Finding Non-overlapping Clusters for Generalized Inference Over Graphical Models

Divyanshu Vats and José M. F. Moura

Abstract

Graphical models use graphs to compactly capture stochastic dependencies amongst a collection of random variables. Inference over graphical models corresponds to finding marginal probability distributions given joint probability distributions. In general, this is computationally intractable, which has led to a quest for finding efficient approximate inference algorithms. We propose a framework for generalized inference over graphical models that can be used as a wrapper for improving the estimates of approximate inference algorithms. Instead of applying an inference algorithm to the original graph, we apply the inference algorithm to a block-graph, defined as a graph in which the nodes are non-overlapping clusters of nodes from the original graph. This results in marginal estimates of a cluster of nodes, which we further marginalize to get the marginal estimates of each node. Our proposed block-graph construction algorithm is simple, efficient, and motivated by the observation that approximate inference is more accurate on graphs with longer cycles. We present extensive numerical simulations that illustrate our block-graph framework with a variety of inference algorithms (e.g., those in the libDAI software package). These simulations show the improvements provided by our framework.

Index Terms

Graphical Models, Markov Random Fields, Belief Propagation, Generalized Belief Propagation, Approximate Inference, Block-Trees, Block-Graphs.

Divyanshu Vats is with the Institute for Mathematics and its Applications, University of Minnesota - Twin Cities, Minneapolis, MN, 55455, USA (email: dvats@ima.umn.edu)

José M. F. Moura is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, USA (email: moura@ece.cmu.edu, ph: (412)-268-6341, fax: (412)-268-3980).

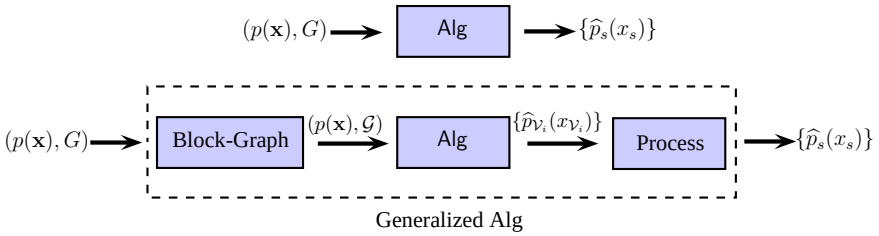


Fig. 1. Alg is an inference algorithm that estimates marginal distributions given a graphical model. We propose a framework that generalizes Alg using block-graphs to improve the accuracy of the marginal estimates.

I. Introduction

A graphical model is a probability distribution defined on a graph such that each node represents a random variable (or multiple random variables), and edges in the graph represent conditional independencies[1]. The underlying graph structure in a graphical model leads to a factorization of the joint probability distribution. Graphical models are used in many applications such as sensor networks, image processing, computer vision, bioinformatics, speech processing, social network analysis, and ecology [1]–[3], to name a few. Inference over graphical models corresponds to finding the marginal distribution $p_s(x_s)$ for each random variable given the joint probability distribution $p(\mathbf{x})$. It is well known that inference over graphical models is computationally tractable for only a small class of graphical models (graphs with low treewidth [4]), which has led to much work to derive efficient approximate inference algorithms.

A. Summary of Contributions

Our main contribution in this paper is a framework that can be used as a wrapper for improving the accuracy of approximate inference algorithms. Instead of applying an inference algorithm to the original graph, we apply the inference algorithm to a block-graph, defined as a graph in which the nodes are non-overlapping clusters of nodes from the original graph. This results in marginal estimates of a cluster of nodes of the original graph, which we further marginalize to get the marginal estimates of each node. Larger clusters, in general, lead to more accurate inference algorithms at the cost of increased computational complexity. Fig. 1 illustrates our proposed block-graph framework for *generalized inference*.

[1]In a graphical model over a graph G, for each edge $(i, j) \notin G$, X_i is conditionally independent of X_j given $X_{V\setminus\{i,j\}}$, where V indexes all the nodes in the graph. We can also say that for each $(i, j) \in G$, X_i is dependent on X_j given X_S, for all S such that $S \subseteq V\setminus\{i, j\}$.

The key component in our framework is to construct a block-graph. It has been empirically observed that approximate inference is more accurate on graphs with longer cycles [5]. This motivates our proposed block-graph construction algorithm where we first find non-overlapping clusters such that the graph over the clusters is a tree. We refer to the resulting block-graph as a block-tree. The block-tree construction algorithm runs in linear time by using two passes of breadth-first search over the graph. The second step in our block-graph construction algorithm is to split large clusters[2] in the block-tree. Using numerical simulations, we show how our proposed algorithm for splitting large clusters leads to superior inference algorithms when compared to an algorithm that randomly splits large clusters in the block-tree and an algorithm that uses graph partitioning to find non-overlapping clusters [6].

As an example, consider applying our block-graph framework to belief propagation (BP) [7], which finds the marginal distribution at each node by iteratively passing messages between nodes. If the graph is a tree, BP computes the exact marginal distributions, however, for general graphs with cycles, BP only approximates the true marginal distribution. Our framework for inference (see Fig. 1) generalizes BP so that message passing occurs between clusters of nodes, where the clusters are non-overlapping. The estimates of the marginal distribution at each node can be computed by marginalizing the approximate marginal distribution of each cluster. Our framework is not limited to BP and can be used as a wrapper for *any* inference algorithm on graphical models as we show in Section V. Using numerical simulations, we show how our block-graph framework improves the marginal distribution estimates computed by current inference algorithms in the literature: BP [7], conditioned belief propagation (CBP) [8], loop corrected belief propagation (LC) [9], [10], tree-structured expectation propagation (TreeEP), iterative join-graph propagation (IJGP) [11], and generalized belief propagation (GBP) [12]–[15].

B. Related Work

There has been significant work in extending the BP algorithm of message passing between nodes to message passing between clusters. It is known that the true marginal distributions of a graphical model minimize the Gibbs free energy [16]. In [12], [17], the authors show that the fixed points of the BP algorithm minimize the Bethe free energy, which is an approximation to the Gibbs free energy. This motivated the generalized belief propagation (GBP) algorithm that minimizes the Kikuchi free energy [18], a better approximation to the Gibbs free energy. In GBP, message passing is between clusters of nodes that are overlapping. A more general approach to GBP is proposed in [14] using region graphs

[2]For discrete graphical models, the complexity of inference is exponential in the maximum cluster size, which is why using the block-tree directly for inference may not be computationally tractable if the maximum cluster size is large.

and in [15] using the cluster variation method (CVM). References [19], [20] propose some guidelines for choosing clusters in GBP. Reference [11] proposes a framework for GBP, called iterative join-graph propagation (IJGP), that first constructs a junction tree, a tree-structured representation of a graph with overlapping clusters, and then splits larger clusters in the junction tree to perform message passing over a set of overlapping clusters.

In the original paper describing GBP [17], the authors give an example of how non-overlapping clusters can be used for GBP, since, when applying the block-graph framework to BP, the resulting inference algorithm (Bm-BP, where m is the cluster size) becomes a class of GBP algorithms where the set of overlapping clusters corresponds to cliques in the block-graph. Our numerical simulations identify cases where Bm-BP leads to superior marginal estimates when compared to a GBP algorithm that uses overlapping clusters. Moreover, since our framework can be applied to any inference algorithm, we show that the marginal estimates computed by GBP based algorithms can be improved by applying the GBP based algorithm to a block-graph. Our block-graph framework is not limited to generalizing BP and we show this in our numerical simulations where we generalize conditioned belief propagation (CBP) and loop corrected belief propagation (LC). Both these algorithms have been shown to empirically perform better than GBP for certain graphical models [8], [10]. In [6], [21], the authors propose using graph partitioning algorithms for finding non-overlapping clusters for generalizing the mean field algorithm for inference [16]. Our numerical results show that our algorithm for finding non-overlapping clusters leads to superior marginal estimates.

We note that our work differs from some other works on studying graphical models defined over graphs with non-overlapping clusters. For example, [22] consider the problem of learning a Gaussian graphical model defined over some block-graph. Similar efforts have been made in [23] for discrete valued graphical models. In [24], the author analyzes properties of a graphical model defined on a block-tree. In all of the above works, the underlying graphical model is *assumed* to be block-structured. In our work, we assume a graphical model defined on an *arbitrary* graph and then find a representation of the graphical model on a block-graph to enable more accurate inference algorithms.

This paper is motivated by our earlier work in studying tree structures for Markov random fields (MRFs) indexed over continuous indices [25]. In [26], we have shown that a natural tree-like representation for such MRFs exists over non-overlapping *hypersurfaces* within the continuous index set. Using this representation, we derived extensions of the Kalman-Bucy filter [27] and the Rauch-Tung-Striebel smoother [28] to Gaussian MRFs indexed over continuous indices.

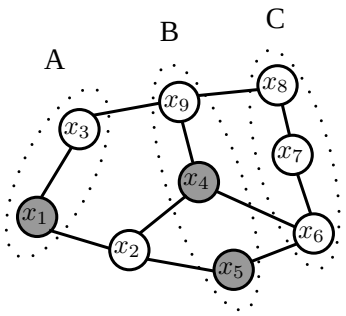


Fig. 2. An example of a graphical model. The global Markov property states that x_A is independent of x_C given x_B since all paths from A to C pass through B.

C. Paper Organization

Section II reviews graphical models and the inference problem. Section III outlines our proposed algorithm for constructing block-trees, a tree-structured graph over non-overlapping clusters. Section IV presents our algorithm for splitting larger clusters in a block-tree to construct a block-graph. Section V outlines our block-graph framework for generalizing inference algorithms. Section VI presents extensive numerical results evaluating our framework on various inference algorithms. Section VII summarizes the paper and outlines some future research directions.

