Microelectronics Reliability xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Microelectronics Reliability

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

Maximum error modeling for fault-tolerant computation using maximum *a posteriori* (MAP) hypothesis

Karthikeyan Lingasubramanian [a,*], Syed M. Alam [b], Sanjukta Bhanja [a]

[a] *Nano Computing Research Group (NCRG), Department of Electrical Engineering, University of South Florida, Tampa, Florida, USA*
[b] *EverSpin Technologies, Austin, Texas, USA*

ARTICLE INFO

Article history:
Received 6 November 2009
Received in revised form 15 July 2010
Accepted 21 July 2010
Available online xxxx

ABSTRACT

The application of current generation computing machines in safety-centric applications like implantable biomedical chips and automobile safety has immensely increased the need for reviewing the worst-case error behavior of computing devices for fault-tolerant computation. In this work, we propose an exact probabilistic error model that can compute the *maximum* error over all possible input space in a circuit-specific manner and can handle various types of structural dependencies in the circuit. We also provide the worst-case input vector, which has the highest probability to generate an erroneous output, for any given logic circuit. We also present a study of circuit-specific error bounds for fault-tolerant computation in heterogeneous circuits using the maximum error computed for each circuit. We model the error estimation problem as a maximum *a posteriori* (MAP) estimate [28,29], over the joint error probability function of the entire circuit, calculated efficiently through an intelligent search of the entire input space using probabilistic traversal of a binary Join tree using *Shenoy–Shafer* algorithm [20,21]. We demonstrate this model using MCNC and ISCAS benchmark circuits and validate it using an equivalent HSpice model. Both results yield the same worst-case input vectors and the highest percentage difference of our error model over HSpice is just 1.23%. We observe that the maximum error probabilities are significantly larger than the average error probabilities, and provides a much tighter error bounds for fault-tolerant computation. We also find that the error estimates depend on the specific circuit structure and the maximum error probabilities are sensitive to the individual gate failure probabilities.

1. Introduction

Why maximum error? *Industries like automotive and health care, which employs safety-centric electronic devices, have traditionally addressed high reliability requirements by employing redundancy, error corrections, and choice of proper assembly and packaging technology. In addition, rigorous product testing at extended stress conditions filters out even an entire lot in the presence of a small number of failures [38]. Another rapidly growing class of electronic chips where reliability is very critical is implantable biomedical chips [40,41]. More interestingly, some of the safety approaches, such as redundancy and complex packaging, are not readily applicable to implantable biomedical applications because of low voltage, low power operation and small form factor requirements. Also in future technologies like NW-FET, CNT-FET [43], RTD [45], hybrid nano devices [15], single electron tunneling devices [16], field coupled computing devices like QCA's [44] (molecular and magnetic) and spin-coupled computing devices, computing components are likely to have higher error rates (both in terms of defect and transient faults) since they operate near the thermal limit and information processing occurs at extremely small volume. Nano-CMOS, beyond 22 nm, is not an exception in this regard as the frequency scales up and voltage and geometry scales down. Also we have to note that, while two design implementation choices can have different average probabilities of failures, the lower average choice may in fact have higher maximum probability of failure leading to lower yield in manufacturing and more rejects during chip burn-in and extended screening.*

1.1. Proposed work

In this work, we present a probabilistic model to study the *maximum* output error over all possible input space for a given logic circuit. We present a method to find out the worst-case input vector, i.e., the input vector that has the highest probability to give an error at the output. In the first step of our model, we convert the circuit into a corresponding *edge-minimal* probabilistic network that represents the basic logic function of the circuit by handling the interdependencies between the signals using random variables of interest in a composite joint probability distribution function $P(y_1, y_2, \ldots, y_N)$. Each node in this network corresponds to a random variable representing a signal in the digital circuit, and each edge corresponds to the logic governing the connected signals. The indi-

* Corresponding author.
E-mail address: klingasu@mail.usf.edu (K. Lingasubramanian).

0026-2714/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.microrel.2010.07.156

vidual probability distribution for each node is given using conditional probability tables.

From this probabilistic network we obtain our probabilistic error model that consists of three blocks, (i) ideal error free logic, (ii) error prone logic where every gate has a gate error probability ε i.e., each gate can go wrong individually by a probabilistic factor ε and (iii) a detection unit that uses comparators to compare the error free and erroneous outputs. The error prone logic represents the real time circuit under test, whereas the ideal logic and the detection unit are fictitious elements used to study the circuit. Both the ideal logic and error prone logic would be fed by the primary inputs $\mathbf{I}$. We denote all the internal nodes, both in the error free and erroneous portions, by $\mathbf{X}$ and the comparator outputs as $\mathbf{O}$. The comparators are based on XOR logic and hence a state "1" would signify error at the output. An evidence set $\mathbf{o}$ is created by evidencing one or more of the variables in the comparator set $\mathbf{O}$ to state "1" ($P(O_i = 1) = 1$). Then performing maximum *a posteriori* (MAP) hypothesis on the probabilistic error model provides the worst-case input vector $\mathbf{i}_{MAP}$ which gives the maximum joint probability $P(\mathbf{i}_{MAP}, \mathbf{o})$ over all possible input combinations $\mathbf{i}$, $\max_{\forall \mathbf{i}} P(\mathbf{i}, \mathbf{o})$. The maximum output error probability can be obtained from $P(O_i = 1)$ after instantiating the input nodes of probabilistic error model with $\mathbf{i}_{MAP}$ and inferencing. The process is repeated for increasing ε values and finally the ε value that makes at least one of the output signals completely random ($P(O_i = 0) = 0.5, P(O_i = 1) = 0.5$) is taken as the error bound for the given circuit.

It is obvious that we can arrive at MAP estimate by enumerating all possible input instantiations and compute the maximum $P(\mathbf{i}, \mathbf{o})$ by any probabilistic computing tool. The attractive feature of the MAP algorithm that we have adapted from [28,29] for digital circuits, lies on eliminating a significant part of the input search-subtree based on an easily available upper-bound of $P(\mathbf{i}, \mathbf{o})$ by using probabilistic traversal of a binary Join tree with *Shenoy–Shafer* algorithm [20,21,27]. The actual computation is divided into two theoretical components. First, we convert the circuit structure into a binary Join tree and employ Shenoy–Shafer algorithm, which is a two-pass probabilistic message-passing algorithm, to obtain multitude of upper bounds of $P(\mathbf{i}, \mathbf{o})$ with partial input instantiations. Next, we construct a Binary tree of the input vector space where each path from the root node to the leaf node represents an input vector. At every node, we traverse the search tree if the upper bound, obtained by Shenoy–Shafer inference on the binary Join tree, is greater than the maximum probability already achieved; otherwise we prune the entire sub-tree. Experimental results on a few standard benchmark show that the worst-case errors significantly deviate from the average ones and also provides tighter bounds for the ones that use homogeneous gate-type (c17 with NAND-only).

Salient features and deliverables are itemized below:

- We have proposed a method to calculate *maximum* output error using a probabilistic model. Through experimental results, we show the importance of modeling maximum output error (Fig. 11).
- Given a circuit with a fixed gate error probability ε, our model can provide the maximum output error probability and the *worst-case* input vector, which can be very useful testing parameters.
- We present the circuit-specific error bounds for fault-tolerant computation and we show that maximum output errors provide a tighter bound.
- We have adapted an efficient design framework for digital circuits, that employs inference in binary Join trees using Shenoy–Shafer algorithm [20,21] to perform MAP hypothesis [28,29] accurately, in order to calculate maximum output error.
- We give a probabilistic error model, where efficient error incorporation is possible, for useful reliability studies. Using our model the error injection and probability of error for each gate can be modified easily. Moreover, we can accommodate both fixed and variable gate errors in a single circuit without affecting computational complexity.

The rest of the paper is structured as follows: Section 2 gives a summary of some of the previous works on error bounds for fault-tolerant computation along with some of the reliability models established from these works, Section 3 explains the structure of our probabilistic error model, Section 4 explains the MAP hypothesis and its complexity, Section 5 provides the experimental results, followed by conclusion in Section 6.

2. Prior work

2.1. State-of-the-art

The study of reliable computation using unreliable components was initiated by von Neumann [1] who showed that erroneous components with some small error probability can provide reliable outputs and this is possible only when the error probability of each component is less than 1/6. This work was later enhanced by Pippenger [2] who realized von Neumann's model using formulas for Boolean functions. This work showed that for a function controlled by k-arguments the error probability of each component should be less than $(k-1)/2k$ to achieve reliable computation. This work was later extended by using networks instead of formulas to realize the reliability model [3]. In [4], Hajek and Weller used the concept of formulas to show that for three-input gates the error probability should be less than 1/6. Later this work was extended for k-input gates [5] where k was chosen to be odd. For a specific even case, Evans and Pippenger [6] showed that the maximum tolerable noise level for two-input NAND gate should be less than $(3-\sqrt{7})/4 = 0.08856\ldots$. Later this result was reiterated by Gao et al. for two-input NAND gate, along with other results for k-input NAND gate and majority gate, using bifurcation analysis [7] that involves repeated iterations on a function relating to the specific computational component. While there exists studies of circuit-specific bounds for circuit characteristics like switching activity [8], the study of circuit-specific error bounds would be highly informative and useful for designing high-end computing machines.

The study of fault-tolerant computation has expanded its barriers and is being generously employed in fields like nano-computing architectures. Reliability models like Triple Modular Redundancy (TMR) and N-Modular Redundancy (NMR) [9] were designed using the von Neumann model. Expansion of these techniques led to models like Cascaded Triple Modular Redundancy (CTMR) [10] used for nanochip devices. In [11], the reliability of reconfigurable architectures was obtained using NAND multiplexing technique and in [12], majority multiplexing was used to achieve fault-tolerant designs for nanoarchitectures. A recent comparative study of these methods [13], indicates that a 1000-fold redundancy would be required for a device error (or failure) rate of 0.01[1]. Many researchers are currently focusing on computing the average error [18,19] from a circuit and also on the expected error to conduct reliability–redundancy trade-off studies. An approximate method based on Probabilistic Gate Model (PGM) is discussed by Han et al. in [14]. Here the PGMs are formed using equations governing the functionality between an input and an output. Probabilistic analysis of digital logic circuits using decision diagrams is

[1] Note that this does *not* mean 1 out of 100 devices will fail, it indicates the devices will generate erroneous output 1 out of 100 times.

proposed in [17]. In [26], the average output error in digital circuits is calculated using a probabilistic reliability model that employs Bayesian Networks.

In testing, the identification of possible input patterns to perform efficient circuit testing is achieved through Automatic Test Pattern Generation (ATPG) algorithms. Some of the commonly used ATPG algorithms like D-algorithm [31], path-oriented decision making (PODEM) algorithm [32] and fanout-oriented test generation (FAN) algorithm [33] are deterministic in nature. There are some partially probabilistic ATPG algorithms [34–36] which are basically used to reduce the input pattern search space. In order to handle transient errors occurring in intermediate gates of a circuit, we need a completely probabilistic model [37].

2.2. Relation to State-of-the-art

Our work concentrates on *estimation of maximum error as opposed to average error*, since for higher design levels it is important to account for maximum error behavior, especially if this behavior is far worse than the average case behavior.

Also our work proposes *a completely probabilistic model as opposed to a deterministic model*, where every gate of the circuit is modeled probabilistically and the worst case input pattern is obtained.

The bounds presented in all the above mentioned works do not consider (i) combination of different logic units like NAND and majority in deriving the bounds and (ii) do not consider circuit structure and dependencies and error masking that could occur in a realistic logic network, making the bounds pessimistic. Our model *encapsulates the entire circuit structure* along with the signal inter dependencies and so is capable of estimating the error bound of the entire circuit as opposed to a single logic unit.

3. Probabilistic error model

The underlying model compares error-free and error-prone outputs. Our model contains three sections, (i) error-free logic where the gates are assumed to be perfect, (ii) error-prone logic where each gate goes wrong independently by an error probability ε and (iii) XOR-logic based comparators that compare the error-free and error-prone primary outputs. When error occurs, the error-prone primary output signal will not be at the same state as the ideal error-free primary output signal. So, *an output of logic "1" at the XOR comparator gate indicates occurrence of error*. For a given digital logic circuit as in Fig. 1a, the error model and the corresponding probabilistic error model are illustrated in Fig. 1b and c respectively. In Fig. 1b and c, block 1 is the error-free logic, block 2 is the error-prone logic with gate error probability ε and block 3 is the comparator logic. In the entire model, the error-prone portion given in block 2 is the one that represents the real-time circuit. The ideal error-free portion in block 1 and the comparator portion in block 3 are fictitious and used for studying the given circuit.

We would like the readers to note that we will be representing a **SET OF VARIABLES** by bold capital letters, **set of instantiations** by bold small letters, any SINGLE VARIABLE by capital letters. Also probability of the event $Y_i = y_i$ will be denoted simply by $P(y_i)$ or by $P(Y_i = y_i)$.

The probabilistic network is a conditional factoring of a joint probability distribution. The nodes in the network are random variables representing each signal in the underlying circuit. To perfectly represent digital signals each random variable will have two states, state "0" and state "1". The edges represent the logic that governs the connecting nodes using conditional probability tables (CPTs). For example, in Fig. 1c, the nodes $X1$ and $X4$ are random variables representing the error-free signal $X1$ and the error-prone signal $X4$ respectively of the digital circuit given in Fig. 1a. The edges connecting these nodes to their parents $I1$ and $I2$ represent the error-free AND logic and error-prone AND logic as given by the CPTs in Table 1.

Let us define the random variables in our probabilistic error model as $\mathbf{Y} = \mathbf{I} \cup \mathbf{X} \cup \mathbf{O}$, composed of the three disjoint subsets $\mathbf{I}$, $\mathbf{X}$ and $\mathbf{O}$ where

1. $I_1, \ldots, I_k \in \mathbf{I}$ are the set of k primary inputs.

Table 1
Conditional Probability Tables (CPTs) for error-free and error-prone AND logic present in the probabilistic error model given in Fig. 1.

