A Memory Constrained Approximate Bayesian Inference Approach Using Incremental Construction of Clique Trees

Shivani Bathla, *Student Member, IEEE*, and Vinita Vasudevan, *Member, IEEE*

Abstract—Bayesian Network (BN) based inference algorithms can be used for estimation of several metrics that are useful in a variety of applications, including circuit design. The complexity of inference using the belief propagation algorithm increases exponentially with clique sizes, necessitating partitioning based on a clique size constraint. Repeated recompilation of the BN to construct the Clique Tree (CT) is expensive. In this paper, we propose a novel algorithm for the incremental construction of the CT model. Nets are added to the CT as long as the clique size constraint is not violated. We show that our algorithm always produces a valid CT that satisfies all properties and gives a large runtime improvement over complete recompilation. When the clique size constraint is violated, we simplify the CT using a combination of exact and approximate marginalization. The approximate CT serves as a starting point for the construction of CT for the remaining nets. The algorithm gives us a set of CTs for each partition. We show that the marginals and the joint probability between variables can be computed in a consistent manner. As a result, we get better estimates of correlations between nets. The proposed framework results in a significant reduction in error for circuits across three benchmark suites, including some very large circuits in the EPFL suite.

Index Terms—Bayesian networks, belief propagation, incremental junction tree, signal probability

I. INTRODUCTION

Bayesian Network (BN) based algorithms have been used for probabilistic inference in a wide variety of applications. A popular method for inference is the Belief Propagation (BP) algorithm [1]. This method does not directly work with the BN model, but with a derived graphical representation called the *clique tree* (CT) model (also called *join tree* or *junction tree*), obtained using a process known as the *compilation* of the network. Inference using the BP algorithms is carried out over this CT network [1]. It is known to be #P complete, necessitating partitioning and simplification.

In this paper, our focus is approximate inference when there are clique size limitations due to memory constraints. The Directed Acyclic Graph (DAG) corresponding to digital circuits can be modeled as a BN [2]–[5]. Many digital circuits (even relatively small ones) have long and nested re-convergent loops, which lead to large clique sizes and pose a huge challenge for the BP algorithm. Therefore, in this paper, we discuss our algorithms and inference tasks in the context of digital circuits.

The authors in [6] propose a partitioning technique called the Multiply Sectioned Bayesian Networks (MSBN) for large domains. Exact inference can be carried out only if the network partitions follow a set of constraints, which is not possible in all cases. In practice, since the partitions can be quite large, the inference algorithms are constrained by the memory and computational resources available. A special case of this method is the hierarchical model proposed in [7]. Both methods do not guarantee any bound on the clique sizes within a partition. Simplification of the network based on queries, followed by exact inference is proposed in [1], [8], but it is query dependent and not suitable when the metric of interest is the marginal/joint probabilities. A related technique is the conditioning method [1]. This is similar to the idea of estimating signal probabilities by identifying supergates and conditioning on fanout nodes that re-converge [9].

One way to mitigate the effect of large clique sizes is to use approximate inference techniques like probabilistic logic sampling, evidence pre-propagated importance sampling, adaptive importance sampling, and likelihood weighting. In the context of digital circuits, these have been used for inference in [4], [10]–[12]. Sampling methods have also been used for approximate inference in MSBNs [13]. However, these methods are inflexible in the sense that any small changes in the circuit would need a complete re-simulation. Moreover, the estimates obtained are highly dependent on parameters like the random sample seed, the number of samples, and the sample generation algorithm. Another approximate technique is the Loopy BP method, which involves iterative message passing over cluster graphs with loops. However, in the presence of multiple loops, convergence may become difficult and, even if achieved, does not guarantee correctness [1], [14].

The authors in [15], [16] propose to cascade BN models of partitions, with an approximate BN model between partitions. There are two issues to be addressed while partitioning BNs. The first issue is to *ensure accuracy*. For accuracy, it is important to model correlations between variables or nets that are required for all subsequent partitions. We will refer to these nets as *interface nets*. Fig. 1 shows partitioning of a sample BN. Nets d, e are the interface nets that lie at the boundary. The simplest model ([17]) assumes that all interface nets are independent, which leads to large estimation errors in many circuits. Another approach suggested in ([15]) is to cascade the BN model of the next partition with a model that approximately captures the JPD of the interface nets via a tree network built using pair-wise correlations of selected nets. As

Fig. 1. Sectioning the Bayesian Network into two partitions. Nodes d and e lie at the interface.

will be seen in the results section, for many circuits, the tree network used in [15] does not approximate the JPD of the interface nets well. Moreover, finding the tree can be time-consuming when the number of interface nets is large. The second issue is to *choose nodes (nets) in a partition* such that the clique size constraint is met. One possibility is to iteratively select larger subgraphs of the BN and recompile them until the size constraint is violated. However, this is a time-consuming process that involves several recomputations. In [15], the partition sizes are chosen from a predetermined set depending on heuristic-based upper bounds on the maximum clique size. This is a pessimistic approach that leads to the creation of smaller partitions, thus requiring more approximations.

It is also possible to use other partitioning and approximation techniques that have been proposed in the context of signal and switching probability estimation in circuits. The method proposed by Costa *et al* ([18]) uses probability polynomials, with a limit on the length of the re-convergent paths. Several local Ordered Binary Decision Diagrams (OBDD) based heuristics have been proposed that limit either the number of input nodes in each partition [19] or the number of topological levels considered for each OBDD [20], [21]. The drawback of these algorithms is that it requires us to decide apriori the number of topological levels used to model correlations. It also does not guarantee a max-clique size within a partition. Besides, BNs allow for a much richer set of approximations than these methods.

To avoid the runtime overhead associated with iterative recompilation of large subgraphs of the BN, it is possible to incrementally modify the clique trees as new nets are added. The method is also attractive since any addition or deletion of gates in parts of the circuit can be easily accommodated. Incremental compilation of the CT model has been explored in some previous works [22], [23]. In [22], incremental addition of links is performed by first forming a cluster graph using a set of rules and then converting the cluster graph into a junction tree. This conversion is performed using heuristic-based graph transformations. A difficulty is to choose a set of heuristics, so that clique size constraints are met. A method to directly obtain the modified junction tree is proposed in [23]. Here, the minimal section of the clique tree that needs modification is identified using the Maximal Prime Subgraph Decomposition (MPSD) [24] of the Bayesian Network. Based

on the moralized graph, the junction tree is converted into another graphical representation called the MPD join tree. This approach requires the storage of multiple intermediate representations corresponding to the entire clique tree.

A. Contributions

Our first contribution is a novel algorithm for the incremental construction of the CT model. We show that the algorithm is sound; that is, it always results in a valid CT after the addition of nets. This method provides a significant speedup over repeated recompilation-based partitioning. In contrast to the MPSD based approach, our algorithm works directly with the clique tree graph and hence does not require multiple intermediate representations. It also reduces the computational effort required for re-triangulation. Our algorithm adds as many nets as possible to an existing CT under the input clique size constraints, thus partitioning the circuit. Unlike previous works [6], [15], we do not need to explicitly partition at the circuit level, where it is difficult to estimate clique sizes.

Our second contribution is a method to simplify the existing CT to obtain the interface CT that captures the JPD of the interface nets, using a combination of exact and approximate marginalization. Since we build the CTs incrementally, it is possible to use a CT-based interface model as a starting point for the next partition. Our algorithm results in a set of clique trees corresponding to each partition. We show that marginals are preserved across partitions; that is, the marginal extracted from any partition has the same value. We also show that the JPD between variables in different partitions can be obtained in a consistent manner by conditioning on the interface variables.

The complexity of the model can be tuned to get a trade-off between accuracy and runtime. The proposed framework allows the flexibility of incorporating incremental addition/deletions to portions of the network. Moreover, it is compatible with existing efforts towards software and hardware acceleration of BP methods [25]–[29].

