Local Conditioning:

Exact Message Passing for Cyclic Undirected Distributed Networks

Matthew G. Reyes

Independent Researcher and Consultant
Ann Arbor, MI 48105, USA
matthewgreyes@yahoo.com
matthewgreyes.com

Abstract. This paper addresses practical implementation of *summing out*, *expanding*, and *reordering* of messages in Local Conditioning (LC) for undirected networks. In particular, incoming messages conditioned on potentially different subsets of the receiving node's relevant set must be *expanded* to be conditioned on this relevant set, then *reordered* so that corresponding columns of the conditioned matrices can be fused through element-wise multiplication. An outgoing message is then reduced by *summing out* loop cutset nodes that are *upstream* of the outgoing edge. The emphasis on implementation is the primary contribution over the theoretical justification of LC given in Fay et al. Nevertheless, the complexity of Local Conditioning in grid networks is still no better than that of Clustering.

Keywords: Local Conditioning, Belief Propagation, Distributed Systems, Message Passing, Cyclic Networks, Recursive Algorithms

1 Introduction

Local Conditioning (LC) has recently been proposed as a method for performing exact truly distributed inference in cyclic undirected networks [12]. Originally introduced in the context of directed networks [4], [5], LC ceased to gain attention in favor of Clustering methods such as the well-known Junction Tree algorithm [8] and generalizations thereof [16]. This was likely due both to the relative complexity of implementing Local Conditioning, and a surge of interest in approximate variations of Belief Propagation for cyclic networks [6], [9]. Moreover, the principle advantage of Local Conditioning over Clustering methods was not as pressing as it is today. That is to say, Local Conditioning is *truly distributed* in the sense of involving messages between individual nodes of the original network. Truly distributed inference is required by physically distributed systems such as autonomous vehicles, industrial warehouse sensors, or networks of delivery drones, for which clustering of nodes is not feasible. Indeed, it is precisely the current preponderance of physically distributed computing that motivates a renewed interest in Local Conditioning. This paper develops

Local Conditioning in the case of undirected cyclic networks. The simpler topological structure of undirected networks and the economy of Gibbs distributions permits a more streamlined presentation than in the case of Bayesian networks [5]. Moreover, the discussion is more detailed than earlier work [12] and lays particular emphasis on the implementation of Local Conditioning.

Belief Propagation (BP) on an acyclic network [11] consists of two main ideas, *fusion* and *propagation*. Each of these can be interpreted as an appropriately defined multiplication of matrices: fusion as an element-wise product of two or more incoming message vectors; propagation as standard matrix multiplication of a fused vector with a matrix associated with an outgoing edge. It is important to distinguish between two senses in which BP can be said to be a *distributed* algorithm. The first is as formalized in the *Generalized Distributive Law* [1], which presents Belief Propagation as a means of solving an otherwise intractable summation[1] by decomposing it into a sequence of much smaller computations. In this sense, BP is a *distributed algorithm* even if it is implemented on a centralized computer. This is significant as by far the most common means of adapting BP for performing exact inference on cyclic networks is to form an acyclic network by clustering nodes [8]. This clustering is possible precisely when BP is implemented on a centralized computer.

Conditioning [10], [13], is an adaption of BP for performing exact inference in a cyclic network by effectively 'opening up' the network. Conditioning was introduced around the same time as Clustering methods. It consists of the same two ideas of fusion and propagation, only instead of vector messages there are matrix messages, each column of which corresponds to, i.e., is conditioned on, a different configuration of a set of so-called *loop cutset* nodes. Conditioning on the loop cutset nodes allows standard acyclic Belief Propagation to be implemented on a cyclic network, by having each loop cutset node interact with its neighbors *as if* there are multiple copies of it, all constrained to have the same value. The multiple copies of a loop cutset node effectively break the cycles going through the node. The constraint that individual copies have the same value results in the increase in the number of message vectors that need to be passed, so that messages corresponding to different common assignments can be aligned.

Local Conditioning [4], [5], [12], reduces the number of columns that need to be included in the message matrices by noting that a message matrix passed over a given edge need only be conditioned on a particular loop cutset node if there is at least one copy of the loop cutset on either side of the edge. In addition to the standard operations of fusion and propagation, there are three additional operations necessary for implementation: *summing out*, *expanding*, and *reordering*. Summing out loop cutset nodes that are *upstream* of a message decreases the number of columns in the message matrix and is part of the propagation step. As a result, incoming messages to a given node will be conditioned on potentially different subsets of the loop cutset nodes. Therefore, incoming message matrices need to be expanded to a common subset. The expanded message matrices then

[1]The reader should consult [1] for the algebraic equivalence of other operations.

need to be reordered so that corresponding columns of the incoming messages in turn correspond to the same configuration on the common subset.

This paper addresses practical implementation of *summing out*, *expanding*, and *reordering* for Local Conditioning. In addition, it discusses the *associated tree* and *stability scaling*, or normalization, important for the practical implementation of general Conditioning. Stability scaling is used in practice with standard Belief Propagation as a means of preventing numerical overflow or underflow in large networks. Scaling the messages of BP does not affect probabilities computed therefrom. However, in Conditioning, columns of the matrix messages are conditioned on different configurations of loop cutset nodes. In order to combine columns of scaled messages in a way that still yields correct probabilities for the cyclic network, messages corresponding to different loop cutset configurations must be scaled with a common factor rather than independently. The present paper does not address finding such a common factor for scaling.

This paper is organized as follows. Section 2 provides an overall goal statement. Sections 3 and 4 provide background on Gibbs distributions and Belief Propagation, respectively. Section 5 provides a brief account of techniques to address cycles in networks. Section 6 provides a discussion of node-specific message passing for Conditioning. Section 7 presents the development of Local Conditioning for cyclic undirected networks. Section 8 concludes with future direction.

2 Goal Statement: Single-Node Inference on Networks

The *inference* addressed in this paper is the computation of probabilities at individual nodes in the network. Such probabilities will be used by nodes for making individual decisions. In general such probabilities will be inferred conditioned upon observed data. However, such conditioning can be incorporated into the framework of an unconditioned model, which is presented here for simplicity.

A network $G = (V, E)$ consists of a set of *nodes* V and a set of *edges* E consisting of pairs of elements of V. For edge $\{i, j\} \in E$, nodes i and j are said to be *neighbors*. The graph $G \setminus \{i, j\}$ is obtained by removing the edge between i and j. For node i, ∂i denotes the set of neighbors of i. Given a neighbor $j \in \partial i$, the set $k \in \partial j \setminus i$ denotes the set of neighbors of j not including i. For each $i \in V$, associate a random variable X_i assuming values in a common *alphabet* $\mathcal{X}$. Let x_i denote a specific value that X_i assumes. An assignment $\mathbf{x} = (x_1, \ldots, x_{|V|})$ to all nodes in the network is referred to as a *configuration*, and $\mathbf{X}$ denotes the *random field* of possible configurations taking values $\mathcal{X}^{|V|}$. For a subset $L \subset V$, $\mathbf{X}_L$ denotes the random field on L, and $\mathbf{x}_L$ a configuration on L. A *path* is a sequence of nodes $k, \ldots, i$ such that any two successive nodes in the sequence are neighbors, and each such pair of neighbors occurs only once in the sequence. A *cycle* is a path that begins and ends at the same node. The goal of this paper is to compute, for each node $i \in V$, the vector of probabilities $p_i(x_i)$ for each value x_i that node i can assume. In particular, we want to find an algorithm for computing these probabilities when the network G has cycles and corresponds to a physically distributed system requiring truly distributed inference.

