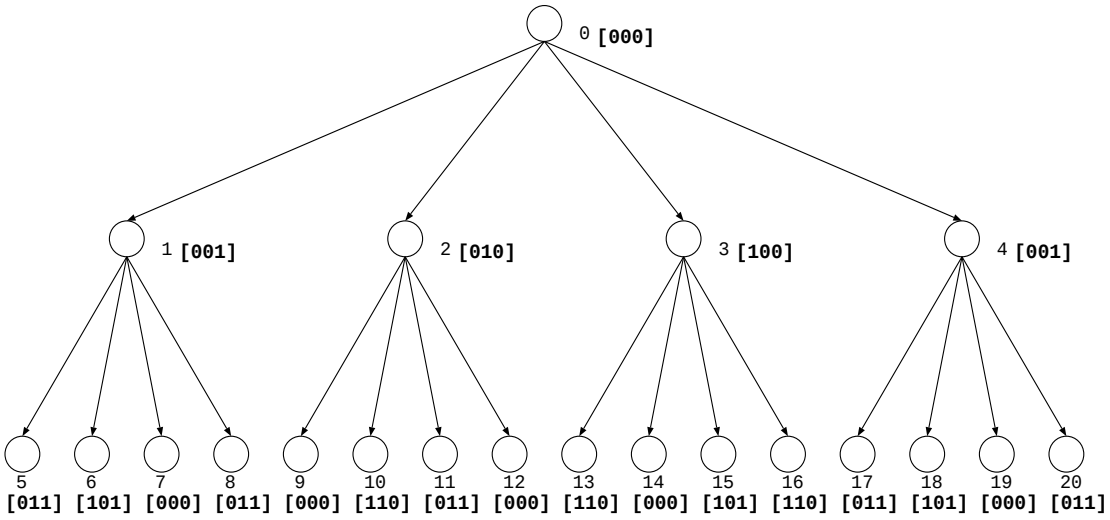

Figure 24. Round robin rotation in a quarternary tree; allocated hypercube node numbers are shown within square brackets

The host uses the above algorithms to assigns levels to the tree, initialize the network and then map on each node of the tree to a node of the parallel machine.

This is done in the beginning, prior to the parallel Bayesian computation and communication. The mapping, therefore, is static mapping and needs to be done only once for any network, making it easy to handle large probabilistic networks without any major task allocation overheads.

4.4. Sequence ordering and deadlock prevention

In a typical message passing application, one of the greatest problems is that of deadlock. Moreover, since all the processors work in parallel, it is also very important to preserve the sequence in which information travels. An out-of-order message can lead to erroneous computation and propagation of errors. At the same time, since messages queue up in buffers, a message that is required first, say A, may arrive after another that is required later, say B, in a parallel environment. This results in a deadlock since the processor keeps waiting for the message it requires first, which is A, which cannot be read because it is behind message B. At the same time, message B is not read because it is required later and cannot be used out of sequence. A cycle is thus created and the entire process stalls.

There is another case that can lead to a deadlock. This can occur during the upward pass when a processor keeps waiting for a child to send a message to it, even though that child has no message to send. It happens for the nodes that do not lie on the shortest path between the root and the instantiated nodes that have received direct evidence. A good way to avoid it is to keep a *change flag* that indicates if a node is expected to change its *belief* in the bottom-up pass or not. All nodes initially have their flags set to FALSE. The nodes that receive direct evidence turn their *change flag* TRUE. Those that have not received evidence but are leaf nodes mark their flags FALSE. The rest of the nodes wait for "change messages" from their child nodes and turn their flags TRUE if any one of the child nodes has its flag marked as TRUE.

Deadlock, due to cycle formation, is prevented with the help of a *receive flag*. To begin with, the *receive flag* of every processor is FALSE. In the bottom-up pass, a processor sets its *receive flag* TRUE only when it has received messages from all its topological children that indicate change. Only after the flag is TRUE can the processor in turn send a message to its topological parent, after which it turns the flag FALSE again. Similarly, in top-down pass, each processor sets the *receive flag* TRUE when it receives a message from its parent and switches it to FALSE again only after sending a message to all its children. This method also ensures the desired sequence of information flow.

4.5. Processor organization

Storing and implementing Bayesian Networks requires dealing with three main properties of the system; (i) probabilistic properties, (ii) structural properties and (iii) communication properties. The structural properties are few and can be defined by stating the following attributes for each node of the Bayesian graph :

a) Name of the node,

b) Total number and list of names of Bayesian Parents of the node,

c) Total number and list of names of Bayesian Children of the node.

Probabilistic properties consist of the following, for each vertex :

a) Number of possible states that can be attained,

b) Belief value,

c) Pi value,

d) Lambda value,

e) Last Pi message from each parent,

f) Last Lambda message from each child,

g) Conditional properties corresponding to each state of each parent.

Only a single communication property is required, which is required for proper sequencing of message passing :

a) Receive flag.

These properties are stored as an entire structure in the processor memory. Each processor stores the above properties for the tree nodes allocated to it. The allocation can be either done by a separate host or by the nodes themselves. In the latter case, each node of the hypercube separately decides which nodes it should store according to the rules specified to it.