II. Background: Graphical Models and Inference

A graphical model is defined using a graph $G = (V, E)$, where the nodes $V = \{1, 2, \ldots, p\}$ index a collection of random variables $\mathbf{x} = \{x_s \in \Omega^d : s \in V\}$ and the edges $E \subseteq V \times V$ encode statistical independencies [3], [29]. The set of edges can be directed, undirected, or both. Since directed graphical models, also known as Bayesian networks, can be mapped to undirected graphical models by moralizing the graph, in this paper, we only consider undirected graphical models, also known as Markov random fields or Markov networks.

The edges in a graphical model imply Markov properties about the collection of random variables. The *local Markov property* states that x_s is independent of $\{x_r : r \in V \backslash \{\mathcal{N}(s) \cup s\}\}$ given $x_{\mathcal{N}(s)}$, where $\mathcal{N}(s)$ is the set of neighbors of s. For example, in Fig. 2, x_2 is independent of $\{x_3, x_6, x_7, x_8, x_9\}$ given $\{x_1, x_4, x_5\}$. The *global Markov property*, which is equivalent to the local Markov property for non-degenerate probability distributions, states that, for a collection of disjoint nodes A, B, and C, if B separates A and C, x_A is independent of x_C given x_B. An example of the sets A, B, and C is shown in Fig. 2. From the *Hammersley-Clifford* theorem [30], the Markov property leads to a factorization of

the joint probability distribution over cliques (fully connected subsets of nodes) in the graph,

$$p(x_1, x_2, \ldots, x_p) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)\,, \tag{1}$$

where $\mathcal{C}$ is the set of all cliques in the graph $G = (V, E)$, $\psi_C(x_C) > 0$ are potential functions defined over cliques, and Z is the partition function, a normalization constant.

Inference in graphical models corresponds to finding marginal distributions, say $p_s(x_s)$, given the probability distribution $p(\mathbf{x})$ for $\mathbf{x} = \{x_1, \ldots, x_p\}$. This problem is of extreme importance in many domains. A classical example is in estimation when we are given noisy observations $\mathbf{y}$ of $\mathbf{x}$ and we want to estimate the underlying random vector. To find the minimum mean square error (mmse) estimate, we need to marginalize the conditional probability distribution $p(\mathbf{x}|\mathbf{y})$ to find the marginals $p_s(x_s|\mathbf{y})$. An algorithm for marginalizing $p(\mathbf{x})$ can be used for marginalizing $p(\mathbf{x}|\mathbf{y})$. In general, exact inference is computationally intractable, however, there has been significant progress in deriving efficient approximate inference algorithms. The main contribution in this paper is the block-graph framework for generalizing inference algorithms (see Fig. 1) so that the performance of approximate inference algorithms can be improved.

III. Block-Trees: Finding Trees Over Non-overlapping Clusters

Section III-A outlines our algorithm for constructing block-trees. Section III-B defines the notion of optimal block-trees by using connections between block-trees and junction trees. Section III-C outlines greedy algorithms for finding optimal block-trees.

A. Main Algorithm

Definition 1 (Block-Graph and Block-Tree): For a graph $G = (V, E)$, a *block-graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph over clusters of nodes in V such that each node in V is associated with only one cluster in $\mathcal{V}$. In other words, the clusters in $\mathcal{V}$ are non-overlapping. If the edge set $\mathcal{E} \subseteq V \times V$ is tree-structured, we call the block-graph a *block-tree*.

Algorithm 1 outlines our construction of a block-tree $\mathcal{G}$ given an arbitrary graph $G = (V, E)$. Without loss in generality, we assume that G is connected, i.e., there exists a path between all non adjacent nodes. The original graph G and a set of nodes $V_1 \subset V$ are the input. The output is the block-tree $\mathcal{G}$. We refer to V_1 as the *root cluster*. The algorithm first finds an initial set of clusters and then splits these clusters to find the final block-tree. We explain the steps of the algorithm.

Forward step: Find clusters $V_1, V_2, \ldots, V_r$ using breadth-first search (BFS) so that $V_2 = \mathcal{N}(V_1), V_3 = \mathcal{N}(V_2)\backslash\{V_1 \cup V_2\}, \ldots, V_r = \mathcal{N}(V_r)\backslash\{V_{r-2} \cup V_{r-1}\}$. These clusters serve as initial clusters for the block-tree. During the BFS step, split each cluster V_k into its connected components $\{V_k^1, \ldots, V_k^{m_k}\}$ using the subgraph $G(V_k)$, which denotes the graph only over the nodes in V_k (Line 2).

Backwards step: We now merge the clusters in each V_k to find the final block-tree. The key intuition in this step is that each cluster V_k should be connected to a single cluster in V_{k-1}. If this is not the case, we merge clusters in V_{k-1} accordingly. Starting at $V_r = \{V_r^1, V_r^2, \ldots, V_r^{m_r}\}$, for each $V_r^j, j = 1, \ldots, m_r$, find all clusters $C(V_r^j)$ in V_{r-1} that are connected to V_r^j (Line 6). Combine all clusters in $C(V_r^j)$ into a single cluster and update the clusters in V_{r-1}^j accordingly. Repeat the above steps for all the clusters in $V_{r-1}, V_{r-2}, \ldots, V_3$.

Algorithm 1: Constructing Block-Trees: BlockTree(G,V_1)

Data: A graph $G = (V, E)$ and a set of nodes V_1.
Result: A block-tree $\mathcal{G} = (\mathcal{C}, \mathcal{E})$
1 Find successive neighbors of V_1 to construct a sequence of r clusters $V_1, V_2, \ldots, V_r$ such that $V_2 = \mathcal{N}(V_1)$, $V_3 = \mathcal{N}(V_2)\backslash\{V_1 \cup V_2\}, \ldots, V_r = \mathcal{N}(V_r)\backslash\{V_{r-2} \cup V_{r-1}\}$.
2 $\{V_k^1, \ldots, V_k^{m_k}\} \leftarrow$ Find m_k connected components of V_k using subgraph $G(V_k)$.
3 **for** $i = r, r-1, \ldots 3$ **do**
4 **for** $j = 1, 2, \ldots m_i$ **do**
5 $C(V_i^j) \leftarrow \mathcal{N}(V_i^j) \cap V_{i-1}$; All nodes in V_{i-1} connected to V_i^j.
6 Combine $C(V_i^j)$ into one cluster and update V_{i-1}.
7 $\mathcal{V} \leftarrow \bigcup_{k=1}^{r} \{V_k^1, V_k^2, \ldots, V_k^{m_k}\}$
8 $\mathcal{E} \leftarrow$ edges between all the clusters in $\mathcal{V}$

The first part of Algorithm 1 finds successive non-overlapping neighbors of the root cluster. This leads to an initial estimate of the block-tree graph. In the backwards step, we split clusters to form a block-tree. We illustrate Algorithm 1 with examples.

Example: Consider the grid graph of Fig. 3(a). Choosing $V_1 = \{1\}$, we get the initial estimates of the clusters as shown in Fig. 3(a). Running the backwards step to identify the final clusters (see Fig. 3(b)), we get the block-tree in Fig. 3(c).

Example: In the previous example, the initial estimates of the clusters matched the final estimates and the final block-tree was a chain-structured graph. We now consider an example where the final block-tree will in fact be tree-structured. Consider the partial grid graph of Fig. 4(a). Choosing $V_1 = \{7\}$, we get the initial estimates of the clusters in Fig. 4(a). We now run the backwards step of the algorithm. Since $V_5 = \{3\}$ is connected to 2 and 6, $C(V_5) = \{2, 6\}$. Thus, $\{2, 6\}$ become a single cluster. We now find

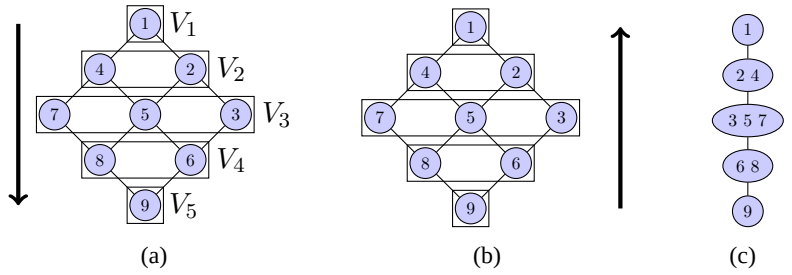


Fig. 3. (a) Original estimates of the clusters in a grid graph when running the forward pass of Algorithm 1. (b) The final clusters after running the backwards pass of Algorithm 1. (c) Final block-tree.

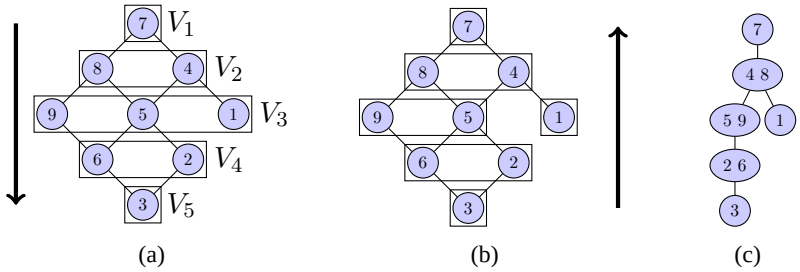


Fig. 4. (a) Original estimates of the clusters in a partial grid when running the forward pass of Algorithm 1. (b) The final clusters after running the backwards pass of Algorithm 1. (c) Final block-tree.

neighbors of $\{2, 6\}$ in $V_3 = \{9, 5, 1\}$. It is clear that only $\{9, 5\}$ are connected to $\{2, 6\}$, so $\{9, 5\}$ become a single cluster. In this way, we have split V_3 into two clusters: $V_3^1 = \{9, 5\}$ and $V_3^2 = \{1\}$. Continuing the algorithm, we find the remaining clusters as shown in Fig. 4(b). The final block-tree is shown in Fig. 4(c).

