Finding Non-overlapping Clusters for Generalized Inference Over Graphical Models

Divyanshu Vats and José M. F. Moura

Abstract

Graphical models compactly capture stochastic dependencies amongst a collection of random variables using a graph. Inference over graphical models corresponds to finding marginal probability distributions given joint probability distributions. Several inference algorithms rely on iterative message passing between nodes. Although these algorithms can be generalized so that the message passing occurs between clusters of nodes, there are limited frameworks for finding such clusters. Moreover, current frameworks rely on finding clusters that are overlapping. This increases the computational complexity of finding clusters since the edges over a graph with overlapping clusters must be chosen carefully to avoid inconsistencies in the marginal distribution computations. In this paper, we propose a framework for finding clusters in a graph for generalized inference so that the clusters are *non-overlapping*. Given an undirected graph, we first derive a linear time algorithm for constructing a block-tree, a tree-structured graph over a set of non-overlapping clusters. We show how the belief propagation (BP) algorithm can be applied to block-trees to get exact inference algorithms. We then show how larger clusters in a block-tree can be efficiently split into smaller clusters so that the resulting graph over the smaller clusters, which we call a block-graph, has lower number of cycles than the original graph. We show how loopy BP (LBP) can be applied to block-graphs for approximate inference algorithms. Numerical simulations show the benefits of running LBP on block-graphs as opposed to running LBP on the original graph. Our proposed framework for generalizing BP and LBP can be applied to other inference algorithms.

Index Terms

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

The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, USA (email: dvats@andrew.cmu.edu, moura@ece.cmu.edu, ph: (412)-268-6341, fax: (412)-268-3980).

I. Introduction

A graphical model is a random vector defined on a graph such that each node represents a random variable (or multiple random variables), and edges in the graph represent conditional independencies. The underlying graph structure in a graphical model leads to a factorization of the joint probability distribution. This property has lead to graphical models being used in many applications such as sensor networks, image processing, computer vision, bioinformatics, speech processing, and ecology [1], [2], to name a few.

The structure of the graph plays an important role in determining the computational complexity of performing various tasks over graphical models. For example, given a probability distribution $p(\mathbf{x})$ of a graphical model $\mathbf{x} = \{x_1, \ldots, x_n\}$, it is of interest in many applications to compute the marginal distribution $p_s(x_s)$. This problem is referred to as *inference* [2]. Algorithms for performing inference over graphical models can be divided into exact inference algorithms and approximate inference algorithms. The need for approximate inference algorithms arises because for a large class of graphs (with high treewidth), performing exact inference is computationally intractable.

Belief propagation (BP) [3] is a popular algorithm for exact inference over *tree-structured* graphical models. The BP algorithm iteratively computes the marginal distribution of each node in the graphical model by passing messages between nodes. When applying BP to graphs with cycles, resulting in the loopy BP (LBP) algorithm [3], it is well known that the algorithm may not converge and the estimated marginal distributions may differ from the true marginal distribution. To improve the accuracy of LBP, the generalized BP (GBP) was proposed in [4], where the authors establish theoretical results showing how by performing message passing between clusters of nodes (instead of individual nodes), the accuracy of inference may be improved. Despite the known theoretical advantages of GBP, there have been limited frameworks for systematically finding clusters in a graph for performing GBP. Further, in many cases, it is not clear how the approach of message passing between clusters instead of nodes can be naturally extended to other inference algorithms such as mean-field inference, tree-based reparameterization belief propagation (TRP-BP) [5], [6], and tree-reweighted belief propagation (TRW-BP) [7].

A. Summary of Contributions

In this paper, we propose a framework that uses *non-overlapping clusters* to generalize inference algorithms over graphical models. The proposed framework is illustrated in Fig. 1. Let Alg be an inference algorithm that marginalizes a joint probability distribution $p(\mathbf{x})$ defined on a graph $G = (V, E)$. The output of Alg is the set of marginal distributions $\{p_s(x_s)\}$. In our proposed framework for generalizing Alg, we

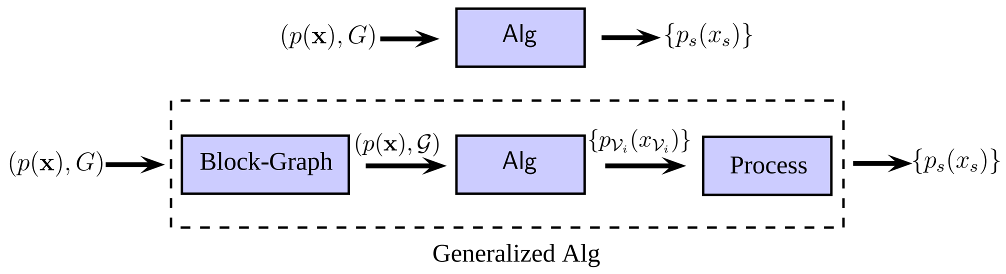


Fig. 1. A framework for generalized inference over graphical models.

map the graph G into a *block-graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, defined as a graph in which each node is a cluster of multiple nodes (from the original graph G) such that the clusters are non-overlapping. We then apply Alg to the block-graph to get marginal distributions of each cluster, which we process to get the marginal distribution of each node.

Our framework in Fig. 1 can be applied to generalize inference algorithms. In this paper, we focus on generalizing belief propagation (BP) and loopy BP. We first consider the problem of exact inference over graphical models. Given a *non tree-structured* graphical model, we propose a *linear time* algorithm for finding non-overlapping clusters in the graph so that the block-graph over these clusters is tree-structured. We call this graph a *block-tree*. Applying BP to the block-tree leads to exact inference algorithms for arbitrary graphical models. We compare the block-tree framework for exact inference to the junction-tree framework [8]. A junction-tree is a tree-structured graph over clusters, where clusters connected by edges are *always overlapping*. We show that constructing block-trees is faster and give trade-offs for using block-trees as opposed to junction-trees when performing inference over graphical models.

Next, we generalize the loopy BP (LBP) algorithm using the block-graph framework. LBP is used extensively in the literature since exact inference using block-trees or junction-trees is computationally intractable when the size of the clusters becomes very large. We propose a simple and efficient algorithm for splitting larger clusters in a block-tree to form a block-graph. Our proposed algorithm for constructing block-graphs reduces the number of cycles in the original graph (by forming non-overlapping clusters) since LBP is known to perform better on nearly tree-structured graphical models. Using numerical simulations, we show how LBP over block-graphs, constructed using our algorithm, leads to more accurate inference algorithms when compared to using LBP on the original graph.

B. Related Work

In the literature, exact inference over graphical models using non-overlapping clusters is referred to as the Pearl's clustering algorithm [3]. In [9] and [10], the authors use non-overlapping clustering for some particular directed graphical models for an application in medical diagnostics. For lattices, [11]–[13] derive inference algorithms by scanning the lattice horizontally (or vertically). Our algorithm in this paper provides a principled way of finding non-overlapping clusters for deriving exact inference algorithms over *arbitrary* (not necessarily lattice graphs and particular directed graphs) graphical models.

There has been significant work in extending the LBP 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 [14]. In [4], [15], the authors showed that the fixed points of the LBP algorithm minimize the Bethe free energy, which is an approximation to the Gibbs free energy. This motivated the Generalized Belief Propagation (GBP) algorithm which minimizes the Kikuchi free energy [16], a better approximator to the Gibbs free energy. In GBP, the message passing is between clusters of nodes that are overlapping. A more general approach to GBP is proposed in [17] using region graphs and in [18] using the cluster variation method. Further, region graphs have been used in [5], [19] to extend the TRP based method for inference in graphical models.

Although [17], [19] show that approximate inference can be improved using clustering, they do not address the problem of choosing regions or clusters in arbitrary graphs over which we want to perform message passing. In our numerical simulations, we show that choosing clusters arbitrarily may not result in better inference algorithms. In [20], [21], the authors have considered methods for choosing regions, however, their algorithms are limited to a small class of graphical models. In [22], the authors propose Iterative Join-Graph Propagation (IJGP), which is a class of GBP algorithms. The IJGP algorithm first computes a junction-tree and then randomly splits appropriate clusters into smaller clusters to build a join-graph (or cluster-graph). However, since IJGP uses junction-trees, the clusters chosen are overlapping and thus it is important to assign edges over these clusters so that the running intersection property is satisfied. This constraint, combined with the fact that we are constructing a junction-tree, makes the construction of a join-graph computationally hard. Thus, for problems where the graph structure changes over time, using junction-tree based algorithms can be computationally infeasible. Another restriction of the IJGP based approach is that it does not generalize other algorithms for message passing such as TRP-BP or TRW-BP.