The general structure of a single processor is shown in Figure 25. This is similar to the structure suggested by Pearl in [39] with an additional block, i.e., the "change" block added. $\lambda_B(A)$ is the lambda message sent from node B to its parent node A. $\pi_B(A)$ is the pi message sent to node B from its parent node A. $\lambda_1(B),\ \lambda_2(B), ...$ are lambda messages that node B receives from its children and $\pi_1(B),\ \pi_2(B), ...$ are pi messages that it sends to its children. $BEL(B)$ is the belief value of B. M is the message matrix and M^T is its transpose. All the lambda messages coming from the child nodes are multiplied together and the product is multiplied with the existing message matrix, made up of the last lambda messages. The result is then forwarded to the parent node as a lambda message. Pi messages are processed in a slightly different way. The pi message coming from a parent node is multiplied with the transpose of the message matrix, obtained from last pi messages. The result is then scaled by a constant and multiplied with the lambda value to yield *belief*. This belief value is then divided by the lambda value of each child and scaled to get the pi values to be sent to each child. The entire computation is done in accordance with the rules described above.

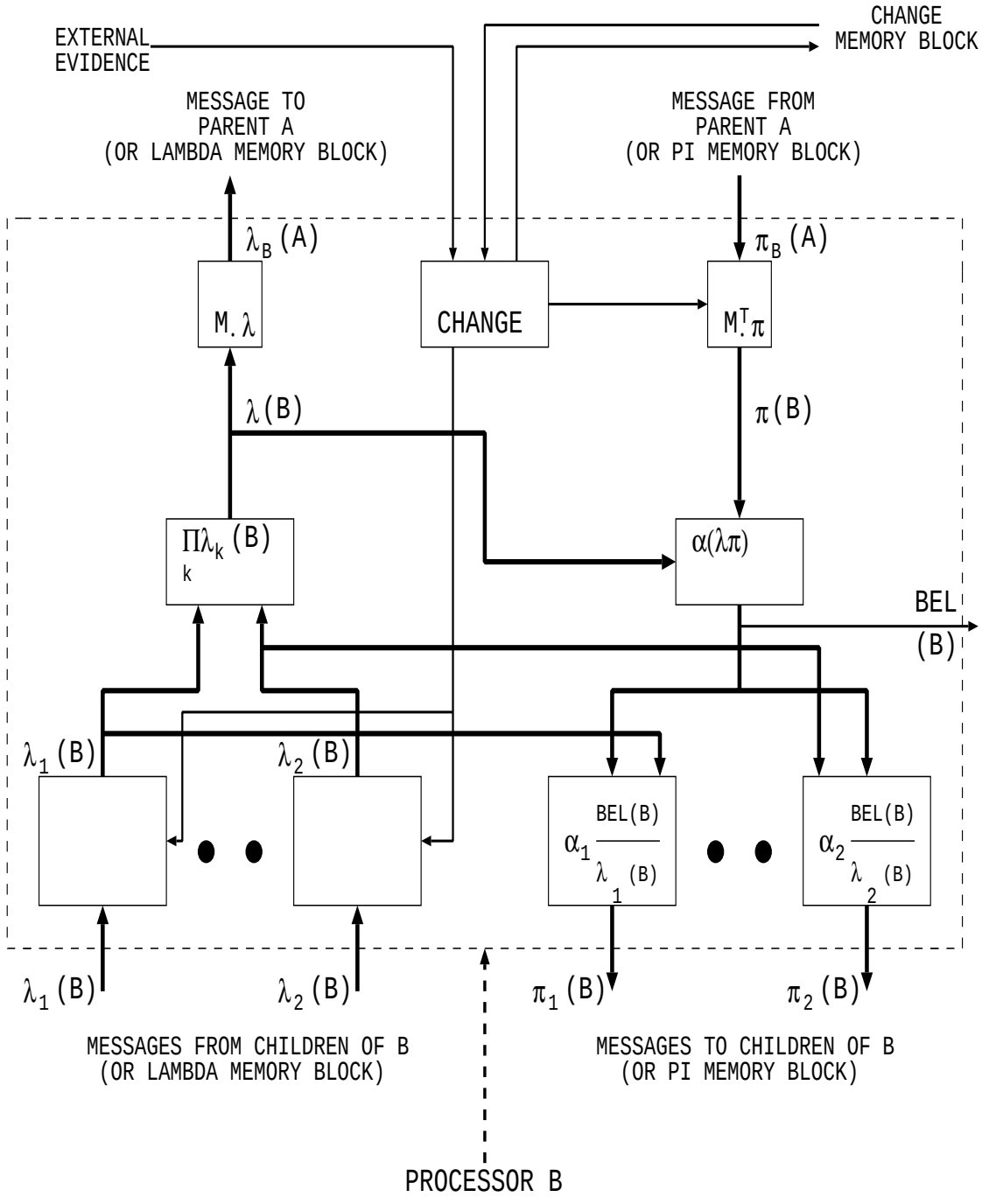

Figure 25. The structure of a single Bayesian node [39]

The function being performed by the processor depends on the category of the node. For example, a leaf node of a tree can be *instantiated* or *non-instantiated*, meaning, one that has received direct evidence and one that has not, respectively. There is a topological root one or more Bayesian roots for every tree. The entire network is initialized by assigning prior probabilities to these Bayesian roots, whereas the topological root is responsible for termination on the up-pass and initiation of the down-pass.

Apart from the above mentioned properties, a processor also needs to have knowledge of the levels allotted to the tree nodes, the names of the nodes that receive direct evidence and the tree nodes that lie on the shortest path between the evidence nodes and the root, as described above. Even though it is primarily concerned with the tree nodes that are allotted to itself, it keeps track of the processors that the other tree nodes are assigned to also, for purposes of specifying destinations during message passing. The data structure and size of the message buffer is also of primary importance since that determines the processing efficiency of incoming messages.