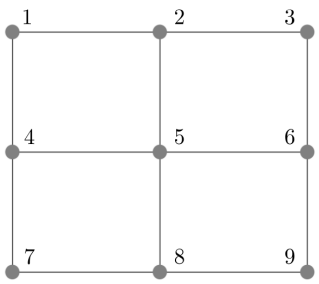

Fig. 1. 3×3 grid network.

As an example, consider a grid graph as shown in Figure 1. The grid topology is a good first-order approximation to a number of distributed networks of practical interest. For example, autonomous vehicles will communicate with those that are close by as well as with base stations positioned along roadsides and medians; in addition, industrial sensors and robots will likely be positioned along warehouse rows and columns. Moreover, the simple geometric structure of grid graphs permits an exploration and articulation of basic principles that, once grasped, can be abstracted to more general topologies.

While the present discussion is couched in terms of probabilities, the contribution of this paper is ultimately in the graph-theoretic manipulation of a network to facilitate communication. As such, the ideas of this paper are applicable to settings in which information other than probabilities are desired.

3 Gibbs Distributions

Probabilities will be computed with respect to *Gibbs* distributions. *Any* multivariate probability distribution that assigns non-zero probability to every configuration of the variables can be parametrized as a Gibbs distribution [7]. Gibbs distributions are *data-driven* in the sense of predicting observed data without making any additional assumptions beyond positivity [3]. Since one should not exclude the possibility of a configuration simply because the configuration has not yet been observed [14], the positivity assumption is a natural one to include in a model, and as such, Gibbs distributions are an extremely general class of models. Furthermore, their conditional independence structure admits a graphical interpretation that maps directly onto problems involving variables whose interdependence arises from communication or physical proximity.

For neighbors i and j there is an *edge potential*, $\Psi_{ij} = [\Psi_{ij}(x_i, x_j)]$, that maps each configuration (x_i, x_j) to a positive number. Likewise, for each node i there is a *self potential* $\Phi_i = [\Phi_i(x_i)]$ that assigns each value of x_i to a positive number. This paper assumes a finite alphabet $\mathcal{X}$, in which case the edge potential Ψ_{ij} can be thought of as a matrix, the self potential Φ_i a vector. To make things

concrete, consider an *Ising* model, in which $\mathcal{X} = \{-1, 1\}$ and

$$\Phi_i = \begin{bmatrix} e^{\alpha_i} \\ e^{-\alpha_i} \end{bmatrix} \quad , \quad \Psi_{ij} = \begin{bmatrix} e^{\theta_{ij}} & e^{-\theta_{ij}} \\ e^{-\theta_{ij}} & e^{\theta_{ij}} \end{bmatrix},$$

for parameters α_i, θ_{ij}.

The *belief* Z_i for node i is a vector with components $[Z_i(x_i)]$ defined as

$$Z_i(x_i) = \Phi_i(x_i) \sum_{\mathbf{x}_{V \setminus i}} \prod_{j,k} \Psi_{jk}(x_j, x_k) \prod_j \Phi_j(x_j).$$

4 Belief Propagation for Acyclic Networks

Let $\{i, j\}$ be an edge such that removing the edge between i and j disconnects the network. Belief Propagation (BP) is ultimately defined for *acyclic* networks, in which the removal of any edge disconnects the network. For such an edge, let $G_{i\setminus j}$ be the component of $G \setminus \{i, j\}$ that contains i. Furthermore, let $Z_i^{i\setminus j}$ be the belief for node i with respect to the Gibbs distribution on $G_{i\setminus j}$ that inherits potentials from the original Gibbs distribution on G in the natural way. It can be shown that

$$Z_i \;\; = \;\; Z_i^{i\setminus j} \Psi_{ji} Z_j^{j\setminus i} \; . \tag{1}$$

We define the *message* from j to i as

$$m_{j\to i} \;\; \stackrel{\Delta}{=} \;\; \Psi_{ji} Z_j^{j\setminus i} \; . \tag{2}$$

Note that this definition does not work if $\{i, j\}$ is not a cut edge. Node k is said to be *upstream* of the message $m_{j\to i}$ if $k \in G_{j\setminus i}$. The message from j to i summarizes information about the potentials in $G_{j\setminus i}$ in that all potentials involving nodes in $G_{j\setminus i}$ have been *summed out*. Likewise, k is said to be *downstream* of $m_{j\to i}$ if $k \in G_{i\setminus j}$. One can see that $Z_j^{j\setminus i}$, and therefore $m_{j\to i}$, is not a function of any downstream variables. These simple observations are critical for the development of Local Conditioning in Section 7.

It is helpful to think of BP as a *fusion* of message vectors incoming to a node and the self potential for that node, followed by *propagation* of the fused vector via multiplication with the corresponding edge potential matrix. Decomposing both $Z_i^{i\setminus j}$ and $Z_j^{j\setminus i}$ recursively according to (1) provides the respective formulas for beliefs and messages:

$$Z_i = \Phi_i \prod_{j \in \partial i} m_{j\to i} \tag{3}$$

and

$$m_{j\to i} = \Psi_{ji} \Phi_j \prod_{k \in \partial j \setminus i} m_{k\to j} \; . \tag{4}$$

Interest in the belief Z_i is due to the fact that normalizing it gives the probabilities that node i assumes value -1 or 1. That is,

$$p_i(x) = \frac{Z_i(x)}{\sum_{x \in \{-1,1\}} Z_i(x)} \ .$$

To prevent numerical overflow or underflow in large networks, the messages in (4) are scaled, or normalized. This does not affect computation of probabilities.

5 Dealing with Cycles

When a network has cycles, the message recursion of (4) will in general not result in correct computation of beliefs. Nevertheless, one can form an acyclic network by grouping the nodes of the original network into *cluster nodes* and creating an edge between two cluster nodes if each contains an endpoint of an edge in the original network [8]. For example, one can cluster the grid network of Figure 1 into a chain network by creating clusters $c_1 = \{1, 4, 7\}$, $c_2 = \{2, 5, 8\}$, and $c_3 = \{3, 6, 9\}$, and edges $\{c_1, c_2\}$ and $\{c_2, c_3\}$. This approach is relatively straightforward if the algorithm is implemented on a centralized computer, where it is simply a matter of creating new variables with larger alphabets. However, if the nodes correspond to distributed units such as autonomous vehicles or industrial sensors, Clustering requires additional units with which clusters of individual units would need to be able to communicate. While such additional layers of infrastructure and communication may be feasible in some settings, it is important to consider *truly distributed* algorithms in which messages are passed between nodes of the original network.

Loopy Belief Propagation (LBP) [9] is a truly distributed algorithm for performing *approximate* inference. In LBP nodes follow the message recursion of (4) *as if* they were part of an acyclic network. While LBP has been extensively studied, there is as yet little understanding of what exactly it computes, most of the results focusing on whether and when it converges and saying little about what it converges to. Another truly distributed algorithm for approximate inference is the popular Tree-Reweighted (TRW) version of BP [15]. In TRW, exact acyclic BP is performed on a sequence of spanning trees of the original cyclic network, where the potentials $\{\Phi_i\}$ and $\{\Psi_{ij}\}$ are *reweighted* at each iteration in such a way as to ensure increasingly better inference. The method of Conditioning, discussed in the next section, can also be viewed as performing standard BP on a sequence of acyclic networks.