The following proposition characterizes the time complexity and correctness of Algorithm 1.

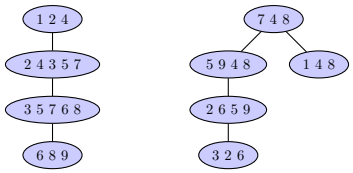
Proposition 1: Algorithm 1 runs in time $O(|E|)$ and always outputs a block-tree.

Proof: Both the forward step and the backwards step involve a breadth first search, which has complexity $O(|E|)$. Algorithm 1 always outputs a block-tree since each cluster in V_k is only connected to a single cluster in V_{k-1}. ■

Block-trees are closely related to junction trees. In the next Section, we explore this connection to define optimal block-trees.

B. Optimal Block-Trees

Junction trees, also known as clique trees or join trees, are tree-structured representations of graphs using a set of overlapping clusters [31], [32]. The *width* of a graph is the maximum cardinality of a



(a) (b)

Fig. 5. (a) Junction tree for the block-tree in Fig. 3(c) (b) Junction tree for the block-tree in Fig. 4(c)

cluster in the junction tree minus one. The *treewidth* of a graph is the minimum width of a graph over all possible junction tree representations. It is well known that several graph related problems that are computationally intractable in general can be solved efficiently when the graph has low treewidth. For the problem of inference over graphical models, [4] showed how junction trees can be used for exact inference over graphical models defined over graphs with small treewidth.

Given a block-tree $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an equivalent junction tree representation can be easily computed by combining all clusters connected along edges into a single cluster. For example, the junction tree representation for the block-trees in Fig. 3(c) and Fig. 4(c) are given in Fig. 5(a) and Fig. 5(b), respectively. Using this junction tree, we can derive exact inference algorithms for graphical models parameterized by block-trees.

It is easy to see that the complexity of inference using block-trees will depend on the maximum sum of cluster sizes of adjacent clusters in the block-tree (since this will correspond to the width of the equivalent junction tree). Thus, an *optimal block-tree* can be defined as a block-tree $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for which $\max_{(i,j)\in\mathcal{E}} (|V_i| + |V_j|)$ is minimized. From Algorithm 1, the construction of the block-tree depends on the choice of the root cluster V_1. Thus, finding an *optimal block-tree* is equivalent to finding an optimal root cluster. This problem is computationally intractable since we need to search over all possible combinations of root clusters V_1.

As an example illustrating how the choice of V_1 alters the block-tree, consider finding a block-tree for the partial grid in Fig. 4(a). In Fig. 4(c), we constructed a block-tree using $V_1 = \{7\}$ as the root cluster. The maximum sum of adjacent cluster sizes in Fig. 4(c) is four. Instead of choosing $V_1 = \{7\}$, let $V_1 = \{7, 4\}$. The initial estimate of the clusters are shown in Fig. 6(a). The final block-tree is shown in Fig. 6(c). Since the clusters $\{9, 6, 2\}$ and $\{8, 5\}$ are adjacent, the maximum sum of adjacent clusters is five.

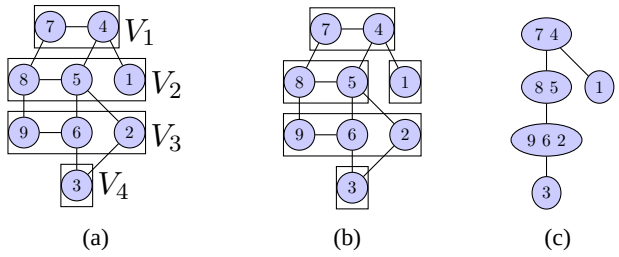


Fig. 6. (a) Original estimates of the clusters in the partial grid using $V_1 = \{7, 4\}$ as the root cluster. (b) Splitting of clusters. (c) Final block-tree.

C. Greedy Algorithms for Finding Optimal Block-Trees

In the previous Section, we saw that finding optimal block-trees is computationally intractable. In this Section, we propose three greedy algorithms for finding optimal block-trees that have varying degrees of computational complexity.

Minimal degree node - MinDegree**:** In this approach, which we call MinDegree, we find the node with minimal degree and use that node as the root cluster. The intuition behind this is that the minimal degree node may lead to the smallest number of nodes being added in the clusters. The complexity of this approach is $O(n)$, where n is the number of nodes in the graph.

The next two algorithms are based on the relationship between junction trees and block-trees outlined in Section III-B. Recall that for every block-tree, we can find a junction tree. This means that an optimal junction tree (a junction tree with minimal width) may be used to find an approximate optimal block-tree. Further, finding optimal junction trees corresponds to finding optimal elimination orders in graphs [33]. Thus, we can make use of greedy algorithms for finding optimal elimination orders to find optimal block-trees.

Using an elimination order - GreedyDegree**:** One of the simplest algorithms for finding an approximate optimal elimination order is known as GreedyDegree [34], [35], where the elimination order corresponds to the sorted list of nodes in increasing degree. The complexity of GreedyDegree is $O(n \log n)$ since we just need to sort the nodes. Using the elimination order, we triangulate[3] the graph to find the cliques. These cliques correspond to the set of clusters in the junction tree representation. We search over a constant number of cliques to find an optimal root cluster.

Using an elimination order - GreedyFillin**:** Another popular greedy algorithm is to find an optimal

[3]A graph is triangulated if each cycle of length four or more has an edge connecting non adjacent nodes.

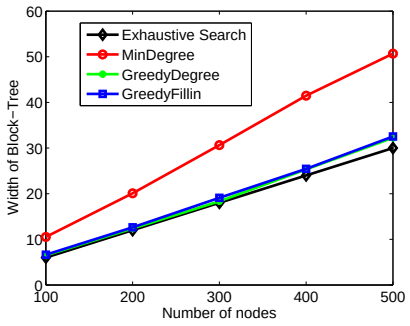


Fig. 7. Plot showing the performance of three different greedy heuristics for finding optimal block-trees.

elimination order such that at each step in the triangulation algorithm, we choose a node that adds a minimal number of extra edges in the graph. This is known as GreedyFillin [36] and has polynomial complexity. Thus, GreedyFillin is in general slower than GreedyDegree, but does lead to slightly better elimination orders on average. To find the block-tree, we again search over a constant number of cliques over the triangulated graph.

We now evaluate the three different greedy algorithms, MinDegree, GreedyDegree, and GreedyFillin, for finding optimal block-trees in Fig. 7. To do this, we create clusters of size k such that the total number of nodes is n (one cluster may have less than k nodes). We then form a tree over the clusters and associate a clique between two clusters connected to each other. We then remove a certain fraction of edges over the graph (not the block-tree), but make sure that the graph is still connected. By construction, the width of the graph constructed is at most $2k$. Fig. 7 shows the performance of MinDegree, GreedyDegree, and GreedyFillin over graphs with different number of nodes and different values of k. We clearly see that both GreedyDegree and GreedyFillin compute widths that are close to optimal. The main idea in this Section is that we can use various known algorithms for finding optimal junction trees to find optimal block-trees.

D. Exact Inference Using Block-Trees

In the literature, exact inference over graphical models using non-overlapping clusters is referred to as the Pearl's clustering algorithm [7]. In [37] and [38], the authors use non-overlapping clustering for some particular directed graphical models for an application in medical diagnostics. For lattices, [39]–[41] derive inference algorithms by scanning the lattice horizontally (or vertically). Our block-tree construction algorithm provides a principled way of finding non-overlapping clusters over arbitrary graphs.

Inference over graphical models defined on block-trees can be done by extending the belief propagation (BP) algorithm [7]. The computational complexity of BP will depend on $\max_{(i,j)\in\mathcal{E}}(|V_i|+|V_j|)$. On the other hand, the computational complexity of exact inference using other frameworks that use overlapping clusters depends on the width of the graph [4], [42]–[44], which is in general less than or equal to $\max_{(i,j)\in\mathcal{E}}(|V_i|+|V_j|)$. The main advantage of using block-trees for exact inference is that the complexity of constructing block-trees is $O(|E|)$, whereas the complexity of constructing tree-decompositions for inference using frameworks that use overlapping clusters is worse than $O(|E|)$ [33], [44], [45]. Thus, block-trees are suitable for exact inference over time-varying graphical models [46], [47] such that the clusters in the block-tree are small.

IV. Block-Graph: Splitting Clusters in a Block-Tree

In this Section, we outline a greedy algorithm for splitting large clusters in a block-tree to form a block-graph. This is an important step in our proposed framework (see Fig. 1) for generalizing inference algorithms since we apply the inference algorithm to the block-graph as opposed to the original graph. Note that we can use the block-tree itself for inference; however, for many graphs this is computationally intractable since the complexity of inference for discrete graphical models using a block-tree is exponential in $\max_{(i,j)\in\mathcal{E}}(|V_i|+|V_j|)$. Thus, when the size of one cluster in the block-tree is large, exact inference using block-trees will be computationally intractable[4].

We modify Algorithm 1 for constructing block-trees to construct block-graphs such that all clusters have cardinality at most m.

Step 1. Using an initial cluster of nodes V_1, find clusters $V_1, V_2, \ldots, V_r$ using breadth-first search (BFS) such that $V_2 = \mathcal{N}(V_1), V_3 = \mathcal{N}(V_2)\backslash\{V_1 \cup V_2\}, \ldots, V_r = \mathcal{N}(V_r)\backslash\{V_{r-2} \cup V_{r-1}\}$. While doing the BFS, write V_k as the set of all connected components in the subgraph $G(V_k)$. Thus, V_k is a set of clusters.

Step 2. For V_r, if there exists any cluster that has cardinality greater than m, partition those components. Let $V_r = \{V_r^1, V_r^2, \ldots, V_r^{m_r}\}$ be the final set of clusters.