CHAPTER 5

RESULTS AND CONCLUSIONS

This chapter deals with performance evaluation of the proposed scheme on the basis of estimated and actual results. It then gives an overview of the contributions of the work along with comparisons with other related schemes and suggests points for future work.

5.1. Performance evaluation

The speedup obtained from the proposed scheme depends on a number of factors such as the number of Bayesian parents and Bayesian children of each node in the Bayesian tree, that is, the number of child nodes of the corresponding node in the single-parent tree, the height of the single-parent tree and the dimension of the hypercube. The first subsection gives an estimate of the expected speedup based on theoretical analysis. The second subsection gives the actual results obtained.

The speedup of very large trees could not be found from direct simulation due to machine problems and was therefore obtained by substituting the CPU timings obtained from smaller trees, for the individual operations. Message passing time for very large trees was also similarly estimated.

5.1.1. Analytical estimate of speedup

The expected speedup obtained by implementing the proposed scheme mainly depends on the distribution of tree nodes over hypercube nodes. The analysis of the same is as follows. Let n be the number of nodes in the Bayesian tree. Let the

average number of children for each tree node be k. Let the height of the Bayesian tree after being hung from its pivotal node be h and the dimension of the hypercube be d. Let n be the number of nodes in the tree. The child nodes of each tree node are distributed over its hypercube neighbours in round robin fashion. Since the total number of neighbours of any node in a d dimension hypercube id d, the distribution is restricted by k or d, whichever is minimum. Hence, for each tree node, its neighbouring processors will each store $\frac{k}{min(k,d)}$ nodes. In a single level, l, there are k^{l-1} clumps. Now, since there is a heightwise round robin distribution in the mapping too, the number of nodes that each processor stores goes down again by a factor of $min(h,d)$. Due to this reason, even though there are h levels, the repititions in each processor will only be $\frac{h}{min(h,d)}$. Also, in a hypercube, the set of processors in a hypercube can always be divided into 2 non-overlapping sets such that no adjacent processors belong to the same set. Hence, due to the adjacency and neighbourhood criteria, alternate levels are mapped onto separate halves of the hypercube, which increases the speedup by an additional factor of 2. The total speedup will therefore be

$$S \;=\; O\left((2 \times n)/(\tfrac{k}{min(k,d)} \times \textstyle\sum_0^h k^{l-1} \times \tfrac{h}{min(h,d)})\right).$$

Since, $n \;=\; \sum_0^h k^l$, we can express speedup as

$$S \;=\; O\left((2 \times \textstyle\sum_0^h k^l)/(\tfrac{k}{min(k,d)} \times \textstyle\sum_0^h k^{l-1} \times \tfrac{h}{min(h,d)})\right) \;=\; O\left(\tfrac{2 \times min(k,d) \times min(h,d)}{h}\right).$$

This speedup can also be expressed in terms of the number of processors, N by substituting d with $log_2 N$.

It is to be noted that the above speedup gives only an average estimate and not a lower or an upper bound, since the actual speedup for any tree may be greater or lesser depending on the number of children of each individual node and the way they are mapped on the different dimension cubes. Since the increase or decrease in

the number of evidences only adds a negligible amount to the overall time, its effect can be neglected.

We can obtain the speedup that includes communication time by taking into account the waiting time for each node. A node at level l waits for a time $O(h)$ for the entire information to reach node 0 in the up pass and then waits for time $O(l)$ for updation in the down pass. Since each message passing takes the same order of time as the CPU time for the operations LFC-RECD, LFC-SENT, PFP-RECD or PFP-SENT; and the number of such operations for a node of k children is $2 \times (k+1)$, we multiply the factor of $\frac{2(k+1)+h+l}{2(k+1)}$ into the number of clusters at each level to obtain the overall speedup with communication overhead. Hence, the speedup now becomes

$$S = O\left((2 \times \sum_0^h k^l)/(\frac{k}{min(k,d)} \times \sum_0^h \frac{2(k+1)+h+l}{2(k+1)} k^{l-1} \times \frac{h}{min(h,d)})\right).$$

In the worst case, that is when there is any direct evidence provided to the leaf nodes at the lowest level, then $l = h$. This happens more often than not. The speedup then becomes

$$S = O\left((2 \times \sum_0^h k^l)/(\frac{k}{min(k,d)} \times \sum_0^h \frac{2(k+1)+2h}{2(k+1)} \times k^{l-1} \times \frac{h}{min(h,d)})\right)$$

$$\text{Or, } S = O\left(\frac{2 \times min(k,d) \times min(h,d) \times (k+1)}{h \times (h+k+1)}\right).$$

5.1.2. Results

Figures 26 to 30 give the actual experimental speedup obtained for trees of various sizes for cubes of dimension 1 to 5 respectively. Figure 31 shows the speedup obtained by mapping a tree of 50 nodes onto cubes of various dimensions. In each graph, the solid line denotes speedup obtained by considering only CPU time and the dotted line denotes speedup obtained by considering the overall time including communication. From the figures, we can see that the results reflect the trend predicted by the analytical estimate.