Our proposed approach of using non-overlapping clusters for approximate inference has been considered before. In the original paper describing GBP [15], the authors give an example of how non-overlapping clusters can be used, however, there have been no algorithms in the literature for finding appropriate clusters. In [23], [24], the authors use disjoint clusters to generalize mean field algorithms for approximate inference. Although we do not demonstrate the use of our algorithm for mean field methods, the clustering algorithm used in [24] is based on standard graph partitioning algorithms that are computationally intensive for large graphs. Our greedy approach to finding clusters is simple, efficient, and motivated by the observation that approximate inference is more accurate on graphs with smaller and longer cycles than on graphs with larger and shorter cycles.

We note that our work differs from some other work on studying graphical models defined over graphs with non-overlapping clusters. For example, [25], [26] consider the problem of learning a Gaussian graphical model defined over some block-graph. Similar efforts have been made in [27] for discrete valued graphical models. In [28], 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.

Our work in this paper is motivated by earlier work done by us in studying tree structures for Markov random fields (MRFs) indexed over continuous indices [29]. In [30], [31], 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 [32] and the Rauch-Tung-Striebel smoother [33] to Gaussian MRFs indexed over continuous indices.

C. Paper Organization

Section II reviews graphical models and the inference problem. Section III presents our algorithm for finding non-overlapping clusters in a graph that form a tree. The resulting graph over the clusters is called a block-tree. Section IV shows how exact inference can be performed on block-trees. Section V presents our algorithm for splitting larger clusters in a block-tree for tractable approximate inference algorithms. Section VI presents numerical simulations showing how effective our clustering algorithm is for improving the accuracy of LBP. Section VII summarizes the paper.

II. Background

Section II-A reviews properties and definitions related to graphs and graphical models necessary for this paper. For a more complete study, we refer the readers to [34]. Section II-B reviews the junction-tree

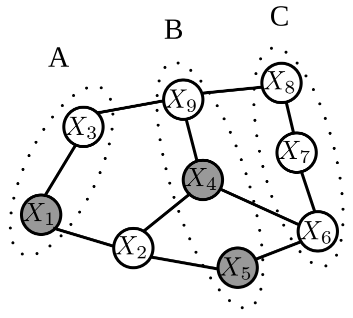


Fig. 2. An example of a graphical model.

algorithm for exact inference over graphical models.

A. Graphical Models and Inference

A graphical model is defined using a graph $G = (V, E)$, where the nodes $V = \{1, 2, \dots, p\}$ index a collection of random variables $\mathbf{x} = \{x_s \in \Omega : s \in V\}$ and the edges $E \subseteq V \times V$ encode statistical dependencies [34], [35]. In this paper, we assume $|\Omega| = K$ is finite. Our results and algorithms easily generalize to continuous valued random variables. The set of edges can be directed, undirected, or both. Since directed graphical models can be mapped to undirected graphical models by moralizing the graph, in this paper, we only present algorithms for undirected graphical models. The extensions to directed graphical models follow immediately.

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, 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 [36], the Markov property leads to a factorization of the joint probability distribution over cliques (fully connected subsets of nodes) in the graph,

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

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 $\mathbf{x} = \{x_1, \ldots, x_n\}$. 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 $p(\mathbf{x}|\mathbf{y})$.

As mentioned before, belief propagation (BP) is a popular algorithm for exact inference over tree-structured graphical models. BP consists of two stages. In the first stage, we pass messages from leaves in a tree to the root. In the second stage, we pass messages from the root to the leaves. To find the marginal distribution of each node, we take the product of all incoming messages in the tree. An extension of BP to more general graphs is provided in [8], where the message passing is done between clusters instead of individual nodes. We review this algorithm, called the junction-tree algorithm, in the next Section. Some other frameworks for exact inference over arbitrary graphical models have been proposed in [37], [38].

B. Junction-Tree

Junction-trees were introduced in [39] for representing graphs with cycles as tree-structured graphs. In [8], the authors showed that inference over undirected graphical models with cycles can be achieved using junction-trees. Formally, a junction-tree is defined as follows [40].

Definition 1 (Junction-Tree): For a graph $G = (V, E)$, a junction-tree $\mathcal{J} = (\mathcal{C}, \mathcal{E})$ is a graph over clusters of nodes in V so that

1) Each node in V is associated with at least one cluster in $\mathcal{C}$.
2) For every edge $(v_1, v_2) \in E$, there exists a cluster $C_k \in \mathcal{C}$ such that $\{v_1, v_2\} \in C_k$.
3) $\mathcal{J}$ satisfies the running intersection property: For all clusters C_i, C_k, and C_j such that C_k separates C_i and C_j, $C_i \cap C_j \subset C_k$.

The idea behind using junction-trees for inference in graphical models is to cluster nodes in the graph and then form a tree-structured graph over the clusters. The clusters in a junction-tree are overlapping, i.e., for two clusters C_i and C_j connected by an edge in the junction-tree, $|C_i \cap C_j| > 0$. Given a tree-structured graph, we can apply a variant of the belief propagation algorithm so that the message passing is between clusters of nodes (instead of individual nodes) [41], [42]. The running intersection property ensures that inference over the junction-tree is consistent so that if a node s is in two clusters, the marginal distribution of x_s computed from each cluster after running message passing is the same. For an example of a junction-tree, we refer to Appendix A, where we review algorithms for constructing junction-trees.

The following proposition summarizes the complexity of constructing junction-trees and that of performing inference over junction-trees.

Proposition 1: Given an undirected graph $G = (V, E)$, the complexity of constructing a junction-tree $\mathcal{J} = (\mathcal{C}, \mathcal{E})$ is $O(|E| + |\mathcal{C}|^2)$. For a random vector $\mathbf{x} \in \Omega^n$, where $|\Omega| = K$, Markov on the graph G, the complexity of cluster based message passing using the junction-tree $\mathcal{J}$ is

$$O\left(\sum_{(i,j)\in\mathcal{E}} K^{|C_i|} + K^{|C_j|}\right) . \tag{2}$$

Proof: See [43] for the complexity of constructing a junction-tree and [44] for the complexity of message passing in a junction-tree. ■

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

In this Section, we present our algorithm for finding a block-tree, which is formally defined as follows.

Definition 2 (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*.

Our proposed algorithm for constructing a block-tree $\mathcal{G}$ given an arbitrary graph $G = (V, E)$ is shown in Algorithm 1. The input to the algorithm is the original graph G and a set of nodes $V_1 \subset V$. The output of the algorithm 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. The various steps in the algorithm are explained as follows.

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 clusters 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 $V_{r-1}, V_{r-2}, \ldots, V_3$.