Step 3. Perform the next steps for each $k = r-1, r-2, \ldots, 1$, starting at $k = r-1$. Let $\widetilde{V_k}$ be the set of all clusters V_k that have cardinality greater than m. Partition all clusters in $\widetilde{V_k}$ into appropriate size clusters of size at most m.

[4]Our algorithm for finding optimal block-trees uses the junction tree construction algorithm, so even if $\max_{(i,j)\in\mathcal{E}}(|V_i|+|V_j|)$ is large and the treewidth of the graph is small, we can detect this and use junction trees for inference.

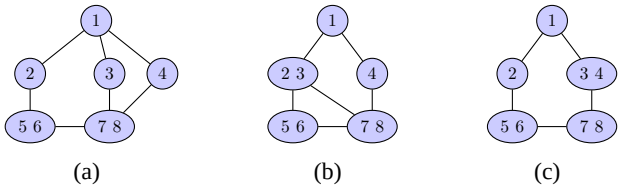


Fig. 8. Explaining Step 4 in the block-graph construction algorithm. Given the block-graph in (a), if we merge nodes 2 and 3, we get the block-graph in (b). If we merge nodes 3 and 4, we get the block-graph in (c). The block-graph in (c) has just one loop.

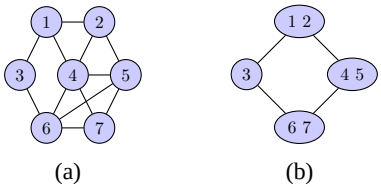


Fig. 9. (a). Original graph. (b) Block-graph representation of the graph in (a).

Step 4. Merge the clusters in the set $V_k \backslash \widetilde{V}_k$. The idea used in merging clusters is that if two clusters are connected to the same cluster in V_{k+1}, by merging these two clusters, we reduce one edge in the final block-graph. Further, if two clusters in V_k are not connected to the same cluster in V_{k+1}, we do not merge these two clusters, since the number of edges in the final block-graph will remain the same. The final clusters constructed using the above rules is denoted as $V_k = \{V_k^1, \ldots, V_k^{m_k}\}$.

Step 5. The block-graph is given by the clusters $\mathcal{V} = \{V_k^1, V_k^2, \ldots, V_k^{m_k}\}_{k=1,\ldots,r}$ and the set of edges $\mathcal{E}$ between clusters.

The key step in the above algorithm is Step 4, where we cluster nodes appropriately. Fig. 8 explains the intuition behind merging clusters with an example. Suppose, we use the block-graph construction algorithm up to Step 3 and now we want to merge clusters in $V_2 = \{2, 3, 4\}$. If we ignore Step 4 and merge clusters randomly, we might get the block-graph in Fig. 8(b) on merging nodes 2 and 3. If we use Step 4, since nodes 3 and 4 are connected to the same node, we merge these to get the block-graph in Fig. 8(c). The graph in Fig. 8(c) has a single cycle with five edges, whereas the graph in Fig. 8(b) has two cycles of size four and three. It has been observed that inference over graphs with longer cycles is more accurate than inference over graphs with shorter cycles [5]. Thus, our proposed algorithm leads to block-graphs that are favorable for inference.

V. Inference Using Block-Graphs

Define a graphical model on a graph $G = (V, E)$ using a collection of p random variables $\mathbf{x} = (x_1, \ldots, x_p)$, where each x_k takes values in Ω^d, where $d \geq 1$. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a block-graph representation of the graph $G = (V, E)$. To derive inference algorithms over the block-graph, we need to define appropriate potentials (or factors) associated with each clique in the block-graph. This can be done by mapping the potentials from the original graph to the block-graph. As an example, let G be the graph in Fig. 9(a) and let the probability distribution over G be given by

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{1,4}(x_1, x_4)\psi_{1,3}(x_1, x_3)\psi_{2,4,5}(x_2, x_4, x_5)\psi_{4,5,6,7}(x_4, x_5, x_6, x_7)\psi_{3,6}(x_3, x_6)\,. \quad (2)$$

Let the clusters in the block-graph representation of G in Fig. 9(b) be $V_1 = \{1, 2\}$, $V_2 = \{4, 5\}$, $V_3 = \{3\}$, and $V_4 = \{6, 7\}$. The probability distribution in (2) can be written in terms of the block-graph as follows:

$$p(\mathbf{x}) = \frac{1}{Z}\Psi_{1,2}(x_{V_1}, x_{V_2})\Psi_{1,3}(x_{V_1}, x_{V_3})\Psi_{2,4}(x_{V_2}, x_{V_4})\Psi_{2,4}(x_{V_2}, x_{V_4})\psi_{3,4}(x_{V_3}, x_{V_4})\,, \quad (3)$$

where

$$\Psi_{1,2}(x_{V_1}, x_{V_2}) = \psi_{1,2}(x_1, x_2)\psi_{1,4}(x_1, x_4)\psi_{2,4,5}(x_2, x_4, x_5) \quad (4)$$

$$\Psi_{1,3}(x_{V_1}, x_{V_3}) = \psi_{1,3}(x_1, x_3) \quad (5)$$

$$\Psi_{2,4}(x_{V_2}, x_{V_4}) = \psi_{4,5,6,7}(x_4, x_5, x_6, x_7) \quad (6)$$

$$\Psi_{2,4}(x_{V_3}, x_{V_4}) = \psi_{3,6}(x_3, x_6)\,. \quad (7)$$

Let Alg be an algorithm for inference over graphical models. Inference over the graph G can be performed using Alg with inputs being the potentials in (2). Inference over the block-graph can be performed using Alg with input being the potentials in (4)-(7). To get the marginal distributions from the block-graph, we need to further marginalize the joint probability distribution over each cluster.

Remark 1: Both the representations (2) and (3) are equivalent, so we are not making any approximations when parameterizing the graphical model using block-graphs.

Remark 2: There is a trade-off in choosing the size of the clusters in the block-graph. Generally, as observed in our numerical simulations, larger clusters lead to better estimates at the cost of more computations.

Remark 3: We presented the block-graph framework using undirected graphical models. The results can be easily generalized to settings where the probability distribution is represented as a factor graph

[48].

VI. Numerical Simulations

In this Section, we provide numerical simulations to show how our proposed block-graph framework for generalizing inference algorithms can be used to improve the performance of current approximate inference algorithms that have been proposed in the literature. Throughout this Section, we assume $x_s \in \{-1, +1\}$ and the probability distribution over $\mathbf{x}$ factorizes as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{p} \phi_i(x_i) \prod_{(i,j) \in E} \psi_{ij}(x_i, x_j)\,. \tag{8}$$

The node potentials are given by $\phi_i(x_i) = \exp(-a_i x_i)$, where $a_i \sim \mathcal{N}(0, 0.1)$ and the edge potentials are given by

$$\text{Repulsive (REP): } \psi_{ij}(x_i, x_j) = \exp(-|b_{ij}| x_i x_j) \tag{9}$$

$$\text{Attractive (ATT): } \psi_{ij}(x_i, x_j) = \exp(|b_{ij}| x_i x_j) \tag{10}$$

$$\text{Mixed (MIX): } \psi_{ij}(x_i, x_j) = \exp(-b_{ij} x_i x_j)\,, \tag{11}$$

where $b_{ij} \sim \mathcal{N}(0, \sigma)$ and σ is the interaction strength. For distributions with attractive (repulsive) potentials, neighboring random variables are more likely to take the same (opposite) value. For distributions with mixed potentials, some neighbors are attractive, whereas some are repulsive. We study several approximate inference algorithms that have been proposed in the literature: Belief Propagation (BP) [7], Iterative Join-Graph Propagation (IJGP-i) [11], Generalized Belief Propagation (GBP-i) [13], [14], Conditioned Belief Propagation (CBP-l) [8], Loop Corrected Belief Propagation (LC) [10], Tree-Structured Expectation Propagation (TreeEP) [49]. In IJGP-i and GBP-i, the integer i refers to the maximum size of the clusters, where the clusters in these algorithms are overlapping. The clusters in GBP-i are selected by finding cycles of length i in the graph. In CBP-l, l is an integer that refers to the number of clamped variables when performing inference: larger l in general leads to more accurate marginal estimates. We use the libDAI software package [50] for all the inference algorithms except for IJGP, where we use the software provided by the authors at [51]. For an inference algorithm Alg, we refer to the generalized inference algorithm as Bm-Alg, where the m is an integer denoting the maximum size of the cluster in the block-graph.

We consider two types of graphs: (i) grid graphs and (ii) random regular graphs, where each node in the graph has the same degree and the edges are chosen randomly. Both these graphs have been

used extensively in the literature for evaluating inference algorithms [8], [10]. We compare inference algorithms using the mean absolute error:

$$\text{Error} = \frac{1}{p}\sum_{s=1}^{p}\sum_{x_s\in\{-1,+1\}} \left|\widehat{p}_s(x_s) - p_s(x_s)\right|, \tag{12}$$

where $\widehat{p}_s$ is the marginal estimate computed by an approximate inference algorithm and $p_s(x_s)$ is the true marginal distribution. To evaluate the computational complexity, we measure the time taken in running the inference algorithms on a 2.66GHz Intel(R) Xeon(R) X5355 processor with 32 GB memory. Since all the approximate inference algorithms we considered are iterative algorithms, we set the maximum number of iterations to be 1000 and stopped the inference algorithm when the mean absolute difference between the new and old marginal estimates is less than 10^{-9}. All the code and graphical models used in the numerical simulations can be downloaded from http://www.ima.umn.edu/~dvats/GeneralizedInference.html.