Our framework supports inference tasks used in BN models. In this paper, we validate the proposed technique by computing two metrics commonly used in circuits. The first is the *signal probability* which is a key metric used in many design automation algorithms, including testing[9], [30], reliability [31]–[35], and probabilistic design[36]–[39]. We demonstrate significantly better accuracies in nearly all circuits belonging to three combinational benchmark suites, including several very large circuits in the EPFL suite. We show that our technique also provides good estimates of the *joint probabilities of gate inputs* which are used to get rapid estimates of reliability metrics [32], [40].

The rest of this paper is organized as follows. Section II provides background on BN models and inference techniques. We describe the proposed overall algorithm in Section III, the proposed incremental approach for building clique tree models in Section IV, and a novel clique tree-based interface model in Section V. The results are presented in Section VI. Finally, we present our conclusions.

II. BACKGROUND

We briefly describe the steps involved in converting a gate-level description of the circuit to a CT model. Fig. 2 shows the steps involved for a sample circuit. First, an undirected model called the *moralized graph* is formed from the BN by removing edge orientations and adding pairwise edges between parents of each net in the graph. Next, the *chordal* completion of the moralized graph is obtained by adding edges to break all loops of size greater than three. A fully connected component (or a clique) in a graph is a maximal clique if it is not contained within any other clique. The maximal cliques present in this chordal completion form the nodes of the CT model. Each node C_i contains a set of nets denoted as $Scope(C_i)$. The cliques are connected such that the resultant CT satisfies the *Running Intersection Property (RIP)* which states that if a variable belongs to two cliques C_i and C_j , it also belongs to the scope of all cliques in the path connecting C_i and C_j in the CT. For example, net d present in cliques C_1 and C_5 is also present in connecting cliques C_3, C_4 in Fig. 2. Each edge in the CT is associated with a set of *sep-set* variables that comprise the intersection of scopes of adjoining cliques. The sep-sets are marked in gray in Fig. 2.

In digital circuits, each net n_i is treated as a binary random variable associated with a conditional probability distribution (CPD) $P(n_i | Par_{n_i})$, where Par_{n_i} are the parents of n_i in the BN model. The CPD is determined by the functionality of the gate driving the net. The CPD corresponding to each net in the BN model is assigned to a clique that contains its scope. The joint probability distribution (JPD) of all nets in the network can be written as

$$P(n_1, n_2, \dots, n_{|\mathcal{N}|}) = \prod_{i=1}^{|\mathcal{N}|} P(n_i | Par_{n_i})$$

Exact BP algorithm involves passing messages along the edges of the CT in two passes, an upward pass (from the leaf nodes to the root node) and a downward pass (from the root node to the leaves). The calibrated clique beliefs (β_i) and sep-set beliefs (μ_{ij}) are obtained after message passing. The clique beliefs can be used to compute marginal probabilities using

$$P(n_i) = \sum_{Scope(C_i) \setminus n_i} \beta_i, \quad \text{for } n_i \in Scope(C_i)$$

All joint probabilities can be obtained from the calibrated CT using a dynamic programming approach [1].

If each parameter is stored using the single-precision format, a clique of size N requires $4 * 2^N$ Bytes to store the clique belief. Thus, both space and time complexity of BP is exponential in the maximum clique size of the CT model. Table I tabulates the max-clique size and the memory required to store the calibrated CT beliefs for various benchmark circuits. The space complexity is very high even for some very small circuits like *int2float*, *x1*, and *c1908*. Hence, this method becomes intractable for circuits with large cliques and requires approximations.

TABLE I
THE MAXIMUM CLIQUE SIZE (MCS) AND MEMORY REQUIRED TO STORE THE CLIQUE TREE REPRESENTATION FOR VARIOUS CIRCUITS

Benchmark	#Nets	MCS	Memory (GB)
int2float	168	36	3.4×10^2
too_large	217	44	7.0×10^4
x1	232	35	1.6×10^2
c1908	237	40	4.8×10^3
alu4	396	69	6.5×10^{12}
vda	510	101	1.0×10^{22}
rot	549	34	1.5×10^2
c3540	562	72	2.2×10^{13}
bar	1277	181	1.1×10^{46}
c6288	1706	50	6.9×10^6
multiplier	13272	311	1.6×10^{85}
div	23859	254	1.1×10^{68}

III. MAIN ALGORITHM

Alg. 1 gives the main steps in our algorithm. It returns the CTs corresponding to each partition, the marginals and the cliques in each interface.

Algorithm 1 MainAlgorithm (G, mcs_p, mcs_{im})

Input: G : Circuit DAG
 mcs_p : Maximum clique size limit for the partition CTF
 mcs_{im} : Maximum clique size limit for the interface CTF
Output: MP : Map of marginal probabilities of nets in G
 L_{CTF} : List of CTF for each partition
 LC_{im} : List of set of cliques in each interface CTF

- 1: Pre-process G
- 2: **Initialize:**
 $MP \leftarrow \langle \rangle$ ▷ Map < net : margProb >
 $CTF = \{\text{Cliques over primary inputs}\}$ ▷ Clique Tree Forest
 $L_{CTF} = []$
 $LC_{im} = []$
- 3: $G.remove(\text{Primary Inputs})$
- 4: **while** $G.size \neq 0$ **do**
- 5: ▷ Add nets and update CTF
- 6: $N_{act} \leftarrow$ Nets in G with zero in-degree ▷ Active nets
- 7: $N_a \leftarrow \emptyset$ ▷ Set of nets added to CTF
- 8: $IN \leftarrow \emptyset$ ▷ Interface nets
- 9: **while** $N_{act} \neq \emptyset$ **do**
- 10: $N_i \leftarrow N_{act}.pop$ nets with min. topological level
- 11: $CTF, N_a \leftarrow \text{ModifyCTF}(CTF, N_i, mcs_p)$
- 12: $IN.add$ parents of nets in $N_i \setminus N_a$
- 13: $G.remove(N_a)$
- 14: $N_f \leftarrow$ Fanout of N_a with zero in-degree in G
- 15: $N_{act}.push(N_f)$
- 16: **end while**
- 17:
- 18: $\beta, \mu \leftarrow$ Calibrate CTF using Belief Propagation
- 19: $L_{CTF}.push(CTF)$
- 20: $MP \leftarrow$ Find marginals from β
- 21: **if** $IN \neq \emptyset$ **then**
- 22: $CTF \leftarrow \text{BuildInterfaceCTF}(CTF, IN, mcs_{im})$
- 23: $LC_{im}.push(\{Scope(C) \mid C \in CTF\})$
- 24: **end if**
- 25: **end while**
- 26: **return** MP, L_{CTF}, LC_{im}

The inputs to Alg. 1 are a circuit and the associated DAG, G , and the clique size constraints. The circuit could have disjoint DAGs, therefore, we use the term *Clique Tree Forest* (CTF) to denote the collection of corresponding CTs. We perform some pre-processing steps described further in Sec. VI. The CTF is

Fig. 2. Compilation of clique tree for an example circuit. The dotted red lines are the moralizing edges between parents, and the dashed blue lines are the fill-in edges introduced during triangulation. The sep-set variables corresponding to each edge in the clique tree are marked in gray.

built incrementally, starting with the primary inputs (line 2). If inputs are independent, CTF contains disjoint cliques. Once inputs are added to the CTF, they are removed from G (line 3). At any point in the construction, nets whose parents are present in the CTF are ready for addition (zero in-degree in G). These nets are called *active nets* (N_{act}). Lines 4-15 describe how the CT is built incrementally. Starting with set of active nets that have the lowest topological level, we add nets to the CTF under a maximum clique size constraint mcs_p using the function `ModifyCTF` (described in Alg. 2). Nets that are successfully added to the CTF (N_a) are removed from G . Parents of the deferred nets are added to the set of interface nets (IN). The list of active nets is updated with fanouts of N_a if both parents are present in CTF. The iteration continues until N_{act} is empty.

BP is then used to obtain the calibrated CT beliefs (β, μ) (line 18). The marginals are obtained after marginalizing appropriate clique beliefs (line 20). If the set of interface nets is not empty, the CTF for the construction of the next partition is obtained by approximating the CTF of the current partition using the function `BuildInterfaceCTF` (described in Alg. 3) (line 22). The algorithm ends once all nets have been added. In order to allow for inference of queries other than the marginals, we store the CTF corresponding to each partition in a list (L_{CTF}) and the set of cliques present in each interface CTF (LC_{im}).