In the first part of Algorithm 1, we find successive non-overlapping neighbors of the root cluster. In the backwards step, we merge clusters to form a final block-tree. We illustrate Algorithm 1 using some

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 $V_1, V_2, \dots, V_r$ such that
   $V_2 = \mathcal{N}(V_1)$, $V_3 = \mathcal{N}(V_2)\backslash\{V_1 \cup V_2\}, \dots, V_r = \mathcal{N}(V_r)\backslash\{V_{r-2} \cup V_{r-1}\}$.
2  $\{V_k^1, \dots, V_k^{m_k}\} \leftarrow$ Find $m_k$ connected components of $V_k$ using subgraph $G(V_k)$.
3  for $i = r, r-1, \dots 3$ do
4      for $j = 1, 2, \dots m_i$ 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  $\mathcal{V} \leftarrow \bigcup_{k=1}^{r}\{V_k^1, V_k^2, \dots, V_k^{m_k}\}$
8  $\mathcal{E} \leftarrow$ edges between all the clusters in $\mathcal{V}$
```

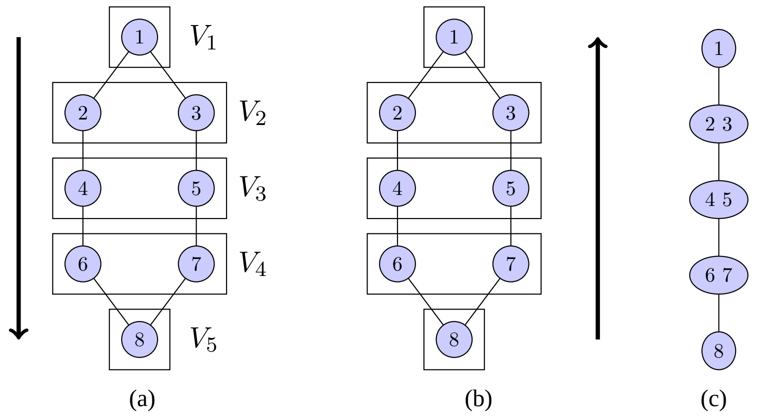


Fig. 3. (a) Original estimates of the clusters in a single loop graph when running the forward pass of Algorithm 1. (b) The final clusters after running the backwards pass of Algorithm 1. (c) Final block-tree.

examples.

Example: Consider the graph with a single loop in Fig. 3(a). Choosing $V_1 = \{1\}$, we can get the initial estimates of the clusters as shown in Fig. 3(a). To get the final estimates, which coincides with the initial estimates of the clusters, we run the backwards step of Algorithm 1. See Fig. 3(b) for the resulting graph. The final block-tree is shown in Fig. 3(c).

Example: Now consider the grid graph of Fig. 15(a). Choosing $V_1 = \{1\}$, we get the initial estimates of the clusters as shown in Fig. 4(a). Running the backwards step to identify the final clusters (see Fig. 4(b)), we get the block-tree in Fig. 4(c).

Example: In the examples considered so far, 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

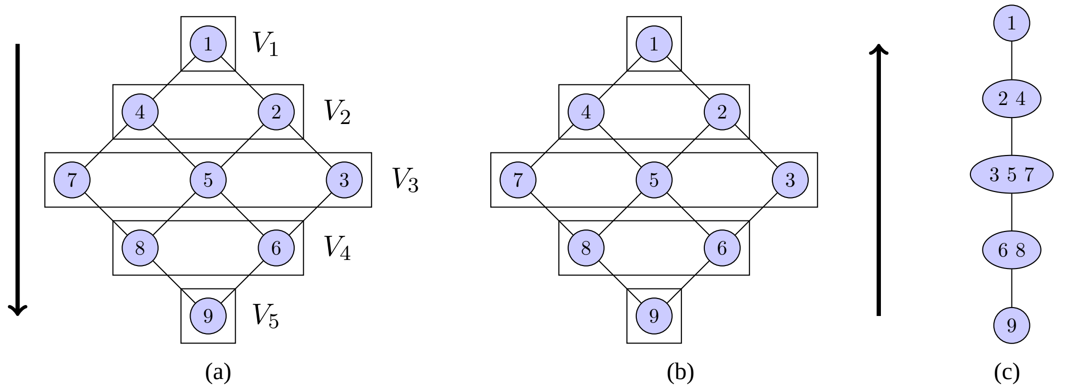


Fig. 4. (a) Original estimates of the clusters in a grid graph when running the forward pass of Algorithm 1. (b) The final clusters after running the backwards pass of Algorithm 1. (c) Final block-tree.

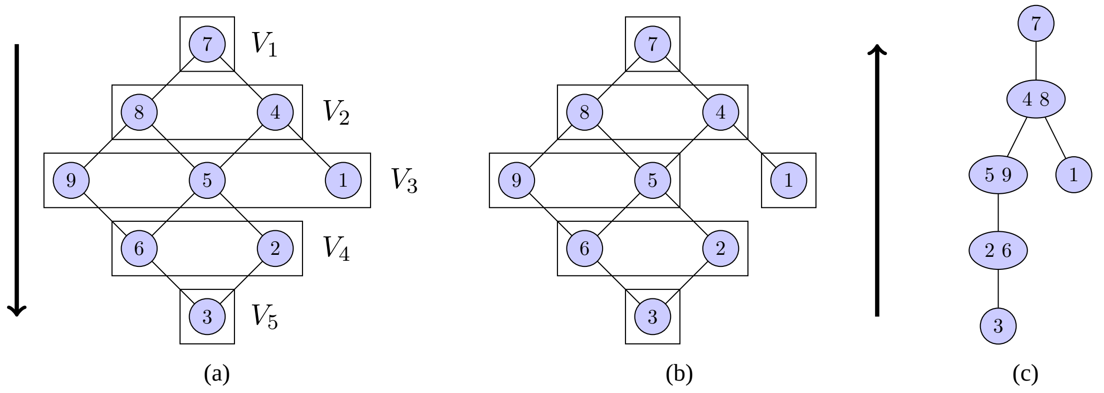


Fig. 5. (a) Original estimates of the clusters in a partial grid when running the forward pass of Algorithm 1. (b) The final clusters after running the backwards pass of Algorithm 1. (c) Final block-tree.

final block-tree will in fact be tree-structured. Consider the partial grid graph of Fig. 5(a). Choosing $V_1 = \{7\}$, we get the initial estimates of the clusters in Fig. 5(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 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 rest of the clusters as shown in Fig. 5(b). The final block-tree is shown in Fig.5(c).

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

Proposition 2: Algorithm 1 for constructing block-trees runs in time $O(|E|)$ and always outputs a block-tree.

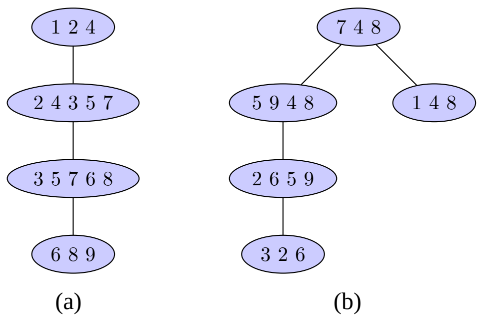


Fig. 6. (a) Junction-tree for the block-tree in Fig. 4(c) (b) Junction-tree for the block-tree in Fig. 5(c)

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}. ■

We make the following remarks regarding Algorithm 1.

Remark 1: If G is a tree to begin with, the output of Algorithm 1 will be the same tree no matter which root cluster we start with. However, when constructing junction-trees, it is important to choose an appropriate elimination order (see Appendix A) to ensure that an optimal junction-tree is constructed.

Remark 2: Different choices of the root cluster V_1 will yield different block-trees. Note that once V_1 is fixed, the block-tree is also fixed. In Section IV-B, we define an optimal block-tree which depends on the choice of the initial cluster V_1.

Remark 3: Comparing the complexity of constructing block-trees (see Proposition 2) to that of constructing junction-trees (see Proposition 1), we notice that constructing junction-trees has an added complexity that is quadratic in the number of clusters in the junction-tree. Thus, constructing block-trees is faster than constructing junction-trees.

Remark 4: Using Algorithm 2 for constructing junction-trees (see Appendix A), we can find a junction-tree for every block-tree. For example, the junction-tree representation for the block-trees in Fig. 4(c) and Fig. 5(c) are given in Fig. 6(a) and Fig. 6(b), respectively. We use this relationship in Section IV-C to find an approximation to the optimal block-trees.

IV. Exact Inference Using Block-Trees

Section IV-A shows how the belief propagation algorithm can be applied to block-trees for inference over graphical models. Section IV-B defines an optimal block-tree. Section IV-C discusses greedy algorithms for finding optimal block-trees. Section IV-D compares block-trees to junction-trees.

A. *Belief Propagation on Block-Trees*

Since block-trees are tree-structured graphs, we can use belief propagation algorithms for inference. Let $\mathbf{x}$ be an undirected graphical model on $G = (V, E)$ with probability distribution $p(\mathbf{x})$. For simplicity, we assume that $\mathbf{x}$ is discrete valued so that $\mathbf{x} \in \Omega$, where $|\Omega| = K$, however, the inference algorithm easily extends to continuous valued random variables. Recall from (1) that the probability distribution factorizes over the cliques of the graph so that

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)\,, \tag{3}$$

where the functions $\psi_C(x_C)$ are known a priori. Suppose we construct a block-tree $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from G using a root cluster V_1. To capture the structure of the original graph G in the block-tree $\mathcal{G}$, we introduce the following notation.

Notation: For every edge $(i, j) \in \mathcal{E}$ connecting two clusters $\mathcal{V}_i$ and $\mathcal{V}_j$, associate a label $(\overline{\mathcal{V}}_i, \overline{\mathcal{V}}_j)$ so that in the original graph G, the nodes in $\overline{\mathcal{V}}_i$ are not connected to the nodes in $\mathcal{V}_j$ and the nodes in $\overline{\mathcal{V}}_j$ are not connected to the nodes in $\mathcal{V}_i$. For example, in Fig. 5(c), all the edge labels will be empty, except the label between $\{1\}$ and $\{4, 8\}$, which will have a label $(\{\}, \{8\})$ since 8 is not connected to 1 in the original graph.

The following steps illustrate how the belief propagation algorithm can be modified for block-trees.

Step 1. Construct a block-tree $\mathcal{G}$ from the undirected graph G using a root cluster V_1.

Step 2. The cliques in the block-tree $\mathcal{G}$ are the edges, so we can write down an alternative factorization for the probability distribution in $p(\mathbf{x})$ as

$$p(\mathbf{x}) = \prod_{k=1}^{l} \Phi_k(x_{\mathcal{V}_k}) \prod_{(i,j) \in \mathcal{E}} \Psi_{i,j}(x_{\mathcal{V}_i \backslash \overline{\mathcal{V}}_i}, x_{\mathcal{V}_j \backslash \overline{\mathcal{V}}_j})\,, \tag{4}$$

where we map each potential function in the original factorization in (3) to an appropriate factor without repeating.

Step 3. Using an arbitrary root node in the block-tree, identify the leaves and pass messages starting at the leaves up to the root. For example, a message from cluster $\mathcal{V}_i$ to cluster $\mathcal{V}_j$ is given as follows:

$$m_{i \to j}(x_{\mathcal{V}_j \backslash \overline{\mathcal{V}}_j}) = \sum_{x_{\mathcal{V}_i \backslash \overline{\mathcal{V}}_i}} \Psi_{i,j}(x_{\mathcal{V}_i \backslash \overline{\mathcal{V}}_i}, x_{\mathcal{V}_j \backslash \overline{\mathcal{V}}_j}) \sum_{x_{\overline{\mathcal{V}}_i}} \prod_{k \in \mathcal{N}(\mathcal{V}_i) \backslash \mathcal{V}_j} \Phi_i(x_{\mathcal{V}_i}) m_{k \to i}(x_{\mathcal{V}_i})\,, \tag{5}$$

