Compressed Sensing for Big Data over Complex Networks

Saeed Basirian Jahromi

Aalto University
School of Science
Master's program in machine learning and data mining

Thesis Supervisor: Prof. Alexander Jung
Espoo, January 2018

AALTO UNIVERSITY
School of Science
Master's program in Machine Learning and Data Mining

ABSTRACT OF THE
MASTER'S THESIS

Author: Saeed Basirian Jahromi

Title of the thesis: Compressed Sensing for Big Data Over Complex Networks

Number of pages: 39 + 7

Date: 27.11.2017

Language: English

Supervisor and advisor: Prof. Alexander Jung

Transductive semi-supervised learning methods aim at automatically labeling large datasets by leveraging information provided by few manually labeled data points and the intrinsic structure of the dataset. Many such methods based on a graph signal representation of a dataset have been proposed, in which the nodes correspond to the data points, the edges connect similar points, and the graph signal is the mapping between the nodes and the labels. Most of the existing methods use deterministic signal models and try to recover the graph signal using a regularized or constrained convex optimization approach, where the regularization/constraint term enforce some sort of smoothness of the graph signal. This thesis takes a different route and investigates a probabilistic graphical modeling approach in which the graph signal is considered a Markov random field defined over the underlying network structure. The measurement process, modeling the initial manually obtained labels, and smoothness assumptions are imposed by a probability distribution defined over the Markov network corresponding to the data graph. Various approximate inference methods such as loopy belief propagation and the mean field methods are studied by means of numerical experiments involving both synthetic and real-world datasets.

Keywords: Semi-supervised learning, graph signal learning, probabilistic graphical models, approximate inference, loopy belief propagation, complex networks

Contents

Acknowledgments

I would like to first thank my supervisor, professor Alexander Jung, for all his support since I started working as a research assistant for the Machine Learning for Big Data group in the fall of 2016. His true passion and hard work has been an inspiration for me and he has presented me with many opportunities to grow and expand my horizons. I would also like to thank my family for their support throughout all of my studies. Finally, I have to thank Mr. Alexandru Mara for his help and suggestions for implementation of the LFR model presented in this thesis.

1 Introduction

The progress of the digital age has resulted in a fast-paced generation of massive datasets with a wide variety and complexity in nature. This includes the typical media we consume daily, such as images and videos, to the data gathered by the millions of different websites on the Internet, especially the social media, to medical datasets which contain patients genetic information, and much more. However, despite the concurrent growth in computing power and resources, there is still a clear need for methods to efficiently process and extract information from these massive, heterogeneous datasets. In this regard, one avenue of research that has attracted considerable attention in recent years is based on the intuition that many datasets have an inherent network structure to them. This idea can be readily seen in datasets that are based on a physical network, such as measurements in a sensor network, but it can be easily extended to others as well; The only requirement is that there exists a similarity measure between the data points, so that if they are represented as nodes in a network, the nodes are connected to each other based on this measure.

This simple but powerful idea has formed the backbone of a recently established branch in signal processing, aptly called "graph signal processing", in which researchers have tried to extend the classical theory and methods of signal processing to signals defined over graphs [32, 30]. On the same note, utilizing a graph representation of datasets has also led to a variety of algorithms in the machine learning community, especially for the task of semi-supervised learning, which is the focus of this work [37, 5, 20, 35]. We can consider semi-supervised learning to lie between unsupervised and supervised learning, in the sense that in semi-supervised learning, only a few of the data points available for training the learning algorithm are labeled. Thus, the algorithm should attempt to also utilize the information contained in the provided unlabeled points to learn the mapping between the features of the data points and the labels, and in turn be able to predict the label of a new point. The recent interest in semi-supervised learning is a natural response to one of the challenges of dealing with the aforementioned massive and heterogeneous datasets: the difficulty of labeling them, since labeling such datasets is costly (and in some cases maybe even impossible) and is also subject to errors.

A graph representation of a data set also allows utilizing one of the powerful tools widely adopted in machine learning, namely probabilistic graphical models (PGM). A PGM combines one of the main fundamental theories in machine learning, probability theory, with graph theory. It is based on defining the data points as random variables over the nodes of a graph, and imposing a probability distribution over these variables, for inference, decoding, or sampling over values of a subset or all of the variables. PGM is a relatively well-developed theory, with a variety of different models, algorithms, and software packages, and it has been employed in various applications such as image processing [24], bio-informatics [15], and more.

Whereas most recent research in this area has focused on deterministic methods based on convex optimization, the major goal of this work is investigating the use of a PGM for transductive (as opposed to inductive) semi-supervised learning of a continuous graph signal (i.e., a graph representation of a dataset with continuous valued labels), empirically identifying the best-performing approximate inference methods for graphical models and comparing them to deterministic methods used for this task. In transductive semi-supervised learning, we are only interested in recovering the labels of the given unlabeled data points and do not attempt to predict labels of points outside the available dataset. One can also draw a natural connection between transductive semi-supervised learning and another recently developed field in signal processing: compressed sensing. In a nutshell, compressed sensing is based on the idea that, provided the signal is sparse (or smooth) enough

and the measurements meet certain conditions, we are able to recover the signal using a number of (possibly noisy) samples far fewer than that required by the Nyquist frequency, by minimizing the ℓ_1 norm of the recovered signal. This discovery has led to a a wide variety of algorithms for signal recovery. Despite this success, little has been done to expand the theory of compressive sensing to massive datasets with an underlying network structure.

There have been attempts in the compressed sensing community to exploit or induce structured sparsity of a signal via a probabilistic framework. In structured sparsity, we generally assume that there is a specific structure underlying the sparse values or coefficients in the signal, for example, the sparse values are partitioned into groups or that we can represent the support of the signal (i.e., the set of non-zero values or coefficients) by a graphical model over a set of latent (hidden) variables. A prominent example of the latter is presented in [10], in which the authors impose an Ising model (which is a simple form of pair-wise Markov random field discussed in Section 3) over the lattice graph representing the support of the signal. As an example which can be considered the combination of these two approaches, [19] proposes using spike-and-slab priors for enforcing the sparsity pattern over each group of regression coefficients in a linear regression problem (which can be considered analogous to the compressed sensing problem) and learn the coefficients using expectation propagation. In this approach, a binary latent variable which follows a Bernoulli distribution is assigned to each group, such that if its value is zero, the coefficients belonging to the group are zero (i.e., they follow the spike distribution centered at zero), otherwise they follow a Gaussian distribution. In a related work, [2] proposes using a Gaussian distribution on the hyper-parameter of the Bernoulli distribution underlying the latent variables of the previous approach, in effect specifying the sparsity pattern via the mean and covariance function of this prior. In a more application-oriented work, in [28], the authors also utilize a pair-wise Markov random field (MRF) with custom potentials for the latent variables specifying the sparsity pattern of coefficients of the wavelet transform of an MRI image, in order to recover the image using a few noisy samples.

In contrast to these works, instead of defining latent variables which encode the sparsity pattern of the signal, the present work utilizes the underlying graph representation of the signal towards this end. To the best of the author's knowledge, the closest works to the present one are those investigating image in-painting (also referred to as image reconstruction on occasion) using Markov random fields (MRF) [24, 36], and a work by Zhu and Ghahramani [38] which also investigates MRFs for semi-supervised learning. In the first case, although image in-painting is a form of graph signal learning (or equivalently transductive semi-supervised learning), it is obviously limited to the grid graph defined over the pixels and discrete labels. In the latter, a MRF-based model is proposed for semi-supervised classification, and unlike that work we assume we are given a graph representation of the dataset.

Outline

The rest of this thesis is organized as follows. In Section 2 we formalize the problem of learning smooth graph signals, introduce (possibly) the two most popular measures for quantifying the signal smoothness, and finally present a brief overview of learning methods based on these smoothness measures, with a few examples. In Section 3, we give a quick review of probabilistic graphical models and their properties (with the emphasis on undirected models) and discusses a few of the most well-known methods for performing inference and sampling in such models. Section 4 presents a graphical model for graph signal recovery, with a brief discussion of two possible smoothness priors

and how to optimize the parameters of the model. Section 5 presents the numerical experiments on a few synthetic graphs and one real world dataset. We conclude and give an outlook for possible follow-up work in Section 6.

Notation

The most important notation used throughout this thesis is listed here for reference.

x	: a scalar variable
x	: a scalar random variable
$\boldsymbol{x}$	: a vector variable
$\mathbf{x}$	: a vector-valued random variable
x_i	: element i of vector $\boldsymbol{x}$
$\boldsymbol{A}$	: a matrix
$A_{i,j}$	: element (i, j) of matrix $\boldsymbol{A}$
$A_{i,:}$	: row i of matrix $\boldsymbol{A}$
$A_{:,j}$	: column j of matrix $\boldsymbol{A}$
$p(\mathbf{x})$	: probability distribution of $\mathbf{x}$
$p(\boldsymbol{x})$	: $p(\mathbf{x} = \boldsymbol{x})$
$\|\boldsymbol{x}\|_p$	: ℓ_p norm of $\boldsymbol{x}$
$\|\mathcal{X}\|$	: the cardinality (i.e., size) of set $\mathcal{X}$
$[x, y]$	: the closed (i.e., inclusive) interval between x and y
$\mathcal{N}(\mathrm{x}, \mu, \sigma)$	: the Gaussian distribution defined on x with mean μ and standard deviation σ
$\Phi(\mathrm{x}, \mu, \sigma)$	: the Gaussian cumulative distribution defined on x with mean μ and standard deviation σ

2 Learning Graph Signals

In this section, first, the graph signal representation of a dataset and the graph signal learning problem is described informally, with a brief detour to present the stochastic block model as a popular synthetic model for clustered graph signals. Next, two popular measures for quantifying the graph signal smoothness, namely total variation and the graph Laplacian quadratic form are discussed. Finally, the general framework of learning graph signals using these smoothness measures in a regularized, deterministic and convex-optimization (as opposed to probabilistic) fashion is discussed.

2.1 Graph signal representation and learning problem

Assume we have a dataset $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$, where $\boldsymbol{x}^{(i)}$ is the features of i'th data point (which can be of any nature, e.g., vectors belonging to $\mathbb{R}^n$ or bag of words representations of documents), and $y^{(i)}$ is the real-valued scalar label associated with the data point (although we can assume that the labels are vectors too). We also assume that the dataset can be represented by an undirected data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, where each node $v_i \in \mathcal{V}$ represents the i'th data point and the real valued edge weight $W_{i,j}$ signifies the strength of the relationship between points i and j (and is therefore 0 if nodes i and j are not connected). In some cases, the edges and the edge weights in the data graph are readily available from the data set or the application. For example, we might have a dataset in which for each data point, the features are some properties of a city such as median income, unemployment rate, among others, and the labels are the average commercial goods prices in that city. In this case, the edges might represent the existence of roads between the cities and hence the edge weights can simply be the length of the roads. However, even if we do not directly have access to the edges and the weights as in this case, we can still assign them, provided we have a similarity measure between the points. For example, we can use a Gaussian kernel

$$W_{i,j} = \begin{cases} \exp(\frac{-\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2}{2\sigma^2}) \text{ if } & \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2 \leq \tau \\ 0 & \text{otherwise,} \end{cases}$$

where $\|.\|_2$ denotes the ℓ_2 norm of a vector and σ, τ are parameters.

In either case, after setting up the data graph, we can define the graph signal. Specifically, the labels of the data points induce a graph signal over the nodes of the data graph, i.e., the graph signal would be $\boldsymbol{y} : \mathcal{V} \to \mathbb{R}^{\mathcal{V}}$, such that $y_i = y^{(i)}$. We also assume the graph signal is smooth, i.e., the labels of tightly connected nodes are close to each other. This is in line with the smoothness hypothesis of (semi-)supervised machine learning, i.e., similar inputs result in similar outputs. For learning or recovering the graph signal, we assume we are given a few noisy measurements $z_j = y_{m_j} + n_j$, taken from a small sampling set $\mathcal{M} = \{m_1, \ldots, m_M\} \subset \mathcal{V}$, such that $M = |\mathcal{M}| \ll N$. Therefore, the goal in graph signal recovery is to find $\boldsymbol{y}$ using $\boldsymbol{z}$ and the data graph, based on the smoothness assumption.

2.2 Stochastic Block Model

While many real world datasets can be represented as graph signals, synthetic graph signals can also be useful for testing a particular learning algorithm. A simple general model for smooth graph signals is based on assuming a clustered form for the signal, i.e., the data graph is composed of clusters and the labels of nodes belonging a cluster are similar. Loosely speaking, a cluster is a

subset of nodes in the data graph that are closely connected to each other. There are several mathematical definitions for measuring how tightly a subset of nodes in a graph are connected, such as the clustering coefficient, and various algorithms for detecting clusters or communities [26].

The important point here is that a clustering assumption readily lends itself to constructing a synthetic smooth graph signal, since we can easily use any model to generate a clustered random graph, and assign similar signal values to nodes belonging to each cluster. One of the most popular models for generating a random graph is the stochastic block model. In a stochastic block model, we have a partition of the nodes into disjoint clusters, $\mathcal{P} = \{\mathcal{C}_1, \dots, \mathcal{C}_r\}$, and a matrix $\boldsymbol{P} \in [0,1]^{r\times r}$ specifying the edge probabilities. Specifically, $P_{i,i}$ is the probability of an edge existing between two nodes in cluster i, and $P_{i,j}$ for $j \neq i$ is the probability that an edge exists between two nodes belonging to clusters i and j. A simplifying assumption often used is to set $P_{i,i} = p$ for $i = 1, \dots, r$, and $P_{i,j} = q$ for $i, j = 1, \dots, r,\ j \neq i$. Once we generate a stochastic block model, we can simply assign similar (or identical) random values to all the nodes belonging to each cluster to construct a smooth graph signal.