6 Conditioning for Undirected Networks

As mentioned in Section 1, Conditioning [10] is an adaptation of BP for exact truly distributed message passing in cyclic networks. As with Clustering, it was

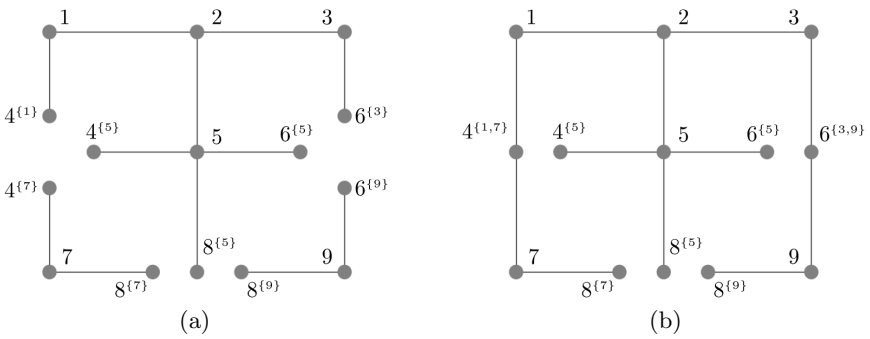

Fig. 2. (a) Completely opened up 3×3 grid network based on loop cutset $\{4, 6, 8\}$; (b) associated tree formed by re-identifying $4^{\{1\}}$ and $4^{\{7\}}$, and $6^{\{3\}}$ and $6^{\{9\}}$. Note that $\mathcal{N}_6 = \{3, 9\}$ and $\mathcal{L}_6 = \{5\}$, while $\mathcal{N}_8 = \{\ \}$ and $\mathcal{L}_8 = \{7, 5, 9\}$.

initially studied in the context of directed networks. Let L be a subset of nodes. The belief at node i can be computed as

$$Z_i = \sum_{\mathbf{x}_L} Z_i^{(\mathbf{x}_L)}, \tag{5}$$

where $Z_i^{(\mathbf{x}_L)}$ is the belief at i *conditioned on* the configuration $\mathbf{x}_L$. If G is acyclic, then as in the previous section, one can use Belief Propagation to compute conditioned beliefs $Z_i^{(\mathbf{x}_L)}$ from conditioned messages $m_{j\to i}^{(\mathbf{x}_L)}$. If G is cyclic, one can still use BP to compute beliefs by choosing L to be a *loop cutset*, a set of nodes whose removal eliminates all cycles in the network. This is because conditioning on a node $l \in L$ effectively *splits* l into multiple copies, each connected to a different subset of l's neighbors ∂l, where the copies of l are constrained to have the same value, the value upon which the original node l is being conditioned. The process of splitting all nodes in L can be interpreted as *opening up* the original cyclic network G at the nodes in L. There was a great deal of research on finding loop cutsets in the context of directed networks [2]. The method of Conditioning can be viewed as performing BP on an opened up version of G once for each configuration of the loop cutset L, and then combining the conditioned beliefs as in (5). However, as with Clustering, actually creating a new network is possible only when the network and processing thereon resides on a centralized computer. Therefore, the opening of G is just a schematic for visualizing node-specific messages that occur on the original cyclic network.

6.1 The Associated Tree and Node-Specific Message Passing

A procedure for finding an *opening* of G based on a loop cutset L is, for each $l \in L$, remove l and all edges incident to l, then for each neighbor $j \in \partial l$, attach a *copy* $l^{\{j\}}$ of l to j. As illustrated in Figure 2 (a), the resulting network may be

disconnected. It is relatively straightforward to show, however, that one can *reidentify* copies of a split loop cutset node to form a connected network that is still acyclic. This is illustrated in Figure 2 (b). In principle one can use a connected or disconnected opening. However, a disconnected opening adds an additional layer of processing that cannot be performed in a truly distributed manner. For this reason one should consider a connected opening, which is referred to as the *associated tree* and denoted by T.

In standard BP all nodes form outgoing messages and compute beliefs in identical ways, using (4) and (3). In Conditioning, loop cutset nodes will form outgoing messages and compute beliefs from incoming messages differently depending on whether its neighbors are leaf or non-leaf nodes in T. For loop cutset node l, let $\mathcal{L}_l \subset \partial l$ denote the neighbors of l for which the copy of l attached to the neighbor is a leaf node in T. Loop cutset node l will interact with j, in the original network, according to standard BP rules *as if* l were a leaf connected only to j. On the other hand, let $\mathcal{N}_l = \partial l \setminus \mathcal{L}_l$ be the neighbors of l that are connected to a non-leaf copy of l in T. In this case, l will interact with neighbors in $\mathcal{N}_l$ according to standard BP rules *as if* they were its only neighbors.

In order to correctly compute beliefs, a loop cutset node must use a consistent rule for dividing its self-potential among the messages it passes to neighbors in $\mathcal{L}_l$ and $\mathcal{N}_l$. In particular, it should use a different potential $\Phi_{l^j} \stackrel{\Delta}{=} \Phi_l^{\alpha_j}$ for each neighbor $j \in \mathcal{L}_l$, and another $\Phi_{l^{\mathcal{N}_l}} \stackrel{\Delta}{=} \Phi_l^{\alpha_{\mathcal{N}_l}}$ for all neighbors in $\mathcal{N}_l$, such that $\sum_{j \in \mathcal{L}_l} \alpha_j + \alpha_{\mathcal{N}_l} = 1$. To account for conditioning on a particular configuration $\mathbf{x}_L$ of the loop cutset, these self potentials will have to be modified, respectively, as $\Phi_{l^{\mathcal{N}_l}}^{(x_l)} \stackrel{\Delta}{=} \Phi_{l^{\mathcal{N}_l}} \delta^{(x_l)}$ and $\Phi_{l^j}^{(x_l)} \stackrel{\Delta}{=} \Phi_{l^j} \delta^{(x_l)}$, where x_l is the value of node l under loop cutset configuration $\mathbf{x}_L$.

Loop cutset node $l \in L$ passes to neighbor $j \in \mathcal{L}_l$ the message

$$m_{l \to j}^{(\mathbf{x}_L)} = \Psi_{lj} \Phi_{l^j}^{(x_l)},$$

while to a neighbor $j \in \mathcal{N}_l$, it passes the message

$$m_{l \to j}^{(\mathbf{x}_L)} = \Psi_{lj} \Phi_{l^{\mathcal{N}_l}}^{(x_l)} \prod_{k \in \mathcal{N}_l \setminus j} m_{k \to l}^{(\mathbf{x}_L)} .$$

A loop cutset node $l \in L$ can compute conditioned beliefs from a neighbor $j \in \mathcal{L}_l$ as

$$Z_l^{(\mathbf{x}_L)} = \Phi_{l^j}^{(x_l)} m_{j \to l}^{(\mathbf{x}_L)} ,$$

or from its non-leaf neighbors as

$$Z_l^{(\mathbf{x}_L)} = \Phi_{l^{\mathcal{N}_l}}^{(x_l)} \prod_{j \in \mathcal{N}_l} m_{j \to l}^{(\mathbf{x}_L)} .$$

For a non loop cutset node $i \notin L$, the belief and outgoing messages conditioned on loop cutset configuration $\mathbf{x}_L$ are computed as

$$Z_i^{(\mathbf{x}_L)} = \Phi_j \prod_{j \in \partial i} m_{j \to i}^{(\mathbf{x}_L)}$$

and

$$m_{i\to j}^{(\mathbf{x}_L)} = \Psi_{ij}\Phi_i \prod_{k\in\partial i\setminus j} m_{k\to i}^{(\mathbf{x}_L)} \ .$$

6.2 Parallel Implementation and Ordering

Conditioning can be implemented both in serial and in parallel. However, the computational savings from Local Conditioning require parallel implementation. In this case, messages $\left[m_{i\to j}^{(\mathbf{x}_L)}\right]$ and beliefs $\left[Z_i^{(\mathbf{x}_L)}\right]$ corresponding to different loop cutset configurations are concatenated as columns in message and belief matrices $M_{i\to j}^{(L)}$ and $Z_i^{(L)}$, respectively. In the parallel implementation of Conditioning, there are $|\mathcal{X}|^{|L|}$ columns in all message and belief matrices.