where $\mathcal{N}(\mathcal{V}_i)$ are neighboring clusters of $\mathcal{V}_i$.

Step 4. Once all messages have reached the root, pass messages from the root back to the leaves using the same message passing equation in (5).

Step 5. The joint distribution for each cluster is given by

$$p_{\mathcal{V}_i}(x_{\mathcal{V}_i}) = \prod_{j \in \mathcal{N}(\mathcal{V}_i)} m_{j \to i}(x_{\mathcal{V}_i}) \Phi_i(x_{\mathcal{V}_i}) \tag{6}$$

Step 6. Marginalize the distribution of each cluster to find the distribution of each node.

The complexity of the message passing algorithm is given as follows.

Proposition 3: For an undirected graphical model $\mathbf{x}$ defined on a graph $G = (V, E)$, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the block-tree constructed using a root cluster V_1. The complexity of message passing over the block-tree is

$$O\left(\sum_{(i,j) \in \mathcal{E}} K^{|\overline{\mathcal{V}}_i|} + K^{|\overline{\mathcal{V}}_j|} + 2K^{|\mathcal{V}_i \backslash \overline{\mathcal{V}}_i| + |\mathcal{V}_j \backslash \overline{\mathcal{V}}_j|} \right) . \tag{7}$$

Proof: From (5), the complexity of passing messages from cluster $\mathcal{V}_i$ to cluster $\mathcal{V}_j$ is

$$O\left(K^{|\overline{\mathcal{V}}_i|} + K^{|\mathcal{V}_i \backslash \overline{\mathcal{V}}_i| + |\mathcal{V}_j \backslash \overline{\mathcal{V}}_j|} \right) \tag{8}$$

Summing over all edges and using the fact that we pass messages twice on an edge (i, j), we get the desired result. ■

We conclude this Section with some remarks regarding the belief propagation algorithm applied to block-trees.

Remark 5: An alternative way to derive the message passing algorithm over block-trees is to transform the block-tree into a junction-tree and then use popular junction-tree based inference algorithms [41], [42]. The complexity of the inference algorithm will be the same, however, by directly using the block-tree, we avoid the step of constructing a junction-tree.

Remark 6: We explain the importance of labeling edges in the block-tree for message passing. Consider the block-tree in Fig. 5(c). Let $\mathcal{V}_a = \{1\}$ and $\mathcal{V}_b = \{4, 8\}$. As mentioned before, the label between $\mathcal{V}_a$ and $\mathcal{V}_b$ is $(\{\}, \{8\})$. From this label, we conclude that the potential $\Psi_{a,b}$ between $\mathcal{V}_a$ and $\mathcal{V}_b$ will be a function of only 1 and 4 since 8 is not connected to 1. The message from $\mathcal{V}_a$ to $\mathcal{V}_b$ will have the form

$$m_{a \to b}(x_4) = \sum_{x_1} \Psi_{a,b}(x_1, x_4) \, . \tag{9}$$

The complexity of passing the above message is $O(K^2)$. Now suppose we do not label the edge. In this case, the algorithm will assume that the potential between $\mathcal{V}_a$ and $\mathcal{V}_b$ is a function of 1, 4, and 8. The message from $\mathcal{V}_a$ to $\mathcal{V}_b$ will have the form

$$m_{a\to b}(x_4, x_8) = \sum_{x_1} \Psi_{a,b}(x_1, x_4, x_8)\,. \tag{10}$$

The complexity of passing the above message is $O(K^3)$. Note that $\Psi_{a,b}(x_1, x_4, x_8 = \theta_1) = \Psi_{a,b}(x_1, x_4, x_8 = \theta_2)$ for all $\theta_1, \theta_2 \in \Omega$, however, this information is not provided in the message passing algorithm since the label is not specified.

Remark 7: In light of Remark 6, an upper bound on the message passing complexity is $O\left(\sum_{(i,j)\in\mathcal{E}} K^{|\mathcal{V}_i|+|\mathcal{V}_j|}\right)$, which is obtained by passing messages without labeling the edges in the block-tree.

B. Optimal Block-Trees

From Proposition 3, we see that the complexity of message passing over block-trees is dominated by the term in (7) whose exponent is maximum. Thus, an optimal block-tree can be defined such that

$$w(\mathcal{G}, V_1) = \max_{(i,j)\in\mathcal{E}} \left\{ \max\left\{ |\overline{\mathcal{V}}_i|, |\overline{\mathcal{V}}_j|, |\mathcal{V}_i \backslash \overline{\mathcal{V}}_i| + |\mathcal{V}_j \backslash \overline{\mathcal{V}}_j| \right\} \right\} \tag{11}$$

is minimized. Notice that (11) depends on the choice of the root cluster V_1 in constructing the block-tree. 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 block-trees for the partial grid in Fig. 5. In Fig. 5(c), we construct a block-tree using $V_1 = \{7\}$ as the root cluster. The complexity of inference in this graph is $O(K^4)$ because of the message passing between the cluster $\{5, 9\}$ and $\{4, 8\}$ and between cluster $\{2, 6\}$ and $\{5, 9\}$. Instead of choosing $V_1 = \{7\}$, let $V_1 = \{7, 4\}$. The initial estimate of the clusters are shown in Fig. 7(a). The final block-tree is shown in Fig. 7(c). It is clear that message passing on the block-tree in Fig. 7(c) will have complexity $O(K^5)$ because of the message passing between the cluster $\{9, 6, 2\}$ and the cluster $\{8, 5\}$.

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 computationaly 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

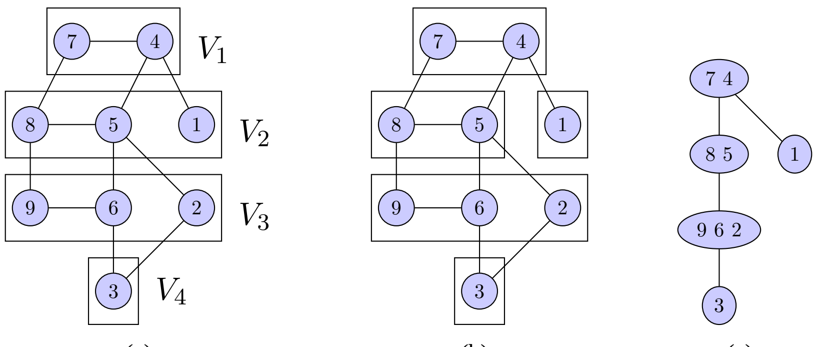


Fig. 7. (a) Original estimates of the clusters in the partial grid using $V_1 = \{7, 4\}$ as the root cluster. (b) Splitting of clusters. (c) Final block-tree.

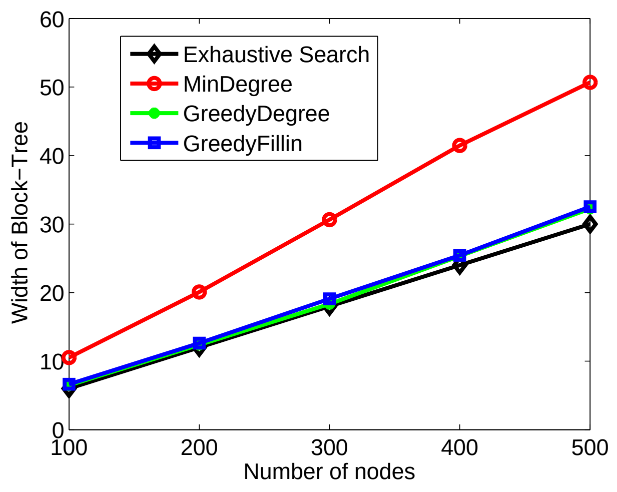


Fig. 8. Plot showing the performance of three different greedy heuristics for finding optimal block-trees.

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 we propose are based on the relationship between junction-trees and block-trees. Recall from Remark 4 that for every block-tree, we can find a junction-tree. This means that an optimal junction-tree may be used to find an approximate optimal block-tree. From Appendix A, we know that an optimal junction-tree can be found using an optimal elimination order. Thus, we make use of algorithms for finding optimal elimination orders to find optimal block-trees.

Using an elimination order - GreedyDegree: One of the simplist algorithms for finding an approximate optimal elimination order is known as GreedyDegree [45], [46], 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 the graph to find the cliques. We then search over a constant number of cliques to find a root cluster that leads to the minimal width as defined in (11).

Using an elimination order - GreedyFillin: Another popular greedy algorithm is to find an optimal elimination order such that at each step in the triangulation algorithm (see Appendix A), we choose a node that adds a minimal number of extra edges in the graph. This is known as GreedyFillin [47] 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. 8. 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 will be at most $2k$. Fig. 8 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 is that we can use various known algorithms for finding optimal junction-trees to find optimal block-trees.

D. Block-Trees vs. Junction-Trees

In this Section, we show how block-trees can lead to computational savings when used for inference over graphical models. The main idea is that for certain graphical models, the complexity of constructing the junction-tree can dominate the complexity of the inference algorithm, thus in these cases, we can use block-trees instead of junction-trees for inference.

1) Example: Single Cycle Graphical Models: Let $G = (V, E)$ be a graphical model on a graph with a single cycle, as in Fig. 3(a). The complexity of inference using a junction-tree is