Error-free AND		
$P(X1 = 1 \mid I1, I2)$	$P(I2 = 0) = 1$	$P(I2 = 1) = 1$
$P(I1 = 0) = 1$	0	0
$P(I1 = 1) = 1$	0	1
Error-prone AND		
$P(X4 = 1 \mid I1, I2)$	$P(I2 = 0) = 1$	$P(I2 = 1) = 1$
$P(I1 = 0) = 1$	ε	ε
$P(I1 = 1) = 1$	ε	$1 - \varepsilon$

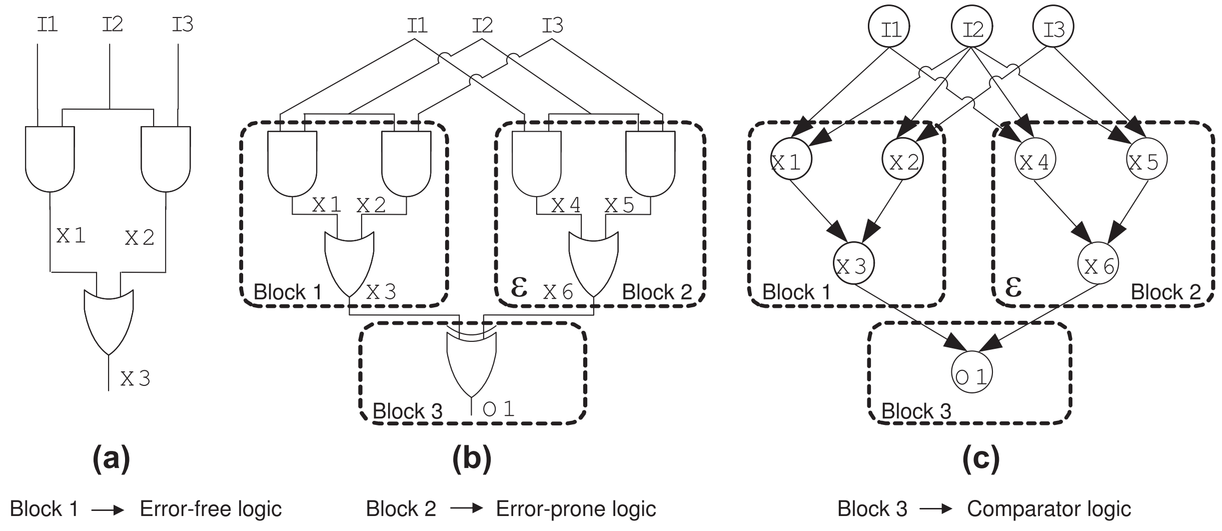

I1
I2
I3
X1
X2
X3
Block 1
X3
ε
X6
Block 2
X4
X5
Block 3
O1
(a)
(b)
(c)
Block 1 → Error-free logic
Block 2 → Error-prone logic
Block 3 → Comparator logic

Fig. 1. (a) Digital logic circuit, (b) error model, and (c) probabilistic error model.

2. $X_1,\ldots,X_m \in \mathbf{X}$ are the m internal logic signals for both the erroneous (every gate has a failure probability ε) and error-free ideal logic elements.
3. $O_1,\ldots,O_n \in \mathbf{O}$ are the n comparator outputs, each one signifying the error in one of the primary outputs of the logic block.
4. $N = k + m + n$ is the total number of network random variables.

Any probability function $P(y_1,y_2,\ldots,y_N)$, where $y_1,y_2,\ldots,y_N$ are random variables, can be written as,

$$P(y_1,\ldots,y_N) = P(y_N|y_{N-1},y_{N-2},\ldots,y_1)P(y_{N-1}|y_{N-2},y_{N-3},\ldots,y_1)\cdots P(y_1) \quad (1)$$

This expression holds for any ordering of the random variables. In most applications, a variable is usually not dependent on all other variables. There are lots of conditional independencies embedded among the random variables, which can be used to reorder the random variables and to simplify the joint probability as,

$$P(y_1,\ldots,y_N) = \prod_v P(y_v|Pa(Y_v)) \quad (2)$$

where $Pa(Y_v)$ indicates the parents of the variable Y_v, representing its direct causes. This factoring of the joint probability function can be denoted as a graph with links directed from the random variable representing the inputs of a gate to the random variable representing the output. To understand it better let us look at the error model given in Fig. 1c. The joint probability distribution representing the network can be written as,

$$P(i1,i2,i3,x1,\ldots,x6,o1) = P(o1|x6,\ldots,x1,i3,i2,i1)P(x6|x5,\ldots,x1,i3,i2,i1)\cdots P(i3)P(i2)P(i1) \quad (3)$$

Here the random variable $O1$ is independent of the random variables $X1$, $X2$, $X4$, $X5$, $I1$, $I2$, $I3$ given its parents $X3$, $X6$. This notion explains the conditional independence between the random variables in the network and it is mathematically denoted by $I(O1,\{X3,X6\},\{X1,X2,X4,X5,I1,I2,I3\})$. So for $O1$, the probability distribution can be rephrased as,

$$P(o1|x6,\ldots,x1,i3,i2,i1) = P(o1|x6,x3) \quad (4)$$

By implementing all the underlying conditional independencies the basic joint probability distribution can be rephrased as,

$$P(i1,i2,i3,x1,\ldots,x6,o1) = P(o1|x6,x3)P(x6|x5,x4)P(x5|i3,i2)P(x4|i2,i1)P(x3|x2,x1) P(x2|i3,i2)P(x1|i2,i1)P(i3)P(i2)P(i1) \quad (5)$$

The implementation of this probability distribution can be clearly seen in Fig. 1c. Each node is connected only to its parents and not to any other nodes. The conditional probability potentials for all the nodes are provided by the CPTs. The attractive feature of this graphical representation of the joint probability distribution is that not only does it make conditional dependency relationships among the nodes explicit but it also serve as a computational mechanism for efficient probabilistic updating.

4. Maximum *a posteriori* (MAP) estimate

As we mentioned earlier, in our probabilistic error model, the network variables, say $\mathbf{Y}$, can be divided into three subsets $\mathbf{I}$, $\mathbf{X}$ and $\mathbf{O}$ where $I_1,\ldots,I_k \in \mathbf{I}$ represents primary input signals; $X_1,\ldots,X_m \in \mathbf{X}$ represents internal signals including the primary output signals; $O_1,\ldots,O_n \in \mathbf{O}$ represents the comparator output signals. *Any primary output node can be forced to be erroneous by fixing the corresponding comparator output to logic "1"*, that is providing an *evidence* $\mathbf{o} = \{P(O_i = 1) = 1\}$ to a comparator output O_i. Given some evidence $\mathbf{o}$, the objective of the maximum *a posteriori* estimate is to find a complete instantiation $\mathbf{i}_{MAP}$ of the variables in $\mathbf{I}$ that gives the following joint probability,

$$MAP(\mathbf{i}_{MAP},\mathbf{o}) = \max_{\forall \mathbf{i}} P(\mathbf{i},\mathbf{o}) \quad (6)$$

The probability $MAP(\mathbf{i}_{MAP},\mathbf{o})$ is termed as the *MAP probability* and the variables in $\mathbf{I}$ are termed as *MAP variables* and the instantiation $\mathbf{i}_{MAP}$ which gives the maximum $P(\mathbf{i},\mathbf{o})$ is termed as the *MAP instantiation*.

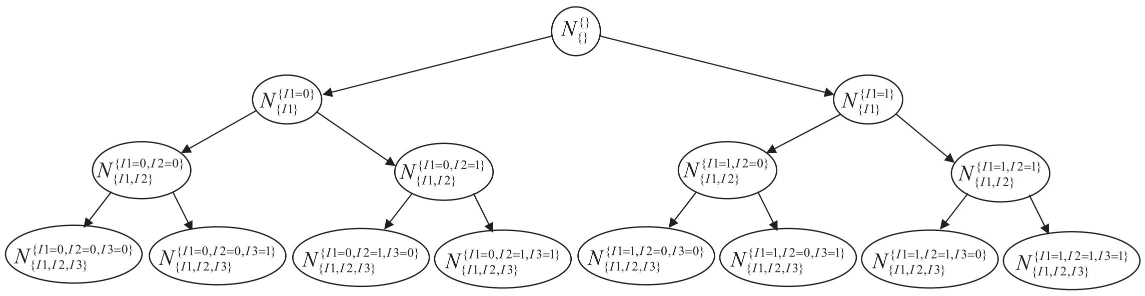
For example, consider Fig. 1. In the probabilistic model shown in Fig. 1c, we have $\{I1,I2,I3\} \in \mathbf{I}$; $\{X1,X2,X3,X4,X5,X6\} \in \mathbf{X}$; $\{O1\} \in \mathbf{O}$. $X3$ is the ideal error-free primary output node and $X6$ is the corresponding error-prone primary output node. Giving an evidence $\mathbf{o} = \{P(O1 = 1) = 1\}$ to $O1$ indicates that $X6$ has produced an erroneous output. The MAP hypothesis uses this information and finds the input instantiation, $\mathbf{i}_{MAP}$, that would give the maximum $P(\mathbf{i},\mathbf{o})$. This indicates that $\mathbf{i}_{MAP}$ is the most probable input instantiation that would give an error in the error-prone primary output signal $X6$. In this case, $\mathbf{i}_{MAP} = \{I1 = 0, I2 = 0, I3 = 0\}$. This means that the input instantiation $\{I1 = 0, I2 = 0, I3 = 0\}$ will most probably provide a wrong output, $X6 = 1$ (since the correct output is $X6 = 0$).

We arrive at the exact maximum *a posteriori* (MAP) estimate using the algorithms by Park and Darwiche [28,29]. It is obvious that we could arrive at MAP estimate by enumerating all possible input instantiations and compute the maximum output error. To make it more efficient, our MAP estimates rely on eliminating some part of the input search-subtree based on an easily available upper-bound of MAP probability by using a probabilistic traversal of a binary Join tree using *Shenoy–Shafer* algorithm [20,21]. The actual computation is divided into two theoretical components.

- First, we convert the circuit structure into a binary Join tree and employ Shenoy–Shafer algorithm, which is a two-pass probabilistic message-passing algorithm, to obtain multitude of upper bounds of MAP probability with partial input instantiations (discussed in Section 4.1). The reader familiar with Shenoy–Shafer algorithm can skip the above mentioned section. To our knowledge, Shenoy–Shafer algorithm is not commonly used in VLSI context, so we elaborate most steps of Join tree creation, two-pass Join tree traversal and computation of upper bounds with partial input instantiations.
- Next, we construct a Binary tree of the input vector space where each path from the root node to the leaf node represents an input vector. At every node, we traverse the search tree if the upper bound, obtained by Shenoy–Shafer inference on the binary Join tree, is greater than the maximum probability already achieved; otherwise we prune the entire sub-tree. The depth-first traversal in the binary input instantiation tree is discussed in Section 4.2 where we detail the search process, pruning and heuristics used for better pruning. Note that the pruning is key to the significantly improved efficiency of the MAP estimates.

4.1. Calculation of MAP upper bounds using Shenoy–Shafer algorithm

To clearly understand the various MAP probabilities that are calculated during MAP hypothesis, let us see the binary search tree formed using the MAP variables. A complete search through the MAP variables can be illustrated as shown in Fig. 2 which gives the corresponding search tree for the probabilistic error model given in Fig. 1c, which has three primary inputs or MAP variables. In this search tree, the root node N will have an empty instantiation; every intermediate node $N^{\mathbf{i}_{inter}}_{\mathbf{I}_{inter}}$ will be associated with a subset $\mathbf{I}_{inter}$ of MAP variables $\mathbf{I}$ and the corresponding partial instantiation $\mathbf{i}_{inter}$; and every leaf node $N^{\mathbf{i}}_{\mathbf{I}}$ will be associated with the entire set $\mathbf{I}$ and the corresponding complete instantiation $\mathbf{i}$. Also each node will



$N^{\{\}}_{\{\}}$
$N^{\{I1=0\}}_{\{I1\}}$
$N^{\{I1=1\}}_{\{I1\}}$
$N^{\{I1=0,I2=0\}}_{\{I1,I2\}}$
$N^{\{I1=0,I2=1\}}_{\{I1,I2\}}$
$N^{\{I1=1,I2=0\}}_{\{I1,I2\}}$
$N^{\{I1=1,I2=1\}}_{\{I1,I2\}}$
$N^{\{I1=0,I2=0,I3=0\}}_{\{I1,I2,I3\}}$
$N^{\{I1=0,I2=0,I3=1\}}_{\{I1,I2,I3\}}$
$N^{\{I1=0,I2=1,I3=0\}}_{\{I1,I2,I3\}}$
$N^{\{I1=0,I2=1,I3=1\}}_{\{I1,I2,I3\}}$
$N^{\{I1=1,I2=0,I3=0\}}_{\{I1,I2,I3\}}$
$N^{\{I1=1,I2=0,I3=1\}}_{\{I1,I2,I3\}}$
$N^{\{I1=1,I2=1,I3=0\}}_{\{I1,I2,I3\}}$
$N^{\{I1=1,I2=1,I3=1\}}_{\{I1,I2,I3\}}$

Fig. 2. Corresponding binary search tree for the probabilistic error model given in Fig. 1, which has three primary inputs or MAP variables.

have v children where v is the number of values or states that can be assigned to each variable I_i. Since we are dealing with digital signals, every node in the search tree will have two children. Since the MAP variables represent the primary input signals of the given digital circuit, one path from the root to the leaf node of this search tree gives one input vector choice. In Fig. 2, at node $N^{01}_{\{I1,I2\}}$, $\mathbf{I}_{inter} = \{I1, I2\}$ and $\mathbf{i}_{inter} = \{I1 = 0, I2 = 1\}$. The basic idea of the search process is to find the MAP probability $MAP(\mathbf{i},\mathbf{o})$ by finding the upper bounds of the intermediate MAP probabilities $MAP(\mathbf{i}_{inter},\mathbf{o})$.

MAP hypothesis can be categorized into two portions. The first portion involves finding intermediate *upper bounds* of MAP probability, $MAP(\mathbf{i}_{inter},\mathbf{o})$, and the second portion involves *improving* these bounds to arrive at the exact MAP solution, $MAP(\mathbf{i}_{MAP},\mathbf{o})$. These two portions are intertwined and performed alternatively to effectively improve on the intermediate MAP upper bounds. These upper bounds and final solution are calculated by performing inference on the probabilistic error model using Shenoy–Shafer algorithm [20,21].