2.3 Quantifying the Graph Signal Smoothness

Since the main assumption behind learning a graph signal is that it is smooth, it is easy to see that the choice of the algorithm for graph signal recovery to a large extent depends on how we measure the smoothness of the signal. A natural way to quantify the graph signal smoothness is to extend the concepts of derivative and gradient (which are used for quantifying smoothness of ordinary continuous signals) to the graph signal setting. In this regard, for a *directed* data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ we can define the edge derivative of signal $\boldsymbol{y}$ with respect to edge $e = (i, j)$ at vertex i as

$$\frac{\partial \boldsymbol{y}}{\partial e}|_i := \sqrt{W_{i,j}}(y_j - y_i).$$

Based on this, we define the gradient of $\boldsymbol{y}$ at vertex i as the vector

$$\nabla_i \boldsymbol{y} := [\{\frac{\partial \boldsymbol{y}}{\partial e}|_i\}_{e\in\mathcal{E} \text{ s.t. } e=(i,j) \text{ for some } j\in\mathcal{V}}].$$

The discrete p-Dirichlet form of $\boldsymbol{y}$ is

$$D_p(\boldsymbol{y}) := \frac{1}{p}\sum_{i\in\mathcal{V}} \|\nabla_i \boldsymbol{y}\|_2^p. \tag{1}$$

2.3.1 Total variation

$D_1(\boldsymbol{y})$ is known as total variation (TV) of signal $\boldsymbol{y}$, which is one of the most commonly used smoothness measures. However, this can be thought of as the ℓ_{2-1} norm definition of TV, as sometimes it is defined simply as

$$\|\boldsymbol{y}\|_{\text{TV}} := \sum_{(i,j)\in\mathcal{E}} W_{i,j}|y_j - y_i|. \tag{2}$$

It is worth noting that both the ℓ_{2-1} norm and definition (2) are not norms, but semi-norms. This is because they do not have the "definite" property of norms, i.e., it is possible that $\|y\|_{\text{TV}} = 0$ but $\boldsymbol{y} \neq 0$, and this happens whenever $\boldsymbol{y}$ is constant across all vertices.

We can define the edge derivative and also restate definition (2) in a simpler way for an undirected graph by defining a directed graph based on the undirected graph. Specifically, for an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ we can define a directed graph $\vec{\mathcal{G}} = (\mathcal{V}, \vec{\mathcal{E}}, \boldsymbol{W})$ by imposing an orientation on the edges, i.e., by considering for each edge $e = \{i, j\} \in \mathcal{E}$ one node as the head e^+ and the other node as the tail e^-, so that the corresponding edge in the directed graph $\vec{e} \in \vec{\mathcal{E}}$ is (e^+, e^-). Subsequently, for the undirected graph we can define the edge derivative with respect to edge $e = \{i, j\}$ at vertex i as

$$\frac{\partial \boldsymbol{y}}{\partial e}|_i := \frac{\partial \boldsymbol{y}}{\partial \vec{e}}|_i,$$

although this ultimately does not affect definition (1). Defining the directed graph for an undirected one also allows us to define a useful matrix known as the graph incidence matrix. For a directed graph $\vec{\mathcal{G}} = (\mathcal{V}, \vec{\mathcal{E}}, \boldsymbol{W})$ based on the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, we define the incidence matrix $\boldsymbol{M} \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{V}|}$ as

$$M_{e,i} = \begin{cases} 1 & \text{if } i = e^+ \\ -1 & \text{if } i = e^- \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Using the incidence matrix, we can rewrite equation (2) as

$$\|\boldsymbol{y}\|_{\text{TV}} = \|\boldsymbol{M}\boldsymbol{y}\|_1. \tag{4}$$

This allows us to see immediately that $\|\boldsymbol{y}\|_{\text{TV}}$ is a convex function, which is extremely important for the performance of the resulting optimization algorithms.

2.3.2 Graph Laplacian quadratic form

Replacing $p = 2$ in equation (1), we get $D_2(\boldsymbol{y})$ which is known as the graph Laplacian quadratic form, another widely used measure for graph signal smoothness. This name comes from the (unnormalized) graph Laplacian matrix $\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{W}$, where $\boldsymbol{D}$ is a diagonal matrix whose ith element is the strength of node i, i.e., $\boldsymbol{D}_{i,i} = \sum_{\{i,j\} \in \mathcal{E}} \boldsymbol{W}_{i,j}$. It is easy to show $\boldsymbol{L} = \boldsymbol{M}^T \boldsymbol{M}$, and therefore it is a positive semi-definite matrix [29]. We can easily show

$$D_2(\boldsymbol{y}) = \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y} = \frac{1}{2} W_{i,j} (y_i - y_j)^2.$$

Based on the quadratic form, we can also define the semi-norm

$$\|\boldsymbol{y}\|_{\boldsymbol{L}} = \sqrt{\boldsymbol{D}_2(\boldsymbol{y})},$$

which is a semi-norm for the same reason explained in the previous subsection for total variation.

While both the TV and the graph Laplacian quadratic form (or norm) of graph signal are convex and tend to small values for a smooth signal, the TV norm seems to be more suitable for cases where the graph signal has abrupt changes or discontinuities across subsets of nodes or clusters, since the Laplacian quadratic form (or norm) tends to produce unnecessarily larger values in such cases, which leads to oversmoothing and consequently can negatively affect the accuracy of the recovery [24].

2.4 Deterministic Graph Signal Learning Methods

Here, we review a few of the most significant recently proposed methods for graph signal learning (or recovery) which are based on optimization directly over the graph signal, as opposed to first considering the signal as a random variable and imposing a graphical model over this variable, which is the subject of the next section. We should first note that a learning method or approach ultimately consists of two parts: (1) the optimization problem that the method attempts to solve, which as we will see is thankfully often a convex problem, and (2) the optimization algorithm, which of course depends on the optimization problem and can be an off-the-shelf linear program or quadratic program solver (itself based on the interior-point method or others), a primal dual method or others.

The unifying theme behind all the learning methods is that they try to find a signal that is close to the measurements (i.e., has low error on the sampling set) and is also smooth, i.e., they generally try to solve the regularized problem of finding

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y}\in\mathbb{R}^N} E(\boldsymbol{y}, \boldsymbol{z}) + \lambda\Omega(\boldsymbol{y}). \tag{5}$$

Here, $\boldsymbol{z}$ is the measurement vector, $E : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$ is the error or loss function, $\Omega : \mathbb{R}^N \to \mathbb{R}$ is the regularization function, and $\lambda > 0$ is a tuning parameter determining the emphasis put on minimizing the regularizer versus the error. Therefore, the difference between the optimization problems that these methods attempt to solve stems from: (1) how we enforce the closeness of the predicted signal and the measurements, i.e., what is the error term in the objective function (2) how we enforce the smoothness (e.g., TV, graph Laplacian, etc.) via the regularizer. Although it should be noted that either of the terms can be defined as a constraint in a constrained optimization problem, therefore a small technical difference also comes from whether the error term or the smoothness term are defined as a constraint or as a regularization term in the objective function, although these forms can usually be converted to each other easily.

If we assume the measurements are given by $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{y}_0 + \boldsymbol{n}$, where $\boldsymbol{y}_0$ is the true graph signal, $\boldsymbol{A}$ is the measurements matrix, and $\boldsymbol{n}$ is a vector of (i.i.d.) noise, then one of the simplest methods for recovering $\boldsymbol{y}$ from $\boldsymbol{z}$ is based on finding [6]

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y}\in\mathbb{R}^N} \|\boldsymbol{z} - \boldsymbol{A}\boldsymbol{y}\|_2^2 + \lambda \boldsymbol{y}^T \boldsymbol{L}\boldsymbol{y}. \tag{6}$$

This is essentially a regularized least-squares problem where the error term (the first term) is the squared sum of differences between the recovered signal and the measurements and the regularization term (with regularization parameter $\lambda > 0$) is the graph Laplacian quadratic from which enforces the smoothness of the graph signal. Since $\boldsymbol{L}$ is positive semi-definite, the objective function is the sum of two convex functions and is convex (in fact we can easily show it is a convex quadratic function). Therefore, taking the gradient of the objective function (and doing basic algebraic manipulation), we arrive at the optimality condition

$$(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{L})\hat{\boldsymbol{y}} = \boldsymbol{A}^T\boldsymbol{z}. \tag{7}$$

If we are dealing with a massive dataset and computing the inverse (or pseudo-inverse, if the inverse does not exist or the matrix is close to being singular) of $(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{L})$ is not feasible, then we can use a method like the block Gauss-Sidel method to find the solution efficiently, as in [13]. For the noiseless case (i.e., $z = \boldsymbol{A}\boldsymbol{y}_0$), if we define I to be the indicator vector for the samples such that

$I_i = 1$ if $i \in \mathcal{M}$ and 0 otherwise, and then apply the Jacobi iterative method [29] to (7), we can easily show that at iteration k we have

$$\hat{y}_i^{(k+1)} = \frac{1}{I_i + \lambda \sum_{j \neq i} \boldsymbol{W}_{i,j}} (I_i y_{0,i} + \lambda \sum_{j \neq i} \boldsymbol{W}_{i,j} \hat{y}_j^{(k)}),$$

which we can consider as (a slightly modified form of) label propagation, one of the first proposed and most well-known methods for transductive semi-supervised learning [11, 37]. A quick inspection of the above iteration shows why this method is called label propagation, since the term $\lambda \sum_{j \neq i} \boldsymbol{W}_{i,j} \hat{y}_j^{(t)}$ corresponds to propagating the labels of the neighbors of node i.

Replacing, the Laplacian quadratic form in (6) with the total variation of the signal (i.e., eq. (4)), we get the optimization problem

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y} \in \mathbb{R}^N} \|\boldsymbol{z} - \boldsymbol{A}\boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{M}\boldsymbol{y}\|_1. \tag{8}$$

While the objective function is convex (since it is a sum of two convex terms), it is not differentiable, so we need to either convert it to a form that is differentiable or use subgradients. The subgradient of a convex function $f : \mathbb{R}^N \to \mathbb{R}$ at a point $\boldsymbol{x}_0$ is a vector $\boldsymbol{v} \in \mathbb{R}^N$ such that for all $\boldsymbol{x} \in \mathbb{R}^N$ we have

$$f(\boldsymbol{x}_0) + \boldsymbol{v}^T(\boldsymbol{x} - \boldsymbol{x}_0) \leq f(\boldsymbol{x}),$$

and the set of all subgradients of f at $\boldsymbol{x}_0$ is its subdifferential $\partial f(\boldsymbol{x}_0)$. If the function is differentiable then we would have $\partial f(\boldsymbol{x}_0) = \nabla f(\boldsymbol{x}_0)$. If we denote the objective function in (8) by $f(\boldsymbol{y})$, then the simplest method (at least conceptually) for finding the solution would be based on the iteration

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} - \alpha_k g(\boldsymbol{y}^{(k)}),$$

where $g(\boldsymbol{y}^{(k)}) \in \partial f(\boldsymbol{y}^{(k)})$, and α_k is a positive step-size which can be constant or chosen by a step-size rule. This method can be considered the analog of the gradient descent algorithm for nondifferentiable convex functions. Unfortunately, the iteration may not be a descent iteration for any α_k, i.e., we might have $f(\boldsymbol{y}^{(k+1)}) > f(\boldsymbol{y}^{(k)})$, although it can be shown that the distance to the optimal point decreases at each iteration. In addition to this, this method may suffer from slow convergence. However, we can thankfully utilize subgradients in other convex optimization methods, such as proximal and proximal gradient methods, augmented lagrangian methods and the closely related and highly popular alternating direction method of multipliers (ADMM), primal-dual methods, among others. In particular, the proximal gradient method and ADMM are well-suited for this problem since they naturally lend themselves to objective functions that consist of a differentiable and a non-differentiable part. For a comprehensive treatment of these algorithms and the subgradient method please refer to [8].

If we do not want to involve subgradients or subdifferentials in solving (8), the simplest approach is to convert it to a constrained quadratic problem

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{t} \in \mathbb{R}^{|\mathcal{E}|}} \|\boldsymbol{z} - \boldsymbol{A}\boldsymbol{y}\|_2^2 + \lambda(\boldsymbol{1}^T \boldsymbol{t}) \tag{9}$$

$$\text{subject to } -\boldsymbol{t} \leq \boldsymbol{M}\boldsymbol{y} \leq \boldsymbol{t}, \tag{10}$$

where $\boldsymbol{1}$ denotes a vector of all ones with the appropriate size. The advantage of this form is that we can apply any off-the-shelf QP solver to this problem to find the solution.

3 Probabilistic Graphical Models

In this section, we review some of the fundamental concepts and algorithms behind the main tool we use in this work for learning graph signals: probabilistic graphical models (PGM). First, a very brief review of the fundamental concepts and definitions behind PGM's is given. Next, Markov networks which are the underlying model behind the method proposed in this work are discussed, including a brief discussion of Lattice models, a simple but very influential type of Markov networks. Finally, a few of the most popular methods for performing approximate inference in graphical models are reviewed.

By merging ideas from probability theory and graph theory, PGM's provide a powerful framework for expressing our modeling assumptions and performing inference and reasoning about a set of variables. We can think of a PGM as a model consisting of union of two parts: (1) a (directed or undirected) graph where each node corresponds to a random variable (either discrete or continuous), and (2) a probability distribution defined over these variables. The major benefit that a PGM provides is that in many cases, we can essentially read off the independence (and conditional independence) relationships between the variables in the probability distribution from the graph, and vise versa. Not only this can help to clarify the independence assumptions in our model, but (as we will see later) this can also aid in doing efficient inference. In order to explain this essential property of graphical models, first we review the concepts of independence and conditional independence between random variables.