A. Evaluating the Block-Graph Construction Algorithm

We first evaluate our proposed algorithm for constructing block-graphs (see Section IV) where we split large clusters in a block-tree. Fig. 10 shows the results of applying our block-graph framework to generalize BP, CBP, LC, and TreeEP on a 5×5 grid graph. We compare our algorithm to an algorithm that randomly splits the clusters in a block-tree and an algorithm proposed in [6] that uses graph partitioning to find non-overlapping clusters. In Fig. 10, the solid lines correspond to our algorithm (see legend B2-BP), the dashed lines correspond to random splitting of clusters (see legend RandB2-BP), and the dotted lines correspond to graph partitioning (see legend GP-B2-BP). The results reported are averages over 100 trials.

Remark 4: It is clear that the graph partitioning approach performs the worst amongst the three different algorithms (the dotted line is above the solid and dashed line). For TreeEP, we observe that the graph partitioning approach performs worse than the original algorithm that does not use block-graphs. This suggests that the graph partitioning algorithm in [6] is not suitable for the inference algorithms considered in Fig. 10. We did not apply the graph partitioning algorithm to LC since the corresponding inference algorithm was very slow.

Remark 5: In most cases, our proposed algorithm for constructing block-graphs performs better than using an algorithm that randomly splits clusters (the solid line is below the dashed line). Interestingly, for TreeEP, the random algorithm performs worse than the original algorithm. We also observed that both the random algorithm and the graph partitioning algorithm took more time than our proposed algorithm. This

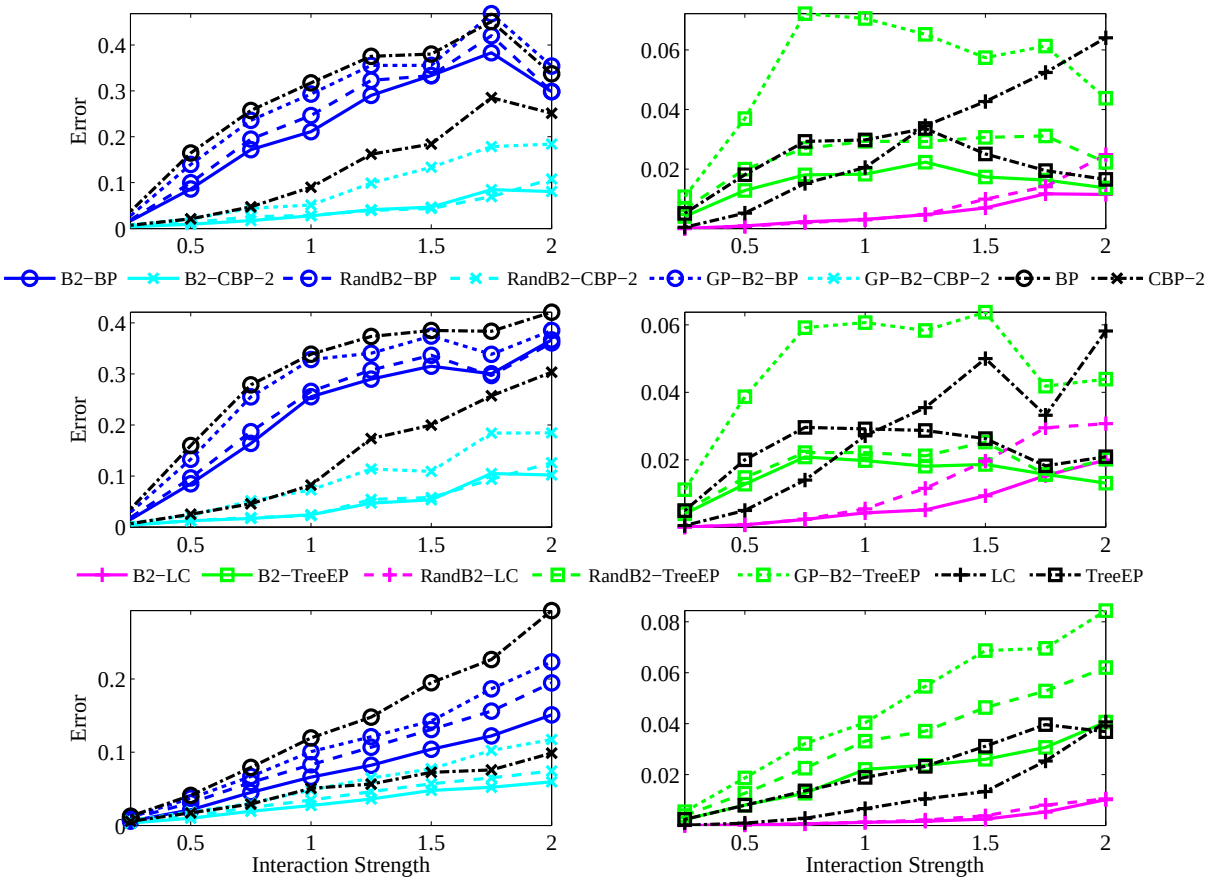


Fig. 10. Evaluating the block-graph construction algorithm in Section IV on a 5×5 grid graph. The solid lines, denoted by Bm-Alg for an inference algorithm Alg, correspond to using our proposed block-graph construction algorithm. The dashed lines correspond to an inference algorithm that randomly splits larger clusters in a block-tree. The dotted lines correspond to an inference algorithm that uses graph partitioning to find clusters. The plots in the top, middle, and bottom row correspond to repulsive, attractive, and mixed potentials, respectively.

suggests that our proposed block-graph construction algorithm leads to block-graphs that are favorable for inference.

B. Grid Graphs

Tables I, II, III, IV, and V show results of applying the block-graph framework for inference over graphical models defined on grid graphs.

Remark 6: In general, we observe that for all cases considered, applying the block-graph framework leads to better marginal estimates. This is shown in the Tables, where for each inference algorithm, we

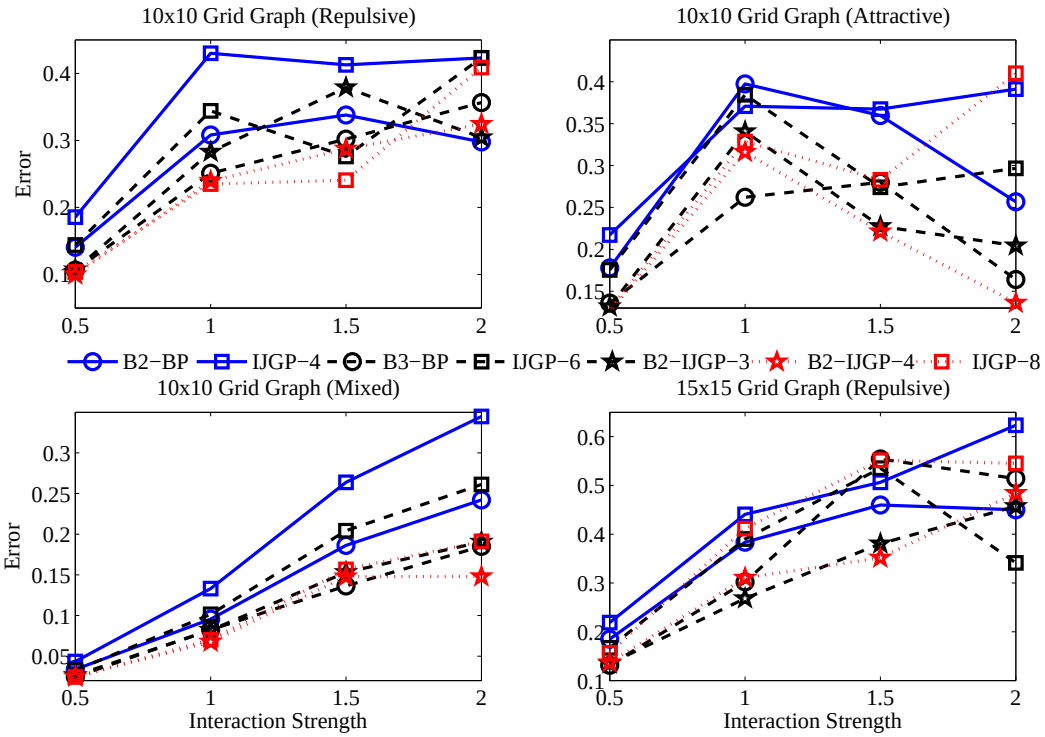


Fig. 11. Comparing the performance of BP, IJGP, and GBP. Algorithms with the same message passing complexity are plotted in the same color.

highlight the algorithm leading to the smallest mean error in bold. For example, when using block-graphs of size two for BP, the error decreases by as much as 25% (see BP vs. B2-BP), whereas when using block-graphs of size three for BP, the error decreases by as much as 50% (see BP vs. B3-BP).

Remark 7: It is interesting to compare BP, IJGP, and GBP, where both IJGP and GBP are based on finding overlapping clusters and IJGP first constructs a junction tree and then splits clusters in the junction tree to find overlapping clusters. Note that for the class of graphical models considered in (8), Bm-BP belongs to the class of GBP-2m algorithms since we can map the block-graph into an equivalent graph with overlapping clusters as done so when converting a block-tree into a junction tree (see Fig. 5). Further, IJBP-2m is also a GBP-2m algorithm [11]. Thus, we want to compare Bm-BP, IJGP-2m, and BP-2m. It is clear that GBP-2m leads to superior marginal estimates, however, this comes at the cost of significantly more computations. Fig. 11 compares Bm-BP to IJGP-2m. We observe that for many cases, Bm-BP leads to better marginal estimates than IJGP-2m. We note that comparing Bm-BP to IJGP-2m may not be appropriate since the stopping criteria for the IJGP may be different than that of the BP

algorithm[5].

Remark 8: We can apply the block-graph framework to generalize GBP based algorithms. Our results show that this leads to better marginal estimates, see GBP-4 vs. B2-GBP-4, IJGP-3 vs. B2-IJGP-3, and IJGP-4 vs. B2-IJGP-4. More specifically, looking at Table V, we notice that the performance of using block-graphs of size two on GBP results in the error being reduced to nearly 15% of the original error.