Shenoy–Shafer algorithm is based on local computation mechanism. The probability distributions of the locally connected variables are propagated to get the joint probability distribution of the entire network from which any individual or joint probability distributions can be calculated. The Shenoy–shafer algorithm involves the following crucial information and calculations.

4.1.1. Valuations

The valuations are functions based on the prior probabilities of the variables in the network. A valuation for a variable Y_i can be given as $\phi_{Y_i} = P(Y_i, Pa(Y_i))$ where $Pa(Y_i)$ are the parents of Y_i. For variables without parents, the valuations can be given as $\phi_{Y_i} = P(Y_i)$. These valuations can be derived from the CPTs (discussed in Section 3) as shown in Table 2.

4.1.2. Combination

Combination is a pointwise multiplication mechanism conducted to combine the information provided by the operand functions. A combination of two given functions f_a and f_b can be written as $f_{a\cup b} = f_a \otimes f_b$, where a and b are set of variables. Table 3 provides an example.

4.1.3. Marginalization

Given a function $f_{a\cup b}$, where a and b are set of variables, marginalizing over b provides a function of a and that can be given as $f_a = f^{mar(b)}_{a\cup b}$. This process provides the marginals of a single variable or a set of variables. Generally the process can be done by summing or maximizing or minimizing over the *marginalizing variables* in b. Normally the summation operator is used to calculate the probability distributions. In MAP hypothesis both summation and maximization operators are involved.

The computational scheme of the Shenoy–Shafer algorithm is based on *fusion* algorithm proposed by Shenoy in [22]. The construction mechanism of the binary Join tree uses this fusion algorithm. *Please note that for the sake of simplicity and better understanding, we use a simpler example as shown in* Fig. 3a *to explain the fusion algorithm, construction of binary Join tree and inference in binary Join tree.*

Given a probabilistic network, like our probabilistic error model in Fig. 3a, the *fusion* method can be explained as follows:

1. The valuations provided are associated with the corresponding variables forming a valuation network as shown in Fig. 3b. In our example, the valuations are ϕ_{I1} for $\{I1\}$, ϕ_{I2} for $\{I2\}$, ϕ_{X1} for $\{X1,I1,I2\}$, ϕ_{X2} for $\{X2,I1,I2\}$, ϕ_{O1} for $\{O1,X1,X2\}$.
2. A variable $Y_i \in \mathbf{Y}$ for which the probability distribution has to be found out is selected. In our example let us say we select $I1$.

Table 2
Valuations of the variables derived from corresponding CPTs.

CPT

Error-free AND

$P(X1=1\|I1,I2)$	$P(I2=0)=1$	$P(I2=1)=1$
$P(I1=0)=1$	0	0
$P(I1=1)=1$	0	1

Error-prone AND

$P(X4=1\|I1,I2)$	$P(I2=0)=1$	$P(I2=1)=1$
$P(I1=0)=1$	ε	ε
$P(I1=1)=1$	ε	1-ε

Input

$P(I1=0)$	0.5
$P(I1=1)$	0.5

Valuation

Error-free AND

$X1$	$I1$	$I2$	ϕ_{X1}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Error-prone AND

$X4$	$I1$	$I2$	ϕ_{X4}
0	0	0	1-ε
0	0	1	1-ε
0	1	0	1-ε
0	1	1	ε
1	0	0	ε
1	0	1	ε
1	1	0	ε
1	1	1	1-ε

Input

$I1$	ϕ_{I1}
0	0.5
1	0.5

Table 3
Combination.

x	y	f_{xy}
0	0	1
0	1	1
1	0	1
1	1	0

y	z	f_{yz}
0	0	1
0	1	0
1	0	0
1	1	0

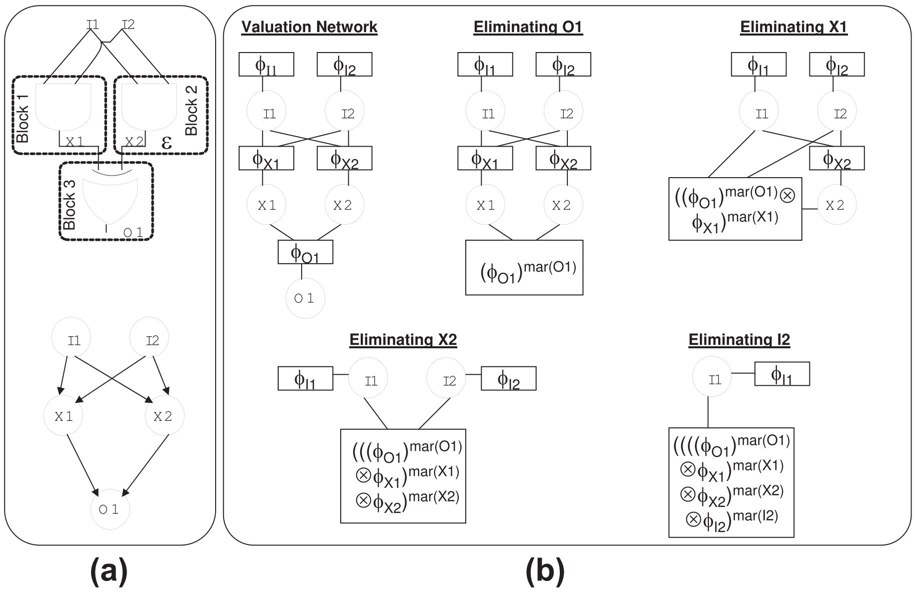
x	y	z	$f_{xyz} = f_{xy} \otimes f_{yz}$
0	0	0	1x1
0	0	1	1x0
0	1	0	1x0
0	1	1	1x0
1	0	0	1x1
1	0	1	1x0
1	1	0	0x0
1	1	1	0x0

3. Choose an arbitrary variable elimination order. For the example network let us choose the order as O1, X1, X2, I2. When a variable Y_i is eliminated, the functions associated with that variable $f^1_{Y_i}, \ldots, f^j_{Y_i}$ are combined and the resulting function is marginalized over Y_i. It can be represented as, $(f^1_{Y_i} \otimes \cdots \otimes f^j_{Y_i})^{mar(Y_i)}$. This function is then associated with the neighbors of Y_i. This process is repeated until all the variables in the elimination order are removed. Fig. 3 illustrates the fusion process.
   Eliminating $O1$ yields the function $(\phi_{O1})^{mar(O1)}$ associated to neighbors $X1$, $X2$.
   Eliminating $X1$ yields the function $((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)}$ associated to neighbors $X2$, $I1$, $I2$.
   Eliminating $X2$ yields the function $(((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)}$ associated to neighbors $I1$, $I2$.
   Eliminating $I2$ yields the function $((((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)} \otimes \phi_{I2})^{mar(I2)}$ associated to neighbor $I1$.
   According to a theorem presented in [21], combining the functions associated with $I1$ yields the probability distribution of $I1$. $\phi_{I1} \otimes ((((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)} \otimes \phi_{I2})^{mar(I2)} = (\phi_{I1} \otimes \phi_{O1} \otimes \phi_{X1} \otimes \phi_{X2} \otimes \phi_{I2})^{mar(O1,X1,X2,I2)} =$ Probability distribution of I1 [21]. Note that the function $\phi_{I1} \otimes \phi_{O1} \otimes \phi_{X1} \otimes \phi_{X2} \otimes \phi_{I2}$ represents the joint probability of the entire probabilistic error model.
4. The above process is repeated for all the other variables individually.

To perform efficient computation, an additional undirected network called *Join tree* is formed from the original probabilistic network. The nodes of the Join tree contains *clusters* of nodes from the original probabilistic network. The information of locally connected variables, provided through valuations, is propagated in the Join tree by *message passing* mechanism. To increase the computational efficiency of the Shenoy–Shafer algorithm, a special kind of Join tree named *binary Join tree* is used. In a binary Join tree, every node is connected to no more than three neighbors. In this framework only two functions are combined at an instance, thereby reducing the computational complexity. We will first explain the method to construct a binary Join tree, as proposed by Shenoy in [21], and then we will explain the inference scheme using message passing mechanism.

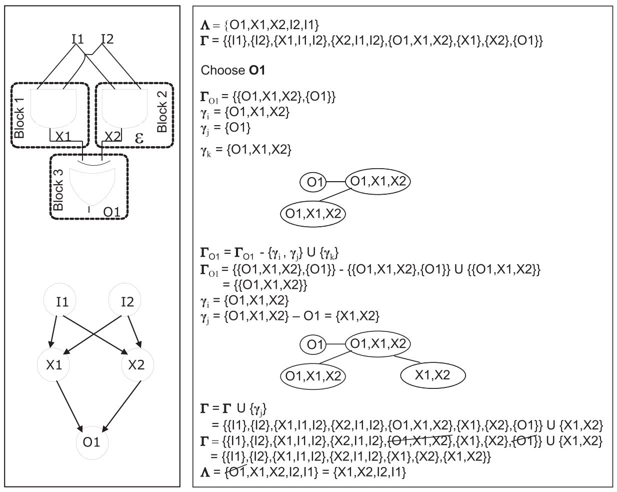
Construction of Binary Join Tree: The binary Join tree is constructed using the fusion algorithm. Please consider the probabilistic error model in Fig. 3a as an example. Fig. 4 illustrates the binary Join tree construction method for the probabilistic error model in Fig. 3a. Fig. 4a explains a portion of the construction method for the first chosen variable, here it is $O1$. Fig. 4b illustrates the entire method. The construction of binary Join tree can be explained as follows:

1. To begin with we have,
   $\Lambda \Rightarrow$ A set that contains all the variables from the original probabilistic network. In our example, $\Lambda = \{I1, I2, X1, X2, O1\}$.
   $\Gamma \Rightarrow$ A set that contains the subsets of variables, that should be present in the binary Join tree. i.e., the subsets that denote the valuations and the subsets whose probability distributions are needed to be calculated. In our example, let us say that we need to calculate the individual probability distributions of all the variables. Then we have, $\Gamma = \{\{I1\}, \{I2\}, \{X1,I1,I2\}, \{X2,I1,I2\}, \{O1,X1,X2\}, \{X1\}, \{X2\}, \{O1\}\}$.
   $\mathcal{N} \Rightarrow$ A set that contains the nodes of the binary Join tree and it is initially null.
   $\mathcal{E} \Rightarrow$ A set that contains the edges of the binary Join tree and it is initially null.
   We also need an order in which we can choose the variables to form the binary Join tree. In our example, since the goal is to find out the probability distribution of I1, this order should

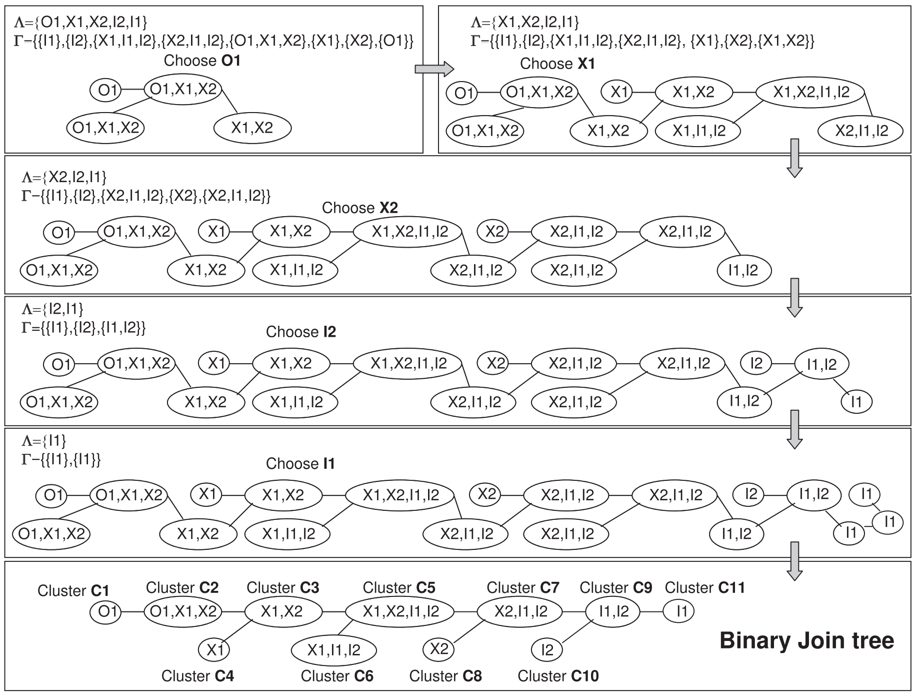


I1
I2
Block 1
Block 2
Block 3
X1
X2
O1
ε
Valuation Network
Eliminating O1
Eliminating X1
Eliminating X2
Eliminating I2
ϕ_{I1}
ϕ_{I2}
ϕ_{X1}
ϕ_{X2}
ϕ_{O1}
$(\phi_{O1})^{mar(O1)}$
$((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)}$
$(((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)}$
$((((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)} \otimes \phi_{I2})^{mar(I2)}$
(a)
(b)

Fig. 3. Illustration of the fusion algorithm.