IV. ALGORITHM FOR INCREMENTAL CONSTRUCTION OF CTF

In this section, we describe the methods used to incrementally construct clique trees. As in all previous works [2], [15], [32], [41], we pre-process the network so that each net has two parent nodes. We first describe the algorithm and then show that the algorithm will always result in a valid CT.

A. Modifying existing CTF to add a single net

Let us first consider the addition of a single net n with parent nodes p_1 and p_2 to a CTF. This will introduce a moralizing edge between p_1 and p_2 and an additional clique C_n with scope equal to $[p_1, p_2, n]$. The CPD of the new net $P(n|p_1, p_2)$ forms the factor associated with the new clique

C_n . The existing CTF now needs to be modified so that this clique can be added while ensuring that the CTF remains valid.

1) *Algorithm:* The modifications required depend on the location of the parent nets in the existing CTF. We identify three possible cases and discuss the algorithm used.

Case 1: If both p_1 and p_2 are contained in the same clique, C , we connect C_n to C as shown in Fig. 3. In case C is contained in C_n , C is replaced by C_n .

Case 2: If p_1, p_2 belong to cliques C_1, C_2 in disconnected CTs, C_n gets connected to both C_1 and C_2 as shown in Fig. 4. If either C_1 or C_2 are non-maximal cliques, they are replaced by C_n .

Case 3: p_1, p_2 belong to different cliques in the same CT. Fig 5a shows an example. In this case, addition of the new clique introduces chordless loops to the chordal graph corresponding to the CT. In the example, chordless loops $p_1abc p_2$ and $p_1dbc p_2$ are introduced as shown in Fig. 5b. Hence, this portion of the graph needs retriangulation (shown in Fig. 5c for the example). The following steps are used to get the modified CTF.

1. Identify two cliques containing the parent nets with the smallest number of intermediate cliques between them. The impacted cliques are in the shortest path (ST) between the parent cliques. The cliques shaded in red in Fig. 5a need to be modified in the example.
2. Construct a set S containing parent nets and all nets in the sep-sets in the path. For the example, $S = \{p_1, p_2, a, d, b, c\}$.
3. Find cliques in the path that contain nets $\notin S$. Such cliques are called *retained cliques*. In the example, C_3 is a retained clique.
4. Find the modified clique tree ST' using the following steps. For the example, ST' is highlighted in teal in Fig. 5d.
 - 4.1 Construct an elimination graph for nets in S as follows. Corresponding to each path clique C in ST , add a fully connected component between nets in $Scope(C) \cap S$ and introduce a moralizing edge between the two parent nets. Fig. 5c shows the elimination graph for the example. This is the portion that needs retriangulation.
 - 4.2 Triangulate the elimination graph and obtain the cor-

Fig. 3. Case 1: Addition of a new net n with both parents in the same clique in the existing CT.

Fig. 4. Case 2: Addition of a new net n , with parents in disconnected CTs.

responding clique tree ST' .

- 4.3 Connect the new clique C_n to the clique in ST' that contains both parents.
- 4.4 For each retained clique C , identify clique C' in ST' that contains $\text{Scope}(C) \cap S$. Connect C to C' . If $\text{Scope}(C') \subset \text{Scope}(C)$, replace C' with C . In the example, clique C_3 is connected to C'_3 .
- 4.5 The factors associated with cliques that are not retained are reassigned to cliques in ST' containing the entire scope of those cliques.
5. Remove the impacted subgraph ST from CTF.
6. Connect ST' to CTF via the set of cliques adjacent to ST in the existing CTF. Cliques in this adjacency set are re-connected to cliques in ST' that contain the corresponding sep-set in the input CTF.

2) *Soundness of the algorithm:* We now argue that if the input CTF is valid, incremental modification using the algorithms described in Sec. IV-A1 results in a valid CTF.

- CTF contains only maximal cliques.
In Cases 1,2, any clique $C \subset C_n$ is removed from the CTF. Similarly, in Case 3, the CT obtained from the elimination graph has only maximal cliques by construction. The final ST' is obtained after connecting retained cliques in step 4.4; where there is an additional check for maximality. Cliques in $\text{CTF} \setminus ST$ contain atleast one net that is not present in ST' , thus remain maximal.
- CTF contains disjoint trees
The new clique is connected to a single clique in Case 1 and it connects two disjoint trees in Case 2. In Case 3, ST' is a tree by construction. It is connected back to CTF via the same set of cliques as ST . Hence, no loops are introduced in either of these cases.
- All CTs in CTF satisfy RIP
Consider the chordal graph corresponding to the existing CTF. In Cases 1,2, this graph remains chordal after a clique is added between the new net and its parents. Thus, RIP is satisfied when C_n is connected to CTF with parents as sep-sets.
For Case 3, let S_a denote the set of all nets in CTF, and S^c

Fig. 5. Case 3: Addition of a new net n , with parents in the CT.

be the set $S_a \setminus S$. The chordal graph corresponding to the existing CTF has a perfect elimination order such that no fill-in edges are introduced on elimination. Even after the addition of the moralizing edge between the parents (p_1, p_2), nets in S^c can be eliminated in this order without adding any fill-in edges. Therefore, cliques containing these nets are retained as is in the final CTF. Elimination of nets in S could potentially introduce fill-in edges as they are a part of chordless loops introduced by the moralizing edge. The graph corresponding to the nets in S is precisely the elimination graph, which is re-triangulated in step 4.1. The clique tree obtained corresponding to this in step 4.2 is valid by construction. ST' obtained after steps 4.3, 4.4 satisfies RIP because cliques are connected such that the sep-sets contain all nets in the intersection with S . The CTF obtained after replacing ST with ST' in step 6 satisfies RIP because ST' is reconnected such that sep-sets are preserved.

- Product of factors gives the correct JPD.

The joint distribution of nets in the existing CTF is correct. In all cases, we reassign the factors corresponding to cliques removed from the existing CT to new cliques containing their scope. CPD of the new net n is assigned to C_n . Thus, the assignment remains valid after modification.

Algorithm 2 ModifyCTF (CTF, N_i, mcs_p)

Input: CTF : Existing CTF
 N_i : Set of active nets to be added
 mcs_p : Maximum clique size limit for CTF

Output: CTF : Modified CTF
 N_a : Subset of N_i added to CTF

```

1: Initialize:  $N_a = []$ 
    $paths = \langle \rangle$  ▷ Map<net : Shortest path b/w net.parents>
2: for  $v_i \in N_i$  do
3:   if ( $v_i \in Case1$ ) or ( $v_i \in Case2$ ) then
4:     Modify  $CTF$  as described in Sec. IV-A1
5:      $N_i.remove(v_i)$ 
6:      $N_a.add(v_i)$ 
7:   else
8:      $paths[v_i] =$  Shortest path between  $v_i$ .parents in  $CTF$ 
9:   end if
10: end for
11:
12:  $L_g \leftarrow$  Group nets in  $N_i$  with overlapping paths
13: while  $L_g \neq \emptyset$  do
14:    $N_g = L_g.pop()$ 
15:    $ST \leftarrow CTF.subgraph(\cup_{v_i \in N_g} paths[v_i])$ 
16:    $ST' \leftarrow$  Find modified subtree using steps 2 - 4 in Sec. IV-A1
17:   if  $ST'.max\_clique\_size \leq mcs_p$  then
18:     Replace  $ST$  by  $ST'$  using steps 5 - 6 in Sec. IV-A1
19:      $N_a.add(N_g)$ 
20:   else
21:      $N_s =$  Choose a subset of  $N_g$  for addition
22:     Group nets in  $N_s$  with overlapping paths and add to  $L_g$ 
23:   end if
24: end while
25: return  $CTF, N_a$ 

```

B. Modifying existing CTF to add multiple nets under clique size constraints

Alg. 2 shows the main steps involved in modifying the CTF to add a set of nets under a given maximum clique size constraint. The addition of nets belonging to the first two cases does not change the size of the existing cliques. Therefore, these nets are sequentially added to the CTF (lines 3-6). For nets belonging to Case 3, we identify and store the path cliques on the shortest path connecting the parents (line 7-8). If there is an overlap in the identified paths, sequentially adding the nets can result in large clique sizes. Therefore, the new nets are divided into subsets, each composed of nets with overlapping paths (line 12). This is achieved by breaking the subgraph of CTF over all path cliques into disjoint subtrees. Each subtree corresponds to a subset of new nets. All nets in a subset N_g are added together. ST is the subgraph of CTF corresponding to the union of path cliques for nets in N_g . The modified subtree ST' is obtained using steps 2 - 4 in Sec. IV-A1 (lines 14-16). The set S will now contain all sep-set nets present in ST and parents for all nets in N_g . If the maximum clique size in ST' is less than mcs_p , ST is replaced by ST' using steps 5 - 6 in Sec. IV-A1 (lines 17-19). Otherwise, we choose a smaller subset N_s for addition and defer the remaining nets to the next partition. Nets in N_s are re-grouped based on overlapping paths, and the new groups are added to the list of subsets (lines 20-22). The algorithm ends when the list becomes empty.