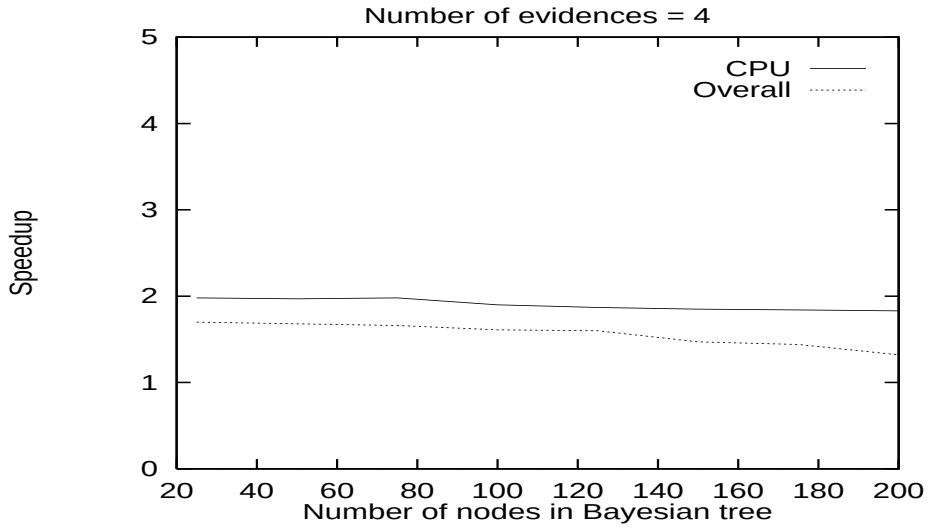

Figure 26. Speedup vs Number of tree nodes for 1d cube

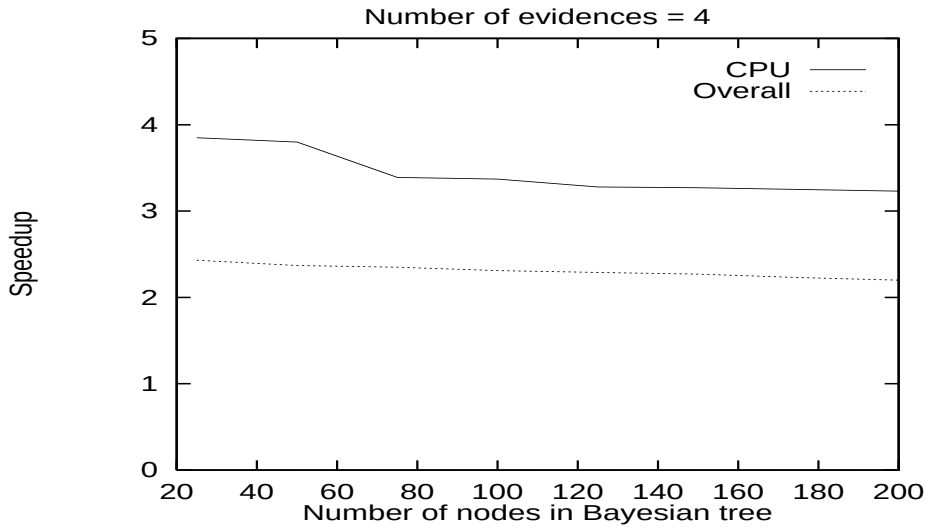

Figure 27. Speedup vs Number of tree nodes for 2d cube

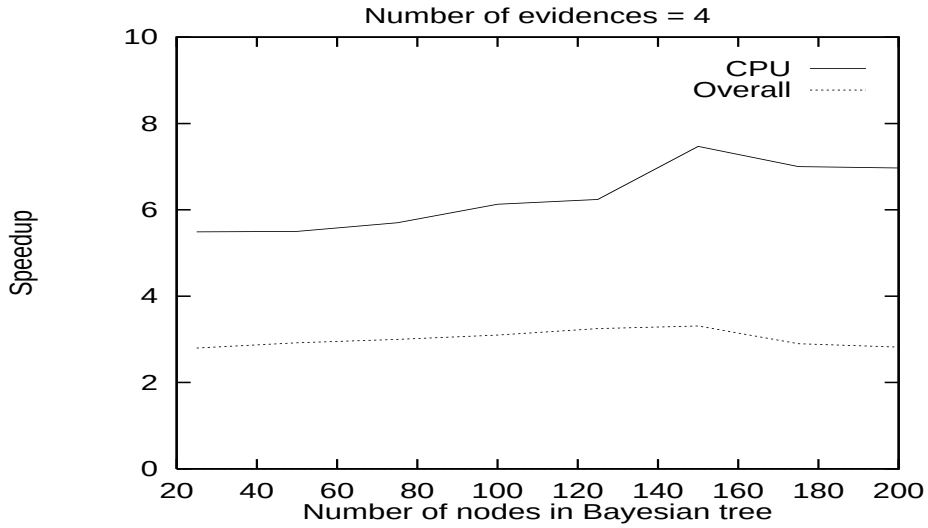

Figure 28. Speedup vs Number of tree nodes for 3d cube

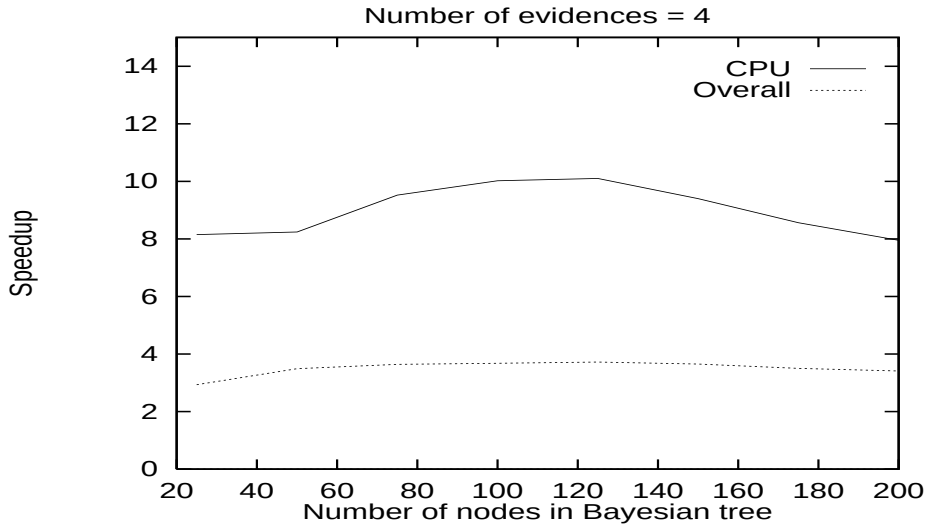

Figure 29. Speedup vs Number of tree nodes for 4d cube

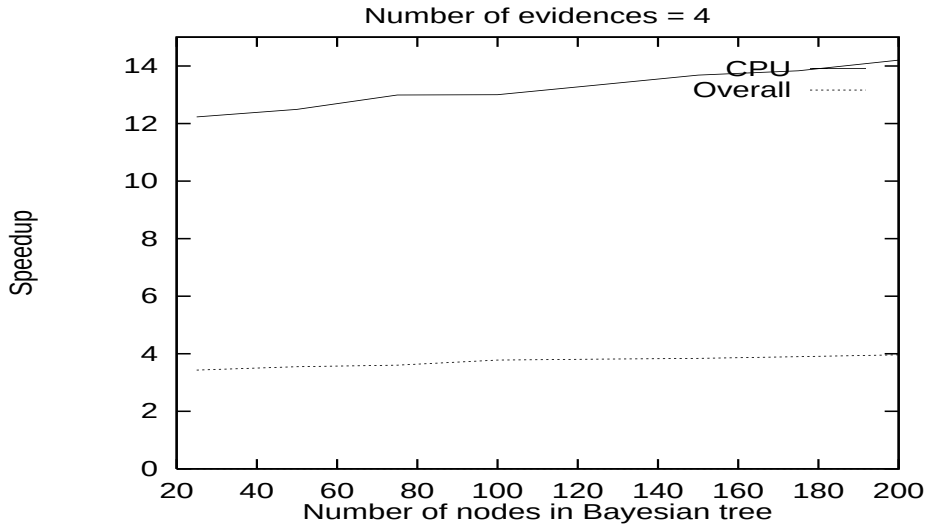

Figure 30. Speedup vs Number of tree nodes for 5d cube

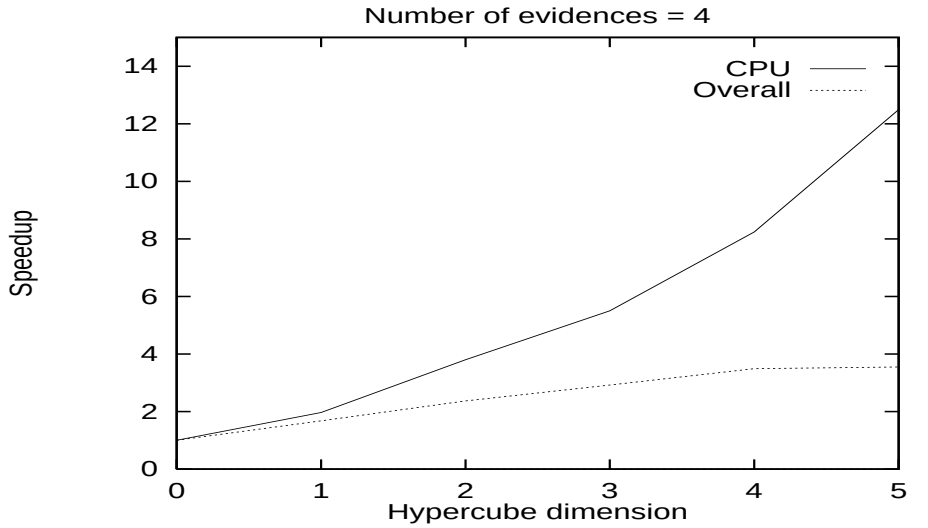

Figure 31. Speedup vs Cube dimension for a tree of 50 nodes