Definition 1 (Potential)**.** *A potential* $\phi(\mathbf{x})$ *is a non-negative function of the random (vector) variable* $\mathbf{x}$*, i.e.,* $\phi(\mathbf{x} = \boldsymbol{x}) \geq 0$ *for all* $\boldsymbol{x} \in dom(\mathbf{x})$*. A probability distribution* $p(\mathbf{x})$ *is a special case of a potential which sums to 1.*

Definition 2 (Independence)**.** *Two random variables* $\mathbf{x}$ *and* $\mathbf{y}$*, with domains* $dom(\mathbf{x})$ *and* $dom(\mathbf{y})$ *are independent if*

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y}) \text{ or equivalently } p(\boldsymbol{x}|\boldsymbol{y}) = p(\boldsymbol{x}),$$

for all $\boldsymbol{x} \in dom(\mathbf{x})$ *and* $\boldsymbol{y} \in dom(\mathbf{y})$*.*

Definition 3 (Conditional independence)**.** *Two random variables* $\mathbf{x}$ *and* $\mathbf{y}$*, with domains* $dom(\mathbf{x})$ *and* $dom(\mathbf{y})$ *are conditionally independent given random variable* $\mathbf{z}$ *with domain* $dom(\mathbf{z})$ *if*

$$p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y}|\boldsymbol{z}) \text{ or equivalently } p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{y}),$$

for all values of $\boldsymbol{x} \in dom(\mathbf{x}), \boldsymbol{y} \in dom(\mathbf{y}),$ *and* $\boldsymbol{z} \in dom(\mathbf{z})$*.*

We denote conditional independence by the symbol $\perp$. Also, we can generalize independence and conditional independence to sets of variables, so that if $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ denote three sets of variables, then $\mathcal{X} \perp \mathcal{Y}$ or $\mathcal{X} \perp \mathcal{Y}|\varnothing$ mean that $\mathcal{X}$ and $\mathcal{Y}$ are (unconditionally) independent, and $\mathcal{X} \perp \mathcal{Y}|\mathcal{Z}$ means that $\mathcal{X}$ and $\mathcal{Y}$ are independent conditioned on $\mathcal{Z}$.

Based on whether the edges in the graphical model are directed or not, we can have three types of graphical models. A directed graphical model is one in which all the edges are directed. By far the most-well known and widely used directed PGM is a Bayesian network, also known as a belief network, in which the underlying graph is a directed acyclic graph and the joint distribution over the variables is factorized into a set of conditional distributions where the conditioning set in each factor is the ancestors of the corresponding variables. In an undirected model or Markov

network, as the name suggests, all the edges are undirected. Finally, we can have both directed and undirected edges in a graphical model, as in chain graphs and factor graphs. The important difference between these models lies in the set of (conditional) independence assumptions they can express. Specifically, Bayesian networks and Markov networks represent different independence assumptions, while chain graphs are more expressive in this sense than both Bayesian networks and Markov networks, and factor graphs are in turn more expressive than chain graphs. Since the proposed model in this work is based on a Markov network, we focus on this model in the subsequent sections and refer to [3] for an introduction to the other models.

3.1 Markov Networks

Informally, a Markov network defined on an undirected graph of variables is a distribution over the variables which is equal to the product of potentials defined on maximal cliques of the graph. A clique is a subset of nodes in a graph in which all the nodes are connected to each other, and a maximal clique is a clique which no larger clique in the graph contains it. More formally, A Markov network is defined as follows.

Definition 4 (Markov network)**.** *A Markov network for a set of random variables $\mathcal{X}$ is defined as*

$$p(\mathcal{X}) = \frac{1}{Z}\prod_c \phi_c(\mathcal{X}_c),$$

where $\phi_c(\mathcal{X}_c)$ is a potential defined on the subset of variables $\mathcal{X}_c \subseteq \mathcal{X}$ belonging to a maximal clique in the undirected graph which represents the model, and Z is the normalizing constant (also known as partition function), i.e.,

$$Z = \int_{\mathcal{X}} \prod_c \phi_c(\mathcal{X}_c).$$

In Fig. 3.1 we depict a simple undirected graph that corresponds to Markov network

$$p(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5) = \frac{1}{Z}\phi(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4)\phi(\mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5).$$

While the definition of the Markov network is based on maximal graphs, in the literature, the definition is typically extended to cases where some potentials are defined for non-maximal cliques, and even potentials concerning a single variable (i.e., unitary potential). For example, Fig. 3.1 can also be considered as a Markov network representing the distribution

$$p(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5) = \frac{1}{Z}\phi(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4)\phi(\mathrm{x}_3, \mathrm{x}_5)\phi(\mathrm{x}_4, \mathrm{x}_5),$$

where the potential $\phi(\mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5)$ over the maximal clique of $\mathrm{x}_3 - \mathrm{x}_4 - \mathrm{x}_5$ is replaced by two pair-wise potentials.

An important question is what kind of conditional independence statements an undirected graph which underlies a Markov network represents. These statements are known as Markov properties, which are all equivalent and hold for any Markov network with positive potentials. The global Markov property states that if $\mathcal{X}$ and $\mathcal{Y}$ are separated by $\mathcal{Z}$, i.e., if every path from any member of $\mathcal{X}$ to any member of $\mathcal{Y}$ passes through a member of $\mathcal{Z}$, then $\mathcal{X} \perp \mathcal{Y} | \mathcal{Z}$. Hence, if no path exists between $\mathcal{X}$ and $\mathcal{Y}$, we have $\mathcal{X} \perp \mathcal{Y}$. The local Markov property states,

$$p(\mathrm{x}|\mathcal{X} \setminus \mathrm{x}) = p(\mathrm{x}|Ne(\mathrm{x})),$$

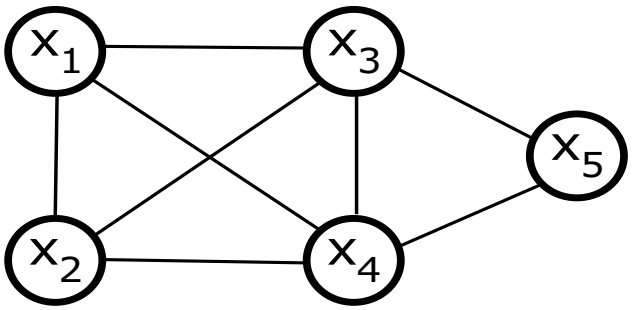

Figure 1: A simple undirected graph corresponding to the Markov network $p(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5) = \frac{1}{Z}\phi(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4)\phi(\mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5)$.

where $\mathcal{X} \setminus \mathrm{x}$ denotes the set $\mathcal{X}$ excluding x, and $Ne(\mathrm{x})$ denotes the neighbors of x. Finally, the pairwise Markov property states that for any non-adjacent variables x and y, $\mathrm{x} \perp \mathrm{y} | \mathcal{X} \setminus \{\mathrm{x}, \mathrm{y}\}$.

The Hammersely-Clifford theorem clarifies the relationship between the conditional statements specified by an undirected graph and the distribution of the Markov network defined for such a graph. It says that given the set of conditional independence statements implied by an undirected graph (i.e., the Markov properties), the most general functional form of a distribution that satisfies these statements is that specified by the Markov network over the graph, i.e., the product of potentials defined on the cliques of the graph, provided the potentials are positive. It also says that the other direction of the argument holds, i.e., given a factorization into clique potentials, the Markov properties on the corresponding graph are implied.

Lattice models

An important type of pair-wise Markov networks, which the proposed model in this work is inspired by, are known as lattice models. As the name suggests, they specify a distribution over a set of variables that are arranged on a lattice, in which the distribution is $p(\mathcal{X}) = \frac{1}{Z}\prod_{i \sim j} \phi_{i,j}(\mathrm{x}_i, \mathrm{x}_j)$, where $i \sim j$ denotes the set of indices in which i and j are neighbors in the undirected lattice graph. $\phi_{i,j}(\mathrm{x}_i, \mathrm{x}_j)$ is typically a potential that forces the value or state of x_i and x_j to be similar. While such models have been popular for a long time in the image processing community for tasks such as image denoising and image in-painting or reconstruction (where the lattice is defined over the image pixels) [17, 36], they originally come from a popular model in statistical physics for describing a system of micro-magnets, known as the Ising model. In the Ising model, each variable x_i is a binary variable corresponding to the spin configuration of a magnet at a specific site in the lattice, and the probability of a configuration of the overall system, $\mathcal{X} = \boldsymbol{x}$, is given by the Boltzmann distribution

$$p(\boldsymbol{x}) = \frac{1}{Z(\beta)} \exp(-\beta E(\boldsymbol{x})),$$

where β is known as the inverse temperature, $Z(\beta) = \int_X \exp(-\beta E(\boldsymbol{x}))$ is the partition function, and $E(\mathcal{X})$ is known as the free energy of the systems defined by

$$E(\boldsymbol{x}) = -\sum_{i \sim j} W_{i,j} I(x_i = x_j) - \sum_i b_i x_i, \tag{11}$$

where $I(s)$ is the indicator function, i.e., $I(s) = 1$ if the statement s is true, and is equal to zero otherwise. In (11), $W_{i,j}$ determines the interaction between the magnets at sites i and j, and b_i is the external magnetic field or bias at site i. It is clear that the free energy definition encourages neighboring sites to have similar magnetic spins while respecting the bias imposed by the external fields. The parameter controlling the state of the system (i.e., what fraction of the magnets are aligned) is β which is inversely proportional to the temperature. If the weights (or interactions) $W_{i,j}$ are nonnegative, then we can find the most likely configuration exactly and efficiently for any arbitrary topology (i.e., not just lattices) by translating the problem into a minimum graph cut problem [3]. Due to the equivalence of the minimum graph cut to the maximum flow we can then use a max-flow algorithm such as the well-known Ford-Fulkerson algorithm and retrieve the solution in $O(N^3)$ time [12].

There are other popular models which are closely related to the Ising model. One of these, the Potts model, is simply the extension of the Ising model to variables with more than two states. This model has also been used for color image denoising or inpainting. Although no efficient exact inference methods exist for such models, a few popular approximate inference methods are based on approximating the problem by a series of binary problem, where each problem is solved via the above graph cut method [9, 22]. Another closely related model is the Boltzmann machine [1], which as the name suggests is also based on the Boltzmann distribution. It can be considered as a stochastic generative neural network where the distribution over the binary variables is

$$p(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{W}, \boldsymbol{b})} \exp(\sum_{i \sim j} W_{i,j} x_i x_j + \sum_i b_i x_i).$$

In this model, $W_{i,j}$ are the weights between the nodes or neurons and $\boldsymbol{b}_i$ are the biases, and both can be considered as parameters that should be learned. Inference and learning in Boltzmann machines is in general intractable and we must resort to approximate methods. A few of the more important approximate inference methods are presented in the next section.

3.2 Approximate Inference Methods

In the model we consider later for graph signal learning and in many other cases in machine learning, we typically have two types of variables: those that are observed, and those that are hidden. The hidden variables could be hidden either because we have missing information about them (as in graph signal learning where we do not know the label of most points), or they could be hidden because they are latent variables which we invent to explain a phenomenon or observation.

In any case, often our ultimate goal is to find the most likely values or states for the hidden variables given the values observed for the visible variables, i.e., to find the maximum a-posteriori (MAP) estimate

$$\hat{\boldsymbol{h}} \in \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}, \Theta) = \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta), \tag{12}$$

where $\boldsymbol{h}$ and $\boldsymbol{v}$ denote the value of the hidden and the visible variables respectively, and Θ denotes all the parameters in the model. It is worth noting that the equality in (12) can make the inference significantly easier in cases where the complete data likelihood $p(\boldsymbol{h}, \boldsymbol{v}|\Theta)$ does not have a simple or tractable form, since for computing the posterior

$$p(\boldsymbol{h}|\boldsymbol{v}, \Theta) = \frac{p(\boldsymbol{h}, \boldsymbol{v}|\Theta)}{p(\boldsymbol{v}|\Theta)}, \tag{13}$$

we need to do an additional calculation to find $p(\boldsymbol{v}|\Theta) = \int_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta)$, which might be intractable. Despite this, finding the exact MAP estimate by solving (12) might still be infeasible if the complete likelihood is not simple enough or the problem size is not small enough.

What complicates things further is that our models usually include parameters that are unknown and we should attempt to learn (In this case we can consider the parameters as additional hidden variables). Probably the most popular approach for learning the parameters is maximum likelihood, which here is equivalent to maximizing the marginal log-likelihood

$$\hat{\Theta} \in \arg\max_{\Theta} \log p(\boldsymbol{v}|\Theta) = \arg\max_{\Theta} \log \int_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta) \tag{14}$$

If the parameters in our model are continuous, then we can use a gradient-based optimization algorithm to learn them. The gradient of the log-likelihood would then be

$$\partial_{\Theta} \log p(\boldsymbol{v}|\Theta) = \frac{1}{p(\boldsymbol{v}|\Theta)} \int_{\boldsymbol{h}} \partial_{\Theta} p(\boldsymbol{v}, \boldsymbol{h}|\Theta) \tag{15}$$

$$= \int_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}, \Theta) \partial_{\Theta} \log p(\boldsymbol{h}, \boldsymbol{v}|\Theta) \tag{16}$$

$$=< \partial_{\Theta} \log p(\mathbf{h}, \boldsymbol{v}|\Theta) >_{p(\mathbf{h}|\boldsymbol{v}, \Theta)} \tag{17}$$

where $< q >_p$ denotes the average of q with respect to the distribution p. However, this average is not tractable to compute for complicated models, and we need to approximate it in some way.