Remark 9: In Fig. 11, we see that for many cases the performance of B2-IJGP-3 (B2-IJGP-4) is better than IJGP-6 (IJGP-8). This suggests that the set of overlapping clusters chosen using the block-graph framework may be better than the clusters chosen using the IJGP framework.

Remark 10: Overall, we observe that block-graph versions of TreeEP lead to the best estimates with reasonable computational time. For example, in Table IV, with $\sigma = 1$, B2-TreeEP results in a mean error of 0.1583 running in an average of 0.455 seconds. In comparison, GBP-4 takes an average of about 95 seconds and the mean error is 0.0884. When compared to other algorithms, B3-CBP-2 runs in 0.32 seconds and results in a mean error of 0.2657. For mixed potentials in Table III, we observe that the generalized versions of TreeEP do not lead to significant improvements in the marginal estimates although the performance of other algorithms does improve. Reference [20] proposes a generalization of TreeEP and gives guidelines for choosing clusters in the GBP algorithm. As shown for IJGP and GBP, our framework can be used in conjunction with frameworks that use overlapping clusters.

Remark 11: To our knowledge, there have been no algorithms for generalizing LC and CBP-l. The computational complexity of LC is exponential in the maximum degree of the graph [10], so it is only feasible to apply LC to a limited number of graphs. We only used LC for the 5×5 grid graph example in Fig. 10. We observe that the CBP-l algorithm improves the estimates of the BP algorithm. Moreover, for regimes where the interaction strength is small, the performance of generalized versions of CBP is comparable to that of TreeEP. For example, in Table IV, for $\sigma = 0.5$, the best TreeEP algorithm has a mean error of 0.0694 and the best CBP based algorithm has a mean error of 0.0864. As another example, in Table V, for $\sigma = 0.5$, the best TreeEP algorithm has a mean error of 0.0624 and the best CBP algorithm has a mean error of 0.0608.

Remark 12: Fig. 12 shows how the error scales as the size of the cluster in the block-graph increases for the 20×20 grid graph. It is clear that the error in general decreases as the cluster size increases; however, for some cases, the error does seem to increase especially when the interaction strength is large.

[5]For IJGP, we used the software available at [51]. We could specify the maximum number of iterations, but not the stopping criteria.

TABLE I
10×10 GRID GRAPH WITH REPULSIVE POTENTIALS: 30 TRIALS

Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.2122	0.0457	0.3714	0.0237	0.4773	0.0150	0.4220	0.0120
B2-BP	0.1405	0.0213	0.3080	0.0073	0.3379	0.0057	0.2978	0.0037
B3-BP	**0.1065**	0.0193	**0.2509**	0.0133	**0.3019**	0.0033	**0.3565**	0.0020
IJGP-3	0.1864	-	0.3784	-	0.4560	-	0.4254	-
B2-IJGP-3	0.1073	-	0.2827	-	0.3789	-	**0.3044**	-
IJGP-6	0.1441	-	0.3442	-	0.2761	-	0.4232	-
IJGP-4	0.1856	-	0.4300	-	0.4128	-	0.4230	-
B2-IJGP-4	**0.0997**	-	0.2394	-	0.2873	-	0.3245	-
IJGP-8	0.1038	-	**0.2349**	-	**0.2407**	-	0.4088	-
CBP-2	0.1345	0.2507	0.2757	0.1710	0.4405	0.1240	0.3801	0.1137
B2-CBP-2	0.0740	0.1687	0.2109	0.1147	0.3263	0.0870	0.2871	0.0660
B3-CBP-2	0.0490	0.1583	0.1872	0.1213	0.2701	0.0890	0.3756	0.0767
CBP-3	0.1056	0.5223	0.2224	0.3537	0.4176	0.2603	0.3459	0.2313
B2-CBP-3	0.0561	0.3470	0.1726	0.2483	0.2824	0.1863	0.2752	0.1470
CBP-4	0.0866	1.0303	0.2094	0.6913	0.3420	0.5337	0.3195	0.4650
B2-CBP-4	**0.0437**	0.6810	**0.1229**	0.5067	**0.2438**	0.3800	**0.2744**	0.3123
TreeEP	0.0678	0.1993	0.1499	0.2513	0.1475	0.1680	0.1067	0.1630
B2-TreeEP	0.0547	0.1110	0.1273	0.1343	0.0489	0.1400	0.0551	0.1120
B3-TreeEP	**0.0542**	0.1463	**0.0878**	0.1683	**0.0485**	0.1447	**0.0414**	0.1527
GBP-4	0.0110	21.0497	0.0439	28.8153	0.0532	25.7290	0.0379	25.5437
B2-GBP-4	**0.0005**	16.7593	**0.0026**	25.5223	**0.0021**	28.8153	**0.0018**	33.1343

TABLE II
10×10 GRID GRAPH WITH ATTRACTIVE POTENTIALS: 30 TRIALS

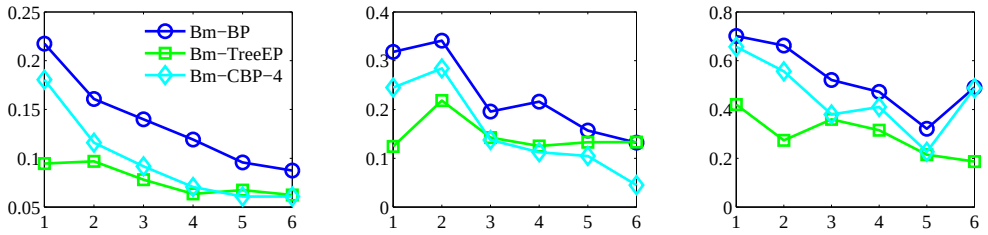
Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.2337	0.0600	0.4482	0.0217	0.3857	0.0160	0.3537	0.0113
B2-BP	0.1778	0.0227	0.3975	0.0080	0.3597	0.0083	0.2567	0.0053
B3-BP	**0.1358**	0.0173	**0.2622**	0.0080	**0.2799**	0.0090	**0.1640**	0.0027
IJGP-3	0.2125	-	0.3710	-	0.3088	-	0.3232	-
B2-IJGP-3	0.1316	-	0.3406	-	0.2275	-	0.2044	-
IJGP-6	0.1755	-	0.3846	-	0.2741	-	0.2967	-
IJGP-4	0.2171	-	0.3708	-	0.3674	-	0.3912	-
B2-IJGP-4	**0.1259**	-	**0.3161**	-	**0.2211**	-	**0.1360**	-
IJGP-8	0.1291	-	0.3287	-	0.2831	-	0.4102	-
CBP-2	0.1468	0.2543	0.3710	0.1590	0.3389	0.1337	0.3252	0.1007
B2-CBP-2	0.0687	0.1647	0.2913	0.1127	0.3177	0.0877	0.2521	0.0707
B3-CBP-2	0.0506	0.1607	0.1979	0.1143	0.2539	0.0943	**0.1092**	0.0763
CBP-3	0.1028	0.5243	0.3289	0.3300	0.2648	0.2713	0.2855	0.2180
B2-CBP-3	0.0490	0.3423	0.2522	0.2400	0.3006	0.1893	0.2416	0.1577
CBP-4	0.0784	1.0373	0.2595	0.6690	**0.2182**	0.5497	0.2662	0.4593
B2-CBP-4	**0.0382**	0.6737	**0.1839**	0.4723	0.2847	0.3857	0.1682	0.3233
TreeEP	0.0804	0.2300	0.2153	0.2470	0.1128	0.1720	0.0803	0.1977
B2-TreeEP	0.0499	0.1327	0.1390	0.1787	0.1104	0.1133	0.0552	0.1020
B3-TreeEP	**0.0427**	0.1407	**0.0989**	0.2117	**0.0686**	0.2090	**0.0546**	0.1337
GBP-4	0.0085	21.4370	0.0500	30.0877	0.0501	25.9627	0.0340	24.5683
B2-GBP-4	**0.0005**	16.7007	**0.0033**	26.8110	**0.0022**	28.9640	**0.0012**	32.2330

TABLE III
10×10 GRID GRAPH WITH MIXED POTENTIALS: 30 TRIALS

Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.0514	0.0237	0.1542	0.2863	0.3178	0.4313	0.3728	0.4750
B2-BP	0.0337	0.0057	0.0955	0.0310	0.1862	0.1653	0.2422	0.2320
B3-BP	**0.0243**	0.0057	**0.0824**	0.0363	**0.1364**	0.1217	**0.1851**	0.1883
IJGP-3	0.0431	-	0.1362	-	0.2719	-	0.3588	-
B2-IJGP-3	0.0264	-	0.0820	-	0.1531	-	0.1905	-
IJGP-6	0.0325	-	0.1015	-	0.2042	-	0.2612	-
IJGP-4	0.0434	-	0.1330	-	0.2639	-	0.3449	-
B2-IJGP-4	0.0246	-	**0.0675**	-	**0.1479**	-	**0.1479**	-
IJGP-8	**0.0243**	-	0.0703	-	0.1567	-	0.1911	-
CBP-2	0.0391	0.2240	0.0988	0.3727	0.1789	0.3817	0.2184	0.3913
B2-CBP-2	0.0240	0.1190	0.0692	0.2107	0.1107	0.2583	0.1572	0.2653
B3-CBP-2	**0.0164**	0.1230	**0.0505**	0.1957	0.1020	0.2440	0.1212	0.2600
CBP-3	0.0351	0.4950	0.0879	0.8083	0.1549	0.8327	0.1984	0.8497
B2-CBP-3	0.0208	0.2660	0.0642	0.4753	0.0988	0.5790	0.1325	0.6020
CBP-4	0.0314	1.0173	0.0780	1.6667	0.1484	1.7300	0.1755	1.7780
B2-CBP-4	0.0191	0.5620	0.0541	0.9970	**0.0986**	1.1933	**0.1004**	1.2360
TreeEP	0.0124	0.1190	0.0350	0.1650	0.0610	0.2177	0.0864	0.2743
B2-TreeEP	**0.0124**	0.0837	0.0386	0.1353	**0.0573**	0.1467	**0.0825**	0.2340
B3-TreeEP	0.0125	0.1057	**0.0342**	0.1490	0.0681	0.1980	0.0887	0.2413
GBP-4	0.0009	10.8840	0.0054	17.2403	0.0091	22.2760	0.0139	23.1837
B2-GBP-4	**0.0000**	10.8890	**0.0002**	15.1400	**0.0008**	17.8040	**0.0013**	18.6687