We use a hash table to perform these operations efficiently. As the CTF is constructed, we build a hash table that is

indexed by the net and contains all the cliques in which the net is present. As a consequence, many of the steps involve a simple look-up and finding set intersections. These operations are fast as both nets and cliques are represented using integer identifiers. All graph operations were performed using the NetworkX library [42].

V. INTERFACE CTF

Once the maximum clique size constraint is reached, the joint probabilities of the cliques and sep-sets (β, μ) are inferred from the CTF using the BP algorithm. At this point, we need an approximate CTF that serves as the starting point for the next partition. We call this approximate CTF as the *interface CTF*. To allow for the addition of nets to the next partition, we need to leave a margin. Therefore, we restrict the clique size in the interface CTF to a threshold ($mcs_{im} < mcs_p$). By definition, the model must contain all *interface nets*, IN (parents of nets that have not yet been added to the CTF). It must also contain a minimal subset of non-interface nets required to ensure that all CTs in CTF remain valid and the JPD between the interface nets is well approximated.

A. Approach to obtain the interface CTF

Alg. 3 describes the proposed methodology to obtain the interface model. It has four main steps, which we now discuss in detail.

Step 1: Identify the minimal sub-graph connecting interface nets: We first identify the set of cliques that contain one or more interface nets and find the subgraph of CTF that connects these cliques (line 2). Though this subgraph in the present form can be used to infer the exact JPD over the interface nodes, it is not the minimal subgraph. This is because the same set of interface nets could be present in multiple cliques that can be removed from the subgraph without violating the RIP property. Starting from the leaf nodes, we recursively remove cliques that contain the same set (or subset) of interface nets as their neighbor (line 3).

Step 2: Collapse cliques and marginalize non-interface nets: In this step, we attempt to marginalize the non-interface nets present in the interface CTF obtained after Step 1.

First (lines 4-7), the non-interface nets present in a single clique are removed from the clique scope, and the corresponding clique beliefs (β) are marginalized. If the resultant clique is a non-maximal clique, then it is removed from CTF, and its neighbors are connected to the containing clique. In case there are multiple containing cliques, we choose the one with the least degree for connection.

On the other hand, if a non-interface net (n) is present in multiple cliques, we form a new clique by collapsing all cliques containing n (lines 8-9). The branches of the collapsed cliques are re-connected to the new clique. The belief corresponding to the new clique is formed by finding the joint probability over the collection of the collapsed cliques and then marginalizing n . While this step preserves the JPD, the size of the new clique can be very large.

Algorithm 3 BuildInterfaceCTF (CTF , IN , mcs_{im})

Input: CTF : Input clique tree forest
 IN : Interface nets
 mcs_{im} : Maximum clique size limit for interface CTF

Output: CTF : Interface CTF

- 1: $\triangleright$ **Step 1: Identify the minimal subgraph connecting IN**
- 2: $CTF \leftarrow$ Subgraph of CTF connecting IN
- 3: Prune CTF by recursively removing redundant leaf nodes
- 4: $\triangleright$ **Step 2: Collapse cliques and marginalize non-interface nets**
- 5: $NI \leftarrow Nets \in CTF \setminus IN$ $\triangleright$ Non-interface nets
- 6: Marginalize subset of NI present in only a single clique
- 7: Remove resultant non-maximal cliques, reconnect neighbors
- 8: $NI_m \leftarrow$ Subset of NI that can be marginalized within mcs_{im}
- 9: Collapse cliques and marginalize out NI_m from CTF
- 10: $\triangleright$ **Step 3: Trim large cliques with size $> mcs_{im}$**
- 11: $L_c \leftarrow$ List of cliques with size $> mcs_{im}$
- 12: Compute metrics $maxMI$, $MLMI$ for nets in all cliques in L_c
- 13: $N \leftarrow$ Non-interface nets present in cliques in L_c
- 14: **while** $L_c \neq \emptyset$ **do** $\triangleright$ CTF has cliques with size $> mcs_{im}$
- 15: **if** $N == \emptyset$ **then** $N \leftarrow$ Interface nets present in cliques in L_c
- 16: $n = N.$ pop net with the least $maxMI$
- 17: $L_{cn} \leftarrow$ List of cliques containing n .
- 18: $L_s \leftarrow$ Subset of L_{cn} with size $\leq mcs_{im}$
- 19: $L_r = \emptyset$ $\triangleright$ List of cliques where n is retained
- 20: **if** $L_s \neq \emptyset$ **then**
- 21: $L_{st} \leftarrow$ Disjoint subtrees in CTF over L_s
- 22: $L_r \leftarrow$ Subtree $\in L_{st}$ containing clique with largest $MLMI$
- 23: **else if** $n \in IN$ **then**
- 24: Add an independent clique C with $Scope(C) = \{n\}$
- 25: **end if**
- 26: Locally marginalize n from all cliques in $L_{cn} \setminus L_r$, sep-sets
- 27: Remove resultant non-maximal cliques, reconnect neighbors
- 28: $L_c.remove(C)$ if $C.size \leq mcs_{im} \forall C \in L_{cn} \setminus L_s$
- 29: **end while**
- 30: $\triangleright$ **Step 4: Re-assign clique factors**
- 31: **for** $CT \in CTF$ **do**
- 32: root $\leftarrow$ Pick a node at random from CT
- 33: $\psi_{root} = \beta_{root}$
- 34: $L_v = [root]$
- 35: **for** $C_i \in L_v$ **do**
- 36: **for** $C_j \in \{CT.neighbors(C_i) \notin L_v\}$ **do**
- 37: $\psi_{C_j} = \beta_j / \mu_{ij}$
- 38: $L_v.add(C_j)$
- 39: **end for**
- 40: **end for**
- 41: **end for**
- 42: **return** CTF

Therefore, this operation is performed on only those non-interface nets for which the new clique size is less than mcs_{im} .