I1
I2
Block 1
Block 2
Block 3
X1
X2
O1
ε
Λ = {O1,X1,X2,I2,I1}
Γ = {{I1},{I2},{X1,I1,I2},{X2,I1,I2},{O1,X1,X2},{X1},{X2},{O1}}
Choose O1
Γ_O1 = {{O1,X1,X2},{O1}}
γ_i = {O1,X1,X2}
γ_j = {O1}
γ_k = {O1,X1,X2}
Γ_O1 = Γ_O1 - {γ_i , γ_j} U {γ_k}
Γ_O1 = {{O1,X1,X2},{O1}} - {{O1,X1,X2},{O1}} U {{O1,X1,X2}}
= {{O1,X1,X2}}
γ_i = {O1,X1,X2}
γ_j = {O1,X1,X2} – O1 = {X1,X2}
X1,X2
Γ = Γ U {γ_j}
= {{I1},{I2},{X1,I1,I2},{X2,I1,I2},{O1,X1,X2},{X1},{X2},{O1}} U {X1,X2}
Γ = {{I1},{I2},{X1,I1,I2},{X2,I1,I2},~~{O1,X1,X2}~~,{X1},{X2},~~{O1}~~} U {X1,X2}
= {{I1},{I2},{X1,I1,I2},{X2,I1,I2},{X1},{X2},{X1,X2}}
Λ = {~~O1~~,X1,X2,I2,I1} = {X1,X2,I2,I1}

(a)



(b)

Fig. 4. (a) Probabilistic error model as shown in Fig. 3 for which the binary Join tree has to be constructed and the partial illustration of Binary Join tree construction method for the first chosen variable. (b) Complete illustration of Binary Join tree construction method.

reflect the variable elimination order ($O1,X1,X2,I2,I1$) used in fusion algorithm.
$|\Gamma_Y|$ indicates the number of elements in set Γ_Y.

```
2. 1:  while $|\Gamma| > 1$ do
   2:    Choose a variable $Y \in \Lambda$
   3:    $\Gamma_Y = \{\gamma_i \in \Gamma | Y \in \gamma_i\}$
   4:    while $|\Gamma_Y| > 1$ do
   5:      Choose $\gamma_i \in \Gamma_Y$ and $\gamma_j \in \Gamma_Y$ such that
         $|\gamma_i \cup \gamma_j| \leqslant |\gamma_m \cup \gamma_n|$ for all $\gamma_m, \gamma_n \in \Gamma_Y$
   6:      $\gamma_k = \gamma_i \cup \gamma_j$
   7:      $\mathcal{N} = \mathcal{N} \cup \{\gamma_i\} \cup \{\gamma_j\} \cup \{\gamma_k\}$
   8:      $\mathcal{E} = \mathcal{E} \cup \{\{\gamma_i, \gamma_k\}, \{\gamma_j, \gamma_k\}\}$
   9:      $\Gamma_Y = \Gamma_Y - \{\gamma_i, \gamma_j\}$
   10:     $\Gamma_Y = \Gamma_Y \cup \{\gamma_k\}$
   11:   end while
   12:   if $|\Lambda| > 1$ then
   13:     Take $\gamma_i$ where $\gamma_i = \Gamma_Y$
   14:     $\gamma_j = \gamma_i - \{Y\}$
   15:     $\mathcal{N} = \mathcal{N} \cup \{\gamma_i\} \cup \{\gamma_j\}$
   16:     $\mathcal{E} = \mathcal{E} \cup \{\{\gamma_i, \gamma_j\}\}$
   17:     $\Gamma = \Gamma \cup \{\gamma_j\}$
   18:   end if
   19:   $\Gamma = \Gamma - \{\gamma_i \in \Gamma | Y \in \gamma_i\}$
   20:   $\Lambda = \Lambda - \{Y\}$
   21: end while
```

3. The final structure will have some duplicate clusters. Two neighboring duplicate clusters can be merged into one, if the merged node does not end up having more than three neighbors. After merging the duplicate nodes we get the binary Join tree.

Note that, even though the binary Join tree is constructed with a specific variable elimination order for finding out the probability distribution of $I1$, it can be used to find out the probability distributions of other variables too.

Inference in binary Join tree: Inference in a binary Join tree is performed using message passing mechanism. Initially all the valuations are associated to the appropriate clusters. In our example, at Fig. 5, which represents the binary Join tree constructed from the probabilistic error model in Fig. 3a, the valuations are associated to these following clusters:

- ϕ_{I1} associated to cluster **C11**
- ϕ_{I2} associated to cluster **C10**
- ϕ_{X1} associated to cluster **C6**
- ϕ_{X2} associated to cluster **C7**
- ϕ_{O1} associated to cluster **C2**

A message passed from cluster b, containing a variable set $\mathbf{B}$, to cluster c, containing a variable set $\mathbf{C}$ can be given as,

$$M_{b\to c} = \left(\phi_b \prod_{a\neq c} M_{a\to b}\right)^{mar(\mathbf{B}\setminus\mathbf{C})} = \sum_{\forall \mathbf{B}\setminus\mathbf{C}} \left(\phi_b \prod_{a\neq c} M_{a\to b}\right) \tag{7}$$

where ϕ_b is the valuation associated with cluster b. If cluster b is not associated with any valuation, then this function is omitted from the equation. The message from cluster b can be sent to cluster c only after cluster b receives messages from all its neighbors other than c. The resulting function is marginalized over the variables in cluster b that are not in cluster c. Note that summation operator is used to marginalize variables. To calculate the probability distribution of a variable Y_i, the cluster having that variable alone is taken as root and the messages are passed towards this root. Probability of Y_i, $P(Y_i)$, is calculated at the root. In our example, at Fig. 5a, to find the probability distribution of $I1$, the cluster **C11** is chosen as the root. The messages from all the leaf clusters are sent towards **C11** and finally the probability distribution of I1 can be calculated as, $P(I1) = M_{\mathbf{C}9\to\mathbf{C}11} \otimes \phi_{I1}$. Also note that the *order of the marginalizing variables* is $O1, X1, X2, I2$ which exactly reflects the elimination order used to construct the binary Join tree. As we mentioned before, this binary Join tree can be used to calculate probability distributions of other variables also. In our example, at Fig. 5b, to find out the probability distribution of $O1$, cluster **C1** is chosen as root and the messages from the leaf clusters are passed towards **C1** and finally the probability distribution of $O1$ can be calculated as, $P(O1) = M_{\mathbf{C}2\to\mathbf{C}1}$. Note that the *order of the marginalizing variables* changes to $I1, I2, X1, X2$. We can also calculate joint probability distributions of the set of variables that forms a cluster in the binary Join tree. In our example, the joint probability $P(I1,I2)$ can be calculated by assigning cluster **C9** as root. In this fashion, the probability distributions of any individual variable or a set of variables can be calculated by choosing appropriate root cluster and sending the messages towards this root. During these operations some of the calculations are not modified and so performing them again will prove inefficient. Using the binary Join tree structure these calculations can be stored thereby eliminating the redundant recalculation. In the binary Join tree, between any two clusters b and c, both the messages $M_{b\to c}$ and $M_{c\to b}$ are stored. Fig. 5c illustrates this phenomenon using our example.

Inference in binary Join tree with evidence: If an evidence set $\mathbf{e}$ is provided, then the additional valuations $\{e_{Y_i} | Y_i \in \mathbf{e}\}$ provided by the evidences has to be associated with the appropriate clusters. A valuation e_{Y_i} for a variable Y_i can be associated with a cluster having Y_i alone. In our example, if the variable $O1$ is evidenced, $\mathbf{e} = O1 = 1$, then the corresponding valuation e_{O1} can be associated with cluster **C1**, as shown in Fig. 6a. While finding the probability distribution of a variable Y_i, the inference mechanism (as explained

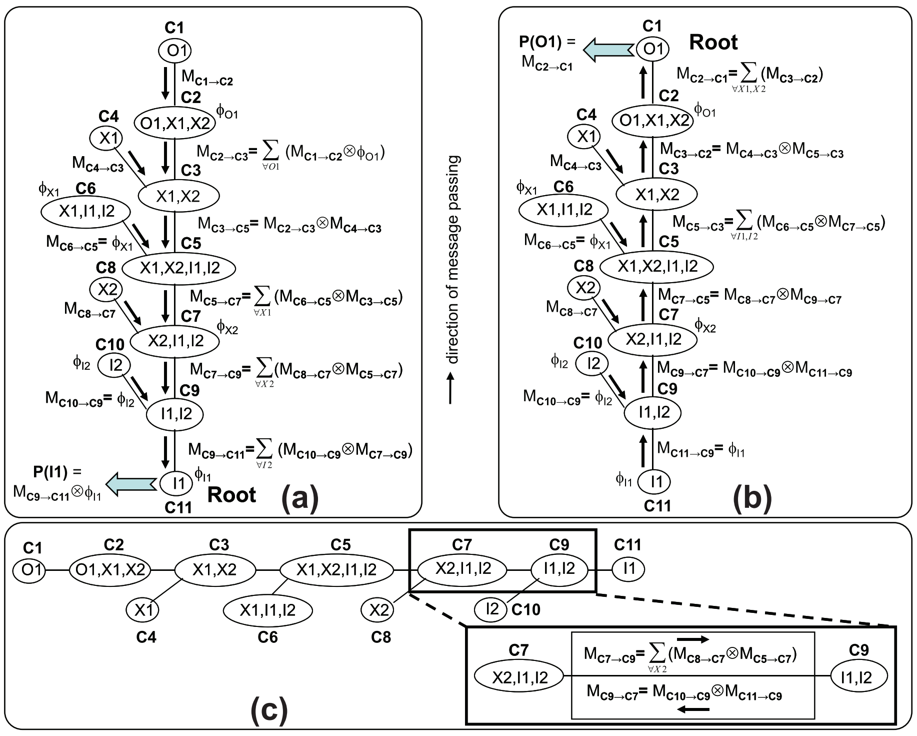
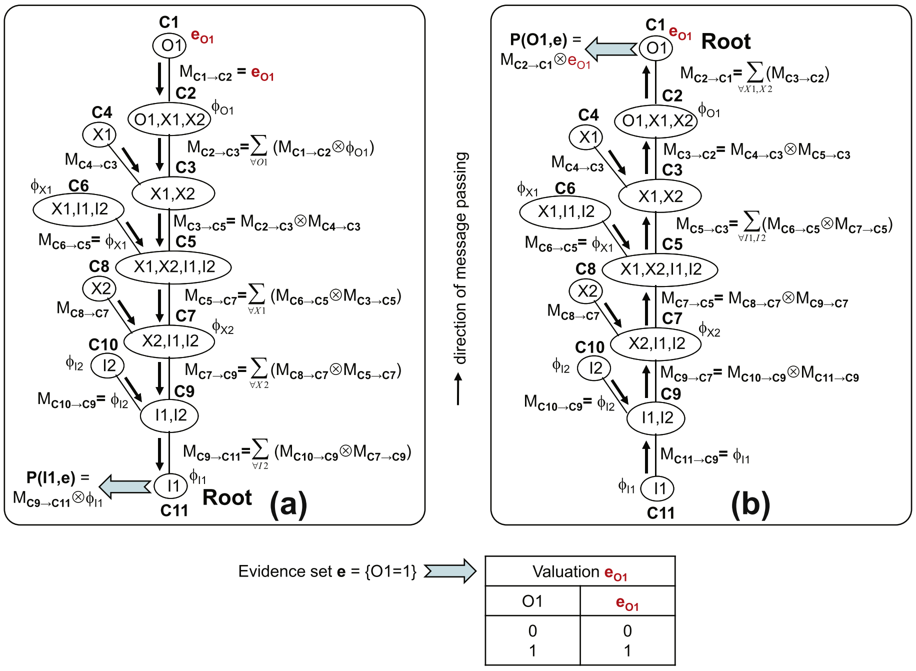


Fig. 5. Illustration of the message passing mechanism in the binary Join tree constructed in Fig. 4 from the probabilistic error model given in Fig. 3a. (a) Message passing with cluster **C11** as root. (b) Message passing with cluster **C1** as root. (c) Message storage mechanism.

before) with an evidence set $\mathbf{e}$ will give the joint probability $P(Y_i,\mathbf{e})$ instead of $P(Y_i)$. From $P(Y_i,\mathbf{e})$, $P(\mathbf{e})$ is calculated as, $P(\mathbf{e}) = \sum_{Y_i} P(Y_i,\mathbf{e})$. Calculation of the probability of evidence $P(\mathbf{e})$ is crucial for MAP calculation.

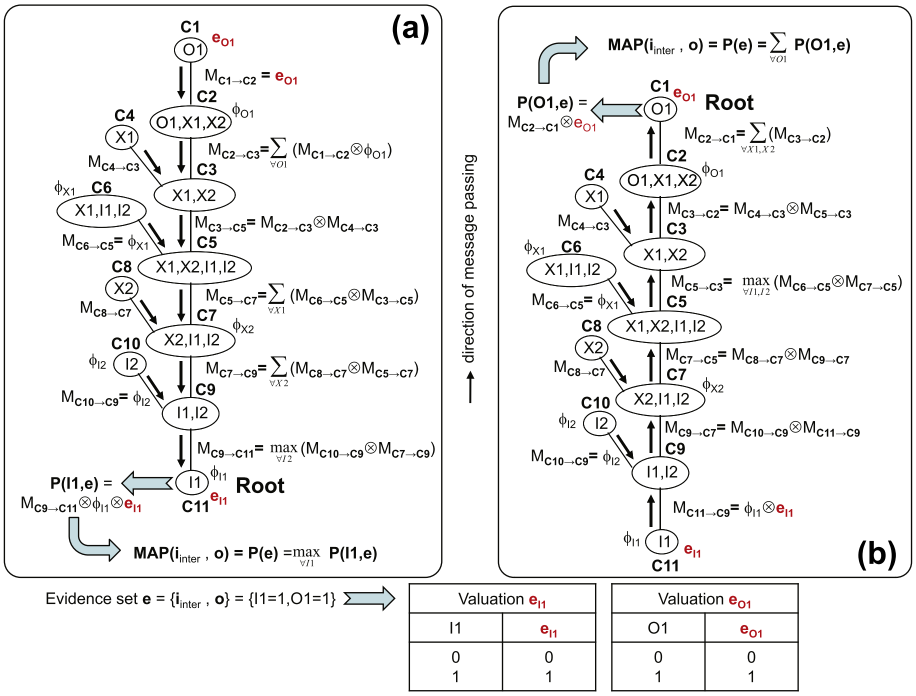
Inference in binary Join tree for MAP calculation: The MAP probabilities $MAP(\mathbf{i}_{inter},\mathbf{o})$ are calculated by performing inference on the binary Join tree with evidences $\mathbf{i}_{inter}$ and $\mathbf{o}$, as shown in Fig. 7. Let us say that we have an evidence set $\mathbf{e} = \{\mathbf{i}_{inter},\mathbf{o}\}$, then $MAP(\mathbf{i}_{inter},\mathbf{o}) = P(\mathbf{e})$. So eventually we have to calculate $P(\mathbf{e})$ by performing inference in the binary join tree. *For a given partial instantiation $\mathbf{i}_{inter}$, $MAP(\mathbf{i}_{inter},\mathbf{o})$ is calculated by maximizing over the MAP variables. This calculation can be done by modifying the message passing scheme to accommodate maximization over MAP variables. So for MAP calculation, the marginalization operation involves both maximization and summation functions. The maximization is performed over the MAP variables in* $\mathbf{I}$ *and the summation is performed over all the other variables in* $\mathbf{X}$ *and* $\mathbf{O}$. For MAP, the message passing scheme from cluster b to cluster c is modified from Eq. (7) as,