TABLE IV
15×15 GRID GRAPH WITH REPULSIVE POTENTIALS: 20 TRIALS

Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.2187	0.1550	0.3977	0.0870	0.5307	0.0760	0.5737	0.0575
B2-BP	0.1848	0.0745	0.3836	0.0590	0.4598	0.0400	0.4500	0.0385
B3-BP	**0.1314**	0.0700	**0.3020**	0.0455	**0.5541**	0.0360	**0.5140**	0.0445
IJGP-3	0.2052	-	0.4239	-	0.4846	-	0.5706	-
B2-IJGP-3	**0.1346**	-	**0.2683**	-	0.3805	-	0.4571	-
IJGP-6	0.1671	-	0.3903	-	0.5354	-	**0.3411**	-
IJGP-4	0.2187	-	0.4406	-	0.5065	-	0.6234	-
B2-IJGP-4	0.1367	-	0.3103	-	**0.3515**	-	0.4842	-
IJGP-8	0.1566	-	0.4115	-	0.5518	-	0.5451	-
CBP-2	0.1774	0.6895	0.3609	0.4940	0.4802	0.4440	0.5789	0.3435
B2-CBP-2	0.1335	0.4550	0.3161	0.3390	0.4749	0.2705	0.4517	0.2425
B3-CBP-2	0.0887	0.4520	**0.2657**	0.3200	0.5948	0.3115	0.4910	0.2515
CBP-3	0.1578	1.4245	0.3234	1.0085	0.4503	0.8665	0.5113	0.7035
B2-CBP-3	0.0946	0.9670	0.2964	0.6650	0.3852	0.5590	0.4319	0.5080
CBP-4	0.1382	2.9145	0.2904	1.9985	0.4150	1.6810	0.5364	1.3610
B2-CBP-4	**0.0864**	1.9710	0.2713	1.3210	**0.3778**	1.1395	**0.3966**	1.0225
TreeEP	0.0954	0.8500	0.2831	0.7995	0.5025	0.5895	0.7151	0.4345
B2-TreeEP	0.0888	0.5265	**0.1583**	0.4550	0.3566	0.4135	0.4293	0.3530
B3-TreeEP	**0.0694**	0.7530	0.1600	0.7835	**0.2584**	0.5290	**0.1375**	0.6025
GBP-4	0.0151	88.3395	0.0884	95.5370	0.0918	86.9865	0.0639	88.9645
B2-GBP-4	**0.0014**	81.1430	**0.0120**	117.4050	**0.0105**	117.1510	**0.0062**	120.3190



(a) $\sigma = 0.5$ (b) $\sigma = 1.0$ (c) $\sigma = 1.5$

Fig. 12. Error as the size of the cluster increases in the 20×20 grid graph. The horizontal axis denotes the cluster size in the block-graph and vertical axis denotes the mean error.

TABLE V
20×20 GRID GRAPH WITH REPULSIVE POTENTIALS: 10 TRIALS

Algorithm	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 1.5$	$\sigma = 2.0$	Algorithm	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 1.5$	$\sigma = 2.0$
BP	0.2174	0.3182	0.7014	0.5926	CBP-2	0.2028	0.3222	0.7034	0.5550
B2-BP	0.1608	0.3412	0.6623	0.5117	B2-CBP-2	0.1387	0.3023	0.6497	0.4465
B6-BP	**0.0874**	**0.1320**	**0.4904**	**0.3102**	B6-CBP-2	0.0638	0.0952	0.5348	0.2926
TreeEP	0.0946	0.1243	0.4201	0.5124	CBP-3	0.1893	0.3042	0.6634	0.5569
B2-TreeEP	0.0969	0.2184	0.2736	0.5154	B2-CBP-3	0.1248	0.2385	0.5538	0.4057
B6-TreeEP	**0.0624**	**0.1330**	**0.1868**	**0.0574**	B6-CBP-3	0.0637	0.0545	0.4947	**0.2919**
GBP-4	0.0175	0.0285	0.1188	0.0259	CBP-4	0.1806	0.2450	0.6571	0.5409
B2-GBP-4	0.0015	0.0034	0.0264	0.0038	B2-CBP-4	0.1160	0.2842	0.5563	0.3990
B3-GBP-4	**0.0003**	**0.0005**	**0.0021**	**0.0017**	B6-CBP-4	**0.0608**	**0.0451**	**0.4871**	0.3116

C. Random Regular Graphs

Tables VI and VII show results of applying the block-graph framework for inference over graphical models defined on random regular graphs with attractive potentials. Table VI considers graphs with 50 nodes and degree 3 and Table VII considers graphs with 70 nodes and degree 3. Just like the grid graph case, we observe that our generalized framework leads to better marginal estimates. This is shown by highlighting the algorithm that leads to minimal mean error for each inference algorithm. We observe that both CBP and TreeEP based algorithms perform the best, even when compared to GBP.

VII. SUMMARY

We proposed a framework for generalizing inference (computing marginal distributions given the joint probability distribution) algorithms over graphical models (see Fig. 1). The key components in our framework are (i) constructing a block-tree, a tree-structured graph over non-overlapping clusters, and (ii) constructing a block-graph, a graph over non-overlapping clusters. We proposed a linear time algorithm for constructing block-trees and showed how large clusters in a block-tree can be split in a

TABLE VI
RANDOM REGULAR GRAPH WITH $p = 50$ NODES, DEGREE 3, AND ATTRACTIVE POTENTIALS: 30 TRIALS

Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.0290	0.0020	0.1683	0.0050	0.1514	0.0023	0.2809	0.0020
B2-BP	0.0217	0.0010	0.1513	0.0037	0.1414	0.0023	**0.2520**	0.0000
B3-BP	**0.0167**	0.0000	**0.1407**	0.0033	**0.1394**	0.0023	0.2817	0.0000
CBP-2	0.0073	0.0630	0.0330	0.0827	0.0712	0.0610	0.2389	0.0467
B2-CBP-2	0.0058	0.0500	0.0272	0.0653	0.0647	0.0487	0.1977	0.0397
B3-CBP-2	0.0056	0.0530	0.0235	0.0670	0.0469	0.0560	0.1590	0.0450
CBP-3	0.0052	0.1357	0.0194	0.1653	**0.0358**	0.1313	0.1074	0.1097
B2-CBP-3	**0.0040**	0.1093	**0.0131**	0.1347	0.0368	0.1103	**0.0919**	0.0947
TreeEP	0.0101	0.0323	0.0687	0.0473	0.0815	0.0437	0.0675	0.0487
B2-TreeEP	0.0100	0.0320	0.0845	0.0553	0.0878	0.0543	0.0728	0.0457
B3-TreeEP	**0.0096**	0.0350	**0.0576**	0.0587	**0.0650**	0.0907	**0.0443**	0.0780
GBP-3	0.0290	1.1053	0.1683	1.4173	0.1514	1.0257	0.2299	0.7763
B2-GBP-3	**0.0217**	0.8870	**0.1513**	1.2520	**0.1414**	0.9837	**0.2280**	0.6937
GBP-4	0.0230	1.0157	0.1548	1.4403	0.1439	1.1250	0.2286	0.7587

TABLE VII
RANDOM REGULAR GRAPH WITH $p = 70$ NODES, DEGREE 3, AND ATTRACTIVE POTENTIALS: 20 TRIALS

Algorithm	$\sigma = 0.5$		$\sigma = 1$		$\sigma = 1.5$		$\sigma = 2.0$	
	Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
BP	0.0172	0.0070	0.1313	0.0200	0.2144	0.0100	0.2410	0.0100
B2-BP	0.0154	0.0040	0.1211	0.0120	0.2071	0.0090	0.2895	0.0040
B3-BP	**0.0106**	0.0025	**0.0871**	0.0120	**0.1866**	0.0065	**0.2037**	0.0040
CBP-2	0.0069	0.0990	0.0397	0.1425	0.1036	0.1185	0.1833	0.0965
B2-CBP-2	0.0069	0.0870	0.0459	0.1220	0.0923	0.1080	0.2571	0.0875
B3-CBP-2	0.0063	0.0870	0.0271	0.1135	0.0882	0.1065	0.1542	0.0880
CBP-3	**0.0057**	0.2120	**0.0247**	0.2840	0.0811	0.2465	0.1547	0.2050
B2-CBP-3	0.0063	0.1875	0.0264	0.2545	**0.0727**	0.2285	**0.0897**	0.1925
TreeEP	**0.0044**	0.0500	0.0404	0.0860	0.1028	0.0985	0.0741	0.1025
B2-TreeEP	0.0054	0.0575	0.0484	0.1130	0.1016	0.0990	0.0969	0.1000
B3-TreeEP	0.0049	0.0715	**0.0341**	0.1005	**0.0682**	0.1575	**0.0548**	0.1730
GBP-3	0.0150	1.4935	0.1081	3.0275	0.2100	2.2155	0.2149	2.0235
B2-GBP-3	**0.0127**	1.4910	**0.0939**	2.8880	**0.1998**	2.1770	**0.2046**	1.6230

systematic manner to construct block-graphs that are favorable for inference. Using numerical simulations, we showed that our framework for generalized inference in general leads to improved marginal estimates for many approximate inference algorithms implemented in the libDAI software package. This suggests that the generalized inference framework can be used as a wrapper for improving the performance of approximate inference algorithms. All the code and graphical models used in the numerical simulations can be downloaded from http://www.ima.umn.edu/~dvats/GeneralizedInference.html. Although the focus in this paper was on computing marginal estimates, our proposed block-graph based framework can also

be used to generalize algorithms for computing the partition function (Z in (1)) [52], [53] or for the problem of MAP inference [54]–[57].