Step 3: Trim large cliques with size $> mcs_{im}$: Next (lines 10-30), we trim large cliques while ensuring that the CTs remain valid and correlations between signals are preserved as much as possible. We use the *Mutual Information (MI)* metric to estimate correlations between nets. This metric is a measure of the average distance between the JPD (p_{XY}) and the product of the marginals ($p_X \cdot p_Y$), defined as below.

$$MI(X; Y) = \sum_{x, y \in \{0, 1\}^2} p_{XY}(x, y) \cdot \log\left(\frac{p_{XY}(x, y)}{p_X(x) \cdot p_Y(y)}\right)$$

While the JPD of signals belonging to the same clique can be obtained easily, computation for signals belonging to

different cliques in a CT requires variable elimination, which is both memory-intensive and time-consuming. Therefore, we trim cliques based on MI between nets that belong to the same clique. We define the following metrics to guide our clique reduction technique.

a. *Maximum Local Mutual Information, $MLMI$*

For each clique in the CTF, we compute the MI between all nets and the interface nets present in the scope. $MLMI$ of a net n contained in clique C is computed as the maximum MI using:

$$MLMI_{n,C} = \begin{cases} \max_{\forall x \in Scope(C) \setminus n} MI(n; x), & n \in IN \\ \max_{\forall x \in Scope(C) \cap IN} MI(n; x), & n \notin IN \end{cases}$$

b. *Maximum Mutual Information, $maxMI$*

Since a net is typically present in multiple cliques, we compute $maxMI$ of a net as:

$$maxMI_n = \max_{\forall C \in CTF} \max_{s.t. n \in Scope(C)} MLMI_{n,C}$$

In this step, the interface CTF is simplified by *locally marginalizing* out nets from individual cliques. Lines 10-29 contain the steps used to trim cliques. We first identify the set of large cliques with size $> mcs_{im}$ (L_c) and compute the metrics $maxMI$, $MLMI$ for all nets contained in them. Then, we find the set of non-interface nets present in these cliques and add them to a list N . We start with the net n that has the least $maxMI$. In case of a tie, we prefer the net present in fewer cliques. We find the list of cliques containing n (L_{cn}) and add cliques with size $\leq mcs_{im}$ to a list L_s . Next, we identify the subgraph of CTF corresponding to cliques in L_s and divide it into a set of disjoint subtrees (L_{st}). Net n is retained in the subtree that contains the clique with largest $MLMI$ (L_r). In case of a tie, we choose the subtree with the largest number of cliques. Net n is locally marginalized out from all other containing cliques and sep-sets. Any resultant non-maximal clique is removed, and its neighbors are connected to the containing clique.

If there still exist large cliques after removing non-interface nets, we add the interface nets present in those cliques to N . Nets are locally marginalized until the maximum clique size constraint is met. An interface net must be contained in atleast one clique in the interface CTF. Therefore, if an interface net is locally marginalized from all containing cliques, an independent clique is added to CTF.

Step 4: Re-assignment of clique factors: For each CT in the CTF, a root node is chosen at random. The factor for the root node is the same as the clique belief (lines 32-34). All other nodes are assigned factors by iterating through them in preorder, i.e., from the root node to leaves. An unvisited neighbor C_j of a node C_i is assigned the conditional probability $P(C_j|C_i)$ as the factor, which can be computed as $\psi_j = \frac{\beta_j}{\mu_{ij}}$ (lines 35-39).

B. Properties of the interface CTF

The following properties are satisfied by the CTF obtained after the simplification steps described in Sec. V-A.

Property I: All CTs in the interface CTF are valid.

Interface CTF obtained after the first step is valid since it is a subgraph of the existing CTF that connects the cliques containing nets in IN . The CTs in CTF obtained after the subsequent steps are valid because of the following reasons.

- It contains only maximal cliques. Any non-maximal cliques generated while simplification of CTF are removed. (lines 7, 27 in Alg. 3).
- It contains disjoint trees.

In steps 2 and 3, neighbors of the collapsed and non-maximal cliques are reconnected to CTF such that connectivity of the CTs is preserved. In Step 3, an edge is removed if the sep-set becomes empty after marginalization, thus further dividing a CT into disjoint CTs (lines 7, 26, 27 in Alg. 3). Therefore, no loops are introduced in the process.
- It satisfies RIP.
 - In Step 2, sep-sets in the input CTF are preserved while re-connecting the neighbors of the collapsed cliques to the new clique, thereby ensuring that RIP is satisfied. Similarly, since neighbors of non-maximal cliques are connected to the containing cliques, sep-sets are preserved.
 - In Step 3, nets are retained in a single connected component of the CTF (lines 21-22 in Alg. 3). Therefore, the subgraph of CTF over any net is connected.
- In Step 4, the re-assignment of factors is done using the calibrated CTF beliefs such that the product of factor beliefs is a valid JPD.

Property II: JPD of nets present within a clique in the interface CTF is the same as that in the partition from which it is derived.

The interface CTF is built from calibrated CTs where each clique belief is the JPD of nets present in the clique. After step 2, the entire JPD is preserved because it involves collapsing all containing cliques before marginalization. In Step 3, nets are locally marginalized from individual cliques without collapsing cliques containing the net. Therefore, in this step also, the JPD of nets contained within a clique is preserved.

In contrast, only the marginals are preserved across partitions in the partitioning approaches described in [15], [16].

Note: The JPD of nets present in different cliques in the interface model may not be preserved across partitions. This is because nets that are locally marginalized from a clique are also locally marginalized from the corresponding sep-sets.

Property III: Marginals are preserved across partitions.

The marginal probabilities can be computed from any containing partition or interface CTF. This directly follows from Property II.

Property IV: JPD of nets can be approximated both within and across partitions.

JPD of nets present in the same CT can be estimated by

performing variable elimination on the CT [1]. Approximate JPD of nets a, b belonging to adjacent partitions can be obtained by conditioning on a subset of interface nets (S_{in}) present within a single clique in the interface CTF via $P(a, b) = \sum_{S_{in}} P(a, S_{in}) * P(b|S_{in})$. This will always result in a valid JPD since joints within any clique in the interface CTF are consistent in adjacent partitions (by Property II). Similarly, JPD of nets belonging to non-adjacent partitions can be obtained using the chain rule by conditioning on cliques in subsequent interface CTFs.

VI. RESULTS

We evaluated the algorithm by computing two commonly used circuit metrics, namely the signal probability and the joint probability between gate inputs. We used circuits in three combinational benchmark suites, namely, ISCAS'85 [43], LGSynth'91 [44], and the recent EPFL'15 [45] benchmarks. The benchmarks were synthesized using the Cadence Genus v15.2 tool using the Faraday 55-nm technology library. All experiments were carried out on a 3.7-GHz Intel i7-8700 Linux system with 64-GB memory. As a proof-of-concept, the proposed algorithms were implemented in Python3 with the NetworkX library. The error is estimated by comparing the inferred signal probabilities with those obtained after zero delay simulation of 100,000 uniformly distributed random vectors. For all experiments presented in this section, we assume independence among the primary inputs.

A. Pre-processing circuit netlist

As in all previous works [2], [15], [32], [41], gates with fan-in greater than two are replaced by equivalent combinations of two-input gates. The signal probability at the output of inverters and buffers is a function of only the input probability. However, chains of inverters introduced while timing optimizations could increase the re-convergence depth. Thus, we remove all such gates from the circuit and connect their fanout nodes to the source node driving the chain. Even though this results in an increased fanout, it can potentially reduce the re-convergence depth. To maintain functional equivalence, we alter the CPD of the fanout gates to take care of inversion.

B. Formation of the clique tree model

The variable elimination algorithm [1] is used for triangulating the elimination/moralized graphs. The elimination order is found using the 'min-fill' metric, and the metric 'min-neighbors' is used in case of a tie [1]. Re-computing the number of fill-in edges each time a variable is eliminated increases the execution time. Therefore, we adopt the methodology suggested in [46] to compute only the change in the number of fill-in edges.

To evaluate the performance of the proposed method for incremental clique tree construction, we compare the results of our algorithm with recompilation based partitioning. In each iteration, a set of nodes is added by incrementally modifying the existing CT in our approach. The augmented BN subgraph is recompiled from scratch to find the CT in the second

Fig. 6. Comparison of the percentage of nets added to the first partition using the proposed incremental (marked in red) and the re-compilation (marked in blue) approaches as the max-clique size constraint (mcs_p) is varied for two sample benchmarks.

TABLE II

COMPARISON OF RUNTIME AND PERCENTAGE OF NETS ADDED TO THE FIRST PARTITION USING THE PROPOSED INCREMENTAL AND THE RE-COMPILATION APPROACHES WITH MAX-CLIQUE SIZE LIMIT SET TO 20.