These issues motivate us to resort to variational methods, in which we attempt to approximate $p(\mathbf{h}|\mathbf{v}, \Theta)$ by a variational distribution $q(\mathbf{h})$ and learn the parameters too, in an iterative fashion. In order to find an accurate approximation to the posterior, we try to minimize the Kullback-Liebler (KL) divergence between $q(\boldsymbol{h})$ and $p(\mathbf{h}|\mathbf{v}, \Theta)$

$$KL(q|p) = \int_{\boldsymbol{h}} q(\boldsymbol{h}) \log \frac{q(\boldsymbol{h})}{p(\boldsymbol{h}|\boldsymbol{v}, \Theta)}. \tag{18}$$

The KL divergence is not a proper distance measure since it is not symmetric, but it can be shown that $KL(q|p) \geq 0$ and is equal to zero if and only if for all fixed $\boldsymbol{v}$ and Θ, $q(\mathbf{h}) = p(\mathbf{h}|\boldsymbol{v}, \Theta)$. Since the infeasibility of computing $p(\boldsymbol{h}|\boldsymbol{v}, \Theta)$ is one of the main reasons we resorted to the variational method in the first place, we use the KL divergence to find a lower bound on $p(\mathbf{h}|\mathbf{v}, \Theta)$. Specifically, we first use (13) in (18) to get

$$KL(q|p) = \int_{\boldsymbol{h}} q(\boldsymbol{h}) \log q(\boldsymbol{h}) - \int_{\boldsymbol{h}} q(\boldsymbol{h}) \log p(\boldsymbol{h}, \boldsymbol{v}|\Theta) + \log p(\boldsymbol{v}|\Theta). \tag{19}$$

Rearranging this equation, we get

$$\log p(\boldsymbol{v}|\Theta) = -F(q, p) + KL(q|p), \tag{20}$$

where

$$F(q, p) = \int_{\boldsymbol{h}} q(\boldsymbol{h}) \log q(\boldsymbol{h}) - \int_{\boldsymbol{h}} q(\boldsymbol{h}) \log p(\boldsymbol{h}, \boldsymbol{v}|\Theta) \tag{21}$$

is commonly known as the free energy or Gibbs free energy. The first integral (or average) in (21) is also known as the entropy, while the second one is known as the energy. Since $KL(q|p) \geq 0$, from eq. (20) we get the lower bound

$$\log p(\boldsymbol{v}|\Theta) \geq -F(q, p). \tag{22}$$

Hence, finding the tightest lower bound for $\log p(\mathbf{v}|\Theta)$ corresponds to minimizing the free energy with respect to both q and Θ.

For minimizing the free energy with respect to (continuous) parameters, we need to compute its gradient. Since the entropy is independent of Θ, the gradient of free energy would be

$$\partial_\Theta F(q,p) = - < \partial_\Theta \log p(\mathbf{h}, \boldsymbol{v}|\Theta) >_{q(\mathbf{h})} .$$

Comparing this with the gradient of the marginal log-likelihood (cf. (15)), we can see that the gradient of the negative of the free energy is an approximation of the gradient of the marginal log-likelihood, since the average with respect to the posterior has been replaced with the average with respect to an approximation of the posterior ($q(\mathbf{h})$). Indeed, taking the gradient of both sides in the inequality (22) shows that the gradient of the marginal log-likelihood is lower bounded by the gradient of negative of the free energy. Therefore, minimizing the free energy with respect to the parameters using a gradient based method is equivalent to approximately maximizing the marginal log-likelihood. Of course, the accuracy of this approximation depends on the accuracy of $q(\mathbf{h})$ in approximating $p(\mathbf{h}|\mathbf{v}, \Theta)$. This is another viewpoint that ultimately provides the same objective as for finding the tightest lower bound in (22), i.e., to minimize the free energy with respect to both q and Θ in an iterative fashion.

Minimizing the free energy with respect to both q and Θ is essentially the unifying concept behind the most commonly used approximate inference methods for graphical model. The difference between these methods stems from the different assumptions on the family of distributions that q comes from, and the way we compute q and Θ. Next, we present a few of these methods. Most of these methods can be considered an EM-type algorithm, where at each iteration, first in the E-step, we fix the parameters and minimize the free energy with respect to q, and in the M-step, we fix q and minimize it with respect to the parameters, and we iterate these steps until convergence or for fixed number of iterations.

3.2.1 Iterated conditional modes

In the most basic form of iterated conditional modes, we iteratively solve for each variable or parameter, trying to find a value that minimizes the free energy, given the current value of all the other variables and parameters. This translates to assuming $q(\boldsymbol{h}) = \delta(\boldsymbol{h} - \hat{\boldsymbol{h}})$, where $\hat{\boldsymbol{h}}$ is the MAP estimate of the hidden variables and $\delta(\cdot)$ is the Dirac delta function. Replacing this into the equation for $F(q,p)$, we maximize with respect to $\hat{\boldsymbol{h}}_i$ and Θ_i iteratively in any order, for a fixed number of iterations or until convergence. Put more simply, the update equation for the i'th hidden variable would be

$$\boldsymbol{h}_i = \arg\max_{\boldsymbol{h}_i} p(\boldsymbol{h}_i|\boldsymbol{h} \setminus \boldsymbol{h}_i, \boldsymbol{v}, \Theta).$$

For a Markov network, using the local Markov property, this can be rewritten as

$$\boldsymbol{h}_i = \arg\max_{\boldsymbol{h}_i} p(\boldsymbol{h}_i|Ne(\boldsymbol{h}_i), \Theta),$$

which can lead to considerable computational savings. While iterated conditional modes is perhaps the simplest approximate inference or learning method in terms of implementation, it can produce poor local maxima since it does not take into account the inter-dependency of the variables in the posterior.

3.2.2 Exact expectation maximization (EM)

It can be shown that if we do not have any assumptions on q other than the fact that it sums (or integrates) to 1, then for any value of Θ, minimizing $F(q, p)$ with respect to q results in $q(\boldsymbol{h}) = p(\boldsymbol{h}|\boldsymbol{v}, \Theta)$. This leads to the highly popular exact EM procedure (commonly known as simply EM), wherein after initializing Θ with some randomly chosen value Θ^0, we perform the following steps until convergence or for a fixed number of iterations:
E-step: compute $q^{(k+1)}(\boldsymbol{h}) = p(\boldsymbol{h}|\boldsymbol{v}, \Theta^{(k)})$.
M-step: find $\Theta^{(k+1)} \in \arg\min_\Theta F(q^{(k+1)}, p)$.

3.2.3 Variational EM

If the exact posterior is intractable, then we can assume $q(\boldsymbol{h})$ comes from a certain family of distributions to simplify the calculations. The most basic assumption we can make is known as the 'naive' mean field assumption, where we assume $q(\boldsymbol{h}) = \prod_i q(\boldsymbol{h}_i)$. Using this form for q in the free energy (eq. (21)) and minimizing for each $q(\boldsymbol{h}_i)$ individually leads to the general mean field equations [3]

$$q(\boldsymbol{h}_i) \propto \exp(< \log p(\boldsymbol{h}, \boldsymbol{v}|\Theta) >_{\prod_{j \neq i} q(\boldsymbol{h}_j)})$$

Hence, we can construct a variational EM algorithm by replacing the E-step of the exact EM algorithm with the mean field equations.

Even if the mean field equations are tractable in some case, they are be based on a naive assumption since they do not account for the dependency between the variables. In these cases, the mean-field approach may not result in an accurate estimate and we might benefit by considering a structured variational EM approach. In a structure variational method, we impose a structure on the form of q via a graphical model, although the more elaborate the structure the more the computational cost would generally be.

3.2.4 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a highly popular general method for inexact sampling, which we can use to sample from a distribution that is either impossible to sample from directly or computationally infeasible to do so. The basic idea in MCMC methods is to use a Markov chain to sample in an iterative fashion. A Markov chain is a sequence of random variables $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ such that the distribution of $\mathbf{x}^{(t)}$ given all previous random variables only depends on $\mathbf{x}^{(t-1)}$. We hope that at each iteration, we sample from a transition distribution $T_t(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$ that is closer to the true distribution $p(\mathbf{x})$. More specifically, we try to choose a Markov chain such that its stationary distribution (the Markov chain distribution in the limit of $t \to \infty$) is close to the real distribution and run the method long enough so that it reaches this stationary distribution.

The most general family of MCMC methods are known as Metropolis-Hastings algorithms. In the more basic form of this algorithm, known simply as the Metropolis algorithm, first we draw a sample $\boldsymbol{x}^{(0)}$ randomly from an approximate starting distribution $p_0(\mathbf{x})$. Then, at iteration t we sample a proposal $\boldsymbol{x}^*$ from a jumping (or proposal) distribution $J_t(\boldsymbol{x}^*|\boldsymbol{x}^{(t-1)})$, and compute the ratio $r = \frac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x}^{(t-1)})}$. Then we set $\boldsymbol{x}^{(t)} = \boldsymbol{x}^*$ with probability equal to $\min(r, 1)$, and otherwise keep the previous value, i.e., set $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-1)}$. In other words, if a proposal increases the true probability density, we accept it, but if it decreases the density, we accept it we probability equal to the density ratio r. The Metropolis-Hastings algorithm works essentially in the same fashion,

with the difference being that in the Metropolis algorithm we require the jumping distribution to by symmetric, i.e., $J_t(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = J_t(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$, and (due to the possible lack of symmetry in the jumping distribution) the ratio we use in the Metropolis-Hastings algorithm is

$$r = \frac{p(\boldsymbol{x}^*)J_t(\boldsymbol{x}^*|\boldsymbol{x}^{(t-1)})}{p(\boldsymbol{x}^{(t-1)})J_t(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^*)}.$$

In order to make sure the samples we take from the Metropolis (or Metropolis-Hastings) algorithm are accurate, besides carefully choosing the jumping distribution and the starting point(s), we need to make sure the algorithm is run long enough. In this regard, we discard the early iterations, which are known as warm-up or burn-in, since they can often be far from the true distribution. A common (but conservative) practice is to discard the entire first half of the simulation. We also need to monitor the convergence of the algorithm. This can be done for example by running simultaneous parallel chains from different starting points, breaking up each chain in half (after discarding the burn-in iterations) and comparing the within and between variance of the chains, to ensure the chains have mixed and are stationary too. For the exact implementation of this method please refer to [16, chapter 11].

Gibbs sampling

One of the simplest and also most popular MCMC algorithms is the Gibbs sampler. In this method, at each iteration we sample each component of $\mathbf{x}$ separately and sequentially, based on the current value of all the other components, i.e., we sample $x_i^{(t)}$ from $p(x_i|\boldsymbol{x}_{-i}^{(t-1)})$, where $\boldsymbol{x}_{-i}^{(t-1)} = (x_1^{(t)}, \cdots, x_{i-1}^{(t)}, x_{i+1}^{(t-1)}, \cdots, x_N^{(t-1)})$. Hence, Gibbs sampling can be thought of as a form of ICM where instead of choosing the MAP value of x_i given all the other variables, we choose it stochastically. We can also think of a Gibbs sampler as a type of Metropolis algorithm if we assume each iteration consists of N steps, where at step i of iteration t we use the jumping distribution $J_{i,t}(\cdot|\cdot)$ to sample along the i'th dimension of $\mathbf{x}$ according to

$$J_{i,t}(\boldsymbol{x}^*|\boldsymbol{x}^{(t-1)}) = \begin{cases} p(x_i^*|\boldsymbol{x}_{-i}^{(t-1)}) & \text{if } \boldsymbol{x}_{-i}^* = \boldsymbol{x}_{-i}^{(t-1)} \\ 0 & \text{otherwise.} \end{cases}$$

Gibbs sampling can work well if the variables (or the coordinates of the vector variable) are not strongly correlated, otherwise it can be very slow. In addition, if the distribution has regions with very low probability surrounded by high probability regions, then single coordinate updates might mean that we might get stuck in one region and have trouble navigating through the whole space.

We can use the samples extracted from any sampling algorithm to find the approximate distribution $q(\mathbf{h})$, as in the E-step of the algorithms above, and hence construct an MCMC-EM or Gibbs-EM algorithm. In particular, if we collect $\boldsymbol{h}^1, \ldots, \boldsymbol{h}^S$ as samples, then if the variables are discrete, we would have

$$q(\mathbf{h} = \boldsymbol{h}_p) = \frac{1}{S}\sum_{i=1}^{S} I(\mathbf{h}^i = \boldsymbol{h}_p),$$

for $\boldsymbol{h}_p \in dom(\mathbf{h})$, where $I(\cdot)$ denotes the indicator function (i.e., $I(x) = 1$ if $x =$ true , and 0 otherwise.)

3.2.5 Loopy belief propagation

Loopy belief propagation (LBP) is another widely popular method for approximate inference on graphical models. It is an iterative method based on message passing between the nodes of the graph. There are two forms of LBP: the sum-product algorithm and the max-product algorithm (or the min-product algorithm if we want to minimize a cost function instead of maximizing a distribution). In the sum-product version we are interested in finding the marginals of the distribution, while in the max-product version we want to find the most likely values or configuration of the variables. First we briefly explain the sum-product form of the algorithm and then point out how we derive the max-product form with only a minor modification. Consider an arbitrary pair-wise Markov network defined as

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{i \sim j} \phi(x_i, x_j).$$

We seek to find an approximation $q(\boldsymbol{x})$ to this distribution. First, all the messages are set to unity (or a uniform distribution). Then, at each update iteration, all the messages are updated according ot a fixed pre-determined schedule. At each such iteration, a message is passed from node i to node j, which we denote by $m_{ij}(x_i)$ when the value of the variable at node j is set to x_J. The message is updated according to

$$m_{ij}(x_j) = \int_{x_i} \phi(x_i, x_j) \prod_{k \in Ne(i), k \neq j} m_{ki}(x_i)$$

The message passing iterations continue according to the schedule for a fixed number of iterations or until convergence is reached. At convergence, the marginal $p(x_i)$ is approximated by

$$q(x_i) \propto \prod_{j \in Ne(i)} m_{ji}(x_i),$$

and the pairwise marginals are approximated by

$$q(x_i, x_j) \propto \prod_{k \in Ne(i), k \neq i} m_{ki}(x_i)\ \phi(x_i, x_j) \prod_{k \in Ne(j), k \neq j} m_{kj}(x_j)$$

After convergence, we can maximize each marginal $q(x_i)$ individually to find the most likely label for each node. However, if we are interested in the joint most likely configuration, we should instead use the max-product algorithm. The max-product algorithm is essentially the same as the sum-product one, except, in the message update iteration the integration operation is replaced by maximizing over x_i, and at convergence, we have a belief function at each node determined by

$$b_i(x_i) \propto \prod_{j \in Ne(i)} m_{ji}(x_i).$$

We get the most likely joint configuration by maximizing each belief function with respect to its variable individually.