There are several interesting research directions that can be further pursued to improve our generalized inference framework. Our algorithm for constructing block-graphs only used the structure of the graph in computing the set of non-overlapping clusters. Using the parameters of the graphical model may result in improved marginal estimates. Further, it may be of interest to design block-graphs that are specific to the inference algorithm of interest. Another interesting research direction is to combine frameworks that choose overlapping clusters with the block-graph framework.

REFERENCES

[1] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications (Monographs on Statistics and Applied Probability)*, 1st ed. Chapman & Hall/CRC, February 2005.

[2] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*. Hanover, MA, USA: Now Publishers Inc., 2008.

[3] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[4] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, 1988.

[5] E. Fabre and A. Guyader, "Dealing with short cycles in graphical codes," in *IEEE International Symposium on Information Theory (ISIT)*, June 2000, p. 10.

[6] E. P. Xing, M. I. Jordan, and S. Russell, "Graph partition strategies for generalized mean field inference," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2004, pp. 602–610. [Online]. Available: http://portal.acm.org/citation.cfm?id=1036843.1036916

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[8] F. Eaton and Z. Ghahramani, "Choosing a variable to clamp: Approximate inference using conditioned belief propagation," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, 2009, pp. 145–152.

[9] A. Montanari and T. Rizzo, "How to compute loop corrections to the Bethe approximation," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 10, p. P10011, 2005.

[10] J. M. Mooij and H. J. Kappen, "Loop corrections for approximate inference on factor graphs," *J. Mach. Learn. Res.*, vol. 8, pp. 1113–1143, May 2007.

[11] R. Mateescu, K. Kask, V. Gogate, and R. Dechter, "Join-graph propagation algorithms," *Journal of Artificial Intelligence Research*, vol. 37, pp. 279–328, 2010.

[12] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized belief propagation," in *NIPS*, 2001, pp. 689–695.

[13] T. Heskes, K. Albers, and B. Kappen, "Approximate inference and constrained optimization," in *UAI*, 2003, pp. 313–320.

[14] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free energy approximations and generalized belief propagation algorithms." *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282–2312, July 2005.

[15] A. Pelizzola, "Cluster variation method in statistical physics and probabilistic graphical models," *Journal of Physics A: Mathematical and General*, vol. 38, no. 33, pp. R309–R339, 2005.

[16] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, Nov 1999.

[17] J. Yedidia, W. Freeman, and Y. Weiss, "Bethe free energy, Kikuchi approximations, and belief propagation algorithms," Mitsubishi Electric Research Laboratories, Tech. Rep. TR2001-16, 2001.

[18] R. Kikuchi, "A theory of cooperative phenomena," *Phys. Rev.*, vol. 81, no. 6, pp. 988– 988–1003, 1951.

[19] M. Welling, "On the choice of regions for generalized belief propagation," in *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence*, 2004, pp. 585–592.

[20] M. Welling, T. Minka, and Y. W. Teh, "Structured region graphs: Morphing EP into GBP," in *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, vol. 21, 2005.

[21] E. P. Xing, M. I. Jordan, and S. Russell, "A generalized mean field algorithm for variational inference in exponential families," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2003, pp. 602–610. [Online]. Available: http://portal.acm.org/citation.cfm?id=1036843.1036916

[22] B. M. Marlin and K. P. Murphy, "Sparse Gaussian graphical models with unknown block structure," in *International Conference on Machine Learning*, 2009, pp. 89–712.

[23] A. Jalali, P. Ravikumar, V. Vasuki, and S. Sanghavi, "On learning discrete graphical models using group-sparse regularization," in *International Conference on Machine Learning*, 2011, pp. 89–712.

[24] D. Malioutov, "Approximate inference in Gaussian graphical models," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2008.

[25] P. Lévy, "A special problem of Brownian motion, and a general theory of Gaussian random functions," in *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, 1954–1955, vol. II*. Berkeley and Los Angeles: University of California Press, 1956, pp. 133–175.

[26] D. Vats and J. M. F. Moura, "Telescoping recursive representations and estimation of Gauss-Markov random fields," *Trans. on Information Theory*, vol. 57, no. 3, pp. 1645 – 1663, 2011.

[27] R. E. Kalman and R. Bucy, "New results in linear filtering and prediction theory," *Transactions of the ASME–Journal of Basic Engineering*, vol. 83, no. Series D, pp. 95–108, 1960.

[28] H. E. Rauch, F. Tung, and C. T. Stribel, "Maximum likelihood estimates of linear dynamical systems," *AIAA J.*, vol. 3, no. 8, pp. 1445–1450, August 1965.

[29] S. L. Lauritzen, *Graphical Models*. Oxford University Press, USA, 1996.

[30] J. Besag, "Spatial interaction and the statistical analysis of lattice systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 192–236, 1974.

[31] R. Halin, "S-functions for graphs," *Journal of Geometry*, vol. 8, pp. 171–186, 1976, 10.1007/BF01917434.

[32] N. Robertson and P. D. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," *Journal of Algorithms*, vol. 7, no. 3, pp. 309 – 322, 1986.

[33] F. Jensen and F. Jensen, "Optimal junction trees," in *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. San Francisco, CA: Morgan Kaufmann, 1994, pp. 360–36.

[34] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, no. 3, pp. 255–269, 1957.

[35] A. Berry, P. Heggernes, and G. Simonet, "The minimum degree heuristic and the minimal triangulation process," in *Graph-Theoretic Concepts in Computer Science*, ser. Lecture Notes in Computer Science, H. Bodlaender, Ed. Springer Berlin / Heidelberg, 2003, vol. 2880, pp. 58–70.

[36] U. B. Kjaerulff, “Triangulation of graphs - algorithms giving small total state space,” Department of Mathematics and Computer Science, Aalborg University, Denmark, Tech. Rep. Research Report R-90-09, 1990.

[37] G. F. Cooper, “Nestor: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge,” Ph.D. dissertation, Department of Computer Science, Stanford University, 1984.

[38] Y. Peng and J. A. Reggia, “Plausibility of diagnostic hypotheses,” in *National Conference on Artificial Intelligence (AAAI’86)*, 1986, pp. 140–145.

[39] J. W. Woods and C. Radewan, “Kalman filtering in two dimensions,” *IEEE Trans. Inf. Theory*, vol. 23, no. 4, pp. 473–482, Jul 1977.

[40] J. M. F. Moura and N. Balram, “Recursive structure of noncausal Gauss-Markov random fields,” *IEEE Trans. Inf. Theory*, vol. IT-38, no. 2, pp. 334–354, March 1992.

[41] B. C. Levy, M. B. Adams, and A. S. Willsky, “Solution and linear estimation of 2-D nearest-neighbor models,” *Proc. IEEE*, vol. 78, no. 4, pp. 627–641, Apr. 1990.

[42] G. Shafer and P. P. Shenoy, “Probability propagation,” *Annals of Mathematics and Artificial Intelligence*, no. 1-4, pp. 327–352, 1990.

[43] N. Zhang and D. Poole, “A simple approach to Bayesian network computations,” in *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, 1994, pp. 171–178.

[44] R. Dechter, “Bucket elimination: A unifying framework for reasoning,” *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, Sep 1999.

[45] K. Kask, R. Dechter, J. Larrosa, and F. Cozman, “Bucket-tree elimination for automated reasoning,” University of California, Irvine, Tech. Rep. R92, 2001.

[46] M. Kolar, L. Song, A. Ahmed, and E. P. Xing, “Estimating time-varying networks,” *Annals of Applied Statistics*, 2009.

[47] S. Zhou, J. Lafferty, and L. Wasserman, “Time varying undirected graphs,” *Machine Learning*, vol. 80, pp. 295–319, 2010, 10.1007/s10994-010-5180-0.

[48] F. Kschischang, B. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498 –519, feb 2001.

[49] T. Minka and Y. Qi, “Tree-structured approximations by expectation propagation,” in *Proc. Neural Information Processing Systems Conf. (NIPS)*, 2003, p. 2003.

[50] J. M. Mooij, “libDAI: A free and open source C++ library for discrete approximate inference in graphical models,” *Journal of Machine Learning Research*, vol. 11, pp. 2169–2173, Aug. 2010.

[51] V. Gogate, “Iterative joint graph propagation,” http://www.hlt.utdallas.edu/~vgogate/ijgp.html.

[52] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, “A new class of upper bounds on the log partition function,” *IEEE Trans. on Information Theory*, vol. 51, no. 7, pp. 2313 – 2335, July 2005.

[53] Q. Liu and A. Ihler, “Bounding the partition function using Holder’s inequality,” in *Proceedings of the 28th International Conference on Machine Learning*, L. Getoor and T. Scheffer, Eds., June 2011, pp. 849–856.

[54] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, “MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches,” *IEEE Trans. Inf. Theory*, vol. 51, no. 11, pp. 3697–3717, Nov. 2005.

[55] V. Kolmogorov, “Convergent tree-reweighted message passing for energy minimization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 1568–1583, 2006.

[56] D. Sontag and T. Jaakkola, “Tree block coordinate descent for MAP in graphical models,” *Journal of Machine Learning Research*, vol. 5, pp. 544–551, 2009.

[57] T. Jebara, “MAP estimation, message passing, and perfect graphs,” in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI ’09, 2009, pp. 258–267.