Circuit	Runtime (sec)			Number of nets (%)		
	Re-compile	Proposed	Δ (%)	Re-compile	Proposed	Δ
C432	0.6	0.1	83	97	92	-5
C1908	0.4	0.1	75	64	48	-16
C3540	1.7	0.2	88	54	46	-8
C5315	12.0	0.7	94	74	69	-5
C6288	2.4	0.3	88	26	24	-2
C7552	23.8	0.7	97	94	85	-9
des	44.6	3.4	92	70	71	1
pair	3.6	0.4	89	49	50	1
cavlc	0.6	0.1	83	35	35	0
i2c	5.9	0.6	90	80	78	-2
voter	723.3	4.3	99	99	98	-1
Mean			89			-4

approach. Nets are added until the clique size constraint is violated. Figure 6 compares the percentage of nets that can be added to the first partition using both approaches as the max-clique size threshold is varied. The results are shown for two sample benchmarks but it holds good for other benchmarks as well. For *vda*, our method compares very well with complete recompilation. In *c6288*, there are significant differences for larger thresholds. However, such clique sizes are impractical for use owing to the associated space complexity. For, mcs_p of 20-30 the difference in percentage of nets added is reasonable.

Table II compares the runtime required and the percentage of nets added to the first partition using both approaches with $mcs_p = 20$. We show the results for a subset of benchmarks; the results for others are similar. Using the incremental approach results in an average of 89% reduction in runtime with an average 4% reduction in the percentage of nets that can be added to the partition while satisfying the max-clique size constraint. Clearly, we pay a very small penalty in the size of the partition for a substantial decrease in the runtime.

C. Impact of clique size constraints

As discussed in Sec. III, the proposed framework uses parameters mcs_p and mcs_{im} as maximum clique size constraints for the partition and the interface CTF, respectively. Table III compares the required runtime and the maximum estimation error in signal probabilities obtained using the

TABLE III

COMPARISON OF THE MAXIMUM ERROR AND RUNTIME USING THE PROPOSED METHODOLOGY WITH DIFFERENT CLIQUE SIZE CONSTRAINTS.

(mcs _p , mcs _{im})	Maximum error				Runtime (sec)			
	(10,5)	(15,10)	(20,15)	(25,20)	(10,5)	(15,10)	(20,15)	(25,20)
C432	0.201	0.129	0.066	0.003	<1	<1	<1	1
C1908	0.009	0.005	0.005	0.005	<1	1	2	29
C3540	0.289	0.148	0.089	0.102	2	2	8	138
C5315	0.125	0.063	0.035	0.005	2	2	6	67
C6288	0.116	0.117	0.064	0.020	4	4	8	101
C7552	0.226	0.036	0.005	0.005	2	2	3	24
des	0.075	0.068	0.039	0.017	9	12	24	274
pair	0.109	0.036	0.009	0.003	2	3	9	148
cavlc	0.110	0.119	0.086	0.081	2	4	9	150
i2c	0.073	0.073	0.061	0.039	2	2	5	84
voter	0.530	0.230	0.030	0.010	9	9	9	32

proposed approach for various values of mcs_p and mcs_{im} . Owing to memory constraints, we limit our experiments to clique sizes of 25. We have shown results for a subset of benchmarks. The results for others are similar. If mcs_p is large, more nets are included in each partition, reducing the error due to approximation. On the other hand, the runtime increases since the number of computations required for calibrating the CT is exponential in the clique sizes. A larger mcs_{im} reduces the number of approximations required to obtain the interface model from the clique tree corresponding to the previous partition. However, it does not leave room for the addition of many nets to the next partition, leading to more partitions. If mcs_{im} is very small, the estimation error is large. As a compromise, we choose mcs_{im} to be 5 less than mcs_p . Based on runtime considerations, we choose $mcs_p = 20$ for further experiments.

D. Comparison with existing interface models

We now compare the performance of the proposed interface model with two existing models. *Hard Partitioning (HP)* ([16]) is the simplest interface approximation. It assumes that all interface nets are independent. In the second model (referred to as *ApproxCBN* in [15]), the JPD of the interface nets is modeled via a *Tree Distribution (TD)* obtained using the pairwise MI between interface nets that have common parents, children, or grandparents. All other interface nets are assumed to be independent.

In [15], partition sizes are quite small due to limitations in computational resources then. For a fair comparison with the proposed model, we use Alg. 1 and modify the BuildInterfaceCTF function to obtain the HP and TD interface models. The first partition is the same in all three cases. Since the interface model is different, the subsequent partitions will be different. Signal probabilities can be inferred exactly for circuits like *parity*, *decode*, *adder*, *alu_ctrl* etc., where the maximum clique size is less than 20. In our results, we have only included circuits where partitioning was required. Table IV compares the number of partitions, the error statistics, and the runtime required for signal probability estimation using the different interface models. Overall, we can achieve average reduction of 46%, 48%, 61% in RMSE, maximum error, and percentage of nets with error > 5% respectively,

TABLE IV

COMPARISON OF REQUIRED RUNTIME AND ERROR STATISTICS OBTAINED WITH THE PROPOSED INTERFACE MODEL WITH THAT OBTAINED USING HARD PARTITIONING (HP) AND TREE DISTRIBUTION (TD) BASED INTERFACE MODELS.