At this point, we highlight some important practical and theoretical issues regarding LBP: (1) for trees, belief propagation is guaranteed to converge to the exact marginals in at most $2|\mathcal{E}|$ iterations, where $|\mathcal{E}|$ is the number of edges in the graph. For this reason it can be a very efficient inference method than naively marginalizing over the states of 2^{N-1} variables to find the marginal for one

variable. However, for graphs that contain loops there are no general convergence guarantees, and the method can diverge or oscillate. Despite this, if the loops are relatively long (so that the change in a variable does not reverberate back to it) and the links between the variables are not too strong, we can expect it to provide a fairly accurate approximation in a relatively short time. This, combined with its amenability to distributed implementation, has allowed it to enjoy a great deal of popularity in the machine learning and image processing communities. (2) The accuracy and convergence of LBP depends on the message updating schedule. (3) For continuous variables, the messages are functions (as opposed to vectors for discrete variables) and this can result in high memory and computation time requirements. This essentially limits the method to parametric distributions that are closed with respect to products and marginalization (meaning that taking the product of and marginalizing distributions belonging to this family results in a distribution that also belongs to the same family), where message passing amounts to parameter passing. For example, if the potentials are Gaussians, then message passing can be implemented as passing means and covariances. However, if the potentials do not belong to such families, the simplest way to deal with this problem is discretization, although this introduces additional quantization errors. Other proposed remedies for this problem include approximate message passing [25], which requires the network to be dense to be accurate, and nonparametric belief propagation which is based on approximating messages by Gaussian mixture models [33].

It is worth mentioning that LPB can be derived as a variational procedure, and it can also be formulated for factor graphs and combined with the EM algorithm, so that in the E-step we approximate the clique marginals with LPB and in the M-step use these approximations to maximize the free energy just as in the previously described methods. For these derivations, please refer to [14].

3.3 Approximate inference in undirected models

Learning the parameters in the model, and by extension approximate inference, is generally (much) more challenging in an undirected model compared to a directed model. To see this, consider an undirected model defined by

$$p(\mathbf{h}, \mathbf{v}|\Theta) = \frac{1}{Z(\Theta)}\tilde{p}(\mathbf{h}, \mathbf{v}|\Theta), \tag{23}$$

$Z(\Theta) = \int_{\mathbf{h},\mathbf{v}} \tilde{p}(\mathbf{h}, \mathbf{v}|\Theta)$ is the partition function. In a Markov network, the unnormalized potential is $\tilde{p}(\mathbf{h}, \mathbf{v}|\Theta) = \prod_c \phi_c(\mathbf{h}_c, \mathbf{v}_c|\Theta)$, where $\phi_c(\mathbf{h}_c, \mathbf{v}_c|\Theta)$ is the potential defined on the set of hidden and visible variables belonging to clique c (see definition 4).

As shown in the appendix, in this case we have

$$\partial_\Theta \log \tilde{p}(\boldsymbol{v}|\Theta) = \partial_\Theta \log \int_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \partial_\Theta \log Z(\Theta) \tag{24}$$

$$= < \partial_\Theta \log \tilde{p}(\mathbf{h}, \boldsymbol{v}|\Theta) >_{p(\mathbf{h}|\boldsymbol{v},\Theta)} - < \partial_\Theta \log \tilde{p}(\mathbf{h}, \mathbf{v}|\Theta) >_{p(\mathbf{h},\mathbf{v}|\Theta)} . \tag{25}$$

The first term in this equation is referred to as the clamped average (since the average is over $\mathbf{h}$ while $\mathbf{v}$ is kept constant at the value $\boldsymbol{v}$), while the second one is called the unclamped average. Using the derivative of the log partition function, i.e., the unclamped average, we can easily show

$$-\partial_\Theta F(q, p) = < \partial_\Theta \log \tilde{p}(\mathbf{h}, \boldsymbol{v}|\Theta) >_{q(\mathbf{h})} - < \partial_\Theta \log \tilde{p}(\mathbf{h}, \mathbf{v}|\Theta) >_{p(\mathbf{h},\mathbf{v}|\Theta)}, \tag{26}$$

where $q(\mathbf{h})$ is an approximation of $p(\mathbf{h}|\mathbf{v}, \Theta)$ that we try to find from an approximate inference (EM-type) algorithm. In practice, for many distributions, both averages are intractable to compute. In

these cases, we can approximate them, and perhaps the easiest way to do so is via approximate sampling. Specifically, assume in our training data we have set $\mathbf{v} = \boldsymbol{v}_c$ and in the current iteration of our EM-type algorithm Θ is set to $\Theta^{(k)}$. Furthermore, assume $\boldsymbol{h}_c^1, \ldots, \boldsymbol{h}_c^S$ and $(\boldsymbol{h}_u^1, \boldsymbol{v}_u^1), \ldots, (\boldsymbol{h}_u^S, \boldsymbol{v}_u^S)$ are approximate samples taken from $p(\mathbf{h}|\boldsymbol{v}, \Theta^{(k)})$ and $p(\mathbf{h}, \mathbf{v}|\Theta^{(k)})$ respectively, acquired from Gibbs sampling or another form of MCMC sampling. Then, we can approximate (26) via

$$-\partial_\Theta F(q,p) = \frac{1}{S}\sum_{s=1}^{S} \partial_\Theta \log \tilde{p}(\boldsymbol{h}_c^s, \boldsymbol{v}_c|\Theta) - \frac{1}{S}\sum_{s=1}^{S} \partial_\Theta \log \tilde{p}(\boldsymbol{h}_u^s, \boldsymbol{v}_u^s|\Theta) \tag{27}$$

4 Proposed Model for Graph Signal Recovery

In this section, first, a graphical model for recovering a smooth graph signal is proposed. The key component of this model, which can be considered as form of pairwise Markov network defined over the data graph, are the prior potentials enforcing the smoothness of the signal. Next, two different choices are presented for this smoothness prior, and finally, the issue of optimizing the parameters of the model is briefly discussed.

Henceforth, assume we have access to the data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, where $\boldsymbol{W} \in \{0,1\}^{N\times N}$ is the adjacency matrix. Further assume the measurements are taken from a small sampling set $\mathcal{M} = \{m_1, \dots, m_M\}$, with $M = |\mathcal{M}| << N$, and assume they are contaminated with additive white Gaussian noise. Denote the graph signal by the random variable $\mathbf{y}$ and the measurements by the random variable $\mathbf{z}$. Therefore, for $i = 1, \dots, M$ we have $\mathrm{z}_i = \mathrm{y}_{m_i} + \mathrm{n}_i$, where $m_i \in \mathcal{M}$, $\mathrm{n}_i \sim \mathcal{N}(0, \sigma_n^2)$ is the noise, and $\sigma_n > 0$ is the standard deviation of the noise.

Denoting all the parameters of the model by Θ, our goal is to find

$$\arg\max_{\mathbf{y}} p(\boldsymbol{y}|\boldsymbol{z}, \Theta) = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}, \boldsymbol{z}|\Theta) \tag{28}$$

We define the joint likelihood via $p(\mathbf{y}, \mathbf{z}|\Theta) = p(\mathbf{z}|\mathbf{y}, \Theta)p(\mathbf{y}|\Theta)$, where $p(\mathbf{z}|\mathbf{y}, \Theta)$ is the likelihood of the measurements given the graph signal and the parameters and $p(\mathbf{y}|\Theta)$ is the graph signal prior. Assuming the measurements are independent and identically distributed (i.i.d.), the first term is readily given by the white Gaussian noise assumption

$$p(\mathbf{z}|\mathbf{y}, \Theta) = \prod_{m_j \in \mathcal{M}} \mathcal{N}(\mathrm{z}_j; \mathrm{y}_{m_j}, \sigma_n^2). \tag{29}$$

Based on the assumption regarding the smoothness of the signal, the prior can also be defined by

$$p(\mathbf{y}|\Theta) \propto \prod_{i\sim j} \phi(\mathrm{y}_i, \mathrm{y}_j|\Theta), \tag{30}$$

where $\phi(\mathrm{y}_i, \mathrm{y}_j|\Theta)$ is a positive potential which enforces the graph signal at the neighboring nodes i and j to be similar. Putting these two equations together, the joint likelihood is given by the pairwise Markov network

$$p(\mathbf{y}, \mathbf{z}|\Theta) = \frac{1}{Z(\Theta)} \prod_{m_j \in \mathcal{M}} \mathcal{N}(\mathrm{z}_j; \mathrm{y}_{m_j}, \sigma_n^2) \prod_{i\sim j} \phi(\mathrm{y}_i, \mathrm{y}_j|\Theta). \tag{31}$$

4.1 Prior

The success of the model in accurately modeling the problem and its computational tractability for inference and learning to a large extent depends on the choice of the prior.

4.1.1 Two state Gaussian mixture prior

A natural and simple choice for the prior potentials enforcing smoothness of the signal would be a two state Gaussian mixture model, i.e.,

$$\phi(\mathrm{y}_i, \mathrm{y}_j|\Theta) = \alpha\mathcal{N}(\mathrm{y}_i - \mathrm{y}_j; 0, \sigma_0) + (1-\alpha)\mathcal{N}(\mathrm{y}_i - \mathrm{y}_j; 0, \sigma_1), \tag{32}$$

with the mixture coefficient $0 \leq \alpha \leq 1$ and the standard deviations $\sigma_1 > \sigma_0 > 0$. The first Gaussian corresponds to a small signal difference over the edge $\{i, j\}$, while the second one corresponds to a large difference. In this way, α can be interpreted as a parameter determining the edge sparsity rate. Hence, the parameter vector would be $\Theta = (\sigma_n^2, \sigma_0^2, \sigma_1^2, \alpha)$. This prior has been previously utilized for one dimensional signal recovery [4], i.e., compressed sensing, however the author is unaware of any case where it has been considered for (continuous) graph signal recovery.

4.1.2 Laplace prior

A distribution which is frequently used for promoting sparsity, is the Laplace prior defined as

$$L(x; \mu, \lambda) = \frac{1}{2\lambda} \exp(\frac{-|x - \mu|}{\lambda}).$$

The Laplace prior is also underlying a probabilistic interpretation of the well-known 'least absolute shrinkage and selection operator' (LASSO) method in statistics and machine learning [34], The LASSO problem can be considered as a form of Bayesian regression, where the likelihood of the measurement $\boldsymbol{y}$ given the data $\mathbf{X}$ and the regression coefficients (or weights) $\boldsymbol{w}$ is given by

$$p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}; \mathbf{X}\boldsymbol{w}, \sigma),$$

for some σ, and the prior is given by

$$p(\boldsymbol{w}) = L(\boldsymbol{w}; 0, \lambda_w).$$

Ignoring the terms that do not depend on $\boldsymbol{w}$ and multiplying by constants, finding the MAP estimate of $\boldsymbol{w}$ from $\log p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{y})$ leads to the (regularized form of) the LASSO problem

$$\hat{\boldsymbol{w}} \in \arg\max_{\boldsymbol{w}} \|\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1,$$

with the regularization parameter λ.

Utilizing the Laplace prior for our problem of graph signal recovery, we define the pairwise potentials by

$$\phi(\mathrm{y}_i, \mathrm{y}_j|\Theta) = L(\mathrm{y}_i - \mathrm{y}_j; 0, \lambda).$$

This is equivalent to defining the prior as

$$p(\boldsymbol{y}|\Theta) = L(\boldsymbol{M}\boldsymbol{y}, 0, \lambda\mathbf{I}), \tag{33}$$

where $\boldsymbol{M}$ is the incidence matrix as defined in Eq. (3). Finding the MAP estimate of $\boldsymbol{y}$ then corresponds to the convex optimization problem defined in Eq. (8).

4.2 Optimizing the Parameters

With this choice of a model, due to the constraints over the parameters, then the E-step of a EM-type algorithm would be a constrained optimization problem. If $\boldsymbol{\theta} \in \mathbb{R}^T$, then the problem is to find

$$\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta} \in \Theta} f(\boldsymbol{\theta}), \tag{34}$$

where $\Theta = \{\theta \mid l_i \leq \theta_i \leq u_i, i = 1, \ldots, T\}$ is the constraint set and $f(\boldsymbol{\theta})$ is equal to $F(q, p)$ as a function of $\boldsymbol{\theta}$ (i.e., with q held constant). One of the simplest and most general methods for solving this problem is the gradient projection method. The k'th iteration of this method is defined as

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \eta^{(k)}(\bar{\boldsymbol{\theta}}^{(k)} - \boldsymbol{\theta}^{(k)}), \tag{35}$$

where

$$\bar{\boldsymbol{\theta}}^{(k)} = Proj(\boldsymbol{\theta}^{(k)} - s^k \nabla f(\boldsymbol{\theta}^{(k)})). \tag{36}$$

Here, $\eta^{(k)} \in [0, 1]$ and $s^{(k)} \geq 0$ both can be viewed as stepsizes, and $Proj(\boldsymbol{x})$ denotes the projection of $\boldsymbol{x}$ over the constraint set. If the constraint set only consists of a box constraint as in problem (34), then the i'th component of the projection would simply be

$$Proj(\boldsymbol{x})_i = \begin{cases} u_i & \text{if } x_i \geq u_i \\ l_i & \text{if } x_i \leq l_i \\ x_i & \text{otherwise.} \end{cases} \tag{37}$$

We can use many different ways to choose the stepsizes, but the simplest one is to choose constant values, i.e., $\eta^{(k)} = 1$ and $s^{(k)} = s > 0$. It can be shown that if $\nabla f(\boldsymbol{\theta})$ satisfies the Lipschitz continuity condition for some constant $L > 0$, i.e., if we have

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall\ \boldsymbol{x}, \boldsymbol{y} \in \Theta,$$

then if $s < \frac{2}{L}$, every limit point of the gradient projection method is a stationary point (i.e., a local optimum) [7].