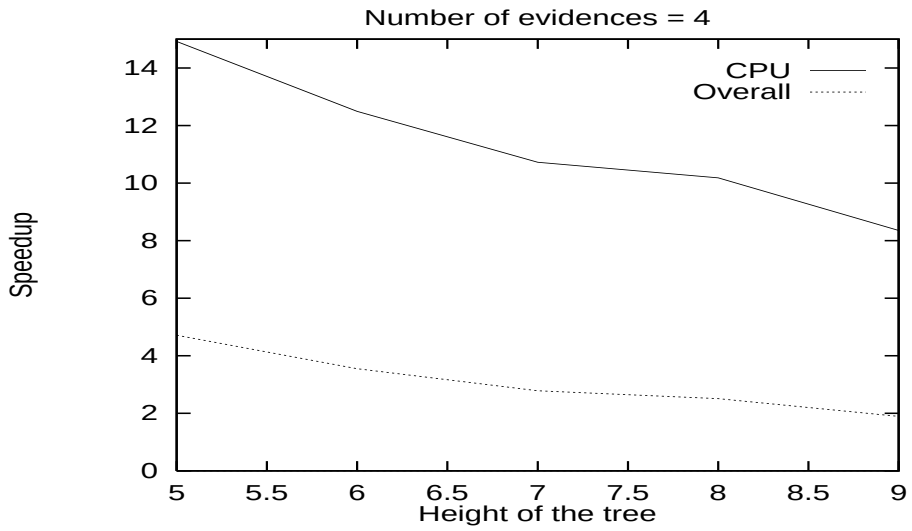

Figure 32. Speedup vs Tree height for a tree of 50 nodes

5.1.3. Timing breakup

In this section, we shall see the actual breakup of the timing and how it results in the speedup that we see for various dimension hypercubes. Table 1 shows the time taken by the nodes of a 22-node Bayesian tree, shown in Figure 32, mapped onto a 2 dimensional hypercube, with direct evidence provided to node 17 of the tree. Since all the computation is done in belief updation on receiving a message or in fetching values for sending a message, the table lists four different fields of CPU time, that is