C1
O1
e_{O1}
$M_{C1\to C2} = e_{O1}$
C2
O1,X1,X2
ϕ_{O1}
C4
X1
$M_{C4\to C3}$
$M_{C2\to C3} = \sum_{\forall O1}(M_{C1\to C2} \otimes \phi_{O1})$
C3
X1,X2
ϕ_{X1}
C6
X1,I1,I2
$M_{C3\to C5} = M_{C2\to C3} \otimes M_{C4\to C3}$
$M_{C6\to C5} = \phi_{X1}$
C5
C8
X1,X2,I1,I2
X2
$M_{C8\to C7}$
$M_{C5\to C7} = \sum_{\forall X1}(M_{C6\to C5} \otimes M_{C3\to C5})$
C7
X2,I1,I2
ϕ_{X2}
C10
ϕ_{I2}
I2
$M_{C7\to C9} = \sum_{\forall X2}(M_{C8\to C7} \otimes M_{C5\to C7})$
C9
$M_{C10\to C9} = \phi_{I2}$
I1,I2
$M_{C9\to C11} = \sum_{\forall I2}(M_{C10\to C9} \otimes M_{C7\to C9})$
P(I1,e) =
$M_{C9\to C11} \otimes \phi_{I1}$
I1
ϕ_{I1}
C11
Root
(a)
direction of message passing
P(O1,e) =
$M_{C2\to C1} \otimes e_{O1}$
Root
$M_{C2\to C1} = \sum_{\forall X1,X2}(M_{C3\to C2})$
$M_{C3\to C2} = M_{C4\to C3} \otimes M_{C5\to C3}$
$M_{C5\to C3} = \sum_{\forall I1,I2}(M_{C6\to C5} \otimes M_{C7\to C5})$
$M_{C7\to C5} = M_{C8\to C7} \otimes M_{C9\to C7}$
$M_{C9\to C7} = M_{C10\to C9} \otimes M_{C11\to C9}$
$M_{C11\to C9} = \phi_{I1}$
(b)
Evidence set **e** = {O1=1}

Valuation e_{O1}	
O1	e_{O1}
0	0
1	1

Fig. 6. Illustration of the message passing mechanism with evidence in the binary Join tree constructed in Fig. 4 from the probabilistic error model given in Fig. 3a. (a) Message passing with cluster **C11** as root. (b) Message passing with cluster **C1** as root.



(a)
C1
O1
e_{O1}
$M_{C1\to C2} = e_{O1}$
C2
O1,X1,X2
ϕ_{O1}
C4
X1
$M_{C4\to C3}$
$M_{C2\to C3} = \sum_{\forall O1}(M_{C1\to C2} \otimes \phi_{O1})$
C3
X1,X2
ϕ_{X1}
C6
X1,I1,I2
$M_{C3\to C5} = M_{C2\to C3} \otimes M_{C4\to C3}$
$M_{C6\to C5} = \phi_{X1}$
C5
C8
X1,X2,I1,I2
X2
$M_{C8\to C7}$
$M_{C5\to C7} = \sum_{\forall X1}(M_{C6\to C5} \otimes M_{C3\to C5})$
C7
X2,I1,I2
ϕ_{X2}
C10
ϕ_{I2}
I2
$M_{C7\to C9} = \sum_{\forall X2}(M_{C8\to C7} \otimes M_{C5\to C7})$
C9
$M_{C10\to C9} = \phi_{I2}$
I1,I2
$M_{C9\to C11} = \max_{\forall I2}(M_{C10\to C9} \otimes M_{C7\to C9})$
P(I1,e) =
$M_{C9\to C11} \otimes \phi_{I1} \otimes e_{I1}$
I1
ϕ_{I1}
Root
C11
e_{I1}
$MAP(i_{inter}, o) = P(e) = \max_{\forall I1} P(I1,e)$
direction of message passing
$MAP(i_{inter}, o) = P(e) = \sum_{\forall O1} P(O1,e)$
P(O1,e) =
$M_{C2\to C1} \otimes e_{O1}$
Root
$M_{C2\to C1} = \sum_{\forall X1,X2}(M_{C3\to C2})$
$M_{C3\to C2} = M_{C4\to C3} \otimes M_{C5\to C3}$
$M_{C5\to C3} = \max_{\forall I1,I2}(M_{C6\to C5} \otimes M_{C7\to C5})$
$M_{C7\to C5} = M_{C8\to C7} \otimes M_{C9\to C7}$
$M_{C9\to C7} = M_{C10\to C9} \otimes M_{C11\to C9}$
$M_{C11\to C9} = \phi_{I1} \otimes e_{I1}$
(b)
Evidence set **e** = {i_{inter} , **o**} = {I1=1,O1=1}

Valuation e_{I1}	
I1	e_{I1}
0	0
1	1

Valuation e_{O1}	
O1	e_{O1}
0	0
1	1

Fig. 7. Illustration of the message passing mechanism for MAP calculation in the binary Join tree constructed in Fig. 4 from the probabilistic error model given in Fig. 3a. (a) Message passing with cluster **C11** as root. (b) Message passing with cluster **C1** as root.

$$M_{b\to c} = \max_{\{\mathbf{I}_b\}\in\{\mathbf{B}\setminus\mathbf{C}\}} \sum_{\{\mathbf{X}_b\cup\mathbf{O}_b\}\in\{\mathbf{B}\setminus\mathbf{C}\}} \phi_b \prod_{a\neq c} M_{a\to b} \tag{8}$$

where $\mathbf{I}_b \subseteq \mathbf{I}$, $\mathbf{X}_b \subseteq \mathbf{X}$, $\mathbf{O}_b \subseteq \mathbf{O}$ and $\{\mathbf{I}_b, \mathbf{X}_b, \mathbf{O}_b\} \in \mathbf{B}$. For example, as shown in Fig. 7, let us consider the binary Join tree constructed from the probabilistic error model given in Fig. 3a. This probabilistic model consists for two MAP variables $\{I1,I2\}$. Let us consider the partial instantiation, $\mathbf{i}_{inter} = \{I1 = 1\}$. To find the intermediate MAP probability, $MAP(\mathbf{i}_{inter},\mathbf{o})$, the evidence set should be $\mathbf{e} = \{\mathbf{i}_{inter}, \mathbf{o}\} = \{I1 = 1, O1 = 1\}$. The corresponding valuations, e_{I1} and e_{O1}, resulting from the evidence set, should be associated with cluster **C**11 and **C**1 respectively. During message passing in the binary Join tree shown in Fig. 7, note that when the marginalizing variable is a MAP variable, the maximizing operator is used instead of summing operator as shown in Figs. 5 and 6. Please note the difference between the messages $M_{\mathbf{C}9\to\mathbf{C}11}$ and $M_{\mathbf{C}5\to\mathbf{C}3}$ in Figs. 6 and 7. This action basically maximizes the resulting joint probability $MAP(\mathbf{i}_{inter},\mathbf{o})$ which is calculated as $P(\mathbf{e})$ by marginalizing over the joint probability acquired at the root cluster, as shown in Fig. 7. Please note that since the marginalizing variable is a MAP variable ($I1$), maximization operator is used in Fig. 7a.

Valid variable elimination order for MAP computation: In the message passing scheme for MAP computation given by Eq. (8), the most important aspect is that the maximization and summation operators in Eq. (8) are non-commutative.

$$\left[\sum_{\mathbf{X}} \max_{\mathbf{I}} P\right](\mathbf{y}) \geqslant \left[\max_{\mathbf{I}} \sum_{\mathbf{X}} P\right](\mathbf{y}) \tag{9}$$

So during message passing in the binary Join tree, the *valid order of the marginalizing variables* or the *valid variable elimination order* should have the summation variables in **X** and **O** before the maximization variables in **I**. A message pass through an invalid variable elimination order can result in a bad upper bound that is stuck at a local maxima and it eventually results in the elimination of some probable instantiations of the MAP variables **I** during the search process. But an invalid elimination order can provide us an initial upper bound of the MAP probability to start with. The closer the invalid variable elimination order to the valid one, the tighter will be the upper bound. In the binary Join tree, any cluster can be chosen as root to get this initial upper bound. For example, in Fig. 7b choosing cluster **C1** as root results in an invalid variable elimination order $I1, I2, X1, X2$ and message pass towards this root can give the initial upper bound. Also it is essential to use a valid variable elimination order during the construction of the binary Join tree so that there is at least one path that can provide a good upper bound.

Now we will explore the variable elimination order through another example as shown in Fig. 8 and explain the mechanism to calculate the MAP solution. Fig. 8 gives the corresponding binary Join tree, for the probabilistic error model given in Fig. 1c, constructed with a valid variable elimination order $(O1,X3,X6,X1,X2,X4,X5,I3,I2,I1)$. In this model, there are three MAP variables $\{I1,I2,I3\}$. The MAP hypothesis on this model results in $\mathbf{i}_{MAP} = \{I1 = 0, I2 = 0, I3 = 0\}$.

The initial upper bound $MAP(\{\},\mathbf{o})$ is calculated by choosing cluster **C2** as root and passing messages towards **C2**. As specified earlier this upper bound can be calculated with any cluster as root. With **C2** as root, an upper bound will most certainly be obtained since the variable elimination order $(I3,I2,I1,X4,X5,X1,X2,X3,X6)$ is an invalid one. But since the maximization variables are at the very beginning of the order, having **C2** as root will yield a looser upper bound. Instead, if **C16** is chosen as root, the elimination order $(O1,X3,X6,X1,I3,X4,X5,I2,I1)$ will be closer to a valid order. So a much tighter upper bound can be achieved. To calculate an intermediate upper bound $MAP(\mathbf{i}_{inter},\mathbf{o})$, the MAP variable I_i newly added to form $\mathbf{i}_{inter}$ is recognized and the cluster having the variable I_i alone is selected as root. By doing this a valid elimination order and proper upper bound can be achieved. For example, to calculate the intermediate upper bound $MAP(\{I1 = 0\},\mathbf{o})$ where the instantiation $\{I1 = 0\}$ is newly added to the initially empty set $\mathbf{i}_{inter}$, a valid elimination order should have the maximization variables $I2, I3$ at the end. To achieve this, cluster **C31** is chosen as root thereby yielding a valid elimination order $(O1,X3,X6,X1,X2,X4,X5,I3,I2)$.

4.2. Calculation of the exact MAP solution

Fig. 9 illustrates the search process for MAP computation in the binary search tree given in Fig. 2 for the probabilistic error model given in Fig. 1c. The calculation of the exact MAP solution $MAP(\mathbf{i}_{MAP},\mathbf{o})$ can be explained as follows:

1. To start with we have the following:
   $\mathbf{I}_{inter}\to$ subset of MAP variables **I**. Initially empty.
   $\mathbf{i}_{inter}\to$ partial instantiation set of MAP variables $\mathbf{I}_{inter}$. Initially empty.
   $\mathbf{i}_{d_1}, \mathbf{i}_{d_2} \to$ partial instantiation sets used to store $\mathbf{i}_{inter}$. Initially empty.
   $\mathbf{i}_{MAP}\to$ MAP instantiation. At first, $\mathbf{i}_{MAP} = \mathbf{i}_{init}$, where $\mathbf{i}_{init}$ is calculated by *sequentially* initializing the MAP variables to a particular instantiation and performing local *taboo search* around the neighbors of that instantiation [29]. Since this method is out of the scope of this paper, we are not explaining it in detail.
   $MAP(\mathbf{i}_{MAP},\mathbf{o})\to$ MAP probability. Initially $MAP(\mathbf{i}_{MAP},\mathbf{o}) = MAP(\mathbf{i}_{init},\mathbf{o})$ calculated by inferencing the probabilistic error model.
   $\upsilon(I_i)\to$ number of values or states that can be assigned to a variable I_i. Since we are dealing with digital signals, $\upsilon(I_i) = 2$ for all i.

2.
```
1:  Calculate $MAP(\mathbf{i}_{inter},\mathbf{o})$. /* This is the initial upper bound
    of MAP probability. */
2:  if $MAP(\mathbf{i}_{inter},\mathbf{o}) \geqslant MAP(\mathbf{i}_{MAP},\mathbf{o})$ then
3:    $MAP(\mathbf{i}_{MAP},\mathbf{o}) = MAP(\mathbf{i}_{inter},\mathbf{o})$
4:  else
5:    $MAP(\mathbf{i}_{MAP},\mathbf{o}) = MAP(\mathbf{i}_{MAP},\mathbf{o})$
6:    $\mathbf{i}_{MAP} = \mathbf{i}_{MAP}$
7:  end if
8:  while $|\mathbf{I}| > 0$ do
9:    Choose a variable $I_i \in \mathbf{I}$.
10:   $\mathbf{I}_{inter} = \mathbf{I}_{inter} \cup \{I_i\}$.
11:   while $\upsilon(I_i) > 0$ do
12:     Choose a value $i_{\upsilon(I_i)}$ of $I_i$
13:     $\mathbf{i}_{d_1} = \mathbf{i}_{inter} \cup \{I_i = i_{\upsilon(I_i)}\}$.
14:     Calculate $MAP(\mathbf{i}_{d_1},\mathbf{o})$ from binary Join tree.
15:     if $MAP(\mathbf{i}_{d_1},\mathbf{o}) \geqslant MAP(\mathbf{i}_{MAP},\mathbf{o})$ then
16:       $MAP(\mathbf{i}_{MAP},\mathbf{o}) = MAP(\mathbf{i}_{d_1},\mathbf{o})$
17:       $\mathbf{i}_{d_2} = \mathbf{i}_{d_1}$
18:     else
19:       $MAP(\mathbf{i}_{MAP},\mathbf{o}) = MAP(\mathbf{i}_{MAP},\mathbf{o})$
20:     end if
21:     $\upsilon(I_i) = \upsilon(I_i) - 1$
22:   end while
23:   $\mathbf{i}_{inter} = \mathbf{i}_{d_2}$
24:   if $|\mathbf{i}_{inter}| = 0$ then
25:     goto line 29
26:   end if
27:   $\mathbf{I} = \mathbf{I} - \{I_i\}$
28: end while
29: if $|\mathbf{i}_{inter}| = 0$ then
```

```
30:         $\mathbf{i}_{MAP} = \mathbf{i}_{MAP}$
31:     else
32:         $\mathbf{i}_{MAP} = \mathbf{i}_{inter}$
33:     end if
```

The pruning of the search process is handled in lines 11–23. After choosing a MAP variable I_i, the partial instantiation set $\mathbf{i}_{inter}$ is updated by adding the best instantiation $I_i = i_{v(I_i)}$ thereby ignoring the other instantiations of I_i.