4.3 The Unnormalized Log-likelihood Gradient

If we consider $\mathbf{y}$ as the hidden variable and $\mathbf{z}$ as the visible one, then we can use equation (27) as an approximate way to learn the parameters of the model, given an observation (or clamping of) the measurements $\mathbf{z} = \boldsymbol{z}_c$. Specifically, casting (31) in the general form of undirected models in (23), we have

$$p(\boldsymbol{y}, \boldsymbol{z}|\Theta) = \frac{1}{Z(\Theta)}\tilde{p}(\boldsymbol{y}, \boldsymbol{z}|\Theta),$$

and using the potential in (32) in (31), it is easy to see that we have

$$\log \tilde{p}(\boldsymbol{y}, \boldsymbol{z}_c|\Theta) = -\frac{M}{2}\log(2\pi) - M\log\sigma_n - \frac{1}{2\sigma_n^2}\sum_{m_j \in \mathcal{M}}(z_{c_j} - y_{i_j})^2 \tag{38}$$

$$+ \sum_{i \sim j} \log(\frac{\alpha}{\sqrt{2\pi}\sigma_0}\exp(-\frac{(y_i - y_j)^2}{2\sigma_0^2}) + \frac{(1-\alpha)}{\sqrt{2\pi}\sigma_1}\exp(-\frac{(y_i - y_j)^2}{2\sigma_1^2})). \tag{39}$$

Using this equation, it is straightforward to show

$$\partial_{\sigma_n} \log \tilde{p}(\boldsymbol{y}, \boldsymbol{z}_c|\Theta) = -\frac{M}{\sigma_n} + \frac{1}{\sigma_n^3} \sum_{m_j \in \mathcal{M}} (z_{c_j} - y_{i_j})^2 \tag{40}$$

$$\partial_{\alpha} \log \tilde{p}(\boldsymbol{y}, \boldsymbol{z}_c|\Theta) = \sum_{i \sim j} (\alpha + \frac{1}{\frac{\sigma_1}{\sigma_0} \exp(\frac{-(y_i - y_j)^2}{2}(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2})) - 1})^{-1} \tag{41}$$

$$\partial_{\sigma_0} \log \tilde{p}(\boldsymbol{y}, \boldsymbol{z}_c|\Theta) = \sum_{i \sim j} \frac{\frac{\alpha}{\sigma_0}(\frac{(y_i - y_j)^2}{\sigma_0^2} - 1)}{\alpha + (1-\alpha)\frac{\sigma_0}{\sigma_1} \exp(\frac{(y_i - y_j)^2}{2}(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}))} \tag{42}$$

$$\partial_{\sigma_1} \log \tilde{p}(\boldsymbol{y}, \boldsymbol{z}_c|\Theta) = \sum_{i \sim j} \frac{\frac{1-\alpha}{\sigma_1}(\frac{(y_i - y_j)^2}{\sigma_1^2} - 1)}{(1-\alpha) + \alpha\frac{\sigma_1}{\sigma_0} \exp(-\frac{(y_i - y_j)^2}{2}(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}))} \tag{43}$$

5 Numerical Experiments

This section presents a few numerical experiments in which the graphical model proposed in the previous section was used for modeling the graph signal on a few different synthetic and real-world graphs. The approximate inference methods described in section 3.2, and two deterministic methods based on total variation and graph Laplacian minimization were compared in the task of recovering the graph signal from a few random noisy measurements, both in terms of recovery accuracy and time. The main tool used for implementing the experiments was UGM [31]. UGM is a Matlab toolbox for modeling and inference with undirected graphical models. Since it assumes discrete random variables, it requires providing a table for node potentials (i.e. unitary potentials) and another table for edge potentials (i.e. pairwise potentials), which specify the respective probabilities for all the states of the variables (The pseudo-code of the algorithms used are presented in the Appendix.)

However, we assume the graph signal is continuous, and hence we have to discretize (or encode) the variables (the graph signal at each node) when providing the probability tables, and we also need to de-discretize or decode the states when evaluating the inference returned by the various UGM methods. In order to discretize the variables, we specify a range for the values they can take and a quantization step. For example, assume $\mathrm{x} \in [x_{min}, x_{max}]$ and S_q is the quantization step. Then, x has $(x_{max} - x_{min})/S_q$ states, and state i corresponds to the value $\mathrm{x} = x_{min} + (i-1) * S_q$. Discretizing the variables means the probability distributions assigned to the potentials should also be quantized. If $p(\mathrm{x})$ is the probability distribution of x, and $p_d(\mathrm{x}_d)$ is the probability mass function of the discrete version of x, then we can define

$$p_d(\mathrm{x}_d = i) = p(\mathrm{x} < x_{min} + i * S_q) - p(\mathrm{x} < x_{min} + (i-1) * S_q.)$$

Since our model is based on a Gaussian distribution for the measurements and a mixture of Gaussians for other edge potentials, these values are easy to compute using the cumulative distribution function of the normal distribution.

The recovery accuracy was evaluated via the empirical mean of the normalized mean squared error (NMSE). Specifically, if $\boldsymbol{y}$ is the true graph signal and $\hat{\boldsymbol{y}}^{(t)}$ is the recovered signal in the t'th simulation run, then the NMSE of $\hat{\boldsymbol{y}}^{(t)}$ is

$$\hat{\varepsilon}^{(t)} = \frac{\|\boldsymbol{y} - \hat{\boldsymbol{y}}^{(t)}\|_2^2}{\|\boldsymbol{y}\|_2^2},$$

and the empirical mean of the NMSE over T simulation runs is

$$\bar{\varepsilon} = \frac{1}{T} \sum_{t=1}^{T} \hat{\varepsilon}^{(t)}.$$

For comparison, two deterministic recovery methods were also test (c.f., Section 2.4). The first one is based on solving

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y} \in \mathbb{R}^N} \|\boldsymbol{z} - \boldsymbol{A}\boldsymbol{y}\|_2^2 + \lambda \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y}, \tag{44}$$

which can be easily solved via

$$\hat{\boldsymbol{y}} = (\boldsymbol{A}^T \boldsymbol{A} + \lambda \boldsymbol{L})^\dagger \boldsymbol{A}^T \boldsymbol{z}, \tag{45}$$

where † denotes the pseudo-inverse.This method is henceforth referred to as the Laplacian method for simplicity, due to using the Laplacian quadratic form in the regularizer. The second method is based on solving

$$\hat{\boldsymbol{y}} \in \arg\min_{\boldsymbol{y}\in\mathbb{R}^N} \|\boldsymbol{z} - \boldsymbol{A}\boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{M}\boldsymbol{y}\|_1, \tag{46}$$

which is henceforth simply referred to as the TV method, due to using total variation (TV) in the regularizer term. This is an unconstrained convex optimization problem, which was solved using CVX, a Matlab-based package for solving convex programs [18].

Before presenting the experiments, it should be mentioned here that, at first, an algorithm for optimizing the parameters of the graphical model was implemented (based on the procedure discussed in the previous section), using both the projected gradient method and the interior point methods (with gradient descent and the BFGS methods both tried in the inner loop) used as the optimization routine, and both Gibbs sampling and MCMC methods for providing the samples needed for approximating the gradient. However, despite these efforts and experimenting with different parameters of the optimization methods and the starting points for optimizations, no satisfactory and stable results were achieved. This could be due to several reasons, such as the landscape of the objective function being badly behaving, the errors in approximating the samples, and the errors incurred during the optimization procedure. For this reason, the following experiments mostly used heuristic settings for the model parameters.

5.1 Experiments on a Chain Graph

Possibly the simplest graph we can use for experimenting with graph signal recovery is a chain graph. In a chain graph, as the name implies, the nodes are linked together in a linear chain. For this experiment, a 100 node chain graph was considered, where the nodes were divided into clusters of 5 nodes, with the graph signal alternating between values of 2 and 4 in each cluster. From each cluster, one node was chosen at random (i.e., sampling ratio of 0.2) and a noisy measurement was taken from the node, with $\sigma_n = 0.1$. σ_n was assumed known, α was set to 0.8 (i.e., approximately the true ratio of sparse edges in the graph), and σ_0 was fixed at 0.01, since we know all the nodes in a cluster have the same signal. The quantization step was set to 0.05, and the range of the measurements was chosen to be $z_{max} = y_{max} + 5\sigma_n = 4.5$ and $z_{min} = y_{min} - 5\sigma_n = 1.5$.

Table 1 displays the empirical mean and standard deviation of NMSE of the recovered graph signal for a few (approximate) inference methods (with Gibbs sampling run with 1000 burn-in iterations and 1000 sampling iterations) and different values of σ_1, over 100 runs. The sum product (LBP) method seems to generally perform better than other methods, although the lowest NMSE was achieved by the mean field method for $\sigma_1 = 1.5$.

Table 2 shows the results for the Laplacian and TV methods, for different values of the tuning parameter λ. The smallest error achieved with these methods was 0.045 and 0.048 for the Laplacian and TV methods respectively, for $\lambda = 0.1$, which is slightly lower than the smallest error achieved with the approximate inference methods.

Finally, figure 3 shows the mean computational time of all the methods tested. For the Laplacian method, this also includes the time to compute the Laplacian matrix, while for the TV method it includes the time to compute the incidence matrix ($\boldsymbol{M}$). As we can see, with the exception of the Gibbs method, the other approximate inference methods have relatively low convergence times, when compared to the deterministic methods. The high convergence time of the Gibbs method may be due to the fact that sampling each variable independently may be too simplistic for the

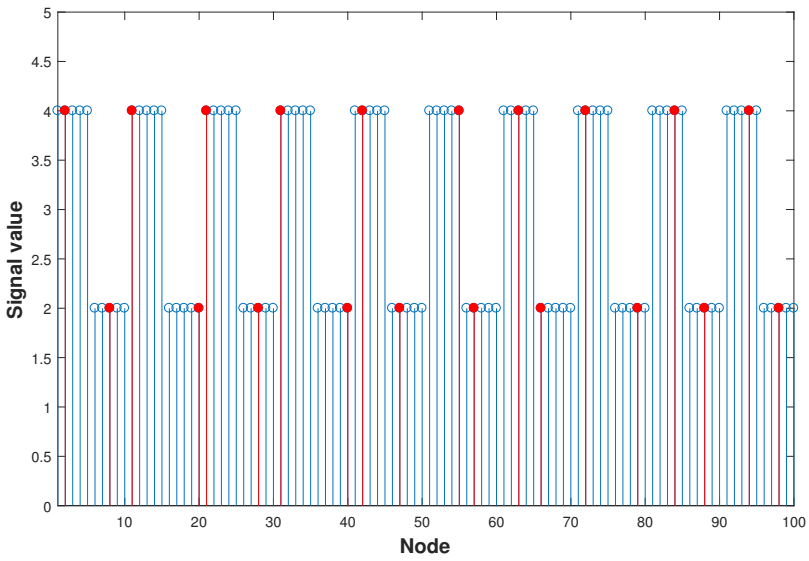

Figure 2: The graph signal defined over the chain graph used in the experiment, with the red circles representing the sampling set.

	σ_1			
	0.5	1	1.5	2
Iterated Conditional Modes	0.152 ± 0.006	0.159 ± 0.002	0.159 ± 0.001	0.159 ± 0.002
Max Product	0.116 ± 0.016	0.128 ± 0.017	0.125 ± 0.018	0.127 ± 0.015
Sum Product	0.055 ± 0.012	0.063 ± 0.013	0.065 ± 0.012	0.067 ± 0.012
Mean Field	0.090 ± 0.016	0.123 ± 0.016	0.022 ± 0.016	0.120 ± 0.014
Gibbs Sampling	0.072 ± 0.012	0.089 ± 0.016	0.094 ± 0.016	0.099 ± 0.017

Table 1: Empirical mean and standard deviation of NMSE of recovered graph signal defined over a chain graph, obtained by different approximate inference methods for different values of σ_1.

distribution here, and so it has a hard time navigating it (c.f., Section 3.2.4). It should be noted however that these times depend on the particular implementation in the software packages used (UGM for the graphical models, and CVX for the TV method), and further optimizations might be possible.

5.2 Experiments on the Stochastic Block Model

For these simulations, in each run, a SBM with 100 nodes and 4 clusters with sizes equal to $10, 20, 30, 40$ was generated, with intra-cluster edge probability $p = 0.3$ and inter-cluster edge probability $q = 0.05$ (an example is shown in Figure 4). α was set to the mean ratio of the number of inter-cluster edges to all the edges, which for the SBM with the given parameters is (approximately) 0.7. For a node i belonging to cluster $\mathcal{C}_i$, the true signal value was chosen as $y_i = b_i + \epsilon_i$, where $b_i \sim U(2, 4)$ is the base signal for cluster $\mathcal{C}_i$, and $\epsilon_i \sim \mathcal{N}(0, 0.05)$. Hence, we fix

	λ			
	0.1	1	10	100
Laplacian	0.045 ± 0.004	0.065 ± 0.003	0.091 ± 0.001	$0.098 \pm 1.56e^{-4}$
TV	0.048 ± 0.004	0.093 ± 0.001	$0.100 \pm 6.6e^{-5}$	$0.100 \pm 6.5e^{-5}$

Table 2: Empirical mean and standard deviation of NMSE of recovered graph signal defined over a chain graph, obtained by the two deterministic recovery methods, for different values of the tuning parameter λ.