$$O\left(n + n^2 + 2nK^3\right) \approx O\left(n^2 + 2nK^3\right) \,, \tag{12}$$

where $O(n + n^2)$ is the complexity of constructing the junction-tree and $O(nK^3)$ is the complexity of belief propagation (see Proposition 1) since all the cliques in the junction-tree will have size three. The

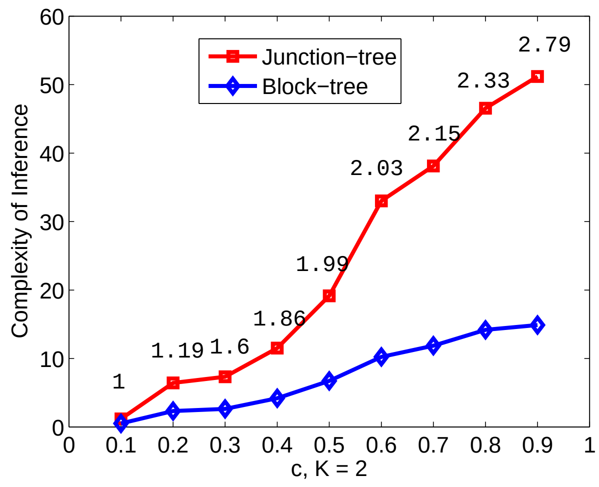


Fig. 9. Comparing the complexity of inference using junction-trees vs. using block-trees for graphs sampled using an Erdős-Rényi model. For each c, we have denoted the average treewidth over 100 randomly generated graphs.

complexity of inference using a block-tree is

$$O\left(n + nK^4\right) \approx O\left(nK^4\right) \,, \tag{13}$$

where $O(n)$ is the complexity of constructing the block-tree (see Proposition 2) and $O(nK^4)$ is the approximate complexity of inference over the block-tree (see Fig. 3(c) for an example). It is clear that there exists some constant $c > 0$ such that if $n > cK^4$, the complexity of inference using the junction-tree will dominate the complexity of inference using the block-tree. Recall that K is the number of states each random variable can take, so this number remains fixed as n increases.

2) Example: Erdős-Rényi Graph: In this Section, we assume that the graph $G = (V, E)$ is a realization of an Erdős-Rényi [48] graph, denoted by $G(n, c/n)$. Here n is the number of nodes and c/n is the probability that an edge appears in the graph (independent of all other edges). Many naturally occurring networks, such as large social networks, may be modeled using Erdős Rényi graphs [49].

It is well known that for $c < 1$ and for *large* n, the treewidth (see Definition 4) of a graph realized using an Erdős-Rényi model is at most two [48]. Thus, for graphical models defined on such random graphs, inference is tractable. The comparison of the complexity of inference using block-trees and junction-trees on a 1000 node graph is shown in Fig. 9. The results shown are over 100 randomly generated graphs for c chosen between 0 and 1. The average treewidth over the 100 graphs is also shown in the graph. We clearly see the benefits of using block-trees over junction-trees. Note that, when reporting complexity

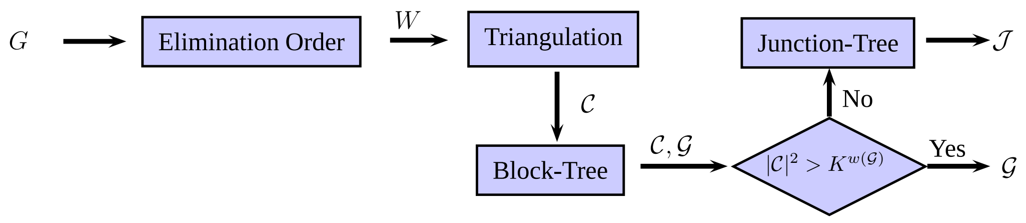


Fig. 10. Algorithm to choose between a block-tree and a junction-tree for inference over graphical models.

results, we factor in the complexity of constructing the junction-tree or the block-tree, in addition to the complexity of performing message passing in the graph. For $c > 1$, the graph realized using an Erdős-Rényi model no longer has low treewidth and the treewidth is known to be of the order $c'n$, where c' is some constant and n is the number of nodes in the graph [50]. Thus, in this case, the complexity of inference will dominate the complexity of constructing the junction-tree, so the block-tree framework will no longer be faster.

3) Discussion: From the examples presented in this Section, we see that, for a certain class of graphical models, the complexity of using block-trees over junction-trees leads to complexity benefits when performing inference over graphical models. The main computational gains are in constructing the block-tree. We note that, for some applications, the junction-tree or the block-tree only needs to be computed once. In these cases, the complexity of inference will be dominated by the message passing algorithm. Thus, the main advantage of the block-tree framework is in studying inference problems where the graph structure varies over time and the inference algorithm needs to compute a new tree-decomposition at each time instant.

Fig. 10 outlines an algorithm for deciding between choosing the junction-tree or block-tree for inference. In the first step of the algorithm, we compute an elimination order from the graph. As outlined in Section IV-C, this can be done using standard algorithms in the literature to compute optimal elimination orders, see [51] for a review of such algorithms. Using the elimination order, we compute a triangulated graph and find the set of cliques $\mathcal{C}$. We use the set of cliques to find a block-tree as outlined in Section IV-C. From the block-tree, we determine if the complexity of constructing the junction-tree dominates the complexity of inference over the block-tree. If this is the case, we use the block-tree for inference, otherwise we construct the junction-tree.

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

In this Section, we show how block-trees can be used to derive approximate inference algorithms. Section V-A discusses our algorithm for splitting larger clusters in a block-tree to form a block-graph. Section V-B shows how approximate inference can be performed over block-graphs.

A. Constructing Block-Graphs

From Remark 7, we know that the complexity of message passing between two non-overlapping clusters $\mathcal{V}_i$ and $\mathcal{V}_j$ is at most exponential in $|\mathcal{V}_i| + |\mathcal{V}_j|$. Thus, when the size of the clusters in a block-tree is large, exact inference using block-trees is computationally intractable. From Proposition 1, we conclude the same about exact inference using junction-trees.

To reduce the computational complexity of message passing, we split larger clusters in the block-tree. The resulting graph over the new clusters will not be tree-structured, hence the BP algorithm can no longer be applied. We will discuss the loopy BP algorithm [3] in Section V-B for approximate inference over non tree-structured graphical models.

To determine the size of the clusters allowed, we assume a user defined maximal message passing complexity m so that the complexity of message passing between any two clusters is at most $O(K^m)$, where K is the number of states each random variable can take. In general, m will be chosen to be sufficiently small so that inference is tractable. We modify Algorithm 1 for constructing block-trees to construct block-graphs, a graph with non-overlapping clusters.

Step 1. Using an initial cluster of nodes V_1, find clusters $V_1, V_2, \dots, V_r$ using breadth-first search (BFS) such that $V_2 = \mathcal{N}(V_1), V_3 = \mathcal{N}(V_2) \backslash \{V_1 \cup V_2\}, \dots, V_r = \mathcal{N}(V_r) \backslash \{V_{r-2} \cup V_{r-1}\}$. 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. To ensure that the user defined maximal message passing complexity is satisfied, for k odd, we split the clusters in V_k so that each cluster has maximal cardinality $\lceil m/2 \rceil$ and for k even, we split clusters in V_k so that each cluster has maximal cardinality $\lfloor m/2 \rfloor$. This ensures that for all messages passed between clusters in different V_k, the message passing complexity is at most $O(K^m)$.

Step 3. For V_r, if there exists any cluster that has cardinality greater than $\lceil m/2 \rceil$ or $\lfloor m/2 \rfloor$, depending on whether r is odd or even, partition those components. Let $V_r = \{V_r^1, V_r^2, \dots, V_r^{m_r}\}$ be the final set clusters.

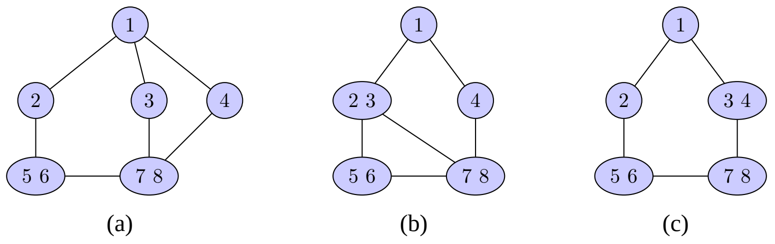


Fig. 11. Explaining Step 6 in the block-graph construction algorithm. Given the block-graph in (a), if we merge nodes 2 and 3, we get the block-graph in (b). If we merge nodes 3 and 4, we get the block-graph in (c). Notice that the block-graph in (c) has just one loop.

Step 4. We 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 $\lceil m/2 \rceil$ or $\lfloor m/2 \rfloor$, depending on whether k is odd or even.