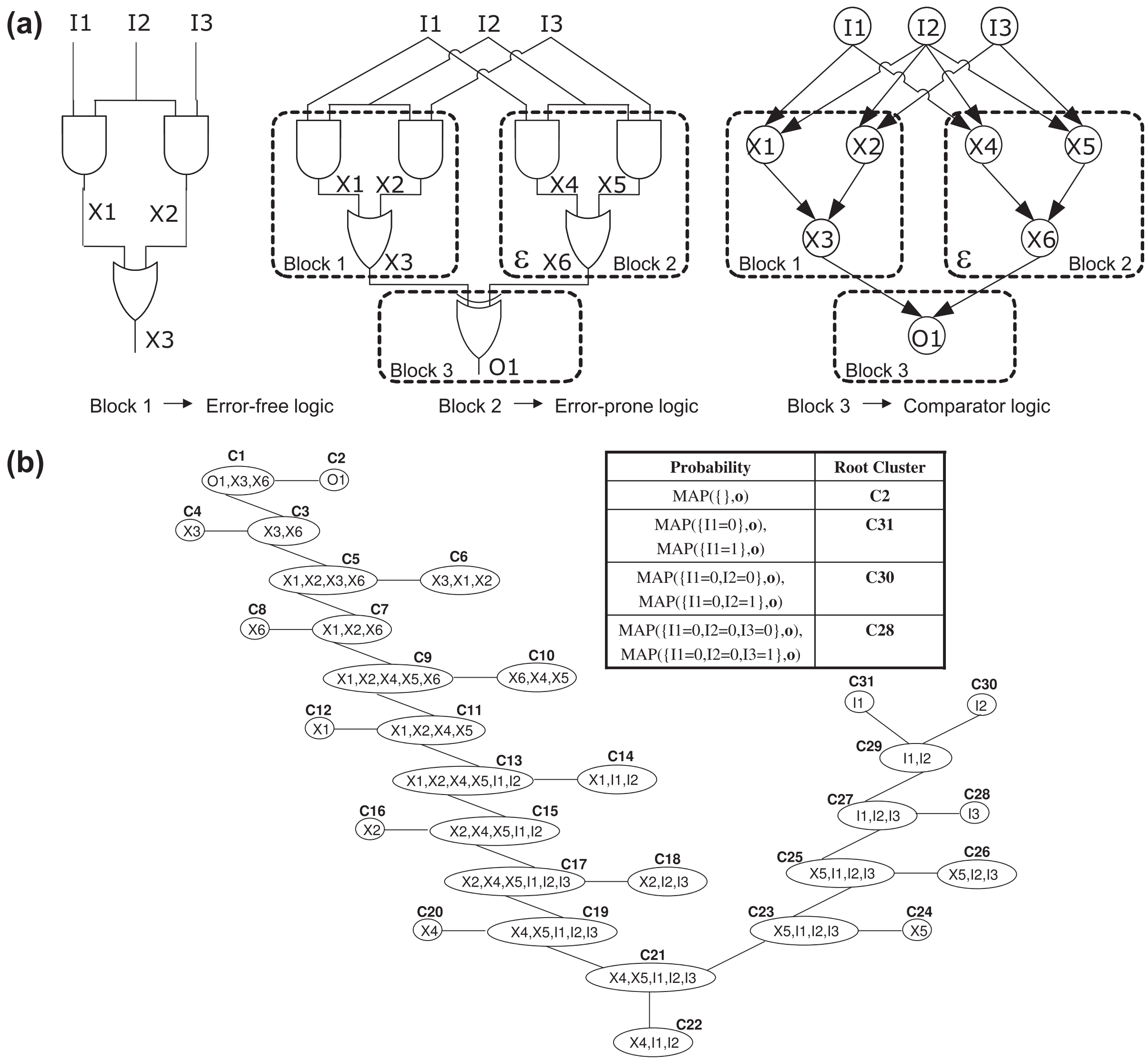
4.3. Calculating the maximum output error probability

According to our error model, the MAP variables represent the primary input signals of the underlying digital logic circuit. So after MAP hypothesis, we will have the input vector which has the highest probability to give an error on the output. The random variables $\mathbf{I}$ that represent the primary input signals are then instantiated with $\mathbf{i}_{MAP}$ and inferenced. So the evidence set for this inference calculation will be $\mathbf{e} = \{\mathbf{i}_{MAP}\}$. The output error probability is obtained by observing the probability distributions of the comparator logic variables $\mathbf{O}$. After inference, the probability distribution $P(O_i, \mathbf{e})$ will be obtained. From this $P(O_i|\mathbf{e})$ can be obtained as, $P(O_i \mid \mathbf{e}) = \frac{P(O_i,\mathbf{e})}{P(\mathbf{e})} = \frac{P(O_i,\mathbf{e})}{\sum_{O_i} P(O_i,\mathbf{e})}$. Finally the maximum output error probability is given by, $\max_i P(O_i = 1|\mathbf{e})$.

4.4. Computational complexity of MAP estimate

The time complexity of MAP depends on that of the depth first branch and bound search on the *input instantiation search tree* and also on that of *inference in binary Join tree*. The former depends on the number of MAP variables and the number of states assigned to each variable. In our case each variable is assigned two states and so the time complexity can be given as $O(2^k)$ where k is the number of MAP variables. This is the worst case time complexity assuming that the search tree is not pruned. If the search tree is pruned, then the time complexity will be $<O(2^k)$.

The time complexity of inference in the binary Join tree depends on the number of cliques q and the size Z of the biggest clique. It can be represented as $q \cdot 2^Z$ and the worst case time complexity can be given as $O(2^Z)$. In any given probabilistic model with N variables, representing a joint probability $P(x_1, \ldots, x_N)$, the corresponding Join tree will have $Z < N$ always [24]. Also depending on the underlying circuit structure, the Join tree of the corresponding



Probability	Root Cluster
MAP({},o)	C2
MAP({I1=0},o), MAP({I1=1},o)	C31
MAP({I1=0,I2=0},o), MAP({I1=0,I2=1},o)	C30
MAP({I1=0,I2=0,I3=0},o), MAP({I1=0,I2=0,I3=1},o)	C28

Fig. 8. (a) Probabilistic error model as shown in Fig. 1c. (b) Corresponding binary Join tree.

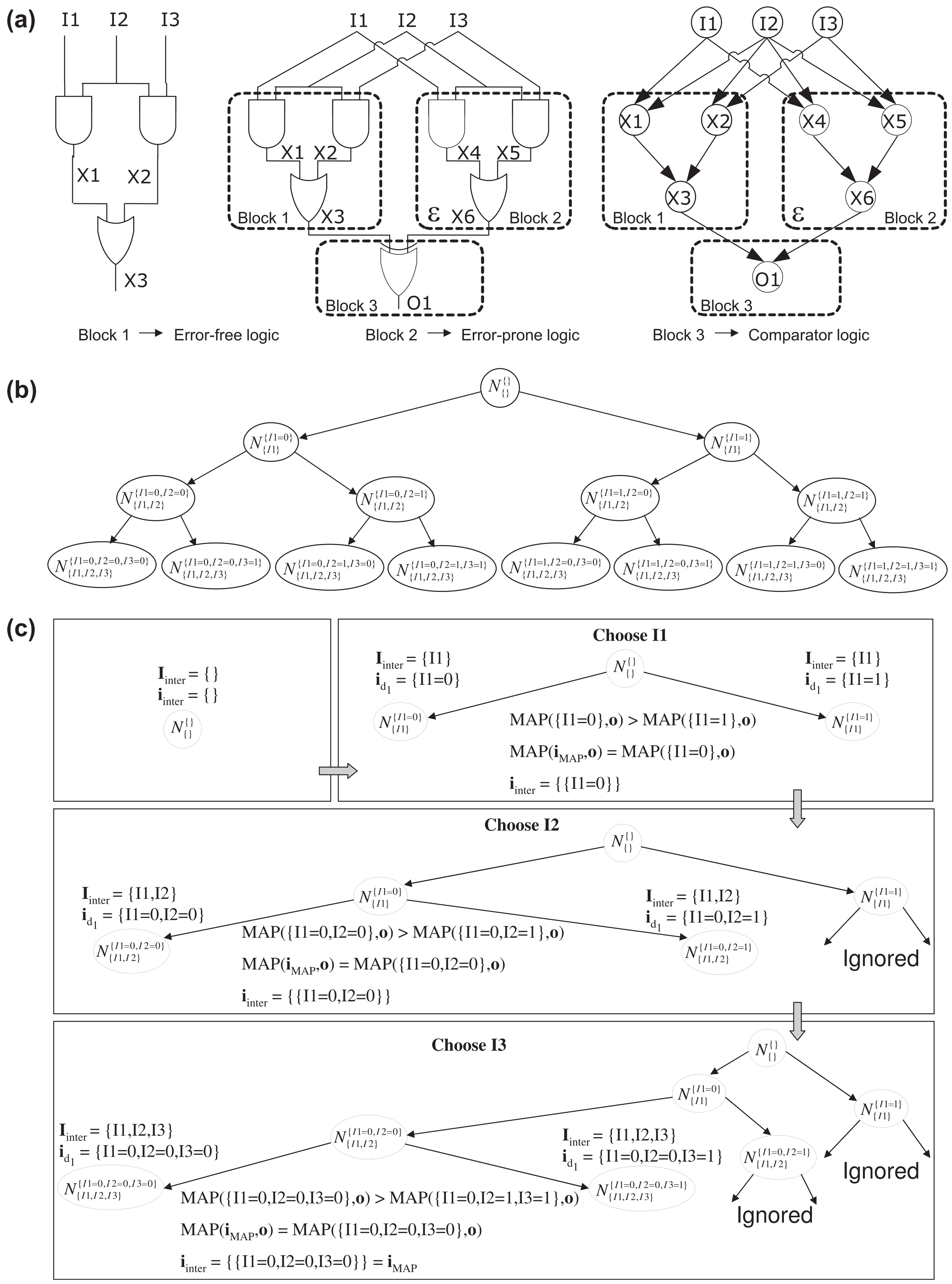

(a)
I1 I2 I3
X1 X2 X3
Block 1 X1 X2 X3
Block 2 X4 X5 ε X6
Block 3 O1
Block 1 → Error-free logic
Block 2 → Error-prone logic
Block 3 → Comparator logic
(b)
$N^{\{\}}_{\{\}}$
$N^{\{I1=0\}}_{\{I1\}}$ $N^{\{I1=1\}}_{\{I1\}}$
$N^{\{I1=0,I2=0\}}_{\{I1,I2\}}$ $N^{\{I1=0,I2=1\}}_{\{I1,I2\}}$ $N^{\{I1=1,I2=0\}}_{\{I1,I2\}}$ $N^{\{I1=1,I2=1\}}_{\{I1,I2\}}$
$N^{\{I1=0,I2=0,I3=0\}}_{\{I1,I2,I3\}}$ … $N^{\{I1=1,I2=1,I3=1\}}_{\{I1,I2,I3\}}$
(c)
$\mathbf{I}_{inter} = \{\}$
$\mathbf{i}_{inter} = \{\}$
Choose I1
$\mathbf{I}_{inter} = \{I1\}$, $\mathbf{i}_{d_1} = \{I1=0\}$
$\mathbf{I}_{inter} = \{I1\}$, $\mathbf{i}_{d_1} = \{I1=1\}$
MAP({I1=0},o) > MAP({I1=1},o)
MAP($\mathbf{i}_{MAP}$,o) = MAP({I1=0},o)
$\mathbf{i}_{inter}$ = {{I1=0}}
Choose I2
$\mathbf{I}_{inter} = \{I1,I2\}$, $\mathbf{i}_{d_1} = \{I1=0,I2=0\}$
$\mathbf{I}_{inter} = \{I1,I2\}$, $\mathbf{i}_{d_1} = \{I1=0,I2=1\}$
MAP({I1=0,I2=0},o) > MAP({I1=0,I2=1},o)
MAP($\mathbf{i}_{MAP}$,o) = MAP({I1=0,I2=0},o)
$\mathbf{i}_{inter}$ = {{I1=0,I2=0}}
Ignored
Choose I3
$\mathbf{I}_{inter} = \{I1,I2,I3\}$, $\mathbf{i}_{d_1} = \{I1=0,I2=0,I3=0\}$
$\mathbf{I}_{inter} = \{I1,I2,I3\}$, $\mathbf{i}_{d_1} = \{I1=0,I2=0,I3=1\}$
MAP({I1=0,I2=0,I3=0},o) > MAP({I1=0,I2=1,I3=1},o)
MAP($\mathbf{i}_{MAP}$,o) = MAP({I1=0,I2=0,I3=0},o)
$\mathbf{i}_{inter}$ = {{I1=0,I2=0,I3=0}} = $\mathbf{i}_{MAP}$
Ignored
Ignored

Fig. 9. illustration of the search process for MAP computation in the binary search tree given in Fig. 2 for the probabilistic error model given in Fig. 1c. (a) Probabilistic error model. (b) Corresponding binary search tree. (c) Search process for MAP computation.

probabilistic error model can have $Z \ll N$ or Z close to N, which in turn determines the time complexity.