1) LFC-RECD or lambda message received from a child,

2) LFC-SENT or lambda message sent to a parent,

3) PFP-RECD or pi message received from a parent,

4) PFP-SENT or pi message sent to a child.

The actual timings obtained for each of these operations are as follows. The time taken for computations for the lambda messages to be sent ranges between 7778 X 50

nsec and 7788 X 50 nsec. Similarly, for pi messages sent, the time is usually between 22774 X 50 nsec and 22813 X 50 nsec. Once the messages are received, the program takes from 12556 X 50 nsec to 12579 X 50 nsec for belief updation due to the lambda messages and from 35062 X 50 nsec to 35084X 50 nsec for belief updation due to the pi messages. Each message passing takes constant amount of time too, since only a single hop is required in every case. In addition to this, each message passing requires about 340401 X 50 nsec to 340488 X 50 nsec. Hence, in any cube of dimension 1 or more, this time gets included too.

As seen from the table, the largest number of tree nodes get mapped on to hypercube node 3. Hence, this processor, with tree nodes 3,5,6,8,16,17,18 and 21, gives the worst case timing and hence is chosen for obtaining the speedup.

It has been noticed that best speedup shows up when:

a) The number of children of the nodes is close to a multiple of the dimension of the hypercube.

b) The number of tree nodes is larger than the number of processors.

c) The tree has a slightly random nature. These trees map on with greater load balancing than complete and fully balanced trees.

Hence, the algorithm works best for task trees of the usual everyday applications.

Sometimes, due to clustering of nodes with larger number of pi messages, the speedup obtained for a dimension may turn out to be lesser than the previous dimension, since pi messages require more computation than lambda messages. However, such cases are not very frequent since the lambda messages being sent and received balance the pi messages out. Also, the problem depends on how and where direct evidences are provided, so it tends to vary within the same tree.

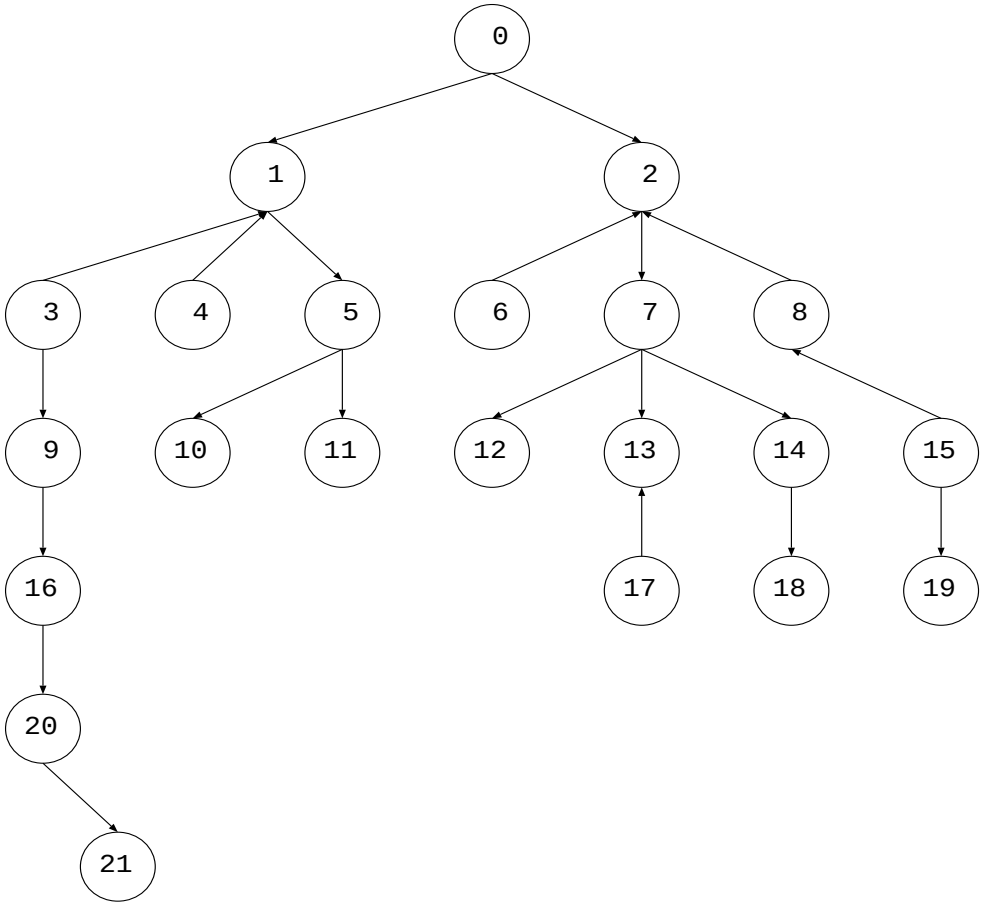

Figure 33. A Bayesian tree with 22 nodes

5.2. Comparison with related work

Peot and Shachter's "Revised Polytree algorithm" [15] reduces computation by handling multiple evidences at one go. The proposed scheme, being a parallel implementation of the Revised Polytree algorithm, can obtain a speedup over the one obtained by a sequential implementation of the algorithm.