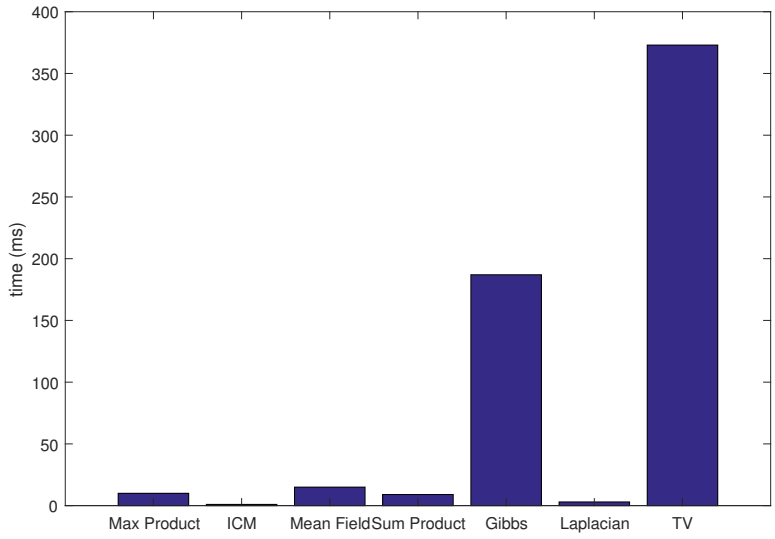

Figure 3: The mean empirical computational time for all the methods tested on the chain graph.

σ_0 at $2 \times 0.05 = 0.1$. The measurement noise σ_n was also set to 0.1, and the range of the signal and the quantization step were the same as the previous experiment.

In each run, after generating the SBM and the true signal value, 20 nodes where chosen uniformly at random as the sampling set, and the measurement vector $\boldsymbol{z}$ was computed according to the Gaussian noise model. Table 3 displays the NMSE of the recovered signal for different methods and values of σ_1. The max product, sum product and mean field methods seem to achieve lower NMSE than the rest, and the smallest NMSE was obtained with the mean field method for $\sigma_1 = 2$. Table 4 displays the results for TV and Laplacian methods. As can be seen from these tables, just as in the case of the chain graph, the smallest errors achieved with these methods is slightly smaller than that of the approximate methods.

The Laplace prior was also tested on the SBM. The same procedure was repeated, except using the Laplace prior for the edge potentials. Table 5 summarizes the result for a few values of the λ parameter of the prior. The errors seem to be slightly lower than in the previous simulations with the two-state mixture model.

Finally, figure 5 shows the mean computational time for all the methods tested on the SBM. As expected the times are higher than those for the chain graph (due to the complexity of the graph), however we can see the same pattern here.

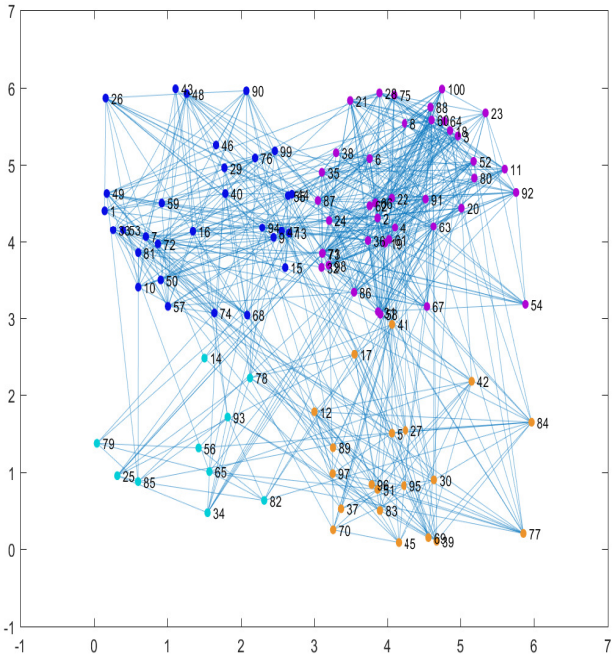

Figure 4: An example of the SBM used for the simulations. Node colors represent the clusters.

	σ_1			
	0.5	1	1.5	2
Iterated Conditional Modes	0.109 ± 0.032	0.110 ± 0.034	0.112 ± 0.033	0.113 ± 0.034
Max Product	0.025 ± 0.022	0.033 ± 0.035	0.030 ± 0.026	0.028 ± 0.027
Sum Product	0.023 ± 0.016	0.030 ± 0.029	0.028 ± 0.024	0.026 ± 0.026
Mean Field	0.025 ± 0.015	0.018 ± 0.014	0.019 ± 0.019	0.017 ± 0.018
Gibbs Sampling	0.027 ± 0.017	0.033 ± 0.026	0.037 ± 0.029	0.042 ± 0.036

Table 3: Empirical mean and standard deviation of NMSE of recovered graph signal defined over a SBM, obtained by different approximate inference methods for different values of σ_1.

	λ			
	0.1	1	10	100
Laplacian	0.019 ± 0.012	0.025 ± 0.019	0.027 ± 0.018	0.026 ± 017
TV	0.026 ± 0.017	0.027 ± 0.020	0.028 ± 0.019	0.026 ± 0.017

Table 4: Empirical mean and standard deviation of NMSE of recovered graph signal defined over a SBM, obtained by the two deterministic recovery methods, for different values of the tuning parameter λ.

	λ			
	0.25	1	4	16
Iterated Conditional Modes	0.120 ± 0.032	0.110 ± 0.035	0.104 ± 0.004	0.107 ± 0.037
Max Product	0.024 ± 0.024	0.025 ± 0.021	0.025 ± 0.030	0.024 ± 0.021
Sum Product	0.019 ± 0.015	0.019 ± 0.013	0.016 ± 0.009	0.021 ± 0.015
Mean Field	0.019 ± 0.015	0.019 ± 0.013	0.016 ± 0.009	0.021 ± 0.014
Gibbs Sampling	0.021 ± 0.016	0.021 ± 0.014	0.027 ± 0.011	0.021 ± 0.016

Table 5: Empirical mean and standard deviation of NMSE of recovered graph signal defined over a SBM, obtained by different approximate inference methods, when utilizing the Laplace prior for edge potentials with different values of the prior parameter λ.

5.3 Experiment on the LFR graph

Most (if not all) real world networks, including social networks, networks of interactions between proteins in a cell, networks in the brain, among many more, besides usually being large, have certain properties which are generally gathered under the umbrella term 'complex networks'. These properties include: (1) High transitivity of the nodes, i.e., the tendency of the neighbors of two nodes to be neighbors with each other, which is usually quantified by the clustering coefficient, a measure of how clustered the graph is. (2) Low average shortest path length between any two nodes, which is referred to as the small-world effect. (3) The distribution of node degrees (i.e., the number of neighbors of a node) follows a power law distribution, i.e., $p(\boldsymbol{k} = k) = Ck^{-\alpha}$, where $\boldsymbol{k}$ denotes the degree of a randomly chosen node, and C and α are constant. Networks with this property are also known as scale-free.

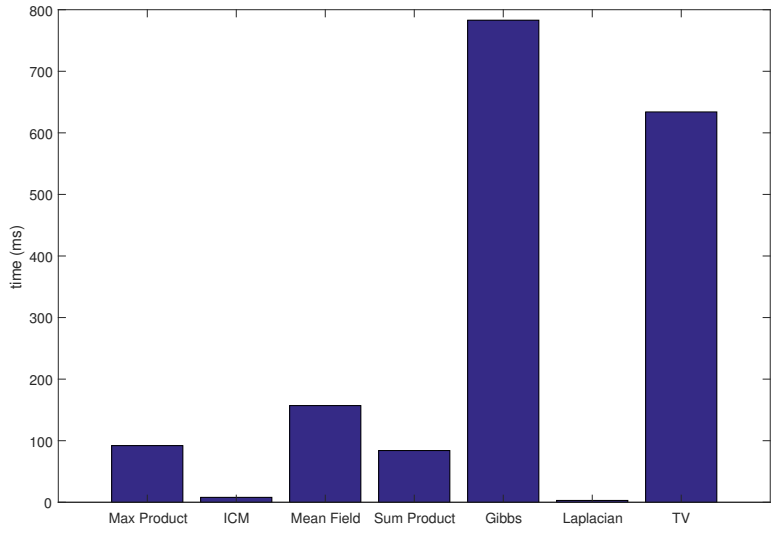

Figure 5: The mean empirical computational time for all the methods tested on the stochastic block model.

The SBM, while possibly the most popular and simplest method for generating synthetic clustered graphs, is not a very good model for complex networks. Most importantly, the degree distribution of each node is a sum of independent binomial distributions, all the nodes have the same expected degree, and the community sizes are fixed a priori. To overcome these shortcomings, the Lancichinetti-Fortunato-Radicci (LFR) model was proposed a few years ago, in which both the node degrees and community sizes follow separate power law distributions, making it a better model of complex networks [21] (There are also other well-known models such as the Barabási-Albert model which follows the power-law distribution but has small average clustering coefficient, and the Watts-Strogatz small-world model has high clustering coefficient and short path lengths but is not scale-free). In addition to the parameters of these distributions, the model has a mixing parameter for determining the proportion of inter-cluster edges to intra-cluster edges, and also provides the option to specify the average expected degree and the maximum degree in the network.

For this experiment, in each run an LFR graph was generated, with the average degree, maximum degree, exponents for the degree and community size distributions, and the mixing parameter set to 7, 15, 2, 1, and 0.3, respectively. The algorithm used automatically outputs the communities (clusters) of the graph, and for generating the graph signal a similar scheme to the one for the SBM was used: For a node i belonging to cluster $\mathcal{C}_i$, the true signal value was set according to $y_i = b_i + \epsilon_i$, where $b_i \sim U(1, 5)$ is the base signal for cluster $\mathcal{C}_i$, and $\epsilon_i \sim \mathcal{N}(0, 0.05)$. In each run, once again, %20 of the nodes were randomly chosen for the sampling set. The mean NMSE over 100 runs were 0.369 ± 0.005, 0.076 ± 0.003, 0.072 ± 0.002, and 0.073 ± 0.003, for the ICM, max product, sum product, and mean field methods respectively. For the deterministic methods, the Laplacian method achieved an NMSE of 0.080 ± 0.003, while the TV method achieved an NMSE of 0.099 ± 0.003. We can see that the max product, sum product, and mean field methods achieve very close and small errors for this synthetic model of a graph signal over a complex network, and

slightly outperform the deterministic methods.

5.4 Experiment on a Real World Dataset

The model was further tested on a real world dataset, namely the Amazon co-purchase dataset, which is a collection of products purchased on the Amazon website during a specific time-period [23]. For each product, it is indicated which other products it is frequently co-purchased with, and what is the average user rating for that product (a number between 0 and 5), which is chosen as the graph signal. First, based on the co-purchase information, an undirected graph was extracted (discarding the products with no co-purchase information). Next, due to computational constraints, a smaller subgraph was chosen by performing a random walk for 2000 iterations, where in each iteration all the neighbors of the randomly chosen node were added to the subgraph. This resulted in a graph with 5227 nodes and 12578 edges. In each simulation run, after randomly choosing $r = 0.2$ of the nodes as the sampling set with the same added measurement noise as in the previous experiment, and setting $\sigma_1 = 2$ while using the same values for the rest of the parameters in the graphical model as in the previous experiment, the previously discussed approximate inference methods were used to recover the graph signal. Figure 6 shows empirical mean and standard deviation of the recovered signal using these methods, over 100 simulation runs. Just as for the synthetic graphs, the mean field and sum product algorithm are the top two performers. The deterministic methods were also tested on this dataset, and the Laplacian method gave an NMSE of 0.341 ± 0.037 while the TV method resulted in an NMSE of 0.421 ± 0.129 (λ was set to 0.1 for both approaches since it resulted in the lowest error for the synthetic graphs). We can see that in this case the mean field and sum product methods actually performed slightly better than the deterministic methods.

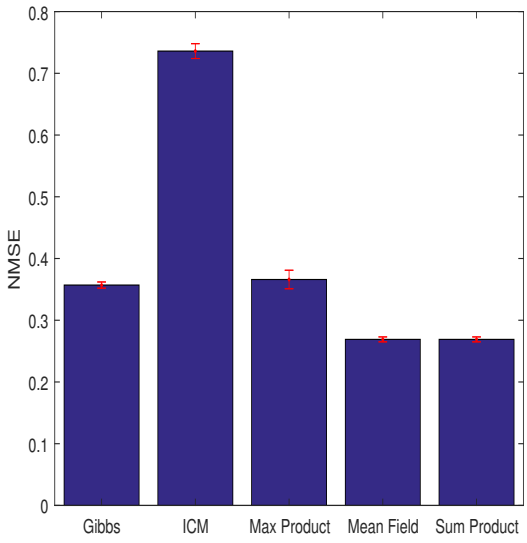

Figure 6: The empirical mean and standard deviation of the NMSE of the signal recovered from the Amazon dataset.

6 Conclusion

This work formulated the problem of learning a smooth, continuous graph signal, or graph-based transductive semi-supervised learning, using a few noisy measurements, in the context of probabilistic graphical models. This translates to a Markov network in which the nodes correspond to random variables representing the signal over all the data points and the provided measurements, and the edges consist of those in the graph representation of the dataset (which represent the similarity between the points), in addition to edges connecting the measurement nodes to the respective nodes where they are sampled from. In this graphical model, prior potentials on the first set of edges enforce the smoothness of the signal, while another set of potentials on the second set of edges enforces the assumption on the noise (such as being Gaussian). Furthermore, approximate methods for learning the graph signal and optimizing the parameters of the model were discussed. Numerical experiments on both synthetic and real data sets demonstrated that, using a relatively small number of noisy samples, these approximate methods can recover the graph signal with a fairly small error (especially for mean field and sum-product methods), which is also very close to the error achieved with deterministic graph signal learning methods. Without quantization (or more accurate quantization) and better setting for parameters, the errors would be even smaller.

There are several avenues for future research on this topic: (1) Designing and implementing an efficient method for optimizing the graphical model parameters. This is a very challenging task, especially for a large graph, due to the need for approximation of the gradients involved, and the nature of the constrained optimization problem. (2) Investigating or designing methods which work directly with continuous variables (such as expectation propagation), without the need for discretizing them, which incurs additional error on top of the main error given by the approximate inference methods. (3) A theoretical or at least comprehensive empirical characterization of the convergence time of the presented inference methods (or others), which could depend on the topology of the data graph. (4) A comprehensive comparison of the presented methods with popular deterministic graph signal learning methods (as introduced in Section 2.4), in terms of accuracy, sensitivity to the number of samples (i.e., efficiency), and convergence time. This comparison would be crucial in determining the real value of graphical models for this task, especially for larger datasets.