Since for every pass in the search tree inference has to be performed in the Join tree to get the upper bound of MAP probability, the worst case time complexity for MAP can be given as $O(2^{k+Z})$. The space complexity of MAP depends on the number of MAP variables for the search tree and on the number of variables N in the probabilistic error model and the size of the largest clique. It can be given by $2^k + N \cdot 2^Z$.

5. Experimental results

The experiments are performed on ISCAS85 and MCNC benchmark circuits. The computing device used is a Sun server with 8 CPUs where each CPU consists of 1.5 GHz UltraSPARC IV processor with at least 32 GB of RAM.

5.1. Experimental procedure for calculating maximum output error probability

Our main goal is to provide the maximum output error probabilities for different gate error probabilities ε. To get the maximum output error probabilities every output signal of a circuit has to be examined through MAP estimation, which is performed through algorithms provided in [30]. The experimental procedure is illustrated as a flow chart in Fig. 10. The steps are as follows:

1. First, an evidence has to be provided to one of the comparator output signal variables in set $\mathbf{O}$ such that $P(O_i = 0) = 0$ and $P(O_i = 1) = 1$. Recall that these variables have a probability distribution based on XOR logic and so giving evidence like this is similar to forcing the output to be wrong.
2. The comparator outputs are evidenced individually and the corresponding input instantiations $\mathbf{i}$ are obtained by performing MAP.
3. Then the primary input variables in the probabilistic error model are instantiated with each instantiation $\mathbf{i}$ and inferenced to get the output probabilities.
4. $P(O_i = 1)$ is noted from all the comparator outputs for each $\mathbf{i}$ and the maximum value gives the maximum output error probability.
5. The entire operation is repeated for different ε values.

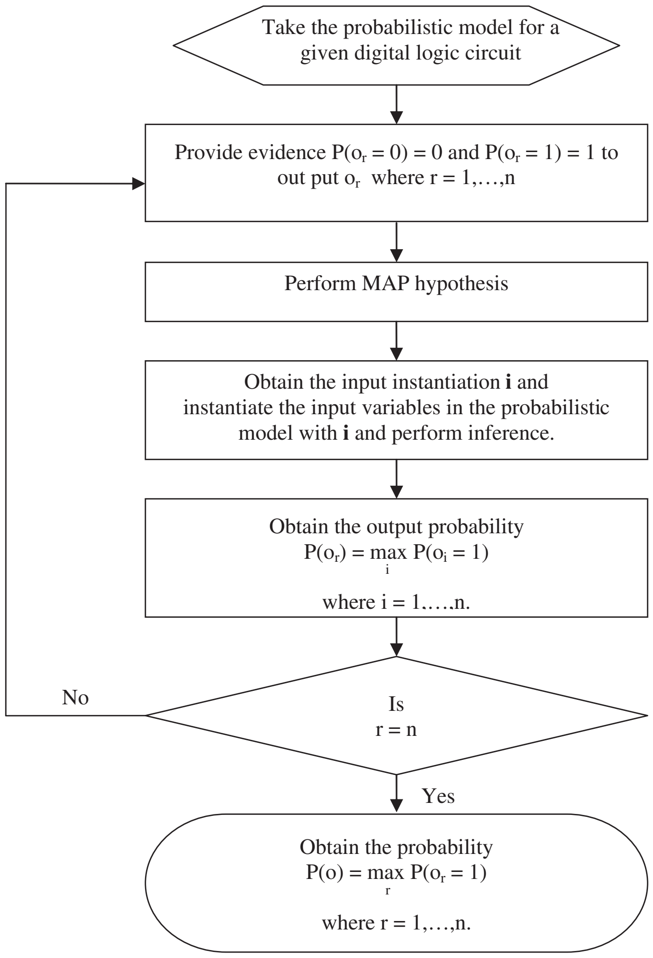

Take the probabilistic model for a given digital logic circuit
Provide evidence $P(o_r = 0) = 0$ and $P(o_r = 1) = 1$ to out put o_r where $r = 1,\dots,n$
Perform MAP hypothesis
Obtain the input instantiation $\mathbf{i}$ and instantiate the input variables in the probabilistic model with $\mathbf{i}$ and perform inference.
Obtain the output probability $P(o_r) = \max_i P(o_i = 1)$ where $i = 1\dots.n$.
Is $r = n$
No
Yes
Obtain the probability $P(o) = \max_r P(o_r = 1)$ where $r = 1,\dots,n$.

Fig. 10. Flow chart describing the experimental setup and process.

5.2. Worst-case input vectors

Table 4 gives the worst-case input vectors got from MAP i.e., the input vectors that gives maximum output error probability. The notable results are as follows:

- In *max_flat* and *voter* the worst-case input vectors from MAP changes with ε, while in *c*17 it does not change.
- In the range {0.005–0.2} for ε, *max_flat* has three different worst-case input vectors while *voter* has two.
- It implies that these worst-case input vectors not only depend on the circuit structure but could dynamically change with ε. This could be of concern for designers as the worst-case inputs might change after gate error probabilities reduce due to error mitigation schemes. Hence, explicit MAP computation would be necessary to judge the maximum error probabilities and worst-case vectors after every redundancy schemes are applied.

5.3. Circuit-specific error bounds for fault-tolerant computation

The error bound for a circuit can be obtained by calculating the gate error probability ε that drives the output error probability of at least one output to a hard bound beyond which the output does not depend on the input signals or the circuit structure. When the output error probability reaches 0.5(50%), it essentially means that the output signal behaves as a non-functional random number generator for at least one input vector and so 0.5 can be treated as a hard bound.

Fig. 11 gives the error bounds for various benchmark circuits. It also shows the comparison between maximum and average output error probabilities with reference to the change in gate error probability ε. These graphs are obtained by performing the experiment for different ε values ranging from 0.005 to 0.1. The average error probabilities are obtained from our previous work by Rejimon et al. [26]. The notable results are as follows:

- The *c*17 circuit consists of 6 NAND gates. The error bound for each NAND gate in *c*17 is $\varepsilon = 0.1055$, which is greater than the conventional error bound for NAND gate, which is 0.08856 [6,7]. The error bound of the same NAND gate in *voter* circuit (contains 10 NAND gates, 16 NOT gates, 8 NOR gates, 15 OR gates and 10 AND gates) is $\varepsilon = 0.0292$, which is lesser than the conventional error bound. This indicates that the error bound for an individual *NAND gate placed in a circuit* can be dependent on the circuit structure. The same can be true for all other logics.

Table 4
Worst-case input vectors from MAP.

Circuits	No. of inputs	Input vector	Gate error probability ε
*c*17	5	01111	0.005–0.2
max_flat	8	00010011	0.005–0.025
		11101000	0.03–0.05
		11110001	0.055–0.2
voter	12	000100110110	0.01–0.19
		111011100010	0.2

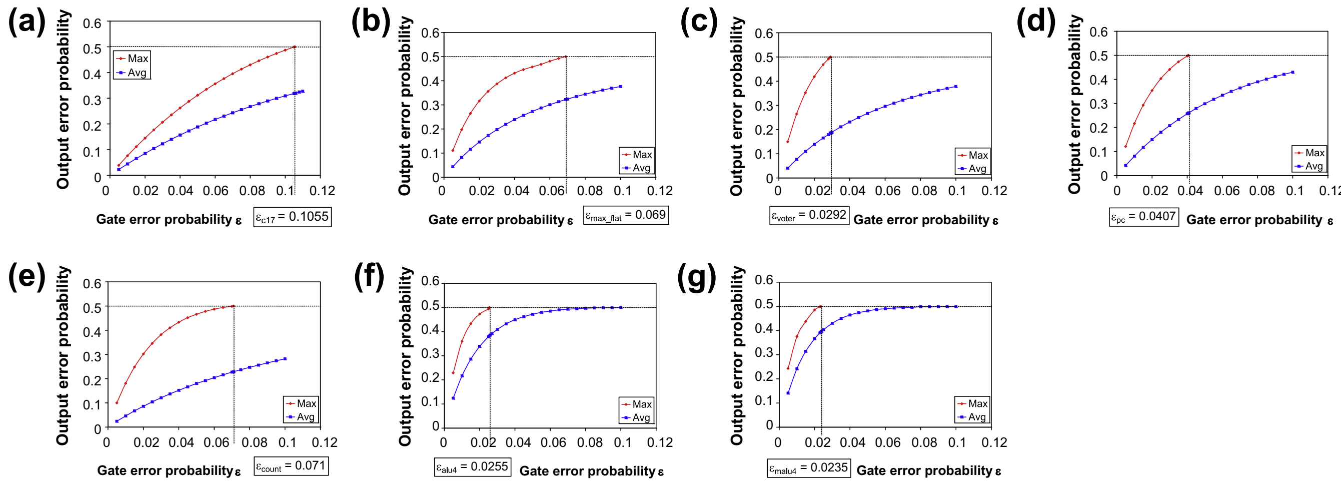


Fig. 11. Circuit-specific error bound for (a) c17, (b) *max_flat*, (c) *voter*, (d) *pc*, (e) *count*, (f) *alu*4, (g) *malu*4. The figures also show the comparison between maximum and average output error probabilities, that indicates the importance of using maximum output error probability to achieve a tighter error bound.

- The maximum output error probabilities are much larger than average output error probabilities, thereby reaching the hard bound for comparatively lower values of ε, making them a very crucial design parameter to achieve tighter error bounds. Only for *alu*4 and *malu*4, the average output error probability reaches the hard bound within $\varepsilon = 0.1(\varepsilon = 0.095$ *for alu*4, $\varepsilon = 0.08$*for malu*4), while the maximum output error probabilities for these circuits reach the hard bound for far lesser gate error probabilities ($\varepsilon = 0.0255$ *for alu*4, $\varepsilon = 0.0235$ *for malu*4).
- While the error bounds for all the circuits, except *c*17, are less than 0.08(8%), the error bounds for circuits like *voter*, *alu*4 and *malu*4 are even less than 0.03(3%) making them highly vulnerable to errors.

Table 5 tabulates the run time for MAP computation. The run time does not change significantly for different ε values and so we provide only one run time which corresponds to all ε values. This is expected as MAP complexity (discussed in Section 4.4) is determined by number of inputs, and number of variables in the largest clique which in turn depends on the circuit complexity. It has to be noted that, even though *pc* has less number of inputs than *count*, it takes much more time to perform MAP estimate due to its complex circuit structure.

5.4. Validation using HSpice simulator

5.4.1. HSpice model

Using external voltage sources error can be induced in any signal and it can be modeled using HSpice [42]. In our HSpice model we have induced error, using external voltage sources, in every gate's output. Consider signal O_f is the original error free output signal and the signal O_p is the error prone output signal and E is the *piecewise linear* (PWL) voltage source that induces error. The basic idea is that the signal O_p is dependent on the signal O_f and the voltage E. Any change of voltage in E will be reflected in O_p. If $E = 0v$, then $O_p = O_f$, and if $E = Vdd(supply\ voltage)$, then $O_p \neq O_f$, thereby inducing error. The data points for the PWL voltage source E are provided by computations on a finite automata which models the underlying error prone circuit where individual gates have a gate error probability ε.

5.4.2. Simulation setup

Note that, for an input vector of the given circuit, a single simulation run in HSpice is not enough to validate the results from our probabilistic model. Also the circuit has to be simulated for each and every possible input vectors to find out the worst-case one. For a given circuit, *the HSpice simulations are conducted for all possible input vectors, where for each vector the circuit is simulated for 1 million runs and the comparator nodes are sampled.* From this data the maximum output error probability and the corresponding worst-case input vector are obtained.

Table 6 gives the comparison between maximum error probabilities achieved from the proposed model and the HSpice simulator at $\varepsilon = 0.05$. The notable results are as follows:

- The simulation results from HSpice almost exactly coincides with those of our error model for all circuits.
- The highest percentage difference of our error model over HSpice is just 1.23%.

Fig. 12a gives the output error probabilities for the entire input vector space of *c*17 with gate error probability $\varepsilon = 0.05$. The notable results are as follows:

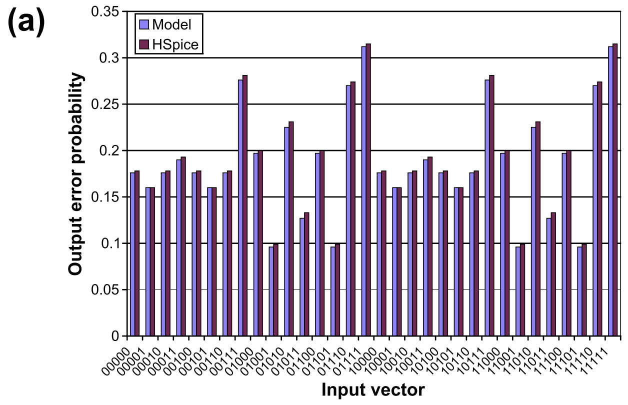
- It can be clearly seen that the results from *both* the probabilistic error model and HSpice simulations show that 01111 gives the maximum output error probability.

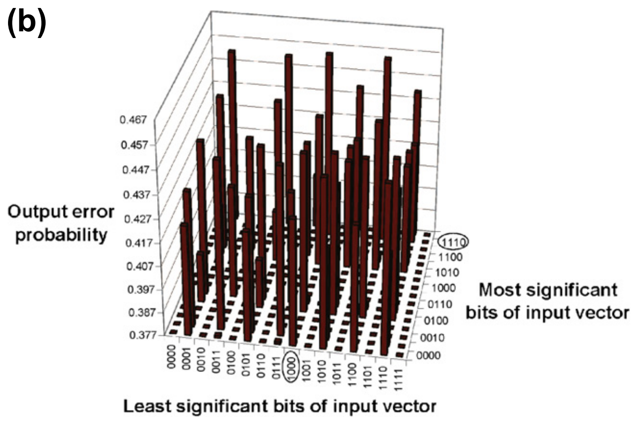
Table 5
Run times for MAP computation.