	Benchmark	#Nets	#Partitions			RMSE				Maximum Error				% Nets with relative error > 5%				Runtime (sec)		
			HP	TD	Proposed	HP	TD	Proposed	$\Delta_{TD}(\%)$	HP	TD	Proposed	$\Delta_{TD}(\%)$	HP	TD	Proposed	$\Delta_{TD}(\%)$	HP	TD	Proposed
I	frg1	113	2	2	4	0.013	0.007	0.001	-86	0.081	0.054	0.006	-89	8	3	0	-100	<1	1	1
	c432	160	2	2	2	0.022	0.023	0.010	-57	0.154	0.161	0.066	-59	5	4	3	-25	<1	1	<1
	int2float	168	2	3	5	0.021	0.019	0.004	-79	0.227	0.227	0.035	-85	4	4	1	-75	<1	8	2
	x1	232	2	2	5	0.012	0.009	0.001	-89	0.123	0.081	0.002	-98	2	2	0	-100	<1	5	2
	c1908	237	2	3	5	0.024	0.016	0.001	-94	0.171	0.155	0.005	-97	9	5	0	-100	<1	2	2
	i9	350	2	2	11	0.015	0.006	0.002	-67	0.067	0.038	0.023	-39	12	0	0	-	1	43	3
	cavlc	490	3	5	20	0.016	0.015	0.013	-13	0.119	0.121	0.086	-29	8	9	7	-22	1	331	9
	priority	501	2	2	6	0.002	0.005	0.001	-80	0.028	0.076	0.003	-96	0	1	0	-100	1	4	3
	apex6	525	2	2	5	0.003	0.003	0.001	-67	0.023	0.022	0.009	-59	1	1	0	-100	1	6	2
	x3	528	2	2	4	0.003	0.002	0.001	-50	0.043	0.016	0.007	-56	1	0	0	-	1	6	1
	rot	549	2	2	7	0.004	0.003	0.001	-67	0.036	0.033	0.003	-91	1	1	0	-100	1	3	2
	c3540	562	3	5	19	0.034	0.033	0.016	-52	0.291	0.295	0.089	-70	23	19	14	-26	1	48	8
	frg2	564	2	2	4	0.001	0.001	0.001	0	0.013	0.013	0.003	-77	0	0	0	-	1	1	1
	i8	577	3	4	17	0.017	0.017	0.004	-76	0.074	0.075	0.016	-79	16	15	1	-93	1	198	9
	c2670	602	2	2	5	0.002	0.002	0.001	-50	0.019	0.023	0.004	-83	0	0	0	-	1	2	1
	c5315	864	2	3	9	0.025	0.019	0.002	-89	0.139	0.130	0.035	-73	13	11	1	-91	1	37	6
	c7552	989	2	2	5	0.020	0.013	0.001	-92	0.233	0.173	0.005	-97	6	3	0	-100	1	12	3
	pair	996	3	3	11	0.014	0.005	0.001	-80	0.110	0.066	0.009	-86	4	2	0	-100	1	13	9
	bar	1277	3	3	52	0.072	0.072	0.001	-99	0.133	0.137	0.005	-96	59	59	0	-100	10	1936	35
	i10	1337	4	5	24	0.021	0.014	0.007	-50	0.200	0.123	0.071	-42	8	8	4	-50	2	50	16
	c6288	1706	3	4	12	0.019	0.015	0.007	-53	0.140	0.095	0.064	-33	13	10	2	-80	2	68	8
	des	1972	2	3	21	0.011	0.010	0.003	-70	0.084	0.083	0.039	-53	6	5	0	-100	7	559	24
	max	2357	2	2	5	0.009	0.004	0.001	-75	0.117	0.069	0.004	-94	2	1	0	-100	8	163	11
	sin	3134	11	23	112	0.046	0.044	0.039	-11	0.352	0.340	0.205	-40	36	36	28	-22	7	1636	76
	arbiter	3667	4	7	114	0.008	0.007	0.007	0	0.120	0.120	0.044	-63	14	11	11	0	13	404	244
	voter	7502	3	2	6	0.019	0.019	0.001	-95	0.434	0.434	0.032	-93	1	1	0	-100	7	13	9
	multiplier	13272	26	96	128	0.030	0.024	0.021	-12	0.371	0.288	0.188	-35	20	18	13	-28	59	15449	310
	log2	16990	26	71	435	0.046	0.045	0.039	-13	0.476	0.412	0.327	-21	35	33	28	-15	62	52078	841
	div	23859	31	31	197	0.023	0.011	0.003	-73	0.180	0.134	0.065	-51	8	3	0	-100	78	11575	237
II	router	152	2	2	2	<0.001	<0.001	<0.001	-	0.002	0.002	0.002	0	0	0	0	-	<1	<1	<1
	dec	331	2	2	17	<0.001	<0.001	<0.001	-	0.002	0.002	0.002	0	0	0	0	-	<1	1	1
	c499	195	2	2	3	0.001	0.001	0.001	0	0.004	0.004	0.004	0	0	0	0	-	<1	1	<1
	c1355	203	2	2	3	0.002	0.002	0.001	-50	0.006	0.006	0.004	-33	0	0	0	-	<1	1	1
	c880	250	2	2	4	0.001	0.001	0.001	0	0.005	0.006	0.004	-33	0	0	0	-	<1	1	1
	dal	545	2	4	17	0.001	0.001	0.001	0	0.013	0.012	0.009	-25	0	0	0	-	1	84	8
	too_large	217	2	3	8	0.009	0.009	0.005	-44	0.07	0.057	0.049	-14	6	6	3	-50	<1	9	3
	i2c	770	2	2	11	0.007	0.005	0.005	0	0.094	0.062	0.061	-2	2	2	1	-50	1	41	5
	alu2	247	3	4	13	0.016	0.014	0.013	-7	0.138	0.101	0.094	-7	11	11	7	-36	<1	42	4
	alu4	396	3	6	20	0.024	0.024	0.022	-8	0.125	0.127	0.117	-8	28	29	23	-21	1	58	7
	k2	942	4	8	47	0.016	0.015	0.014	-7	0.122	0.118	0.119	1	10	10	8	-20	2	398	28
	vda	510	3	6	22	0.014	0.014	0.012	-14	0.143	0.141	0.132	-6	10	10	7	-30	1	241	11
	square	11812	22	35	238	0.025	0.026	0.023	-12	0.173	0.131	0.124	-5	33	35	31	-11	29	3845	546
	sqr	26944	40	43	502	0.047	0.046	0.036	-22	0.227	0.221	0.199	-10	47	46	38	-17	49	744	314
	mem_ctrl	21032	13	39	232	0.057	0.052	0.047	-10	0.447	0.447	0.427	-4	27	25	22	-12	102	27165	1481
Mean									-46				-48				-61			

#Nets: Number of nets before pre-processing the circuit netlist.

$\Delta_{TD}(\%)$: Percentage difference in the statistics obtained with the proposed model with respect to the TD [15].

when compared with the TD method. Looking at the results in more detail, we have divided the circuits into two categories.

- *Category I*: For all circuits in this category, the proposed method results in a substantial reduction in maximum error and percentage of nets with large errors compared to both HP and TD. In some of the benchmarks like *c1908*, *c7552*, *bar* (128-bit barrel shifter), *max* (128-bit), *voter*, and *int2float* converter, there is a dramatic drop in error. Significant reduction in the maximum error is also seen for some very large arithmetic EPFL circuits like divisor (64-bit) and multiplier (64-bit). There is a reduction of error in *log2*, but the maximum error continues to be large.
- *Category II*: In terms of error, all methods give similar results. However, we get a reduction in the percentage of nets with large errors and a significant improvement in runtime compared to TD. There are a few circuits (*sqr*, *mem_ctrl*) where the maximum error is still large.

The first six circuits have short reconvergent paths, which are typically taken care of by all methods. Other methods like the Correlation Coefficient Method (CCM) [41], the Weighted Averaging Algorithm (WAA) [47] also work well for these circuits. OBDD based methods [20], [21] can also easily handle these circuits. On the other hand, circuit *mem_ctrl* has long and nested reconvergent paths. Most of the short paths are taken care of, as evidenced by the low RMSE, but there is not much reduction in the maximum error.

As expected, the runtime for the proposed model is larger when compared to HP, and the number of partitions is higher than both TD and HP models. The runtime is lower than TD for nearly all circuits. Even though it is implemented in Python, the runtime is of the order of a few seconds for most circuits. The maximum runtime is for circuits *log2* and *mem_ctrl* but it is substantially lower than that required for TD. This is because, in the proposed model, we find corre-

TABLE V

COMPARISON OF MAXIMUM ERROR AND RMSE IN JOINT PROBABILITIES OF THE GATE INPUTS THAT DRIVE THE INTERNAL NETS.

Circuit	Max Error	RMSE (P ₀₀)	RMSE (P ₀₁)	RMSE (P ₁₀)	RMSE (P ₁₁)
c432	0.066	0.007	0.008	0.007	0.008
c499	0.003	0.001	0.001	0.001	0.001
c880	0.003	0.001	0.001	0.001	0.001
c1355	0.004	0.001	0.001	0.001	0.001
c1908	0.004	0.001	0.001	0.001	0.001
c2670	0.004	0.001	0.001	0.001	0.001
c3540	0.100	0.013	0.012	0.013	0.015
c5315	0.031	0.002	0.002	0.002	0.002
c6288	0.064	0.006	0.006	0.006	0.005
c7552	0.005	0.001	0.001	0.001	0.001

lations only between variable pairs that belong to a common clique. On the other hand, TD requires variable elimination to compute joint probabilities for structurally related variable pairs. The runtime increases with the number of variable pairs for which correlations need to be computed.

An important application of signal probability estimation is reliability analysis and the analysis of probabilistic circuits. Signal correlation coefficients are often used to get rapid estimates of reliability [32], [40]. These methods basically require the JPD of gate inputs. Table V shows the errors obtained for ISCAS'85 benchmarks. For most circuits, the error is small. While the maximum error is 0.1 in the circuit c3540, the RMSE is relatively low, indicating that the error is small for most input pairs. Therefore, the proposed methodology can be used in conjunction with existing techniques to obtain accurate reliability estimates.

VII. CONCLUSIONS

We propose an algorithm for incremental construction of the CT model under clique size constraints. We show that the resultant CT is valid. This approach gives a significant runtime improvement over complete recompilation-based partitioning. We also propose a method to find the interface CT such that the marginals are preserved, and the beliefs within cliques are consistent across partitions. Thus, a valid JPD can be extracted, which is needed for inference of evidence-based queries. The input max-clique size constraints can be altered to trade-off accuracy and runtime. The proposed framework provides a significant improvement in the accuracy of signal probabilities. It also gives good estimates of joint probabilities of gate inputs which are used to get rapid estimates of reliability metrics.

Some circuits still pose a challenge in terms of the maximum error achieved. Constructing separate BNs for different outputs or a cluster of outputs can help achieve better estimates. Other possibilities that can help reduce the space complexity are compression of clique beliefs and use of approximate message passing algorithms.