In mapping the Bayesian tree onto the parallel machine, this work employs a strategy that is superior to the current task allocation strategies in the following

Table 1. Computation breakup for the 22-node tree mapped onto a 2-d hypercube with direct evidence at Node 17

TREE NODE #	LFC-RECD	LFC-SENT	PFP-RECD	PFP-SENT	CPU-TIME (X 50ns)	HCUBE NODE #
0	1	0	0	2	58163.00	
4	1	0	0	0	12563.00	0
7	1	1	1	3	123821.00	
19	0	0	1	0	35072.00	
1	0	2	1	1	73438.00	
9	0	0	1	1	57878.00	1
10	0	0	1	0	35072.00	
13	0	0	1	2	80700.00	
2	1	3	1	1	93826.00	
11	0	0	1	0	35085.00	
12	0	2	2	0	85713.00	2
14	0	0	1	0	35077.00	
15	1	0	0	1	35387.00	
20	0	0	1	1	57875.00	
3	1	0	0	1	35383.00	
5	0	0	1	2	80690.00	
6	1	0	0	0	12567.00	
8	1	1	0	0	55412.00	3
16	0	0	1	1	57880.00	
17	1	0	0	1	35319.00	
18	0	0	1	0	35068.00	
21	0	0	1	0	35851.00	

ways. The approach in [20], if applied to Bayesian networks, would require multiple message hops between parent and child nodes. However, the proposed approach maintains adjacency properties when mapped and thus requires only single message hop throught the network. Hence, the *Dilation Bound*, defined in Chapter 2 as the number of interprocessor links required to map an edge of the task tree, remains constant at 1. Also, since the technique discussed in this thesis allows mapping of Bayesian trees with a large number of nodes onto a relatively smaller number of processors, the *Expansion Ratio*, also explained in Chapter 2 as the ratio of the

number of processors to the number of tasks, is usually lesser than 1. The proposed scheme also has an advantage over the one suggested in [17] based on the minimization of communication overhead. By maintaining adjacency, we reduce the total message traffic on the parallel machine links. Mapping multiple tasks on single processors has been described in [22], which uses cluster mapping for the same. However, the method given in the paper requires the computation and the communication to be done at separate times and to be fully synchronized. The present work, however, was implemented free of these restrictions. Table 2 summarizes the above comparisons.

Table 2. Comparison with other mapping approaches

Method/ scheme	Number of hops for messages between adjacent nodes	Ratio of tasks to hypercube processors	Synchronization requirement for computation and communication
Kavianpour and Bagherzadeh's	Multiple	Less than or equal to 1	Not reqd.
Horiike's	Multiple	Less than or equal to 1	Not reqd.
Kawaguchi, Tamura and Utsumiya's	None within a cluster; multiple between clusters	More than or equal to 1	Required
Proposed	Single	Almost always more than 1	Not reqd.

5.3. Future work

This thesis mainly concerns itself with static mapping. However, dynamic mapping can yield very good results if the sequence of message passing and communication is maintained. Apart from this, with the use of dynamic mapping, the network will

need extra storage to maintain the current status of each node and computation stage. Hence, a good area of future work can be to have an intelligent dynamic mapping that takes care of the above mentioned factors.

Another scope of improvement appears in the grouping of nodes. An efficient way of separating the nodes with a large number of pi messages will remove anamolies such as a larger dimension having lesser speedup than the smaller dimension. Also, this thesis only considers tree structured Bayesian networks. Hence, one may want to extend the work to cover other graphs too.

Since the overhead in implementing Bayesian networks on parallel machines like hypercubes can be very large, a VLSI chip can be developed which handles all the message passing efficiently so as to reduce overall time. Message passing can sometimes become a problem due to queueing up of messages in the processor buffers. Hence, it requires a setup in which there are separate memory blocks for each message type within each processor. A design of this kind eliminates the need for a FIFO buffer, on each processor, to store the different messages. In this way, messages will not have to queue up in the processor buffers. This will reduce the waiting time.

Since Bayesian belief networks have a large number of applications, there is a great scope of specific application-based architectures too that can yield better results by exploiting the knowledge of the structure of the task graph.

LIST OF REFERENCES

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann Publishers, first ed., 1988.

[2] G. Cooper, "Computational complexity of probabilistic inference using bayesian belief networks (research note)," *Artificial Intelligence 42*, pp. 393–405, 1990.

[3] P. Dagum and M. Luby, "Approximately probabilistic reasoning in bayesian belief networks is np-hard," *Artificial Intelligence*, pp. 141–153, 1993.

[4] D. E. Heckerman, "A tractable algorithm for diagnosing multiple diseases," *Artificial Intelligence*, pp. 174–181, 1989.

[5] H. J. Suermondt and G. F. Cooper, "A combination of exact algorithms for inference on bayesian belief networks," *Int. J. Approximate Reasoning*, pp. 521–542, 1991.

[6] K. Ramamurthi and A. M. Agogino, "Real time expert system for fault tolerant supervisory control," *Computers in Engineering*, pp. 333–339, 1988.