Step 5. Partition all clusters in $\widetilde{V_k}$ into appropriate size clusters of size $\lceil m/2 \rceil$ or $\lfloor m/2 \rfloor$ and also ensuring that the message passing complexity between clusters created is at most $O(K^m)$.

Step 6. We now 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}, then 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 7. The block-graph is given by the clusters $\mathcal{V} = \bigcup_{k=1}^{r} \{V_k^1, V_k^2, \ldots, V_k^{m_k}\}$ and the set of edges $\mathcal{E}$ between clusters.

The key step in the above algorithm is Step 6, where we cluster nodes appropriately. Fig. 11 explains the intuition behind merging clusters with an example. Suppose, we use the block-graph construction algorithm up to Step 5 and now we want to merge clusters in $V_2 = \{2, 3, 4\}$. If we ignore Step 6 and merge clusters randomly, we might get the block-graph in Fig. 11(b) on merging nodes 2 and 3. If we use Step 6, then since nodes 3 and 4 are connected to the same node, we merge these to get the block-graph in Fig. 11(c). Notice that Fig. 11(c) is a graph with a single cycle with five edges, whereas Fig. 11(b) is a graph 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 [52]. Thus, our proposed algorithm leads to block-graphs that are favorable for inference.

B. Approximate Inference Using Block-Graphs

In this Section, we review the loopy belief propagation (LBP) algorithm [3] applied to block-graphs. The algorithm uses the same message passing updates as in the normal belief propagation (see Section IV-A) applied to block-trees, but ignores the fact that the graph is not tree-structured.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the block-graph constructed using the algorithm in Section V. Recall the notation adopted in Section IV-A for labeling edges in a block-tree. Just like in Step 2 in Section V, reparameterize the joint probability distribution as in (4). Let $m^t_{i\to j}(x_{\mathcal{V}_j\backslash\overline{\mathcal{V}}_j})$ denote the message passed from cluster $\mathcal{V}_i$ to $\mathcal{V}_j$. Assume the initial messages at $t = 0$ are unity, i.e., $m^0_{i\to j}(x_{\mathcal{V}_j\backslash\overline{\mathcal{V}}_j}) = 1$ for all $x_{\mathcal{V}_j\backslash\overline{\mathcal{V}}_j} \in \Omega^{|\mathcal{V}_j\backslash\overline{\mathcal{V}}_j|}$ and $(i, j) \in \mathcal{E}$. For each iteration, the message passing updates are given as

$$m^t_{i\to j}(x_{\mathcal{V}_j\backslash\overline{\mathcal{V}}_j}) = \sum_{x_{\mathcal{V}_i\backslash\overline{\mathcal{V}}_i}} \Psi_{i,j}(x_{\mathcal{V}_i\backslash\overline{\mathcal{V}}_i}, x_{\mathcal{V}_j\backslash\overline{\mathcal{V}}_j}) \sum_{x_{\overline{\mathcal{V}}_i}} \prod_{k\in\mathcal{N}(\mathcal{V}_i)\backslash\mathcal{V}_j} \Phi_i(x_{\mathcal{V}_i}) m^{t-1}_{k\to i}\,, \tag{14}$$

After T iterations, the joint distribution of each cluster $\mathcal{V}_i$ is proportional to the product of all incoming messages:

$$p_{\mathcal{V}_i}(x_{\mathcal{V}_i}) \propto \prod_{j\in\mathcal{N}(\mathcal{V}_i)} \Phi_i(x_{\mathcal{V}_i}) m^T_{j\to i} \tag{15}$$

To find the marginal distribution of each node, we marginalize the joint distribution $p_{\mathcal{V}_i}(x_{\mathcal{V}_i})$, which has complexity $O(|\mathcal{V}_i|K^{|\mathcal{V}_i|})$.

VI. Numerical Simulations

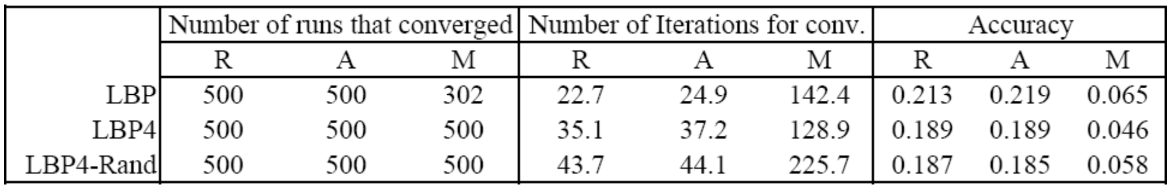
In this Section, we provide numerical simulations to show the performance gains when using block-graphs for inference over graphical models. We study inference over binary valued graphical models defined on grid graphs. For each random variable, let $x_s \in \{-1, +1\}$. We assume that each node has a potential given by $\phi_i(x_i) = \exp(-a_i x_i)$, where $a_i \sim \mathcal{N}(0, (0.25)^2)$ and the edge potentials are given by

$$\text{Repulsive: } \psi_{ij}(x_i, x_j) = \exp(-|b_{ij}|x_i x_j) \tag{16}$$

$$\text{Attractive: } \psi_{ij}(x_i, x_j) = \exp(|b_{ij}|x_i x_j) \tag{17}$$

$$\text{Mixed: } \psi_{ij}(x_i, x_j) = \exp(-b_{ij} x_i x_j)\,, \tag{18}$$

where $b_{ij} \sim \mathcal{N}(0, 1)$. 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



	Number of runs that converged			Number of Iterations for conv.			Accuracy		
	R	A	M	R	A	M	R	A	M
LBP	500	500	302	22.7	24.9	142.4	0.213	0.219	0.065
LBP4	500	500	500	35.1	37.2	128.9	0.189	0.189	0.046
LBP4-Rand	500	500	500	43.7	44.1	225.7	0.187	0.185	0.058

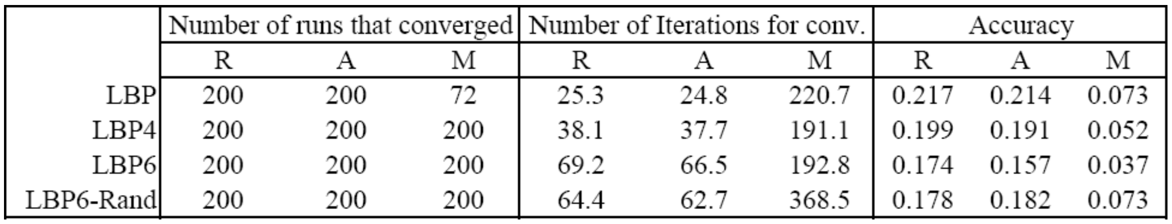
Fig. 12. Comparison of convergence and accuracy of using LBP on graphs vs. using LBP on block-graphs for a 7×7 grid graph. Results are over 500 runs. The iterations and accuracy results are when all the algorithms converge.

are attractive, whereas some are repulsive. We consider three types of graphs: 7×7 grid graphs, 8×8 grid graphs, and 30×30 grid graphs. We analyze the results of the numerical simulations as follows:

Notation: LBP refers to running loopy belief propagation on the original graph. LBPm refers to running LBP on a block-graph constructed using our proposed algorithm in Section V such that the maximal message passing complexity is $O(K^m)$. LBPm-Rand refers to using a modification of the algorithm in Section V so that instead of merging clusters in a structured manner using Step 5 and Step 6, we instead merge clusters randomly. Note that each iteration of the LBP algorithm has complexity $O(2|E|K^2)$. On the other hand, running LBP on block-graphs has higher complexity (see Proposition 2). To objectively compare results on the number of iterations, we rescale the values. Thus, if the LBP on a block-graph converges in T iterations, we rescale this number by $T \cdot C'/C$, where C' is the complexity of inference using the block-graph and C is the complexity of using just the graph. We declare convergence if the absolute difference between the messages passed in successive iterations is less than a threshold of $\epsilon = 10^{-4}$. If the algorithm does not converge in 3000 iterations, we say the algorithm failed to converge. Given the true marginal distribution $p_s(x_s)$ of each node and an estimate $\widehat{p}_s(x_s)$, the accuracy is measured by

$$\text{Accuracy} = \frac{1}{2|V|} \sum_{x_s \in \{-1,+1\}} \left| p_s(x_s) - \widehat{p}_s(x_s) \right| . \qquad (19)$$

7×7 **grid graph:** Fig. 12 shows results of running LBP on block-graphs and running LBP on the original 7×7 grid graph. The first three columns report the number of times the algorithm converged out of 500 runs. LBP using mixed potentials converged only 60% of the time, whereas LBP4 and LBP4-Rand converged on all 500 runs. This suggests that mixed potentials cause Ising models to sometimes not converge. Comparing the mean number of iterations (which are rescaled) required for convergence, we note that for both attractive and repulsive potentials, LBP converges faster. However, for mixed potentials, block-graphs converge faster. Comparing LBP4 and LBP4-Rand, we see the benefits of merging clusters



	Number of runs that converged			Number of Iterations for conv.			Accuracy		
	R	A	M	R	A	M	R	A	M
LBP	200	200	72	25.3	24.8	220.7	0.217	0.214	0.073
LBP4	200	200	200	38.1	37.7	191.1	0.199	0.191	0.052
LBP6	200	200	200	69.2	66.5	192.8	0.174	0.157	0.037
LBP6-Rand	200	200	200	64.4	62.7	368.5	0.178	0.182	0.073

Fig. 13. Comparison of convergence and accuracy of using LBP on graphs vs. using LBP on block-graphs for a 8×8 grid graph. Results are over 200 runs. The iterations and accuracy results are when all the algorithms converge.