Parallel implementation requires that messages and beliefs are ordered so that corresponding columns of incoming messages themselves correspond to the same loop cutset configuration. We adopt the convention that column indices of message and belief matrices are ordered with respect to an ordering (L) of L, given by their $|L|$-digit $|\mathcal{X}|$-ary representations in which the first node of (L) corresponds to the most significant digit, and so on. If the ordering (L) is agreed upon in advance, the ordering of nodes in L does not need to be communicated to neighboring nodes.

As with standard BP, for large networks messages in Conditioning will need to be scaled to avoid numerical underflow or overflow. If the columns of a message matrix $M_{i\to j}^{(L)}$ are scaled independently of one another, then an additional layer of non-distributed processing is required to compute beliefs for nodes in the network. Therefore, in order for beliefs to be computed in a truly distributed manner, all columns of a message matrix must be scaled with the same factor. We do not address this problem in the present paper.

7 Local Conditioning for Undirected Networks

Local Conditioning is an adaption of Conditioning that achieves exponential savings in complexity by conditioning message and belief matrices only on *local* subsets of the loop cutset nodes. Diez [4] introduced the main ideas of Local Conditioning for directed networks through examples, and Fay and Jaffray [5] subsequently proved that indeed exact beliefs can be computed with message matrices of reduced size. However, the demonstration in [5] was theoretical, showing that messages and beliefs *could be* computed only conditioning on local subsets of loop cutset nodes, and did not address the details of *how* nodes would actually compute beliefs and outgoing messages from incoming messages of different sizes and orderings. This section provides, in the context of undirected networks, both a more formal description of Local Conditioning than in [4], and a more practical account than that given in [5]. In particular, it discusses the *summing*

out of *upstream* loop cutset nodes when passing a message matrix over an edge, so that the message contains only columns corresponding to loop cutset nodes that are *relevant* for the edge; and the *expansion* of incoming message matrices to account for loop cutset nodes that are *downstream* of the incoming messages yet *relevant* for the receiving node. Furthermore, it discusses the *ordering* of message matrices with respect to the given relevant sets, and the *re*-ordering of incoming message matrices so that they are aligned prior to fusion.

Let T be the associated tree dictating the conditioned message passing. Message passing in Local Conditioning still obeys the topology of T and as such a loop cutset node l computes outgoing messages to neighbors, and beliefs from incoming messages, depending on whether l is connected to its neighbors as a leaf or non-leaf in T. To simplify the ensuing discussion, message and belief computations will be presented agnostically, without reference to whether a node is in the loop cutset or not. However, it should be clear from the discussion in Section 6.1 that if one wishes to specifically consider a loop cutset node, either the set $\mathcal{N}_l$, or $\{j\}$ for some $j \in \mathcal{L}_l$, can be substituted for ∂i. Moreover, care will of course need to be applied to self-potentials of loop cutset nodes, both in dividing them up between $\mathcal{L}_l$ and $\mathcal{N}_l$, and in modifying them to ensure consistency with the respective loop cutset configurations.

7.1 Relevant Nodes and Reduced Complexity

Recall from Section 4 that a node k is *upstream* of the message $m_{j\to i}$ if $k \in T_{j\setminus i}$. Likewise, k is said to be *downstream* of $m_{j\to i}$ if $k \in T_{i\setminus j}$. Note that if k is upstream of $m_{j\to i}$, then k is downstream of $m_{i\to j}$. Let $L_{j\setminus i} \subset L$ denote the subset of loop cutset nodes *all of whose copies* are contained in $T_{j\setminus i}$, or in other words, the set of loop cutset nodes upstream of $m_{j\to i}$ and downstream of $m_{i\to j}$. Note that $L_{j\setminus i}$ and $L_{i\setminus j}$ are disjoint. For an edge $\{i, j\}$, the *relevant set*

$$R_{ij} \stackrel{\Delta}{=} L \setminus \left(L_{i\setminus j} \cup L_{j\setminus i}\right) \tag{6}$$

is the set of loop cutset nodes at least one copy of which is upstream and at least one copy of which is downstream of the messages $m_{j\to i}$ and $m_{i\to j}$ passed over the edge. Clearly, R_{ij} and R_{ji} are the same. For node i, the relevant set is

$$R_i = \bigcup_{j\in\partial i} R_{ij} \ . \tag{7}$$

A message matrix $M_{j\to i}^{(R_{ji})}$ passed over edge $\{i, j\}$ will only be conditioned on the relevant set R_{ij}, and as such will have $|\mathcal{X}|^{|R_{ij}|}$ columns, one for each configuration $\mathbf{x}_{R_{ij}}$ on the relevant set for the edge. When node i receives message matrices $M_{j\to i}^{(R_{ji})}$ from its neighbors, each will be conditioned on a potentially different subset of the relevant set R_i for node i. Node i will then *expand* each of the incoming $M_{j\to i}^{(R_{ji})}$ to a message matrix $M_{j\to i}^{(R_i)}$ conditioned on node i's relevant set. After expanding each of the incoming message matrices, node i will then

reorder the expanded message matrices so that a given column of the incoming messages correspond to the same configuration $\mathbf{x}_{R_i}$. The belief matrix $Z_i^{(R_i)}$ is then computed by multiplying element-wise the reordered incoming messages and the self-potential for i as

$$Z_i^{(\mathbf{x}_{R_i})} = \Phi_i \prod_{j \in \partial i} m_{j \to i}^{(\mathbf{x}_{R_{ji}}, \mathbf{x}_{R_i \setminus R_{ji}})} ,$$

and then computing unconditioned beliefs as

$$Z_i = \sum_{x_{R_i}} Z_i^{(\mathbf{x}_{R_i})} .$$

Node i will also form the outgoing message column

$$m_{i \to j}^{(\mathbf{x}_{R_{ij}})} = \Psi_{ji} \Phi_i \sum_{\mathbf{x}_{R_i \setminus R_{ij}}} \prod_{k \in \partial i \setminus j} m_{k \to i}^{(\mathbf{x}_{R_{ki}}, \mathbf{x}_{R_i \setminus R_{ki}})} ,$$

to neighbor j by multiplying element-wise expanded and reordered message matrices from its other neighbors $k \in \partial i \setminus j$, and then *summing out* those loop cutset nodes in R_i but not in R_{ij}.

7.2 The Details

Details of *summing out*, *expanding*, and *reordering* for Local Conditioning are discussed. For a given node i in T, let $j_1, j_2, \ldots, j_{n_i}$ indicate the neighbors of i. L can be partitioned as $(R_i, L_{j_1 \setminus i}, \ldots, L_{j_{n_i} \setminus i})$ In Conditioning, the message $M_{j_m \to i}^{(L)}$ from j_m to i is a matrix with $|\mathcal{X}|^{|L|}$ columns, one for each configuration of the loop cutset L. By definition, L can be partitioned as $(L_{i \setminus j_m}, R_{i,j_m}, L_{j_m \setminus i})$.