[7] R. D. Shacter, S. K. Anderson, and K. L. Poh, "Directed reduction algorithms and decomposable graphs," *Artificial Intelligence*, pp. 237–244, 1990.

[8] Wright, "Correlation and causation," *Journal of Agricult. Research, 20*, pp. 557–585, 1921.

[9] J. Pearl, "Evidential reasoning using stochastic simulation of causal models," *Artificial Intelligence 33*, pp. 173–215, 1987.

[10] J. Pearl, "On evidential reasoning in a hierarchy of hypotheses," *Artificial Intelligence 28*, pp. 9–15, 1986.

[11] J. Pearl, "Distributed revision of composite beliefs," *Artificial Intelligence 29*, pp. 241–288, 1986.

[12] J. Pearl, "Embracing causality in default reasoning," *Artificial Intelligence 35*, pp. 259–271, 1988.

[13] D. Heckerman, A. Mamdani, and M. P. Wellman, "Introduction (bayesian networks)," *Communications of the ACM*, pp. 24–26, March 1995.

[14] D. Heckerman, "A tutorial on learning bayesian networks." Technical Report, March 1995.

[15] M. A. Peot and R. D. Shachter, "Fusion and propagation with multiple observations in belief networks (research note)," *Artificial Intelligence 48*, pp. 299–318, 1991.

[16] K. R. Pattipati, T. Kurien, R.-T. Lee, and P. B. Luh, "On mapping a tracking algorithm onto parallel processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, pp. 774–791, September 1990.

[17] S. Horiike, "A task mapping method for a hypercube," *Systems and Computers in Japan*, vol. 22, no. 9, pp. 14–22, 1991.

[18] W. K. Chen and E. F. Gehringer, "A graph oriented mapping strategy for a hypercube," *3rd Hypercube Concurrent Computers and Applications*, pp. 200–209, 1988.

[19] F. Ercal, J. Ramanuja, and P. Sadayappan, "Task allocation onto a hypercube by recursive mincut bipartitioning," *3rd Hypercube Concurrent Computers and Applications*, pp. 210–221, 1988.

[20] A. Kavianpour and N. Bagherzadeh, "A systematic approach for mapping application tasks in hypercubes," *IEEE Transactions on Computers*, vol. 42, pp. 742–746, June 1993.

[21] T. Kawaguchi, "Static allocation of a task tree onto a linear array," *IEEE International Symposium on Circuits and Systems*, vol. 3, no. PART 3/40, pp. 1921–1924, 1993.

[22] T. Kawaguchi, Y. Tamura, and K. Utsumiya, "A task mapping algorithm for linear array processors," *IEICE Transactions on Information and Systems*, vol. E77-D, pp. 546–554, May 1994.

[23] M.-S. Chen and K. G. Shin, "Subcube allocation and task migration in hypercube multiprocessors," *IEEE Transactions on Computers*, vol. 39, pp. 1146–1155, September 1990.

[24] A. Sivasubramaniami, U. Ramachandran, and H. Venkateswaran, "A computational model for message-passing," *IEEE Proceedings of the International Conference on Parallel Processing*, pp. 358–361, 1992.

[25] K. H. Kim and A. Kavianpour, "A distributed recovery block approach to fault-tolerant execution of application tasks in hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 104–111, January 1993.

[26] O. K. Hejlesen, S. Andreassen, and S. K. Andersen, "Implementation of a learning procedure for multiple observations in a diabetes advisory system based on causal probabilistic networks," in *Artificial Intelligence in Medicine*, Munich: IOS Press, 1993.

[27] U. G. Oppel, A. Hierle, L. Janke, and W. Moser, "Transformation of compartmental models into sequences of causal probabilistic networks," in *Artificial Intelligence in Medicine*, Munich: IOS Press, 1993.

[28] B. Abramson, "The design of belief network-based systems for price forecasting," *Computers Elect. Engng.*, vol. 20, no. 2, pp. 163–180, 1994.

[29] F. Nadi, A. M. Agogino, and D. A. Hodges, "Use of influence diagrams and neural networks in modeling semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 4, pp. 52–58, February 1991.

[30] R. Fung and B. D. Favero, "Applying bayesian networks to information retrieval," *Communications of the ACM*, pp. 42–48, March 1995.

[31] S. Sarkar and K. L. Boyer, "Using perceptual inference networks to manage vision processes," *Computer Vision and Image Understanding*, vol. 62, pp. 27–46, July 1995.

[32] L. Burnell and E. Horvitz, "Structure and chance : Melding logic and probability for software debugging," *Communications of the ACM*, pp. 31–41, March 1995.

[33] D. Heckerman, J. S. Breese, and K. Rommelse, "Decision-theoretic troubleshooting," *Communications of the ACM*, pp. 49–57, March 1995.

[34] D. Heckerman, J. Breese, and K. Rommelse, "Troubleshooting under uncertainty." Technical Report, January 1994.

[35] D. Heckerman and R. Shachter, "Decision-theoretic foundations for causal reasoning." Technical Report, March 1994.

[36] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, "Bayesian updating in recursive graphical models by local computations," *Comput. Stat. Q.*, 1990.

[37] F. V. Jensen, K. G. Olesen, and S. K. Andersen, "An algebra of bayesian belief universes for knowledge based systems," *Networks*, 1990.

[38] J. H. Kim and J. Pearl, "A computational model for casual and diagnostic reasoning in inference engines," *Proceedings IJCAI-83*, 1983.

[39] J. Pearl, "Fusion, propagation and structuring in belief networks," *Artificial Intelligence 29*, pp. 241–288, 1986.