REFERENCES

- [1] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [2] S. Bhanja and N. Ranganathan, "Dependency preserving probabilistic modeling of switching activity using bayesian networks," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pp. 209–214, 2001.
- [3] S. Bhardwaj, S. Vruthula, and D. Blaauw, "Probability distribution of signal arrival times using bayesian networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1784–1794, 2005.
- [4] T. Rejimon, K. Lingasubramanian, and S. Bhanja, "Probabilistic Error Modeling for Nano-Domain Logic Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 55–65, 2009.
- [5] W. Ibrahim, V. Beiu, and A. Beg, "Greda: A fast and more accurate gate reliability eda tool," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 509–521, 2012.
- [6] Y. Xiang, D. Poole, and M. P. Beddoes, "Multiply sectioned bayesian networks and junction forests for large knowledge-based systems," *Computational Intelligence*, vol. 9, no. 2, pp. 171–220, 1993.
- [7] S. Srinivas, "A probabilistic approach to hierarchical model-based diagnosis," in *Uncertainty Proceedings 1994*, pp. 538–545, Elsevier, 1994.
- [8] A. Darwiche, "Dynamic jointrees," *CoRR*, vol. abs/1301.7369, 2013.
- [9] S. C. Seth and V. D. Agrawal, "A new model for computation of probabilistic testability in combinational circuits," *Integration*, vol. 7, no. 1, pp. 49–75, 1989.
- [10] T. Rejimon and S. Bhanja, "A timing-aware probabilistic model for single-event-upset analysis," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 14, no. 10, pp. 1130–1139, 2006.
- [11] S. S. Ramani and S. Bhanja, "Any-time probabilistic switching model using bayesian networks," in *Proceedings of the 2004 international symposium on Low power electronics and design*, pp. 86–89, 2004.
- [12] S. Chaudhuri, J. S. Wong, and P. Y. Cheung, "Timing speculation in fpgas: Probabilistic inference of data dependent failure rates," in *2011 International Conference on Field-Programmable Technology*, pp. 1–8, IEEE, 2011.
- [13] K. H. Jin, D. Wu, and L. Wu, "On designing approximate inference algorithms for multiply sectioned bayesian networks," in *2009 IEEE International Conference on Granular Computing*, pp. 294–299, IEEE, 2009.
- [14] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, (San Francisco, CA, USA), pp. 467–475, Morgan Kaufmann Publishers Inc., 1999.
- [15] S. Bhanja and N. Ranganathan, "Cascaded bayesian inferencing for switching activity estimation with correlated inputs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 12, pp. 1360–1370, 2004.
- [16] S. Bhanja and N. Ranganathan, "Switching activity estimation of large circuits using multiple bayesian networks," in *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15th International Conference on VLSI Design*, pp. 187–192, IEEE, 2002.
- [17] S. Bhanja and N. Ranganathan, "Switching activity estimation of vlsi circuits using bayesian networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 4, pp. 558–567, 2003.
- [18] J. Costa, L. Silveira, S. Devadas, and J. Monteiro, "Power estimation using probability polynomials," *Design Automation for Embedded Systems*, vol. 9, no. 1, pp. 19–52, 2004.
- [19] B. Kapoor, "Improving the accuracy of circuit activity measurement," in *Proceedings of the 31st annual Design Automation Conference*, pp. 734–739, 1994.
- [20] R. Marculescu, D. Marculescu, and M. Pedram, "Probabilistic modeling of dependencies during switching activity analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 73–83, 1998.
- [21] C.-S. Ding, C.-Y. Tsui, and M. Pedram, "Gate-level power estimation using tagged probabilistic simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 11, pp. 1099–1107, 1998.
- [22] D. L. Draper, "Clustering without (Thinking about) Triangulation," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, (San Francisco, CA, USA), pp. 125–133, Morgan Kaufmann Publishers Inc., 1995.
- [23] M. J. Flores, J. A. Gámez, and K. G. Olesen, "Incremental Compilation of Bayesian Networks," in *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI'03, (San Francisco, CA, USA), pp. 233–240, Morgan Kaufmann Publishers Inc., 2002.
- [24] K. G. Olesen and A. L. Madsen, "Maximal prime subgraph decomposition of bayesian networks1," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 32, pp. 21–31, feb 2002.
- [25] J. Choi and R. A. Rutenbar, "Configurable and scalable belief propagation accelerator for computer vision," in *2016 26th International*

- Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, IEEE, 2016.
- [26] O. U. Khan and D. D. Wentzloff, “Hardware accelerator for probabilistic inference in 65-nm cmos,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 837–845, 2015.
 - [27] L. Zheng and O. Mengshoel, “Optimizing parallel belief propagation in junction trees using regression,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 757–765, 2013.
 - [28] L. Zheng, O. Mengshoel, and J. Chong, “Belief propagation by message passing in junction trees: Computing each message faster using gpu parallelization,” in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI’11*, (Arlington, Virginia, USA), pp. 822–830, AUAI Press, 2011.
 - [29] N. Ma, Y. Xia, and V. K. Prasanna, “Data parallelism for belief propagation in factor graphs,” in *2011 23rd International Symposium on Computer Architecture and High Performance Computing*, pp. 56–63, IEEE, 2011.
 - [30] K. P. Parker and E. J. McCluskey, “Probabilistic treatment of general combinational networks,” *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 668–670, 1975.
 - [31] D. T. Franco, M. C. Vasconcelos, L. Naviner, and J.-F. Naviner, “Signal probability for reliability evaluation of logic circuits,” *Microelectronics Reliability*, vol. 48, no. 8, pp. 1586–1591, 2008.
 - [32] M. R. Choudhury and K. Mohanram, “Reliability analysis of logic circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 3, pp. 392–405, 2009.
 - [33] R. Xiao and C. Chen, “Gate-level circuit reliability analysis: A survey,” *VLSI Design*, vol. 2014, 2014.
 - [34] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, “A stochastic computational approach for accurate and efficient reliability evaluation,” *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336–1350, 2014.
 - [35] A. Vijayan, S. Kiamehr, M. Ebrahimi, K. Chakrabarty, and M. B. Tahoori, “Online Soft-Error Vulnerability Estimation for Memory Arrays and Logic Cores,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 499–511, 2018.
 - [36] S. Bhanja and S. Sarkar, “Probabilistic modeling of qca circuits using bayesian networks,” *IEEE Transactions on Nanotechnology*, vol. 5, no. 6, pp. 657–670, 2006.
 - [37] N.-Z. Lee and J.-H. R. Jiang, “Towards formal evaluation and verification of probabilistic design,” *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1202–1216, 2018.
 - [38] Z. Vasicek, “Formal methods for exact analysis of approximate circuits,” *IEEE Access*, vol. 7, pp. 177309–177331, 2019.
 - [39] L. Tan, Z. Li, G. Su, and D. Wang, “Asymptotically linear analysis and gate probability allocation schemes in probabilistic circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 596–606, 2020.
 - [40] C. Chen and R. Xiao, “A fast model for analysis and improvement of gate-level circuit reliability,” *Integration*, vol. 50, pp. 107–115, 2015.
 - [41] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco, “Estimate of signal probability in combinational logic networks,” in *Proceedings of the 1st European test conference*, pp. 132–133, IEEE Computer Society, 1989.
 - [42] D. A. Schult and P. Swart, “Exploring network structure, dynamics, and function using networkx,” in *Proceedings of the 7th Python in science conferences (SciPy 2008)*, vol. 2008, pp. 11–16, Pasadena, CA, 2008.
 - [43] F. Brglez, “A neural netlist of 10 combinational benchmark circuits,” *Proc. IEEE ISCAS: Special Session on ATPG and Fault Simulation*, pp. 151–158, 1985.
 - [44] S. Yang, “Logic synthesis and optimization benchmarks user guide: Version 3.0,” tech. rep., MCNC Technical Report, Jan. 1991.
 - [45] L. Amarú, P.-E. Gaillardon, and G. De Micheli, “The eplf combinational benchmark suite,” in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.
 - [46] K. Kask, A. Gelfand, L. Otten, and R. Dechter, “Pushing the power of stochastic greedy ordering schemes for inference in graphical models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, 2011.
 - [47] B. Krishnamurthy and I. G. Tollis, “Improved techniques for estimating signal probabilities,” *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1041–1045, 1989.