in a structured manner rather than merging clusters randomly since LBP4 converges faster than LBP4-Rand. For mixed potentials, the convergence speed is almost twice as that of LBP4-Rand. Comparing the accuracy, we see that using block-graphs leads to smaller errors. Comparing LBP4 and LBP4-Rand, for repulsive and attractive potentials there is almost no difference between the error, however, for mixed potentials, LBP4 has lower error.

8×8 **grid graph:** Fig. 12 shows results of running LBP on block-graphs and running LBP on the original 8×8 grid graph. The interpretation of the results are similar to that of the results for the 7×7 grid graph. We notice that increasing the message passing complexity to 6, compared to 4, improves the estimates of the node potentials at the cost of higher computational cost. Comparing LBP6-Rand to LBP4, we notice that for mixed potentials LBP4 performs better suggesting and LBP6-Rand performs as good as LBP.

30×30 **grid graph:** Fig. 12 shows results of running LBP on block-graphs and running LBP on the original 30×30 grid graph. In this case, over 100 runs, the LBP failed to converge in 3000 iterations. The interpretation of the results is the same as before. We note that to compare the quality of the estimates, we did not have the true estimates available since it is computationally intractable to find these since the graph has high treewidth. Instead, we ran LBP on a block-graph with maximal message passing complexity $O(K^{10})$ and used these estimates as the ground truth. Note that, even though LBP did not get converge in 3000 iterations, the quality of the estimate is almost the same as that of using LBP8-Rand.

From the discussion above, it is clear that using LBP on block-graphs leads to more accurate estimates of the marginal distributions. For repulsive and attractive node potentials, we observed that higher accuracy comes at the cost of more computations. For mixed potentials, we observed that using block-graphs was more accurate and faster than using the original graph. Further, we evaluated our proposed algorithm for constructing block-graphs and noticed that our proposed algorithm leads to more accurate estimates

	Number of runs that converged			Number of Iterations for conv.			Accuracy*		
	R	A	M	R	A	M	R	A	M
LBP	100	100	0	86.85	83.74	-	0.232	0.243	0.114
LBP4	100	100	100	126.6	120.2	1971.8	0.157	0.181	0.078
LBP8	100	100	100	454.1	512.2	7769.9	0.083	0.068	0.059
LBP8-Rand	100	100	100	493.9	491.5	5574	0.258	0.257	0.101

Fig. 14. Comparison of convergence and accuracy of using LBP on graphs vs. using LBP on block-graphs for a 30×30 grid graph. Results are over 100 runs. Since finding the true node potentials is computationally intractable, we use a higher order block-graph to estimate the ground truth.

when compared to an approach of randomly selecting clusters. This shows the advantage of using the block-tree as an initial estimate of the block-graph.

VII. Summary

We have built a framework for performing inference, i.e., computing marginal distributions, over graphical models using non-overlapping clusters. Our main contribution is in proposing an efficient algorithm for finding clusters in a graph to generalize various inference algorithms. We first derived a linear time algorithm for mapping an arbitrary graph into a tree-structured graph over non-overlapping clusters. We call this graph a block-tree. Belief propagation (BP) on block-trees lead to an alternative to the junction-tree algorithm for performing exact inference over arbitrary graphical models. We showed that constructing block-trees is faster than constructing junction-trees and identified domains where using a block-tree framework for inference is more suitable than using a junction-tree framework.

Next, we considered the problem of generalizing approximate inference algorithms. We modified the block-tree construction algorithm so that larger clusters are appropriately split so that the message passing complexity can be reduced. We call the resulting graph over non-overlapping clusters a block-graph. Our proposed algorithm for finding block-graphs tries to minimize the number of cycles in the original graph since it is known that performing approximate inference on graphs that are approximately tree like is more accurate. Using numerical simulations, we showed how loopy belief propagation (LBP) on block-graphs lead to more accurate inference algorithms than when performing LBP on the original graph. Further, in some cases, we showed that performing LBP on block-graphs was computationally more efficient. Although we only showed how BP and LBP can be generalized, as shown in Fig. 1, other inference algorithms can be easily generalized using the framework of choosing non-overlapping clusters in a graph. In fact, the block-graph framework can also be used to generalize maximum a posteriori

(MAP) inference algorithms such as [53]–[55], where the objective is to find a $\widehat{\mathbf{x}}$ that maximizes the joint probability distribution $p(\mathbf{x})$.

APPENDIX A
CONSTRUCTING JUNCTION-TREES

Before reviewing the algorithm for constructing junction-trees, we first review some standard graph theoretic definitions. A triangulated graph, also known as a chordal graph, is defined as follows [40]:

Definition 3 (Triangulated graph): A graph is triangulated if all cycles of length four or more have an edge connecting non-adjacent nodes in the cycle.

Given any graph G, we can triangulate it using the following steps [56].

Step 1: Choose an *elimination order*, which is a permutation of nodes in the graph.

Step 2: For each node in the elimination order, form an edge between all the nodes in its neighbors and update the edge set.

Step 3: Remove the node and all its edges to other nodes and repeat step 2 until we exhaust all the nodes in the elimination order.

The resulting graph after steps 1-3 will be triangulated. Using the triangulated graph, width and treewidth of a graph are defined as follows [39].

Definition 4 (Width and Treewidth): Width of a triangulated graph is the size of the largest clique minus 1. Treewidth, $\text{tw}(G)$, of a graph G is the minimum width among all possible triangulations of the graph G.

An algorithm to construct a junction-tree is outlined in Algorithm 2. In summary, we first triangulate a graph using an elimination order, then find the cliques in the triangulated graph, and then use a maximum weight spanning tree (MWST) algorithm to find an appropriate tree over the cliques. Note that from Proposition 1, we see that the complexity of inference over junction-trees is exponential in the number of nodes in the largest clique. From Definition 4, this means that the complexity of inference is exponential in the width of the triangulated graph. Thus, an *optimal junction-tree* corresponds to choosing an elimination order in the triangulation algorithm leading to the minimum width. The elimination order leading to an optimal junction-tree is called an *optimal elimination order*.

To illustrate Algorithm 2, we consider finding the junction-tree for the grid graph in Fig. 15(a). A triangulated graph, constructed using the elimination order $\{1,3,7,9,4,5,2,6,8\}$, is shown in Fig. 15(b). It can be shown that the elimination order chosen is optimal. The weighted graph over cliques is shown in Fig. 15(c). The junction tree constructed using the triangulated graph is shown in Fig. 15(d). The width

Algorithm 2: Constructing a Junction-Tree: JunctionTree(G,W)