In Local Conditioning, the message matrix $M_{j_m \to i}^{(L_{i \setminus j_m}, R_{i,j_m}, L_{j_m \setminus i})}$ need not be conditioned on $\mathbf{x}_{L_{i \setminus j_m}}$. For any two configurations $(\mathbf{x}_{L_{i \setminus j_m}}, \mathbf{x}_{R_{j_m,i}}, \mathbf{x}_{L_{j_m \setminus i}})$ and $(\mathbf{x}'_{L_{i \setminus j_m}}, \mathbf{x}_{R_{j_m,i}}, \mathbf{x}_{L_{j_m \setminus i}})$ that differ only on $L_{i \setminus j_m}$, the corresponding message columns $m_{j_m \to i}^{(\mathbf{x}_{L_{i \setminus j_m}}, \mathbf{x}_{R_{j_m,i}}, \mathbf{x}_{L_{j_m \setminus i}})}$ and $m_{j_m \to i}^{(\mathbf{x}'_{L_{i \setminus j_m}}, \mathbf{x}_{R_{j_m,i}}, \mathbf{x}_{L_{j_m \setminus i}})}$ are identical. This is because all potentials involving nodes in $L_{i \setminus j_m}$ are downstream of the message from j_m to i. We can eliminate this redundancy, and node j_m can pass to node i a message matrix $M_{j_m \to i}^{(R_{j_m,i}, R_{L_{j_m \setminus i}})}$ with only $|\mathcal{X}|^{|R_{j_m i}| + |R_{L_{j_m \setminus i}}|}$ columns.

Considering the message matrix $M^{(R_{j_m,i}, R_{L_{j_m \setminus i}})}$, node j_m can sum the columns

$$\{m_{j_m \to i}^{(\mathbf{x}_{R_{i,j_m}}, \mathbf{x}'_{L_{j_m \setminus i}})} : \mathbf{x}'_{L_{j_m \setminus i}} \in \mathcal{X}_{L_{j_m \setminus i}}\}$$

of message matrix $M_{j_m \to i}^{(R_{i,j_m}, L_{j_m \setminus i})}$ that agree on R_{i,j_m} and differ on $L_{j_m \setminus i}$. That is, since all potentials involving nodes in $L_{j_m \setminus i}$ are upstream of the message from j_m to i, it is not necessary to retain separate columns for different configurations

on the these loop cutset nodes. This results in a message matrix $M_{j_m \to i}^{(R_{i,j_m})}$ with $|\mathcal{X}|^{|R_{i,j_m}|}$ columns, a substantial reduction from the original number of $|\mathcal{X}|^{|L|}$.

As in Section 6.2, columns of the message $M_{j_m \to i}^{(R_{j_m,i})}$ are ordered with respect to an ordering $(R_{j_m,i})$ of the relevant set $R_{j_m,i}$ for edge $\{i, j_m\}$. Each incoming $M_{j_m \to i}^{(R_{j_m,i})}$ is expanded to a message $M_{j_m \to i}^{(R_i)}$ by iteratively replicating $M_{j_m \to i}^{(R_{j_m,i})}$ for each node in $R_i \setminus R_{i,j_m}$. Let $(R_i)_{j_m}$ denote the ordering of R_i with respect to which the columns of expanded $M_{j_m \to i}^{(R_i)}$ are ordered, and (R_i) the common ordering of R_i with which all incoming message matrices need to be aligned. For each neighbor $j_m \in \partial i$, node i computes $P_{(R_i)_{j_m}}$, the permutation matrix that converts $(R_i)_{j_m}$ to (R_i). Using $P_{(R_i)_{j_m}}$, indices of the expanded $M_{j_m \to i}^{(R_i)_{j_m}}$ are mapped to the reordered $M_{j_m \to i}^{(R_i)}$ in the following way. A column $m_{j_m \to i}^{(\mathbf{x}_{R_i})_{j_m}}$ of expanded $M_{j_m \to i}^{(R_i)_{j_m}}$ has an index $c = (c_1 \cdots c_{|R_i|})$, where $c_k \in \{0, \ldots, |\mathcal{X}|\}$. Applying $P_{(R_i)_{j_m}}$ to c gives the index within reordered $M_{j_m \to i}^{(R_i)}$ for the conditioned message $m_{j_m \to i}^{(\mathbf{x}_{R_i})}$.

We stated above that node j_m sums columns of message matrix $M_{j_m \to i}^{(R_{j_m,i}, L_{j_m \setminus i})}$ corresponding to different configurations on $L_{j_m \setminus i}$, resulting in a message matrix $M_{j_m \to i}^{(R_{j_m,i})}$ of reduced size. In reality, the summing out of loop cutset nodes in $L_{j_m \setminus i}$ will have been performed recursively, neighbors $k \in \partial j_m \setminus i$ summing out some of the nodes in $L_{j_m \setminus i}$, and so on back to the leaves of $T_{j_m \setminus i}$.

For a given neighbor $k \in \partial j_m \setminus i$, the set $R_{j_m,i} \cup L_{j_m \setminus i}$ can equivalently be partitioned as $(L_{j_m \setminus k}^{j_m \setminus i}, R_{k,j_m}, L_{k \setminus j_m})$, where $L_{j_m \setminus k}^{j_m \setminus i} \triangleq L_{j_m \setminus i} \cap L_{j_m \setminus k}$ is the set of loop cutset nodes all of whose copies are upstream of the message from j_m to i and downstream of the message from k to j_m. For example, consider *another* neighbor $k' \in \partial j_m \setminus \{i, k\}$. The set $L_{k' \setminus j_m}$ consists of loop cutset nodes all of whose copies are in both $T_{j_m \setminus i}$ and $T_{j_m \setminus k}$. Using similar arguments as before, the message $M_{k \to j_m}^{(\mathbf{x}_{L_{j_m \setminus k}^{j_m \setminus i}}, \mathbf{x}_{R_{k,j_m}}, \mathbf{x}_{L_{k \setminus j_m}})}$ need not be conditioned on $L_{j_m \setminus k}^{j_m \setminus i}$, and the loop cutset nodes in $L_{k \setminus j_m}$ can be summed out *before* node k sends its message to j_m. As a result, node j_m receives from each $k \in \partial j_m \setminus i$ a message $M_{k \to j_m}^{(R_{k,j_m})}$. The remaining nodes in $L_{j_m \setminus i}$ to be summed out before node j_m passes its message to i are those loop cutset nodes that are relevant for at least one $k \in \partial j_m \setminus i$ and upstream of the message from j_m to i, which we denote by $L^{j_m \setminus i} \triangleq \left(\cup_{k \in \partial j_m \setminus i} R_{k,j_m}\right) \setminus R_{i,j_m}$. That is, the incoming message matrices $M_{k \to j_m}^{(R_{k,j_m})}$, $k \in \partial j_m \setminus i$, are each expanded to $\cup_{k \in \partial j_m \setminus i} R_{k,j_m}$, reordered so that corresponding columns align, then multiplied element-wise along with the self-potential Φ_{j_m} for j_m. Finally, conditioning on $L^{j_m \setminus i}$ is removed by summing out, resulting in the outgoing message matrix $M_{j_m \to i}^{(R_{i,j_m})}$.

We now show the algebraic details of computing beliefs and messages.