Circuit	No. of inputs	No. of gates	Time (s)
*c*17	5	6	0.047
max_flat	8	29	0.110
voter	12	59	0.641
pc	27	103	225.297
count	35	144	36.610
*alu*4	14	63	58.626
*malu*4	14	92	588.702

Table 6
Comparison between maximum error probabilities achieved from the proposed model and the HSpice simulator at $\varepsilon = 0.05$.

Circuit	Model	HSpice	% Diff. over HSpice
*c*17	0.312	0.315	0.95
max_flat	0.457	0.460	0.65
voter	0.573	0.570	0.53
pc	0.533	0.536	0.56
count	0.492	0.486	1.23
*alu*4	0.517	0.523	1.15
*malu*4	0.587	0.594	1.18





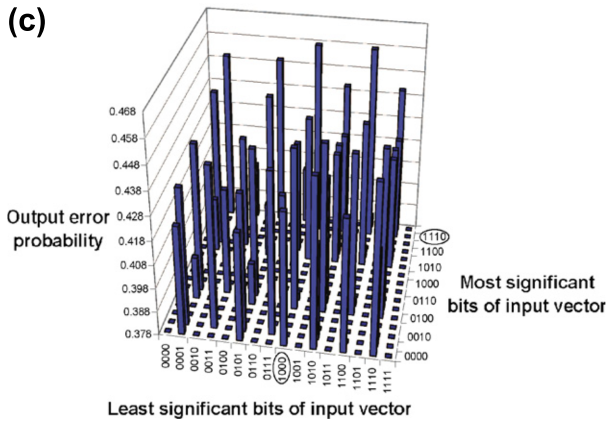


Fig. 12. (a) Output error probabilities for the entire input vector space with gate error probability $\varepsilon = 0.05$ for *c*17. (b) Output error probabilities $\geqslant(\mu + \sigma)$, calculated from probabilistic error model, with gate error probability $\varepsilon = 0.05$ for *max_flat*. (c) Output error probabilities $\geqslant(\mu + \sigma)$, calculated from HSpice, with gate error probability $\varepsilon = 0.05$ for *max_flat*.

Fig. 12b and c give the output error probabilities, obtained from the probabilistic error model and HSpice respectively, for *max_flat* with gate error probability $\varepsilon = 0.05$. In order to show that *max_flat* has large number of input vectors capable of generating maximum output error, we plot output error probabilities $\geqslant((\mu) + (\sigma))$, where μ is the mean of output error probabilities and σ is the standard deviation. The notable results are as follows:

- It is clearly evident from Fig. 12b that *max_flat* has a considerably large amount of input vectors capable of generating output error thereby making it error sensitive. Equivalent HSpice results from Fig. 12c confirms this aspect.

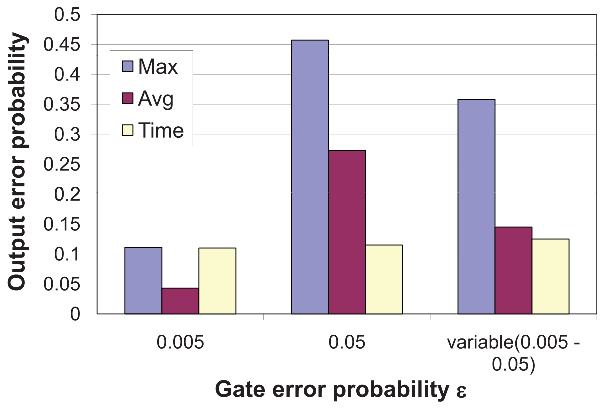


Fig. 13. Comparison between the average and maximum output error probability and run time for $\varepsilon = 0.005$, $\varepsilon = 0.05$ and variable ε ranging for 0.005–0.05 for *max_flat*.

- It is clearly evident that the results from probabilistic error model and HSpice show the same worst-case input vector, 11101000, that is obtained through MAP hypothesis.

5.5. Results with multiple ε

Apart from incorporating a single gate error probability ε in all gates of the given circuit, our model also supports to incorporate different ε values for different gates in the given circuit. Ideally these ε values has to come from the device variabilities and manufacturing defects. Each gate in a circuit will have an ε value selected in random from a fixed range, say 0.005–0.05.

We have presented the result in Fig. 13 for *max_flat*. Here we compare the average and maximum output error probability and run time with $\varepsilon = 0.005$, $\varepsilon = 0.05$ and variable ε ranging for 0.005–0.05. The notable results are as follows:

- It can be seen that the output error probabilities for variable ε are closer to those for $\varepsilon = 0.05$ than for $\varepsilon = 0.005$ implicating that the outputs are affected more by the erroneous gates with $\varepsilon = 0.05$.
- The run time for all the three cases are almost equal, thereby indicating the efficiency of our model.

6. Conclusion

We have proposed a probabilistic model that computes the exact maximum output error probabilities for a logic circuit and map this problem as maximum *a posteriori* hypothesis of the underlying joint probability distribution function of the network. We have demonstrated our model with standard ISCAS and MCNC benchmarks and provided the maximum output error probability and the corresponding worst-case input vector. We have also studied the circuit-specific error bounds for fault-tolerant computing. The results clearly show that the error bounds are highly dependent on circuit structure and computation of maximum output error is essential to attain a tighter bound.

6.1. Possible applications

In IC testing, the usage of a probabilistic error model and the information about the worst-case input vector can help to improve testing techniques like scan chains, burn-in test and hierarchical testing. Scan chains are widely used in Design for Test (DFT) methodologies for IC testing. The basic idea is to form a chain of flip-flops that are made scan-able and the desired test pattern can be serially inserted into the flip-flop chain. The test pattern is applied to the logic circuits driven by the flip-flop chain after which the logic circuit outputs can also be captured into the same or different flip-flop chain for serial shift-out. In such a setup, including the worst-case input vector in the test patterns can speed up the testing process, since the most hazardous behavior of the circuit-under-test can be detected with the worst-case input vector. Burn-in tests are performed to find out devices with inherent defects or manufacturing defects [46]. These devices will go faulty when subjected to high stress. The IC is subjected to long test time and stress conditions, such as extreme supply voltages and temperatures, during a burn-in test. To aid the burn-in test, a probabilistic error model that can target and exercise individual device fault modes would help to expedite the failure mechanisms and to screen for inherent faults in a shorter test time. More specifically, the worst case input vectors generated according to our method is well suited for application during the burn-in test. Finally, in hierarchical testing, the entire circuit-under-test is divided into several internal modules where these modules can be tested individually. Such a hierarchical division reduces the size of circuit-under-test facilitating rigorous probabilistic error analysis and the application of worst input vectors to the targeted internal modules.

6.2. Future work

Extending our proposed algorithm one can also obtain a set of, say N, input patterns which are highly likely to produce an error in the output. Circuit designers will have to pay extra attention in terms of input redundancy for these set of vulnerable inputs responsible for the high end of error spectrum. We are already working on the stochastic heuristic algorithms for both average and maximum error for mid-size benchmarks where exact algorithms are not tractable. This work should serve as a baseline exact estimate to judge the efficacy of the various stochastic heuristic algorithms that will be essential for circuits of higher dimensions. Our future effort is to model the gate error probabilities derived from the physics of the device and fabrication methods. This will be achieved by calibrating the effect of variations in device and process parameters like threshold voltage, temperature, effective channel length, gate oxide thickness and doping concentration, on individual gates. We will also focus on modeling delay faults due to timing violations and modeling variability in error probabilities.

References

[1] von Neumann J. ‘Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon CE, McCarthy J, editors. Automata studies. Princeton (NJ): Princeton Univ. Press; 1954. p. 43–98.
[2] Pippenger N. Reliable computation by formulas in the presence of noise. IEEE Trans Inform Theory 1988;34(2):194–7.
[3] Feder T. Reliable computation by networks in the presence of noise. IEEE Trans Inform Theory 1989;35(3):569–71.
[4] Hajek B, Weller T. On the maximum tolerable noise for reliable computation by formulas. IEEE Trans Inform Theory 1991;37(2):388–91.
[5] Evans W, Schulman LJ. On the maximum tolerable noise of *k*-input gates for reliable computation by formulas. IEEE Trans Inform Theory 2003;49(11):3094–8.
[6] Evans W, Pippenger N. On the maximum tolerable noise for reliable computation by formulas. IEEE Trans Inform Theory 1998;44(3):1299–305.
[7] Gao JB, Qi Y, Fortes JAB. Bifurcations and fundamental error bounds for fault-tolerant computations. IEEE Trans Nanotechnol 2005;4(4):395–402.
[8] Marculescu D, Marculescu R, Pedram M. Theoretical bounds for switching activity analysis in finite-state machines. IEEE Trans VLSI Syst 2000;8(3):335–9.
[9] Depledge PG. Fault-tolerant computer systems. IEE Proc A 1981;128(4):257–72.
[10] Spagocci S, Fountain T. Fault rates in nanochip devices. In: Electrochemical society; 1999. p. 354–68.
[11] Han J, Jonker P. A defect- and fault-tolerant architecture for nanocomputers. Nanotechnology 2003;14:224–30.
[12] Roy S, Beiu V. Majority multiplexing-economical redundant fault-tolerant designs for nano architectures. IEEE Trans Nanotechnol 2005;4(4):441–51.
[13] Nikolic K, Sadek A, Forshaw M. Fault-tolerant techniques for nanocomputers. Nanotechnology 2002;13:357–62.
[14] Han J, Taylor E, Gao J, Fortes JAB. Reliability modeling of nanoelectronic circuits. In: IEEE Conference on Nanotechnology; 2005.
[15] Simsir MO, Cadambi S, Ivancic F, Roetteler M, Jha NK. Fault-tolerant computing using a hybrid nano-CMOS architecture. In: International conference on VLSI design; 2008. p. 435–40.
[16] Chen C, Mao Y. A statistical reliability model for single-electron threshold logic. IEEE Trans Electron Dev 2008;55:1547–53.
[17] Abdollahi A. Probabilistic decision diagrams for exact probabilistic analysis. In: Proceedings of the 2007 IEEE/ACM international conference on computer-aided design; 2007. p. 266–72.
[18] Choudhury MR, Mohanram K. Accurate and scalable reliability analysis of logic circuits. DATE 2007:1454–9.
[19] Lazarova-Molnar S, Beiu V, Ibrahim W. A strategy for reliability assessment of future nano-circuits. In: WSEAS international conference on circuits; 2007. p. 60–5.
[20] Shenoy PP, Shafer G. Propagating belief functions with local computations. IEEE Expert 1986;1(3):43–52.
[21] Shenoy PP. Binary join trees for computing marginals in the Shenoy–Shafer architecture. Int J Approx Reason 1997:239–63.
[22] Shenoy PP. Valuation-based systems: a framework for managing uncertainty in expert systems. Fuzzy Logic Manage Uncertain 1992:83–104.
[24] Jensen FV, Lauritzen S, Olesen K. Bayesian updating in recursive graphical models by local computation. Comput Stat Quart 1990:269–82.
[26] Rejimon T, Bhanja S. Probabilistic error model for unreliable nano-logic gates. In: IEEE conference on nanotechnology; 2006. p. 717–22.
[27] Lingasubramanian K, Bhanja S. Probabilistic maximum error modeling for unreliable logic circuits. In: ACM great lake symposium on VLSI; 2007. p. 223–6.
[28] Park JD, Darwiche A. Solving MAP exactly using systematic search. In: Proceedings of the 19th annual conference on uncertainty in artificial intelligence; 2003.
[29] Park JD, Darwiche A. Approximating MAP using local search. In: Proceedings of 17th annual conference on uncertainty in artificial intelligence; 2001. p. 403–10.
[30] Sensitivity analysis, modeling, inference and more. <http://reasoning.cs.ucla.edu/samiam/>.
[31] Roth JP. Diagnosis of automata failures: a calculus and a method. IBM J Res Dev 1966;10(4):278–91.
[32] Goel P. An implicit enumeration algorithm to generate tests for combinational logic circuits. IEEE Trans Comput 1981;C-30(3):215–22.
[33] Fujiwara H, Shimono T. On the acceleration of test generation algorithms. IEEE Trans Comput 1983;C-32(12):1137–44.
[34] Agrawal VD, Seth SC, Chuang CC. Probabilistically guided test generation. In: Proceedings of IEEE international symposium on circuits and systems; 1985. p. 687–90.
[35] Savir J, Ditlow GS, Bardell PH. Random pattern testability. IEEE Trans Comput 1984;C-33(1):79–90.
[36] Seth C, Pan L, Agrawal VD. PREDICT – Probabilistic estimation of digital circuit testability. In: Proceedings of IEEE international symposium on fault-tolerant computing; 1985. p. 220–5.
[37] Chakradhar ST, Bushnell ML, Agrawal VD. Automatic test generation using neural networks. In: Proceedings of IEEE international conference on computer-aided design, vol. 7 (10); 1988. p. 416–9.
[38] Mason M. FPGA reliability in space-flight and automotive applications. FPGA Program Logic J 2005.
[40] Gerrish P, Herrmann E, Tyler L, Walsh K. Challenges and constraints in designing implantable medical ICs. IEEE Trans Dev Mater Reliab 2005;5(3):435–44.

[41] Stotts L. Introduction to implantable biomedical IC design. IEEE Circ Dev Mag 1999:12–8.

[42] Cheemalavagu S, Korkmaz P, Palem KV, Akgul BES, Chakrapani LN. A probabilistic CMOS switch and its realization by exploiting noise. In: Proceedings of the IFIP international conference on very large scale integration; 2005.

[43] Martel R, Derycke V, Appenzeller J, Wind S, Avouris Ph. Carbon nanotube field-effect transistors and logic circuits. In: Proceedings of the 39th conference on design automation; 2002.

[44] Kummamuru RK, Orlov AO, Ramasubramaniam R, Lent CS, Bernstein GH, Snider GL, et al. Shift registers and analysis of errors. IEEE Trans Electron Dev 1993;50(59):1906–13.

[45] Mazumder P, Kulkarni S, Bhattacharya M, Sun Jian Ping, Haddad GI. Digital circuit applications of resonant tunneling devices. Proc IEEE 1998;86(4):664–86.

[46] Military standard (MIL-STD-883). Test methods and procedures for microelectronics; 1996.