```
  Data: A graph $G = (V, E)$ and an elimination order $W$.
  Result: A junction-tree $\mathcal{J} = (\mathcal{C}, \mathcal{E})$
1 Triangulate the graph $G$ using the elimination order $W$.
2 Identify the set of cliques $\mathcal{C}$ in the triangulated graph.
3 For all tuples $C_i, C_j \in \mathcal{C}$, construct a weighted graph with weights $w(i,j) = |C_i \cap C_j|$.
4 $\mathcal{E} \leftarrow \arg\max_{\mathcal{E}} \sum_{(i,j)\in\mathcal{E}} w(i,j)$.
```

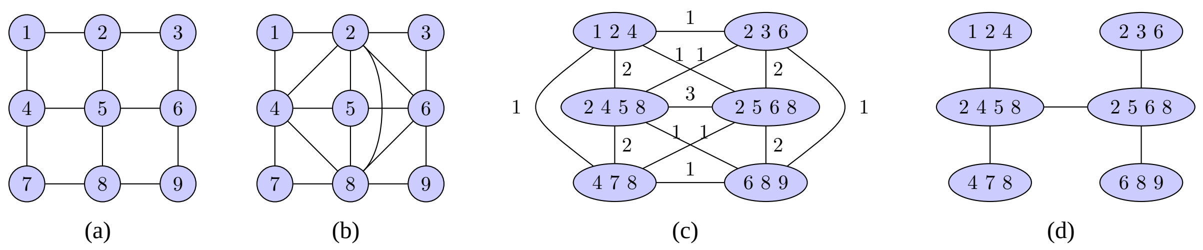


Fig. 15. (a) A grid graph (b) Triangulated graph of (a). (c) Weighted graph over cliques. (d) Junction-tree

of the triangulated graph in Fig. 15(c) is three. Since the elimination order is optimal, the treewidth of the graph in Fig. 15(a) is also three.

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

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

[5] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Trans. Inf. Theory*, vol. 45, no. 9, pp. 1120–1146, May 2003.

[6] E. B. Sudderth, M. J. Wainwright, and A. S. Willsky, "Embedded trees: estimation of Gaussian processes on graphs with cycles," *IEEE Trans. Signal Process.*, vol. 52, no. 11, pp. 3136–3150, Nov. 2004.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] M. J. Wainwright, “Stochastic processes on graphs: Geometric and variational approaches,” Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002.

[20] 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.

[21] 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.

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

[23] 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

[24] ——, “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

[25] 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.

[26] J. Friedman, T. Hastie, and R. Tibshirani, “Applications of the lasso and grouped lasso to the estimation of sparse graphical models,” pp. 1–22, 2010. [Online]. Available: http://www-stat.stanford.edu/~tibs/ftp/ggraph.pdf

[27] 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.

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

[29] 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.

[30] 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.

[31] D. Vats, "Tree-structured like representations for continuous and graph indexed Markov random fields," Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 2011.

[32] 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.

[33] 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.

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

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

[36] 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.

[37] 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.

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

[39] 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.

[40] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2000.

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

[42] F. V. Jenson, S. L. Lauritzen, and K. G. Oleson, "Bayesian updating in causal probabilistic networks by local computation," *Computational Statistics Quarterly*, vol. 4, no. 4, pp. 269–282, 1990.

[43] 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.

[44] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks (research note)," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 393–405, 1990.

[45] 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. [Online]. Available: http://www.jstor.org/stable/2627454

[46] 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.

[47] 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.

[48] P. Erdős and A. Rényi, "On the evolution of random graphs," in *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 1960, pp. 17–61.

[49] M. O. Jackson, *Social and Economic Networks*. Princeton University Press, 2008.

[50] C. Lee, J. Lee, and S. il Oum, "Rank-width of random graphs," *arXiv:1001.0461v1*, 2010.

[51] H. L. Bodlaender and A. M. Koster, “Treewidth computations I. upper bounds,” *Information and Computation*, vol. 208, no. 3, pp. 259 – 275, 2010.

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

[53] 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.

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

[55] 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.

[56] A. Becker and D. Geiger, “A sufficiently fast algorithm for finding close to optimal junction trees,” in *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 1996, pp. 81–89.