Computing Beliefs The belief Z_i of node i can be expressed as

$$\begin{aligned}
Z_i &= \sum_{\mathbf{x}_L} Z_i^{(\mathbf{x}_L)} \\
&= \sum_{\mathbf{x}_L} \Phi_i \prod_{j \in \partial i} m_{j \to i}^{(\mathbf{x}_L)} \\
&= \sum_{\mathbf{x}_{R_i}} \sum_{\mathbf{x}_{L_{j_1 \setminus i}}} \cdots \sum_{\mathbf{x}_{L_{j_{n_i} \setminus i}}} \Phi_i \prod_{m=1}^{n_i} m_{j_m \to i}^{(\mathbf{x}_L)} \\
&= \sum_{\mathbf{x}_{R_i}} \Phi_i \sum_{\mathbf{x}_{L_{j_1 \setminus i}}} \cdots \sum_{\mathbf{x}_{L_{j_{n_i} \setminus i}}} m_{j_1 \to i}^{(\mathbf{x}_L)} \cdots m_{j_{n_i} \to i}^{(\mathbf{x}_L)} \\
&= \sum_{\mathbf{x}_{R_i}} \Phi_i \sum_{\mathbf{x}_{L_{j_1 \setminus i}}} \cdots \sum_{\mathbf{x}_{L_{j_{n_i} \setminus i}}} m_{j_1 \to i}^{(\mathbf{x}_{L_{i \setminus j_1}}, \mathbf{x}_{R_{i,j_1}}, \mathbf{x}_{L_{j_1 \setminus i}})} \cdots m_{j_{n_i} \to i}^{(\mathbf{x}_{L_{i \setminus j_{n_i}}}, \mathbf{x}_{R_{i,j_{n_i}}}, \mathbf{x}_{L_{j_{n_i} \setminus i}})} .
\end{aligned}$$

Recall from preceding discussion, that the message $m_{j_1 \to i}^{(\mathbf{x}_L)}$ is not a function of $\mathbf{x}_{L_{j_2 \setminus i}}, \ldots, \mathbf{x}_{L_{j_{n_i} \setminus i}}$. Therefore, each message $m_{j_m \to i}^{(\mathbf{x}_{L_{i \setminus j_m}}, \mathbf{x}_{R_{i,j_m}}, \mathbf{x}_{L_{j_m \setminus i}})}$ becomes $m_{j_m \to i}^{(\mathbf{x}_{R_{i,j_m}}, \mathbf{x}_{L_{j_m \setminus i}})}$, the summations $\sum_{L_{j_1 \setminus i}} \cdots \sum_{L_{j_{n_i} \setminus i}}$ distribute over the multiplications $m_{j_1 \to i}^{(\mathbf{x}_{R_{i,j_1}}, \mathbf{x}_{L_{j_1 \setminus i}})} \cdots m_{j_{n_i} \to i}^{(\mathbf{x}_{R_{i,j_{n_i}}}, \mathbf{x}_{L_{j_{n_i} \setminus i}})}$, and we continue as

$$\begin{aligned}
Z_i &= \sum_{\mathbf{x}_{R_i}} \Phi_i \sum_{\mathbf{x}_{L_{j_1 \setminus i}}} m_{j_1 \to i}^{(\mathbf{x}_{R_{i,j_1}}, \mathbf{x}_{L_{j_1 \setminus i}})} \cdots \sum_{\mathbf{x}_{L_{j_{n_i} \setminus i}}} m_{j_{n_i} \to i}^{(\mathbf{x}_{R_{i,j_{n_i}}}, \mathbf{x}_{L_{j_{n_i} \setminus i}})} \\
&= \sum_{\mathbf{x}_{R_i}} \Phi_i \prod_{m=1}^{n_i} \sum_{\mathbf{x}_{L_{j_m \setminus i}}} m_{j_m \to i}^{(\mathbf{x}_{R_{j_m,i}}, \mathbf{x}_{L_{j_m \setminus i}})} \\
&= \sum_{\mathbf{x}_{R_i}} \Phi_i \prod_{m=1}^{n_i} m_{j_m \to i}^{(\mathbf{x}_{R_{j_m,i}})} .
\end{aligned} \tag{8}$$

Summing out loop cutset nodes in the $L_{j_m \setminus i}$ results in incoming message matrices $M_{j_m \to i}^{(R_{j_m,i})}$ with $|\mathcal{X}|^{|R_{j_m,i}|}$ columns, respectively.

After receiving incoming messages $M_{j \to i}^{(R_{ji})}$ from its neighbors $j \in \partial i$, node i *expands* each of them to matrices $M_{j \to i}^{(R_i)_j}$ conditioned on its relevant set R_i by duplicating the columns of $M_{j \to i}^{(R_{ji})}$ repeatedly for each node in $R_i \setminus R_{ij}$. The expanded message matrices are then *reordered* so that they can be fused through element-wise multiplication. Node i then computes its belief as

$$Z_i = \sum_{\mathbf{x}_{R_i}} \Phi_i \prod_{m=1}^{n_i} m_{j_m \to i}^{(\mathbf{x}_{R_i})} , \tag{9}$$

which corresponds to the operation of node i summing the $|\mathcal{X}|^{|R_i|}$ columns of its belief matrix $Z_i^{(R_i)}$, the columns of which correspond to different configurations

$\mathbf{x}_{R_i}$, formed by multiplying element-wise its self-potential Φ_i and the incoming message matrices $M_{j\to i}^{(R_i)}$ from its neighbors.

Computing Outgoing Messages In the transition from (8) to (9) above, we see that node j passes to node i the message

$$\begin{aligned}
m_{j\to i}^{(\mathbf{x}_{R_{ji}})} &= \sum_{\mathbf{x}_{L_{j\setminus i}}} m_{j\to i}^{(\mathbf{x}_{R_{ji}}\ ,\ \mathbf{x}_{L_{j\setminus i}})} \\
&= \sum_{\mathbf{x}_{L_{j\setminus i}}} \Psi_{j,i}\Phi_j \prod_{k\in\partial j\setminus i} m_{k\to j}^{(\mathbf{x}_{R_{ji}}\ ,\ \mathbf{x}_{L_{j\setminus i}})} \\
&= \Psi_{j,i}\Phi_j \sum_{\mathbf{x}_{L^{j\setminus i}}} \sum_{\mathbf{x}_{L_{k_1\setminus j}}} \cdots \sum_{\mathbf{x}_{L_{k_{n_j}\setminus j}}} m_{k_1\to j}^{(\mathbf{x}_{R_{ji}}\ ,\ \mathbf{x}_{L_{j\setminus i}})} \cdots m_{k_{n_j}\to j}^{(\mathbf{x}_{R_{ji}}\ ,\ \mathbf{x}_{L_{j\setminus i}})} \\
&= \Psi_{j,i}\Phi_j \sum_{\mathbf{x}_{L^{j\setminus i}}} \prod_{k_m\in\partial j\setminus i} \sum_{\mathbf{x}_{L_{k_m\setminus j}}} m_{k_m\to j}^{(\mathbf{x}_{R_{ji}}\ ,\ \mathbf{x}_{L_{j\setminus i}})} \\
&= \Psi_{j,i}\Phi_j \sum_{\mathbf{x}_{L^{j\setminus i}}} \prod_{k_m\in\partial j\setminus i} \sum_{\mathbf{x}_{L_{k_m\setminus j}}} m_{k_m\to j}^{(\mathbf{x}_{L_{j\setminus k_m}^{j\setminus i}}\ ,\ \mathbf{x}_{R_{k_m,j}}\ ,\ \mathbf{x}_{L_{k_m\setminus j}})}
\end{aligned}$$