7 Appendix

Derivation of marginal log-likelihood gradient in undirected models (Eq. 24):

As in section 3.3, we assume

$$p(\mathbf{h}, \mathbf{v}|\Theta) = \frac{1}{Z(\Theta)}\tilde{p}(\mathbf{h}, \mathbf{v}|\Theta).$$

Hence, we have

$$\begin{aligned}
\partial_\Theta \log p(\boldsymbol{v}|\Theta) &= \partial_\Theta \log \int_{\boldsymbol{h}} \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \partial_\Theta \log Z(\Theta) && (47)\\
&= \frac{1}{\tilde{p}(\boldsymbol{v}|\Theta)} \int_{\boldsymbol{h}} \partial_\Theta \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \frac{1}{Z(\Theta)} \int_{\boldsymbol{h},\boldsymbol{v}} \partial_\Theta \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) && (48)\\
&= \int_{\boldsymbol{h}} \frac{\tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta)}{Z(\Theta)p(\boldsymbol{v}|\Theta)} \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \frac{1}{Z(\Theta)} \int_{\boldsymbol{h},\boldsymbol{v}} \tilde{p}(\boldsymbol{h}, \boldsymbol{v}) \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) && (49)\\
&= \int_{\boldsymbol{h}} \frac{p(\boldsymbol{h}, \boldsymbol{v}|\Theta)}{p(\boldsymbol{v}|\Theta)} \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \int_{\boldsymbol{h},\boldsymbol{v}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta) \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) && (50)\\
&= \int_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}, \Theta) \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) - \int_{\boldsymbol{h},\boldsymbol{v}} p(\boldsymbol{h}, \boldsymbol{v}|\Theta) \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) && (51)\\
&= < \partial_\Theta \log \tilde{p}(\mathbf{h}, \boldsymbol{v}|\Theta) >_{p(\mathbf{h}|\boldsymbol{v},\Theta)} - < \partial_\Theta \log \tilde{p}(\mathbf{h}, \mathbf{v}|\Theta) >_{p(\mathbf{h},\mathbf{v}|\Theta)} . && (52)
\end{aligned}$$

In the third line we have used the fact that $\tilde{p}(\boldsymbol{v}|\Theta) = Z(\Theta)p(\boldsymbol{v}|\Theta)$ and

$$\partial_\Theta \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) = \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta) \partial_\Theta \log \tilde{p}(\boldsymbol{h}, \boldsymbol{v}|\Theta).$$

Algorithms:

Algorithm 1 Augmented data graph and edge potential generation

Input: Data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, sampling set $\mathcal{M}$, measurement vector $\boldsymbol{z}$, model parameters σ_n, σ_0, σ_1, α, quantization range x_{min}, x_{max}, quantization step S_q.

Initialize: $\mathcal{V}' \leftarrow \mathcal{V}$, $\mathcal{E}' \leftarrow \mathcal{E}$, $S \leftarrow \frac{(x_{max}-x_{min})}{S_q} + 1$, $p \leftarrow \{0\}^{(|\mathcal{V}|+|\mathcal{M}|)\times(|\mathcal{V}|+|\mathcal{M}|)\times S\times S}$

for $s_1 \in [1, S]$ **do**
  $x_1 \leftarrow x_{min} + (s_1 - 1)S_q$
  for $s_2 \in [1, S]$ **do**
    $x_2 \leftarrow x_{min} + (s_2 - 1)S_q$
    $e \leftarrow |x_1 - x_2|$
    for $i \in \mathcal{V}$ **do**
      for $j \in \mathcal{V}$ **do**
        $p_{i,j,s_1,s_2} = p_{j,i,s_1,s_2} \leftarrow \alpha(\Phi(e + S_q, 0, \sigma_0) - \Phi(e, 0, \sigma_0)) + (1 - \alpha)(\Phi(e + S_q, 0, \sigma_1) - \Phi(e, 0, \sigma_1))$
      end for
    end for
  end for
end for
$n \leftarrow |\mathcal{V}| + 1$
for $i \in \mathcal{M}$ **do**
  Add node n to $\mathcal{V}'$.
  Add edge $\{i, n\}$ to $\mathcal{E}'$
  for $s_1 \in [1, S]$ **do**
    $z \leftarrow x_{min} + s_1 * S_q$
    for $s_2 \in [1, S]$ **do**
      $x \leftarrow x_{min} + (s_2 - 1)S_q$
      $p_{i,n,s_1,s_2} = p_{n,i,s_1,s_2} \leftarrow \Phi(z, x, \sigma_n) - \Phi(z - S_q, x, \sigma_n)$;
    end for
  end for
  $n \leftarrow n + 1$
end for

Output: $\mathcal{G}', p$

Algorithm 2 Graph Signal Recovery via Iterated Conditional Modes

Input: Data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, sampling set $\mathcal{M}$, measurement vector $\boldsymbol{z}$, σ_n, σ_0, σ_1, α, x_{min}, x_{max}, S_q.

Initialize: Generate the augmented data graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{W}')$ and the edge potential tensor p using Algorithm 1.

$S \leftarrow \frac{(x_{max} - x_{min})}{S_q} + 1$

Initialize $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{V}'|}$ randomly.

for $i \in \mathcal{V}' - \mathcal{V}$ **do**

$y_i = 1 + \lfloor \frac{z_i - x_{min}}{S_q} \rfloor$

end for

repeat

for $i \in \mathcal{V}$ **do**

$q \leftarrow \{\frac{1}{S}\}^S$

for $j \in \mathrm{Ne}(i)$ **do**

$q \leftarrow q \odot p_{i,j,:,y_j}$

end for

$y_i \leftarrow \max q$

end for

until Stopping criterion is satisfied.

Output: $\boldsymbol{y}$

Algorithm 3 Graph Signal Recovery via Loopy Belief Propagation

Input: Data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, sampling set $\mathcal{M}$, measurement vector $\boldsymbol{z}$, σ_n, σ_0, σ_1, α, x_{min}, x_{max}, S_q.

Initialize: Generate the augmented data graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{W}')$ and the edge potential tensor p using Algorithm 1.

$S \leftarrow \frac{(x_{max}-x_{min})}{S_q} + 1$, $m_{i,j,k} \leftarrow \frac{1}{S}$, $q \leftarrow \{0\}^{|\mathcal{V}|\times S}$

repeat

  for $i \in \mathcal{V}$ **do**

    for $j \in \mathrm{Ne}(i)$ **do**

      for $k \in [1, S]$ **do**

        For sum product method: $m_{i,j,k} \leftarrow \sum_{l\in\mathcal{X}} p_{i,j,l,k} \prod_{n\in Ne(i), n\neq j} m_{n,i,l}$

        For max product method: $m_{i,j,k} \leftarrow \max_{l\in\mathcal{X}} p_{i,j,l,k} \prod_{n\in Ne(i), n\neq j} m_{n,i,l}$

      end for

    end for

    Normalize $m_{i,j,:}$

  end for

until Stopping criterion is satisfied.

for $i \in \mathcal{V}' - \mathcal{V}$ **do**

  $j = Ne(i)$

  $s = 1 + \lfloor \frac{z_j - x_{min}}{S_q} \rfloor$

  for $k \in [1, S]$ **do**

    $m_{i,j,k} \leftarrow p_{i,j,s,k}$

  end for

end for

for $i \in \mathcal{V}$ **do**

  for $k \in [1, S]$ **do**

    $q_{i,k} \leftarrow prod_{j\in Ne(i)} m_{j,i,k}$

  end for

  $y_i \leftarrow \max q_{i,:}$

end for

Output: $\boldsymbol{y}$

Algorithm 4 Graph Signal Recovery via Mean Field Method

Input: Data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, sampling set $\mathcal{M}$, measurement vector $\boldsymbol{z}$, σ_n, σ_0, σ_1, α, x_{min}, x_{max}, S_q.

Initialize: Generate the augmented data graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{W}')$ and the edge potential tensor p using Algorithm 1.

$S \leftarrow \frac{(x_{max} - x_{min})}{S_q} + 1$, $q \leftarrow \{0\}^{|\mathcal{V}| \times S}$

repeat

  for $i \in \mathcal{V}$ **do**

    for $j \in \mathrm{Ne}(i)$ **do**

      for $k \in [1, S]$ **do**

        if $j \in \mathcal{V}' - \mathcal{V}$ **then**

          $s = 1 + \lfloor \frac{z_j - x_{min}}{S_q} \rfloor$

          $q_{i,k} \leftarrow q_{i,k} + p_{i,j,k,s}$

        else

          $q_{i,k} \leftarrow q_{i,k} q_{j,:} \log(p_{i,j,k,:})$ ▷ log is applied element-wise.

        end if

      end for

    end for

    Normalize $q_{i,:}$

  end for

until Stopping criterion is satisfied.

for $i \in \mathcal{V}$ **do**

  $y_i \leftarrow \max q_{i,:}$

end for

Output: $\boldsymbol{y}$

Algorithm 5 Graph Signal Recovery via Gibbs Sampling

Input: Data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, sampling set $\mathcal{M}$, measurement vector $\boldsymbol{z}$, σ_n, σ_0, σ_1, α, x_{min}, x_{max}, S_q.

Initialize: Generate the augmented data graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{W}')$ and the edge potential tensor p using Algorithm 1.

$S \leftarrow \frac{(x_{max} - x_{min})}{S_q} + 1$

Initialize $\tilde{\boldsymbol{y}} \in \mathbb{R}^{|\mathcal{V}'|}$ randomly.

for $i \in \mathcal{V}' - \mathcal{V}$ **do**

$\tilde{y}_i = 1 + \lfloor \frac{z_i - x_{min}}{S_q} \rfloor$

end for

repeat

for $i \in \mathcal{V}$ **do**

$q \leftarrow \{\frac{1}{S}\}^S$

for $j \in \mathrm{Ne}(i)$ **do**

$q \leftarrow q \odot p_{i,j,:,\tilde{y}_j}$

end for

Normalize q

$\tilde{y}_i \leftarrow$ Sample from q

end for

Add $\tilde{\boldsymbol{y}}$ to the samples

until Stopping criterion is satisfied.

Discard the first half of the samples as warm-up.

$\boldsymbol{y} \leftarrow$ the most occurring sample

Output: $\boldsymbol{y}$

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] M. R. Andersen, O. Winther, and L. K. Hansen. Bayesian inference for structured spike and slab priors. In *Advances in Neural Information Processing Systems*, pages 1745–1753, 2014.

[3] D. Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[4] D. Baron, S. Sarvotham, and R. G. Baraniuk. Bayesian compressive sensing via belief propagation. *IEEE Transactions on Signal Processing*, 58(1):269–280, 2010.

[5] S. Basirian and A. Jung. Random walk sampling for big data over networks. *arXiv preprint arXiv:1704.04799*, 2017.

[6] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT*, volume 3120, pages 624–638. Springer, 2004.

[7] D. P. Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[8] D. P. Bertsekas and A. Scientific. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[10] V. Cevher, M. F. Duarte, C. Hegde, and R. Baraniuk. Sparse signal recovery using markov random fields. In *Advances in Neural Information Processing Systems*, pages 257–264, 2009.

[11] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. The MIT Press, Cambridge, Massachusetts, 2006.

[12] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[13] G. B. Eslamlou, A. Jung, and N. Goertz. Smooth graph signal recovery via efficient laplacian solvers. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 5915–5919. IEEE, 2017.

[14] B. J. Frey and N. Jojic. A comparison of algorithms for inference and learning in probabilistic graphical models. *IEEE Transactions on pattern analysis and machine intelligence*, 27(9):1392–1416, 2005.

[15] N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303(5659):799–805, 2004.

[16] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*, volume 2. CRC press Boca Raton, FL, 2014.

[17] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[18] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, version 2.0, 2012.

[19] D. Hernández-Lobato, J. M. Hernández-Lobato, and P. Dupont. Generalized spike-and-slab priors for bayesian group feature selection using expectation propagation. *The Journal of Machine Learning Research*, 14(1):1891–1945, 2013.

[20] A. Jung, A. Mara, and S. Jahromi. Semi-supervised learning via sparse label propagation. *arXiv preprint arXiv:1612.01414*, 2016.

[21] A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[22] I. Leichter. The swap and expansion moves revisited and fused. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2264–2271. IEEE, 2009.

[23] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[24] S. Z. Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[25] A. Montanari. Graphical models concepts in compressed sensing. *Compressed Sensing: Theory and Applications*, pages 394–438, 2012.

[26] M. E. J. Newman. *Networks: an Introduction*. Oxford Univ. Press, 2010.

[27] L. Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[28] M. Panić, J. Aelterman, V. Crnojević, and A. Pižurica. Compressed sensing in mri with a markov random field prior for spatial clustering of subband coefficients. In *Signal Processing Conference (EUSIPCO), 2016 24th European*, pages 562–566. IEEE, 2016.

[29] D. Poole. *Linear Algebra: A Modern Introduction*. Cengage Learning, 2014.

[30] A. Sandryhaila and J. M. Moura. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE Signal Processing Magazine*, 31(5):80–90, 2014.

[31] M. Schmidt. UGM: A matlab toolbox for probabilistic undirected graphical models, 2007.

[32] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, May 2013.

[33] E. B. Sudderth, A. T. Ihler, M. Isard, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. *Communications of the ACM*, 53(10):95–103, 2010.

[34] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[35] N. Tran, S. Basirian, and A. Jung. When is network lasso accurate: The vector case. *arXiv preprint arXiv:1710.03942*, 2017.

[36] M. Yasuda, J. Ohkubo, and K. Tanaka. Digital image inpainting based on markov random field. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 747–752. IEEE, 2005.

[37] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[38] X. Zhu and Z. Ghahramani. Towards semi-supervised classification with markov random fields. 2002.