where we have used the fact, mentioned above, that for a neighbor $k \in \partial j \setminus i$, $R_{ji} \cup L_{j\setminus i}$ can be partitioned as $(L_{j\setminus k}^{j\setminus i}, R_{kj}, L_{k\setminus j})$. As before, note that the set $L_{j\setminus k}^{j\setminus i}$ is downstream of the message from k to j and as such for any two configurations $\mathbf{x}_{L_{j\setminus k}^{j\setminus i}}$ and $\mathbf{x}'_{L_{j\setminus k}^{j\setminus i}}$, the messages $m_{k\to j}^{(\mathbf{x}_{L_{j\setminus k}^{j\setminus i}}\ ,\ \mathbf{x}_{R_{k,j}}\ ,\ \mathbf{x}_{L_{k\setminus j}})}$ and $m_{k\to j}^{(\mathbf{x}'_{L_{j\setminus k}^{j\setminus i}}\ ,\ \mathbf{x}_{R_{k,j}}\ ,\ \mathbf{x}_{L_{k\setminus j}})}$ are identical. Therefore the message from k to j does not need to be conditioned on $L_{j\setminus k}^{j\setminus i}$, leaving us with

$$\begin{aligned}
m_{j\to i}^{(\mathbf{x}_{R_{ji}})} &= \Psi_{j,i}\Phi_j \sum_{\mathbf{x}_{L^{j\setminus i}}} \prod_{k_m\in\partial j\setminus i} \sum_{\mathbf{x}_{L_{k_m\setminus j}}} m_{k_m\to j}^{(\mathbf{x}_{R_{k_m,j}}\ ,\ \mathbf{x}_{L_{k_m\setminus j}})} \\
&= \Psi_{j,i}\Phi_j \sum_{\mathbf{x}_{L^{j\setminus i}}} \prod_{k_m\in\partial j\setminus i} m_{k_m\to j}^{(\mathbf{x}_{R_{k_m,j}})}.
\end{aligned}$$

Node j *expands* each of the incoming message matrices $M_{k_m\to j}^{(R_{k_m,j})}$ to message matrix $M_{k_m\to j}^{(\cup_{k\in\partial j\setminus i} R_{kj})_{k_m}}$, *reorders* the expanded message matrices to facilitate element-wise multiplication, then sums out the loop cutset nodes in $L^{j\setminus i} = R_j \setminus R_{ij}$. This yields the outgoing message matrix $M_{j\to i}^{(R_{ji})}$ and establishes recursive computation of Local Conditioning messages.

We summarize belief and message computation in the following theorem.

Theorem 1 (Local Conditioning Sum-Product BP).
For a non loop cutset node $j \notin L$, local conditioned beliefs and messages are computed as

$$Z_j^{(\mathbf{x}_{R_j})} = \Phi_j \prod_{k \in \partial j} m_{k \to j}^{(\mathbf{x}_{R_{kj}})}$$

and

$$m_{j \to i}^{(\mathbf{x}_{R_{ji}})} = \Psi_{ji} \Phi_j \sum_{\mathbf{x}_{L^{j \setminus i}}} \prod_{k \in \partial j \setminus i} m_{k \to j}^{(\mathbf{x}_{R_{jk}})},$$

where $L^{j \setminus i} \triangleq \left(\cup_{k \in \partial j \setminus i} R_{kj} \right) \setminus R_{ij}$ is the set of loop cutset nodes all of whose copies are upstream of the message from j to i but are not in any of the $L_{k \setminus j}$ for $k \in \partial j \setminus i$. Unconditioned beliefs are computed as

$$Z_j = \sum_{\mathbf{x}_{R_j}} Z_j^{(\mathbf{x}_{R_j})}.$$

For a loop cutset node $l \in L$, conditioned beliefs are computed as

$$Z_l^{(\mathbf{x}_{R_l})} = \Phi_{lj}^{(x_l)} m_{j \to l}^{(\mathbf{x}_{R_{jl}})},$$

for some $j \in \mathcal{L}_l$, or as

$$Z_l^{(\mathbf{x}_{R_l})} = \Phi_{l^{\mathcal{N}_l}}^{(x_l)} \prod_{j \in \mathcal{N}_l} m_{j \to l}^{(\mathbf{x}_{R_{jl}})}.$$

The message to a neighbor $i \in \mathcal{N}_l$ is computed as

$$m_{l \to i}^{(\mathbf{x}_{R_{li}})} = \Psi_{li} \Phi_{l^{\mathcal{N}_l}}^{(x_l)} \sum_{\mathbf{x}_{L^{l \setminus i}}} \prod_{k \in \mathcal{N}_l \setminus i} m_{k \to l}^{(\mathbf{x}_{R_{kl}})},$$

where $L^{l \setminus i} \triangleq \left(\cup_{k \in \partial l \setminus i} R_{kl} \right) \setminus R_{il}$ is the set of loop cutset nodes all of whose copies are upstream of the message from l to i but are not in any of the $L_{k \setminus l}$ for $k \in \partial l \setminus i$. Unconditioned beliefs are computed as

$$Z_l = \sum_{\mathbf{x}_{R_l}} Z_l^{(\mathbf{x}_{R_l})}.$$

7.3 Complexity of Local Conditioning

The complexity of Local Conditioning depends upon the sizes of the relevant sets for nodes and edges with respect to the particular associated tree used for the node-specific message passing of LC. For a given loop cutset L, different

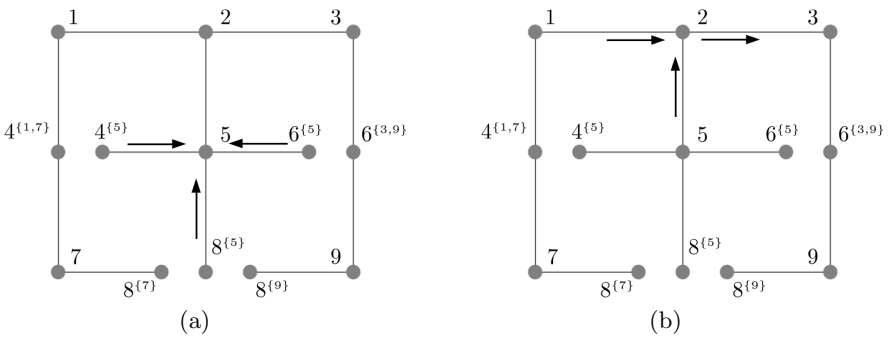

Fig. 3. (a) Messages $M^{(4)}_{4\to 5}$, $M^{(8)}_{8\to 5}$, and $M^{(6)}_{6\to 5}$ incoming to node 5 from neighbors 4, 8, and 6, respectively; (b) outgoing message $M^{(6,8)}_{2\to 3}$ computed from $M^{(4,8)}_{1\to 2}$ and $M^{(4,6,8)}_{5\to 2}$.

associated trees will result in different computational complexities. We have not addressed the question of finding an optimal associated tree for a given loop cutset, nor the further question of finding an optimal loop cutset. Considerable work has been done in optimizing loop cutsets in the context of directed networks [2], and our hope is that some of this can be leveraged for the undirected case.

For a given associated tree, we have shown that the message passed over edge $\{i, j\}$ is a matrix with $|\mathcal{X}|^{|R_{ij}|}$ columns. Each column has $|\mathcal{X}|$ elements. Moreover, the belief at node i is a matrix with $|\mathcal{X}|^{|R_i|}$ columns, again each column with $|\mathcal{X}|$ elements. Since $R_{ij} \subset R_i$ for all $j \in \partial i$, the complexity is dominated by a term that is exponential in $\max_{i\in V} |R_i|$. In our preliminary analysis so far, we have found that for an $M \times N$ grid network, there is a loop cutset and an associated tree for that loop cutset with $\max_i |R_i| = M + 1$. In other words, the complexity of Local Conditioning on an $M \times N$ grid network is the same as that of Clustering.

7.4 An Example: Expanding, Reordering, and Summing Out

Figure 3 illustrates Local Conditioning messages passed with respect to the associated tree from Figure 2. In (a), node 5 receives incoming message matrices

$$M^{(4)}_{4\to 5} = \left[m^{(-1)}_{4\to 5}, m^{(1)}_{4\to 5} \right], \quad M^{(6)}_{6\to 5} = \left[m^{(-1)}_{6\to 5}, m^{(1)}_{6\to 5} \right], \quad M^{(8)}_{8\to 5} = \left[m^{(-1)}_{8\to 5}, m^{(1)}_{8\to 5} \right]$$

from neighbors 4, 6, and 8, respectively. Each of these is expanded to a message matrix conditioned on $\{4, 6, 8\}$ in the following way:

$$M^{(6,8,4)}_{4\to 5} = \left[m^{(-1)}_{4\to 5}, m^{(1)}_{4\to 5}, m^{(-1)}_{4\to 5}, m^{(1)}_{4\to 5}, m^{(-1)}_{4\to 5}, m^{(1)}_{4\to 5}, m^{(-1)}_{4\to 5}, m^{(1)}_{4\to 5} \right],$$

$$M^{(4,8,6)}_{6\to 5} = \left[m^{(-1)}_{6\to 5}, m^{(1)}_{6\to 5}, m^{(-1)}_{6\to 5}, m^{(1)}_{6\to 5}, m^{(-1)}_{6\to 5}, m^{(1)}_{6\to 5}, m^{(-1)}_{6\to 5}, m^{(1)}_{6\to 5} \right],$$

and

$$M_{8\to5}^{(4,6,8)} = \left[m_{8\to5}^{(-1)} , m_{8\to5}^{(1)} , m_{8\to5}^{(-1)} , m_{8\to5}^{(1)} , m_{8\to5}^{(-1)} , m_{8\to5}^{(1)} , m_{8\to5}^{(-1)} , m_{8\to5}^{(1)} \right].$$

These expanded message matrices will be ordered with respect to the ordering $(4, 6, 8)$ of R_5. The expanded messages $M_{4\to5}^{(6,8,4)}$ and $M_{6\to5}^{(4,8,6)}$ from nodes 4 and 6 are reordered using the permutation matrices

$$P_{(6,8,4)} = \begin{bmatrix} 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{bmatrix} \qquad , \qquad P_{(4,8,6)} = \begin{bmatrix} 1\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \end{bmatrix} ,$$

that respectively permute $(6, 8, 4)$ and $(4, 8, 6)$ into $(4, 6, 8)$. That is, columns of $M_{4\to5}^{(6,8,4)}$ are indexed as $000, 001, 010, 011, 100, 101, 110$ and 111. Multiplying each of these indices by $P_{(6,8,4)}$ shows that the columns of $M_{4\to5}^{(6,8,4)}$ are respectively reindexed as in

$$M_{4\to5}^{(4,6,8)} = \left[m_{4\to5}^{(-1)} , m_{4\to5}^{(1)} , m_{4\to5}^{(-1)} , m_{4\to5}^{(1)} , m_{4\to5}^{(-1)} , m_{4\to5}^{(1)} , m_{4\to5}^{(-1)} , m_{4\to5}^{(1)} \right] .$$

Likewise for $M_{6\to5}^{(4,8,6)}$ and $P_{(4,8,6)}$.

In (b), node 2 forms preliminary outgoing message

$$M_{2\to3}^{(4,6,8)} = \left[\bar{m}_{4\to5}^{(0)} , \bar{m}_{4\to5}^{(1)} , \bar{m}_{4\to5}^{(2)} , \bar{m}_{4\to5}^{(3)} , \bar{m}_{4\to5}^{(4)} , \bar{m}_{4\to5}^{(5)} , \bar{m}_{4\to5}^{(6)} , \bar{m}_{4\to5}^{(7)} \right] ,$$

where the configuration on $\{4, 6, 8\}$ is indicated by enumerating the combinations of $\{-1, 1\}^3$, the binary representation of a configuration index having 4 has the most significant digit. Since we want to sum out node 4, node 2 will form the outgoing message matrix $M_{2\to3}^{(6,8)}$ in the following way:

$$M_{2\to3}^{(6,8)} = M_{2\to3}^{(4,6,8)} \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix} .$$

8 Concluding Remarks and Future Directions

We have addressed issues related to the practical implementation of Local Conditioning. As more and more systems are fielded comprised of distributed entities, such as sensor networks or roadways of autonomous vehicles, an understanding of Local Conditioning will become increasingly important. Further work in this area must address suboptimal implementations for networks of prohibitive topology, for example by prematurely summing out relevant loop cutset nodes.

Acknowledgements

The author would like to thank David Neuhoff for comments on an earlier draft.

References

1. S.M Aji and R.J. McCliese, "The Generalized Distributive Law", *IEEE Trans. Info. Thy.*, vol. 46, no. 2, 2000.
2. A. Becker and D. Geiger, "Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem," *Artificial Intelligence*, vol. 83, pp. 167-188, 1996.
3. T.M. Cover and J.A. Thomas, "Elements of Information Theory," Wiley, 1991.
4. F.J. Diez, "Local Conditioning in in Bayesian Networks", *Artificial Intelligence*, vol. 87, pp. 1-20, 1996.
5. A. Fay and J.-Y. Jaffray, "A justification of local conditioning in Bayesian networks," *International Journal of Approximate Reasoning*, vol. 24, 2000.
6. B.J. Frey and D.J.C. MacKay, "A Revolution: Belief Propagation in Graphs With Cycles," *Advances in Neural Information Processing Systems 10*, MIT Press, Denver, 1997.
7. G.R. Grimmett, "A theorem on random fields," *Bulletin of London Mathematical Society*, vol. 5, pp. 81-84, 1973.
8. S.L. Lauritzen and D.J. Spiegelhalter, "Local Computations with Probabilities on Graphical Structure and their Application to Expert Systems", *Jrnl. Royal Stat. Soc. B*, vol. 50, no. 2, 1988.
9. K. Murphy, Y. Weiss, and M. Jordan, "Loopy Belief Propagation for Approximate Inference: An Empirical Study", *UAI*, 1999.
10. J. Pearl, "A Constraint Propagation Approach To Probabilistic Reasoning", *Uncertainty in Artificial Intelligence*, Elsevier, New York, pps. 357-369, 1986.
11. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, June 2014.
12. M.G. Reyes and D.L. Neuhoff, "Local Conditioning for Undirected Graphs," *Information Theory and Applications* workshop, San Diego, February 2017.
13. H.J. Suermondt and G.F. Cooper, "Probabilistic Inference in Multiply Connected Belief Networks Using Loop Cutsets", *Intl. Jrnl. of Approximate Reasoning*, vol. 4, 1990, pps. 283-306.
14. N.N. Taleb, "The Black Swan: The Impact of the Highly Improbably", Random House, 2007.
15. M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky, "Tree-Based Reparametrization Framework for Analysis of Sum-Product and Related Algorithms," *IEEE Trans. on Info. Thy.*, vol. 49, no. 5, May 2003.
16. J.S. Yedidia, W.T. Freeman, and Y. Weiss, "Constructing Free-Energy Approximations and Generlized Belief Propagation Algorithms", *IEEE Trans. Info. Thy.*, vol. 51, no. 7, July 2005.